Strongly Equivalent Logic Programs

Vladimir Lifschitz
University of Texas at Austin, USA

David Pearce
DFKI, Saarbrücken, Germany

Agustín Valverde
University of Málaga, Spain

Abstract

A logic program Π_1 is said to be equivalent to a logic program Π_2 in the sense of the answer set semantics if Π_1 and Π_2 have the same answer sets. We are interested in the following stronger condition: for every logic program Π, $\Pi_1 \cup \Pi$ has the same answer sets as $\Pi_2 \cup \Pi$. The study of strong equivalence is important, because we learn from it how one can simplify a part of a logic program without looking at the rest of it. The main theorem shows that the verification of strong equivalence can be accomplished by checking the equivalence of formulas in a monotonic logic, called the logic of here-and-there, which is intermediate between classical logic and intuitionistic logic.

1 Introduction

This paper is about logic programs with negation as failure under the answer set ("stable model") semantics [Gelfond and Lifschitz, 1988]. A program Π_1 is said to be equivalent to a program Π_2 if Π_1 and Π_2 have the same answer sets. We are interested here in the following stronger condition: *for every program* Π, $\Pi_1 \cup \Pi$ *is equivalent to* $\Pi_2 \cup \Pi$.

Consider, for instance, the one-rule programs $p \leftarrow not\ q$ and p. These programs have the same answer set $\{p\}$, but they are not equivalent in the strong sense: if we add the rule q to each of the two programs, the answer set for the first will become $\{q\}$, and the answer set for the second will become $\{p, q\}$. On the other hand, the program

$$\begin{array}{l} p \leftarrow q\ , \\ q \end{array}$$

is strongly equivalent to

$$\begin{array}{l} p\ , \\ q\ . \end{array}$$

The rule $p \leftarrow q, not\ q$ is strongly equivalent to the empty set: removing this rule from a program does not affect the program's answer sets.

There is an interesting analogy between the strong equivalence of logic programs and equivalence of propositional formulas. In each of the examples above, let's think of rules as propositional formulas: replace every *not* with $\neg$, every comma with $\wedge$, and turn every rule

$$Head \leftarrow Body$$

into the implication

$$Body \rightarrow Head.$$

The programs in the first example above turn into

$$\{\neg q \rightarrow p\} \text{ and } \{p\}$$

—two sets of formulas that are not equivalent in propositional logic. On the other hand, the second example turns into a pair of equivalent sets of formulas

$$\{q \rightarrow p, q\} \text{ and } \{p, q\}.$$

And so does the third example:

$$\{q \wedge \neg q \rightarrow p\} \text{ and } \emptyset$$

are propositionally equivalent.

This analogy is not complete, however. In propositional logic, $\neg p \rightarrow q$ is equivalent to $\neg q \rightarrow p$, but the rules $q \leftarrow not\ p$ and $p \leftarrow not\ q$ are not strongly equivalent; they are not even equivalent in the weak sense.

In connection with the last example one might remember that *intuitionistically* the formulas $\neg p \rightarrow q$ and $\neg q \rightarrow p$ are *not equivalent*. Could intuitionistic logic be a better tool for the study of strong equivalence than classical logic?

Using intuitionistic logic for this purpose is indeed a good idea, but even better results will be achieved if we use a stronger subsystem of classical propositional logic—"the logic of here-and-there," HT. The main theorem of this paper shows that there is a perfect match between the logic of here-and-there and the strong equivalence of logic programs: two programs are strongly equivalent if and only if these programs, viewed as sets of propositional formulas, are equivalent in HT.

The logic of here-and-there can be defined in terms of 3-valued truth tables. These truth tables were originally introduced by Heyting [1930] as a technical device, for the purpose of demonstrating that intuitionistic logic is weaker than classical. Heyting remarks that the truth values in these tables "can be interpreted as follows: 0 denotes a correct proposition, 1 denotes a false proposition, and 2 denotes a proposition that cannot be false but whose correctness is not proved." The proof of the fact that intuitionistic logic cannot be described by a finite set of truth values in [Gödel, 1932] uses an infinite monotonically decreasing sequence of systems whose first member is classical logic, and whose second member happens to be HT. The logic of here-and-there is known also as "the logic of present and future" or "the Smetanich logic." It was apparently first axiomatised in [Łukasiewicz, 1941]. The importance of HT for the study of logic programming can be seen from the results of Pearce [1997, 1999].

