Strong Equivalence for Logic Programs and Default Theories (Made Easy)

Hudson Turner

Computer Science Department
University of Minnesota, Duluth
Duluth, MN 55812, USA
hudson@d.umn.edu

Abstract. Logic programs P and Q are strongly equivalent if, given any logic 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. Recently, Lifschitz, Pearce and Valverde showed that Heyting's logic of here-and-there can be used to characterize strong equivalence of logic programs. This paper offers a more direct characterization, and extends it to default logic. In their paper, Lifschitz, Pearce and Valverde study a very general form of logic programs, called "nested" programs. For the study of strong equivalence of default theories, it is convenient to introduce a corresponding "nested" version of default logic, which generalizes Reiter's default logic.

1 Introduction

Logic programs P and Q are "strongly equivalent" if, given any logic program R, $P \cup R$ and $Q \cup R$ are equivalent. Recent work by Lifschitz, Pearce and Valverde (2001) uses Heyting's logic of here-and-there to characterize strong equivalence of logic programs under the answer set semantics (Gelfond & Lifschitz 1991, Lifschitz et al. 1999). Their proof utilizes Pearce's equilibrium logic (1997, 1999). In the current paper, strong equivalence of logic programs is characterized more directly, in terms of concepts used in the definition of answer sets—no knowledge of the logic of here-and-there is required. This simplifies the proof of the main strong equivalence theorem, and may also make the result easier to apply to specific cases. Moreover, this alternative characterization of strong equivalence is easily extended to default logic (Reiter 1980).

Strong equivalence can help us reason about correctness of logic programs and default theories. For example, as discussed in (Lifschitz et al. 2001), it can be used to establish the fact that in any logic program with a constraint of the form

$$\bot \leftarrow F, G\,,$$

the disjunctive rule

$$F; G \leftarrow \top$$

can be replaced by the pair of rules

$$F \leftarrow not\, G$$
$$G \leftarrow not\, F$$

without affecting the program's answer sets.

Lifschitz, Pearce and Valverde consider strong equivalence for a very general form of logic programs, called "nested" programs (Lifschitz et al. 1999). For the study of strong equivalence of default theories, it is convenient to introduce similarly general "nested" default theories.

Section 2 reviews definitions for nested logic programming. Section 3 states and proves a simple characterization of strong equivalence for logic programs. Section 4 makes precise the relationship between our strong equivalence theorem and that obtained using the logic of here-and-there. Section 5 briefly investigates strongly equivalent transformations of logic programs. Taking advantage of the strong similarities between definitions for logic programming and default logic, Section 6 introduces "nested" default logic, and shows that it extends both nested logic programming and disjunctive default logic (Gelfond et al. 1991), which in turn extends Reiter's default logic. Section 7 states a characterization of strong equivalence for nested default theories similar to that for nested logic programs. Section 8 briefly investigates strongly equivalent transformations of default theories. Proofs related to nested default logic appear in Section 9.

2 Nested Logic Programming

This paper employs the definition of logic programs from (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"). *NLP formulas* are built from elementary formulas using the unary connective *not* and the binary connectives , (conjunction) and ; (disjunction). An *NLP rule* is an expression of the form

$$F \leftarrow G$$

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

A *nested logic program* is a set of NLP rules.

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

2.2 Semantics

Let us first define recursively when a consistent set X of literals *satisfies* an NLP 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 $X \models G$.
- $X \models not\, F$ iff $X \not\models F$.

A consistent set X of literals is *closed under* a program P if, for every rule $F \leftarrow G$ in P, $X \models F$ whenever $X \models G$.

The *reduct* of a formula F relative to a consistent set X of literals (written F^X) is obtained by replacing every maximal occurrence in F of a formula of the form *not* G with $\bot$ if $X \models G$ and with $\top$ otherwise. The *reduct* of a 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.

A consistent set X of literals is an *answer set* for P if it is minimal among the consistent sets of literals closed under 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 Logic Programs

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

The following terminology is convenient. For program P, and consistent sets X, Y of literals with $X \subseteq Y$, call the pair (X, Y) an *HT-model* of P if both X and Y are closed under P^Y.