Since HT is a 3-valued logic, there is an exponential time algorithm for deciding whether two propositional formulas are equivalent in HT. Consequently, the strong equivalence of logic programs can be verified in exponential time.

Interesting questions about the strong equivalence of programs arise in disjunctive logic programming. Compare the disjunctive rule

$$p; q \tag{1}$$

with the program

$$\begin{aligned} p &\leftarrow \mathit{not}\ q\ , \\ q &\leftarrow \mathit{not}\ p\ . \end{aligned} \tag{2}$$

Each of these two programs has the answer sets $\{p\}$, $\{q\}$, but (1) and (2) are not strongly equivalent: the program obtained by adding the rules

$$\begin{aligned} p &\leftarrow q\ , \\ q &\leftarrow p \end{aligned} \tag{3}$$

to (1) has the answer set $\{p, q\}$, which is not an answer set for the program obtained by adding the same rules to (2). On the other hand, the program

$$\begin{aligned} &p; q\ , \\ &\bot \leftarrow p, q \end{aligned} \tag{4}$$

is strongly equivalent to

$$\begin{aligned} p &\leftarrow \mathit{not}\ q\ , \\ q &\leftarrow \mathit{not}\ p\ , \\ \bot &\leftarrow p, q\ . \end{aligned} \tag{5}$$

This fact illustrates the possibility of eliminating "exclusive disjunctions" from a logic program.

Although program (1) is disjunctive, the counterexample (3) proving that (1) and (2) are not strongly equivalent does not contain disjunctive rules. It does not even contain negation as failure. We'll see that this is a general phenomenon: if there exists a program Π such that $\Pi_1 \cup \Pi$ is not equivalent to $\Pi_2 \cup \Pi$ then we can always find a program Π with this property such that the rules of Π are syntactically very simple. Furthermore, we'll show that the relationship between logic HT and the strong equivalence of logic programs discussed above holds in the presence of disjunctive rules, if we agree to identify the semicolon with $\vee$. Written as sets of propositional formulas, (4) and (5) become

$$\{p \vee q, \neg(p \wedge q)\} \tag{6}$$

and

$$\{\neg p \to q, \neg q \to p, \neg(p \wedge q)\}. \tag{7}$$

These two sets are equivalent in the logic of here-and-there (although they are not equivalent in the weaker intuitionistic logic).

Several published results on the answer set semantics address, implicitly, the topic of strong equivalence. Theorem 1 from [Erdem and Lifschitz, 1999], discussed in Section 5 below, is similar to the assertion about the strong equivalence between programs (4) and (5). Corollary 4.10 from [Inoue and Sakama, 1998] gives an example of strong equivalence for programs with negation as failure in the heads of rules. Theorems stated in Section 4 of [Lifschitz *et al.*, 1999] provide numerous examples of strongly equivalent programs with nested expressions in the heads and bodies of rules. We'll return to some of these examples in Section 4.2.

The study of strong equivalence is important, because we learn from it how one can simplify a part of a logic program without looking at the rest of it.

In the next two sections, we review the logic of here-and-there and the answer set semantics of logic programs. In Section 4 we state and prove the main theorem and give examples of its use. In Section 5 the main theorem is extended to programs with two kinds of negation.

2 Logic of Here-and-There

Propositional formulas are built from propositional atoms and the 0-place connective $\bot$ using the binary connectives $\wedge$, $\vee$ and $\to$. We write $\top$ for $\bot \to \bot$, and $\neg F$ for $F \to \bot$. A *theory* is a set of propositional formulas.

Recall that, in classical propositional logic, interpretations can be viewed as sets of atoms. The satisfaction relation between an interpretation I and a formula F can be then defined recursively, as follows:

- for an atom p, $I \models p$ if $p \in I$,
- $I \not\models \bot$,
- $I \models F \wedge G$ if $I \models F$ and $I \models G$,
- $I \models F \vee G$ if $I \models F$ or $I \models G$,
- $I \models F \rightarrow G$ if $I \not\models F$ or $I \models G$.

A *model* of a theory Γ in the sense of classical logic is an interpretation that satisfies every formula in Γ.