In Section 4, we will see that HT-models correspond to models in the logic of here-and-there.

Theorem 1. *Logic programs are strongly equivalent iff they have the same HT-models.*

Proof. Right to left: Assume that programs P and Q have the same HT-models. Take any program R. We need to show that $P \cup R$ and $Q \cup R$ are equivalent. Assume that X is an answer set for $P \cup R$. That is, X is a consistent set of literals closed under $(P \cup R)^X$, and no proper subset of X is closed under $(P \cup R)^X$. Since $(P \cup R)^X = P^X \cup R^X$, X is closed under both P^X and R^X. Since X is closed under P^X, it follows by assumption that X is closed under Q^X. So X is closed under $Q^X \cup R^X = (Q \cup R)^X$. Suppose a proper subset of X is closed under $(Q \cup R)^X$. Then it is closed under both Q^X and R^X. By assumption it is also closed under P^X, and thus under $(P \cup R)^X$, contradicting the choice of X. We conclude that every answer set for $P \cup R$ is an answer set for $Q \cup R$. By symmetry, every answer set for $Q \cup R$ is an answer set for $P \cup R$.

Left to right: Assume (wlog) that (X, Y) is an HT-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 is not closed under Q^Y. Then Y is not closed under $(Q \cup Y)^Y = Q^Y \cup Y$, and so is not an answer set for $Q \cup Y$. On the other hand, one easily verifies that Y is an answer set for $P \cup Y$. Hence P and Q are not strongly equivalent.

Case 2: Y is closed under Q^Y. Take $R = X \cup \{F \leftarrow G : F, G \in Y \setminus X\}$. Clearly Y is closed under $(Q \cup R)^Y$. Let Z be a subset of Y closed under $(Q \cup R)^Y = Q^Y \cup R$. By choice of R, $X \subseteq Z$, and by assumption X is not closed under Q^Y, so $X \neq Z$. Hence some $L \in Y \setminus X$ 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 is closed under $(P \cup R)^Y = P^Y \cup R$. So Y is not an answer set for $P \cup R$, and we conclude again that P and Q are not strongly equivalent.

Although simpler (due to simpler definitions), this proof resembles in many details the proof of the main theorem in (Lifschitz et al. 2001), including the fact that it demonstrates that if logic programs P and Q are not strongly equivalent then they can be distinguished by adding a logic program in which both the head and body of each rule is a literal.

4 HT-Models and the Logic of Here-and-There

Lifschitz, Pearce and Valverde identify logic program rules with formulas in the logic of here-and-there, and show that programs are strongly equivalent iff they are equivalent in the logic of here-and-there.

They consider nested programs, as described in Section 2, except that they do not allow classical negation. (That is, their programs do not contain the symbol $\neg$.) Accordingly, they define answer sets using sets of atoms in place of consistent sets of literals. For convenience, the term "stable model" will be used to refer to an answer set in their sense.

After establishing their strong equivalence theorem (with respect to stable models) for nested programs without classical negation, they explain that the result can be extended to all nested programs as follows. Take any nested program P. For each atom A in the language of P, add a new atom A', and let P' be the program in this extended language obtained by (i) replacing each occurrence of each negative literal $\neg A$ with atom A', and (ii) adding the rule $\bot \leftarrow A, A'$ for every new atom A'. The answer sets for P are in one-to-one correspondence with the stable models of P'. More precisely, given any set X of literals, let X' be obtained by replacing each negative literal $\neg A \in X$ by A'. Then X is an answer set for P iff X' is a stable model of P'.

It follows that nested programs P and Q are strongly equivalent (in the sense of this paper) iff P' and Q' are strongly equivalent wrt stable models. Moreover, for any nested programs P and Q without classical negation, P and Q are strongly equivalent wrt stable models iff P' and Q' are.

In (Lifschitz et al. 2001), 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 logic program in the sense of the logic of here-and-there. Although it is not done here, one can easily verify that their Lemmas 1 and 2 together imply the following.

Proposition 1. *For any nested logic program P, (X, Y) is an HT-model of P (as defined in this paper) iff (X', Y') is a model of P' in the logic of here-and-there.*

So these approaches are essentially equivalent with regard to logic programs. Each has advantages.

The primary advantage of the approach introduced here is its relative simplicity. The definition of HT-model 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. Moreover, the (relatively) simple definition can make the theorem easier to apply to specific cases.

The definition introduced in this paper takes advantage of the special status of the symbol $\leftarrow$ in 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 establish strong equivalence in particular cases.

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

5 Equivalent Transformations of Logic Programs

To demonstrate the use of Theorem 1, let us consider again the example from the introduction: for any NLP formulas F and G, programs P_1 and P_2 below have the same HT-models.

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

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

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

When strong equivalence is characterized using the logic of here-and-there, we immediately obtain a replacement theorem: strong equivalence is preserved under substitution of formulas that are equivalent in the logic of here-and-there. And of course it follows that if formulas F and G are satisfied by the same (here-and-there) models of a program P, then, for any program Q, occurrences of F in Q can be replaced by G without affecting the answer sets of $P \cup Q$. One can provide a similar facility using HT-models. Let us begin with two definitions.

We say that NLP formulas F and G are *equivalent* relative to logic program P if, for every HT-model (X, Y) of P, $X \models F^Y$ iff $X \models G^Y$.

An occurrence of a formula is *regular* if it is not an atom preceded by $\neg$.

Theorem 2. *Let P be a program, and let F and G be formulas equivalent relative to P. For any program Q, and any 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 2 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. Although it is not done here, a proof of Theorem 2 can be easily constructed based on the corresponding proof from the earlier paper. (Section 9 does include a similar proof—of the corresponding theorem for "nested" default logic.) Alternatively, just as Proposition 1 related the HT-models (X, Y) of a program P with the models (X', Y') of program P' under the logic of here-and-there, one can show that NLP formulas F and G are equivalent relative to program P iff the corresponding formulas F' and G' are satisfied by the same models of P' in the logic of here-and-there.

Many formula equivalences are proved in (Lifschitz et al. 1999, Proposition 4), and of course they also hold under our (weaker) definition (relative to the empty program). Thus, Theorem 2 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 2, 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.

6 Nested Default Logic

For the study of strong equivalence of default theories, it is convenient to introduce a "nested" version of default logic that generalizes disjunctive default logic (Gelfond et al. 1991), which in turn generalizes Reiter's default logic.

The relatively uniform syntax of nested default logic will make it more convenient for stating and using strong equivalence results. (We don't have to deal separately with a prerequisite and a set of justifications—they are expressed in a single formula.)

As one might expect, the definitions for nested default logic are almost exactly as for nested logic programs—essentially, allow arbitrary formulas of classical logic in place of literals, and use consistent, logically closed sets of formulas in place of consistent sets of literals. Accordingly, the strong equivalence theorem (and its proof!) is nearly identical too.

6.1 Syntax

Let us say *classical formula* to mean a formula of classical propositional logic.

NDL formulas are built from classical formulas using the unary connective *not* (negation as failure) and the binary connectives $|$ (strong disjunction) and $\wedge$ (conjunction). (There is no need for a distinct "strong conjunction" connective.) An *NDL rule* is an expression of the form

$$\frac{F}{G}$$

where F and G are NDL formulas, called the *condition* and the *conclusion* of the rule.

A *nested default theory* is a set of NDL rules.

When convenient, a rule of the form $\frac{\top}{F}$ will be identified with formula F.

6.2 Semantics

Let us use the term *candidate set* for a consistent set of classical formulas that is closed under classical propositional logic.

We can recursively define when a candidate set X *satisfies* an NDL formula F (symbolically, $X \models F$), as follows.