2.1 Semantics

An *HT-interpretation* is an ordered pair $\langle I^H, I^T \rangle$ of sets of atoms such that $I^H \subseteq I^T$. Intuitively, such a pair describes "two worlds": the atoms in I^H are true "here," and the atoms in I^T are true "there." Accordingly, we call the symbols H and T *worlds*.[1] The worlds are ordered by $H < T$.

For any HT-interpretation $\langle I^H, I^T \rangle$, any world w, and any formula F, we define when the triple $\langle I^H, I^T, w \rangle$ *satisfies* F recursively, as follows:

- for any atom F, $\langle I^H, I^T, w \rangle \models F$ if $F \in I^w$,
- $\langle I^H, I^T, w \rangle \not\models \bot$,
- $\langle I^H, I^T, w \rangle \models F \wedge G$ if $\langle I^H, I^T, w \rangle \models F$ and $\langle I^H, I^T, w \rangle \models G$,
- $\langle I^H, I^T, w \rangle \models F \vee G$ if $\langle I^H, I^T, w \rangle \models F$ or $\langle I^H, I^T, w \rangle \models G$,
- $\langle I^H, I^T, w \rangle \models F \rightarrow G$ if, for every world w' such that $w \leq w'$, $\langle I^H, I^T, w' \rangle \not\models F$ or $\langle I^H, I^T, w' \rangle \models G$.

We say that an HT-interpretation $\langle I^H, I^T \rangle$ *satisfies* F if $\langle I^H, I^T, H \rangle$ satisfies F. A *model* of a theory Γ in the sense of logic HT is an HT-interpretation that satisfies every formula in Γ.

A formula F is a *consequence* of a set Γ of formulas in logic HT (symbolically, $\Gamma \models_{HT} F$) if every model of Γ in the sense of HT satisfies F also. We say that Γ is *equivalent* to Δ in the sense of HT if Γ and Δ have the same models in the sense of HT.

An HT-interpretation $\langle I^H, I^T \rangle$ is said to be *total* if $I^H = I^T$.

Every consequence of Γ in the logic of here-and-there is a consequence of Γ in the sense of classical logic. (Proof: An interpretation I satisfies F iff the total interpretation $\langle I, I \rangle$ satisfies F.) But the converse is not true.

[1] This terminology, and the definitions that follow, will not be new to the readers familiar with Kripke models. HT-interpretations are a special case of that concept.

For instance, $p \vee \neg p$ and $\neg\neg p \to p$ are not theorems of HT, that is to say, they are not consequences of the empty set of formulas; $\neg q \to p$ is not a consequence of $\neg p \to q$. (Proof: in each case, take $I^H = \emptyset$, $I^T = \{p\}$.)

In the definition of HT in terms of 3 truth values, the value assigned to an atom p is determined by whether p belongs to I^H, $I^T \setminus I^H$, or neither.

2.2 Deduction

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. A formalisation of HT can be obtained from intuitionistic logic by adding the axiom schema

$$F \vee (F \to G) \vee \neg G \tag{8}$$

(Lex Hendriks, personal communication).

The most useful, for our purposes, consequence of (8) is the weak law of the excluded middle

$$\neg F \vee \neg\neg F \tag{9}$$

(in (8), take G to be $\neg F$). All proofs in HT given below are actually proofs in intuitionistic logic extended with (9). There is a good reason for that: for any two propositional formulas F_1, F_2 that correspond to logic programs in the sense of Section 3, if F_1 is equivalent to F_2 in the logic of here-and-there then the equivalence $F_1 \leftrightarrow F_2$ can be derived from the weak law of the excluded middle in intuitionistic logic (Dick de Jongh and Lex Hendriks, personal communication).

It is easy to see that De Morgan's laws

$$\begin{aligned} \neg(F \vee G) &\leftrightarrow \neg F \wedge \neg G, \\ \neg(F \wedge G) &\leftrightarrow \neg F \vee \neg G \end{aligned}$$

are provable in HT. Indeed, the first equivalence and one half of the second are provable intuitionistically; to derive $\neg F \vee \neg G$ from $\neg(F \wedge G)$, consider two cases $\neg F$, $\neg\neg F$.

Using De Morgan's laws, we can verify that (6) is equivalent to (7): these two sets of formulas can be rewritten as

$$\{p \vee q, \neg p \vee \neg q\}$$

and

$$\{\neg p \to q, \neg q \to p, \neg p \vee \neg q\},$$

which are intuitionistically equivalent.

Another interesting equivalence provable in HT is

$$\neg F \vee G \leftrightarrow \neg\neg F \rightarrow G.$$

Left-to-right, it is provable intuitionistically; right-to-left, consider two cases $\neg F$, $\neg\neg F$.

3 Logic Programs and Answer Sets

The presentation below follows essentially [Lifschitz *et al.*, 1999], except that the second kind of negation is not allowed here.[2]

Formulas are built from propositional atoms and the 0-place connectives $\top$ and $\bot$ using negation as failure (*not*), conjunction (,) and disjunction (;). A *rule* is an expression of the form

$$Head \leftarrow Body \tag{10}$$

where *Head* and *Body* are formulas. If *Body* is $\top$ then we identify rule (10) with formula *Head*. A *program* is a set of rules.

We define when a set X of atoms *satisfies* a formula F (symbolically, $X \models F$) recursively, as follows:

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

The *reduct* Π^X of a program Π with respect to a set X of atoms is obtained by replacing every maximal occurrence of a formula of the form *not* F in Π (that is, every occurrence of *not* F that isn't in the range of another *not*)

- with $\bot$ if $X \models F$,
- with $\top$, otherwise.[3]

[2] Adding the second negation is discussed in Section 5.

[3] As observed in [Ferraris and Lifschitz, 2000], this definition is equivalent to the recursive definition of a reduct given in [Lifschitz *et al.*, 1999].

The concept of an answer set is defined first for programs not containing negation as failure. A set X of atoms is *closed* under such a program Π if, for every rule (10) in Π, $X \models \mathit{Head}$ whenever $X \models \mathit{Body}$. An *answer set* for a program Π without negation as failure is a minimal set closed under Π.

For an arbitrary program Π, we say that X is an *answer set* for Π if X is an answer set for the reduct Π^X.

Let us check, for instance, that $\{p\}$ is an answer set for (2). The reduct of (2) with respect to $\{p\}$ is

$$\begin{array}{l} p \leftarrow \top\ , \\ q \leftarrow \bot\ . \end{array}$$

Clearly $\{p\}$ is a minimal set closed under this reduct.

4 Strong Equivalence

4.1 Main Theorem

A program Π is *unary* if, in every rule of Π, the head is an atom and the body is either $\top$ or an atom.

In the statement of the theorem, we identify formulas and rules in the sense of Section 3 with propositional formulas, as described in the introduction. Accordingly, programs become a special case of theories, and we can talk about the equivalence of programs in the logic of here and there.

Theorem 1 *For any programs Π_1 and Π_2, the following conditions are equivalent:*

(a) *for every program Π, programs $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets,*

(b) *for every unary program Π, programs $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets,*

(c) *Π_1 is equivalent to Π_2 in the logic of here-and-there.*

The fact that (b) implies (a) shows that the strong equivalence condition we are interested in ("for every Π, $\Pi_1 \cup \Pi$ is equivalent to $\Pi_2 \cup \Pi$") does not depend very much on what kind of program Π is assumed to be: it does not matter whether Π is required to belong to the narrow class of unary programs or is allowed to be an arbitrary program with nested expressions. The fact that (a) is equivalent to (c) expresses the correspondence between the strong equivalence of logic programs and the equivalence of formulas in the logic of here-and-there.

The proof of the theorem is given in Section 4.5 below.

4.2 Some Examples

We have already seen how the equivalence of conditions (a) and (c), along with the properties of the logic of here-and-there stated in Section 2, can be used to prove the strong equivalence of logic programs. Here are further examples.

Replacing a rule of the form

$$not\ F_1; \dots ; not\ F_k \leftarrow G$$

in any program by the constraint[4]

$$\bot \leftarrow F_1, \dots , F_k, G$$

does not affect the program's answer sets [Inoue and Sakama, 1998, Section 4.2], because the corresponding formulas are equivalent in HT. This fact is interesting because the role of constraints in a logic program is well understood: adding a constraint to a program eliminates its answer sets that "violate" the constraint [Lifschitz *et al.*, 1999, Proposition 2].