- For classical F, $X \models F$ iff $F \in X$.
- $X \models (F \wedge G)$ 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 $X \not\models F$.

A candidate set X is *closed under* a default theory P if, for every rule $\frac{F}{G}$ in P, $X \models F$ implies $X \models G$.

The *reduct* of an NDL formula F relative to a candidate set X (written F^X) is obtained by replacing every maximal occurrence in F of a formula of the

form *not* G with $\bot$ if $X \models G$ and with $\top$ otherwise. The reduct of a default theory P relative to X (written P^X) is obtained by replacing the condition and conclusion of each rule in P by their reducts relative to X.

A candidate set X is an *extension* of P if it is minimal among the candidate sets closed under P^X.

6.3 Relation to (Nested) Logic Programming

Essentially, nested logic programming is a special case of nested default logic. Every NLP formula F corresponds to the NDL formula $d(F)$ obtained by replacing occurrences of the connectives ; and , with | and $\wedge$ respectively. A nested logic program corresponds to the default theory obtained by replacing each NLP rule $F \leftarrow G$ with $\frac{d(G)}{d(F)}$. A consistent set of literals corresponds to the candidate set whose formulas are its consequences (in classical logic).

Proposition 2. *The answer sets for any nested logic program correspond to the extensions of the corresponding nested default theory.*

6.4 Relation to (Disjunctive) Default Logic

Nested default logic generalizes disjunctive default logic (Gelfond et al. 1991), which in turn generalizes Reiter's default logic. Here we review the definition of disjunctive default logic and relate it to nested default logic.

A *disjunctive default rule* is an expression of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma_1 | \cdots | \gamma_n} \tag{1}$$

where $\alpha, \beta_1, \ldots, \beta_m, \gamma_1, \ldots, \gamma_n$ are classical formulas ($m \geq 0, n \geq 1$). Reiter's default logic corresponds to the special case when $n = 1$.[1]

A disjunctive default rule (1) corresponds to the NDL rule

$$\frac{\alpha \wedge \mathit{not}\, \neg\beta_1 \wedge \cdots \wedge \mathit{not}\, \neg\beta_m}{\gamma_1 | \cdots | \gamma_n}.$$

A *disjunctive default theory* is a set of disjunctive default rules.

Let P be a disjunctive default theory and X a set of classical formulas. Define

$$P^X = \left\{ \frac{\alpha :}{\gamma_1 | \cdots | \gamma_n} \ : \ \frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma_1 | \cdots | \gamma_n} \in P \text{ and } \neg\beta_1, \ldots, \neg\beta_m \notin X \right\}.$$

A set Y of classical formulas is *closed under* P^X if, for every member of P^X, if $\alpha \in Y$ then at least one of $\gamma_1, \ldots, \gamma_n$ belongs to Y.

We say X is an *extension* of P if X is minimal among sets of formulas closed under propositional logic and closed under P^X.

[1] In Reiter's formulation, a default theory is a pair (P, W), where the second component W is a set of classical formulas. Here we suppress the second component, since every $\phi \in W$ can be equivalently represented in P by the rule $\frac{\top :}{\phi}$.

Proposition 3. *A candidate set X is an extension of a disjunctive default theory P iff it is an extension of the corresponding nested default theory.*

Proposition 3 restricts attention to candidate sets (which are by definition consistent) because, unlike nested default logic, disjunctive default logic allows for the possibility of an inconsistent extension.

7 Strong Equivalence of Default Theories

Nested default theories P and Q are *equivalent* if they have the same extensions. They are *strongly equivalent* if, for any nested default theory R, $P \cup R$ and $Q \cup R$ are equivalent.

For nested default theory P, and candidate sets X, Y with $X \subseteq Y$, the pair (X, Y) is an *HT-model* of P if both X and Y are closed under P^Y.

Theorem 3. *Nested default theories are strongly equivalent iff they have the same HT-models.*

A proof of Theorem 3 is easily obtained from the proof of Theorem 1, and so is not presented in this paper. (Essentially, replace references to "consistent sets of literals" with references to "candidate sets.")

The proof shows that any two nested default theories that are not strongly equivalent can be distinguished by adding a nested default theory in which the conditions and conclusions of all rules are classical formulas.

8 Equivalent Transformations of Default Theories

As with logic programs (using Theorem 1), Theorem 3 can be used, for example, to show that in any default theory containing the rule

$$\frac{F \wedge G}{\bot},$$

the rule

$$\frac{\top}{F \mid G}$$

can be replaced by the rules

$$\frac{not\ F}{G}, \frac{not\ G}{F}.$$

Moreover, it is clear that replacing any occurrence of one classical formula with another that is logically equivalent (in classical logic) yields a strongly equivalent default theory.

We can formulate an additional replacement theorem, similar to Theorem 2 for logic programming, thus extending our account of when an occurrence of one formula may be safely replaced by another. Again we need some definitions first.

We say that NDL formulas F and G are *equivalent* relative to nested default theory P if, for every HT-model (X, Y) of P, $X \models F^Y$ iff $X \models G^Y$.

An occurrence of a subformula in an NDL formula is called *regular* if it is not a proper subpart of an occurrence of a subformula formed by an application of $\neg$ or $\vee$.

Theorem 4. *Let P be a nested default theory, and let F and G be formulas equivalent relative to P. For any nested default theory Q, and any nested default theory Q' obtained from Q by replacing some regular occurrences of F by G, nested default theories $P \cup Q$ and $P \cup Q'$ are strongly equivalent.*

As with Theorem 2, the restriction to regular occurrences is essential. (And essentially the same example shows this.)

Theorem 4 can be used to show, for example, that in any nested default theory with rules

$$\frac{not\ F}{\neg F}, \frac{not\ \neg F}{F}$$

any occurrence of NDL formula $F|G$ (for any classical formula G) can be safely replaced with $F \vee G$.

9 Proofs Related to Nested Default Logic

Proposition 2. *The answer sets for any nested logic program correspond to the extensions of the corresponding nested default theory.*

For any candidate set X, let $l(X)$ denote the set of all literals in X.

Lemma 1. *For any NLP formula F and candidate set X, $l(X) \models F$ iff $X \models d(F)$.*

Proof. Straightforward, by structural induction.

Lemma 2. *For any NLP formula F and candidate set X, $d(F^{l(X)}) = d(F)^X$.*

Proof. Follows easily from Lemma 1 and the definitions.

Lemma 3. *For any nested logic program P and candidate sets X and Y, $l(X)$ is closed under $P^{l(Y)}$ iff X is closed under $d(P)^Y$.*

Proof. Follows easily from Lemmas 1 and 2, and the definitions.

Proof of Proposition 2: Take any nested logic program P. Assume X is an answer set for P. So X is a consistent set of literals closed under P^X, and no proper subset of X is closed under P^X. Let Y be the candidate set corresponding to X. By Lemma 3, Y is closed under $d(P)^Y$. Suppose a candidate set Z with $Z \subseteq Y$ is closed under $d(P)^Y$. By Lemma 3, $l(Z)$ is closed under P^X. Since $Z \subseteq Y$, $l(Z) \subseteq X$. We conclude by choice of X that $l(Z) = X$. It follows that $Z = Y$. So Y is an extension of $d(P)$. Proof in the other direction is similar.

Proposition 3. *A candidate set X is an extension of a disjunctive default theory P iff it is an extension of the corresponding nested default theory.*

Proof. It is clear that for any disjunctive default theory P and corresponding nested default theory Q, and any candidate sets X and Y, X is closed under P^Y iff X is closed under Q^Y, from which the result follows.

Theorem 4. *Let P be a nested default theory, and let F and G be formulas equivalent relative to P. For any nested default theory Q, and any nested default theory Q' obtained from Q by replacing some regular occurrences of F by G, nested default theories $P \cup Q$ and $P \cup Q'$ are strongly equivalent.*

The proof of Theorem 4 is very similar to the proof of Proposition 3 from (Lifschitz et al. 1999), and illustrates how a proof of Theorem 2 might go.

We begin with an easily verified lemma.

Lemma 4. *For any NDL formula F and candidate set X, $X \models F$ iff $X \models F^X$.*

Lemma 5. *Let F and G be NDL formulas equivalent relative to nested default theory P. If an NDL formula H' can be obtained from an NDL formula H by replacing some regular occurrences of F by G, then H and H' are equivalent relative to P.*

Proof. Consider any HT-model (X, Y) of P. We need to show that $X \models H^Y$ iff $X \models (H')^Y$. Proof is by structural induction on H.

Case 1: H is an atom or $H = \neg H_1$ or $H = H_1 \vee H_2$. Then the only regular occurrence of a formula in H is H itself. Consequently $H = F$ and $H' = G$, and we're done.

Case 2: $H = H_1 \wedge H_2$. If $H = F$ and $H' = G$ we're done. Otherwise, $H' = H_1' \wedge H_2'$ and, by the induction hypothesis, H_1 and H_1' are equivalent relative to P, as are H_2 and H_2'. Then

$$\begin{array}{rl} X \models H^Y & \text{iff } X \models (H_1 \wedge H_2)^Y \\ & \text{iff } X \models H_1^Y \wedge H_2^Y \\ & \text{iff } X \models H_1^Y \text{ and } X \models H_2^Y \\ & \text{iff } X \models (H_1')^Y \text{ and } X \models (H_2')^Y \\ & \text{iff } X \models (H_1')^Y \wedge (H_2')^Y \\ & \text{iff } X \models (H_1' \wedge H_2')^Y \\ & \text{iff } X \models (H')^Y. \end{array}$$

Case 3: $H = H_1 | H_2$. Similar to Case 2.

Case 4: $H = not\, H_1$. If $H = F$ and $H' = G$ we're done. Otherwise, $H' = not\, H_1'$ and, by the induction hypothesis, H_1 and H_1' are equivalent relative to P. Assume that $X \models (not\, H_1)^Y$. Then $(not\, H_1)^Y = \top$, so $Y \not\models H_1$. It follows by Lemma 4 that $Y \not\models H_1^Y$. Since (X, Y) is an HT-model of P, so is (Y, Y). Since H_1 and H_1' are equivalent relative to P, we can conclude that $Y \not\models (H_1')^Y$. By Lemma 4, $Y \not\models H_1'$. So $(not\, H_1')^Y = \top$, and thus $X \models (not\, H_1')^Y$. The other direction is symmetric.

Proof of Theorem 4: Assume that Q' can be obtained from Q by replacing some regular occurrences of F by a formula G that is equivalent relative to P. We must show that $P \cup Q$ and $P \cup Q'$ have the same HT-models.

Consider any HT-model (X, Y) of P. It is enough to show that both X and Y are closed under Q^Y iff both are closed under $(Q')^Y$. So consider any rule $\frac{H_1}{H_2} \in Q$, along with the corresponding rule $\frac{H_1'}{H_2'} \in Q'$. By Lemma 5, $X \models (H_1)^Y$ iff $X \models (H_1')^Y$, and similarly $X \models (H_2)^Y$ iff $X \models (H_2')^Y$. We conclude that X is closed under Q^Y iff it is closed under $(Q')^Y$. Since (Y, Y) is also an HT-model of P, the same argument shows that Y is closed under Q^Y iff it is closed under $(Q')^Y$.

Acknowledgements

I am grateful to Vladimir Lifschitz and Michael Gelfond for a number of helpful comments. This work partially supported by NSF Career Grant #0091773.

References

Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

Michael Gelfond, Vladimir Lifschitz, Halina Przymusińska, and Miroslaw Truszczyński. Disjunctive defaults. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proc. of the 2nd Int'l Conference*, pages 230–237, 1991.

Vladimir Lifschitz, L.R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(2–3):369–390, 1999.

Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, To appear, 2001.

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.

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

Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132, 1980.