In any program, replacing a subformula of the form $not(F, G)$ with $not\ F; not\ G$ does not change its answer sets, as well as replacing $not(F; G)$ with $not\ F, not\ G$ [Lifschitz *et al.*, 1999, Proposition 4(iv)]. Indeed, as we saw in Section 2, the logic of here-and-there satisfies De Morgan's laws.

In any program containing the rules

$$\begin{array}{l} F; G\ , \\ \bot \leftarrow F, G\ , \\ F \leftarrow H \end{array}$$

replacing the last rule by the constraint

$$\bot \leftarrow not\ F, H$$

does not affect the program's answer sets, because the corresponding sets of formulas are equivalent in HT (and even in intuitionistic logic).

On the other hand, Theorem 1 can be used to prove that some pairs of programs are not strongly equivalent. For instance, the program

$$\begin{array}{l} p \leftarrow q\ , \\ p \leftarrow not\ q \end{array} \tag{11}$$

is not strongly equivalent to p, because p is not a consequence of the formulas $q \to p$, $\neg q \to p$ in the logic of here-and-there. (Proof: Take $I^H = \emptyset$, $I^T = \{p, q\}$.) This fact can be also proved directly, by adding $q \leftarrow p$ to each of the two programs.

[4] A *constraint* is a rule whose head is $\bot$.

4.3 A Remark on Intuitionistic Logic

The example of programs (4) and (5) shows that the logic of here-and-there in the statement of Theorem 1 cannot be replaced by intuitionistic logic: these programs are strongly equivalent, but not intuitionistically equivalent. Program (4) is syntactically rather complicated: it contains a disjunctive rule and a constraint. But we can give a similar counterexample without using rules like these. Consider the program

$$\begin{array}{l} q \leftarrow \mathit{not}\ p\ , \\ p \leftarrow \mathit{not}\ q\ , \\ r \leftarrow p, q\ , \\ s \leftarrow p\ , \\ s \leftarrow q \end{array} \tag{12}$$

and the program obtained from (12) by adding the rule

$$s \leftarrow \mathit{not}\ r\ . \tag{13}$$

These programs are not equivalent to each other intuitionistically, which can be proved using a Kripke model with 3 worlds. But they are strongly equivalent. To derive $\neg r \rightarrow s$ from

$$\neg p \rightarrow q,\ \neg q \rightarrow p,\ (p \wedge q) \rightarrow r,\ p \rightarrow s,\ q \rightarrow s \tag{14}$$

in HT, assume $\neg r$. Using the third of formulas (14) and de Morgan's laws, we derive $\neg p \vee \neg q$. Using the first two of formulas (14), we derive $p \vee q$. Using the last two of formulas (14), we derive s.

4.4 Equilibrium Logic

In order to prove the main theorem we shall make use of some results about a system of nonmonotonic reasoning called *equilibrium logic*, developed in Pearce [1997, 1999].

Equilibrium logic can be described as a special kind of minimal model reasoning in logic HT, as follows. An *equilibrium* model of a theory Γ is a total HT-interpretation $\langle I, I\rangle$ such that

(i) $\langle I, I\rangle$ is a model of Γ, and

(ii) for every proper subset J of I, $\langle J, I\rangle$ is not a model of Γ.

Consider, for instance, the theory comprising a single atomic formula p. Some of its models in the sense of logic HT are

$$\begin{array}{c} \langle \{p\}, \{p\}\rangle, \\ \langle \{p\}, \{p, q\}\rangle, \\ \langle \{p, q\}, \{p, q\}\rangle. \end{array}$$

The first of them is an equilibrium model. The second is not, because it is not total. The third is not an equilibrium model either, because it does not satisfy the minimality condition (ii).

As another example, take the theory $\neg\neg p$. Some of its models in the sense of logic HT are

$$\langle \emptyset, \{p\} \rangle,$$
$$\langle \{p\}, \{p\} \rangle.$$

Neither is an equilibrium model: the first is not total, and the second is not minimal. In fact, this theory has no equilibrium models.

Equilibrium logic is defined by that which holds in all equilibrium models.

The main property of equilibrium logic that we need here is captured in Lemma 3 below, which asserts essentially that the concept of an equilibrium model is a generalization of the concept of an answer set.

Lemma 1 *For any program Π without negation as failure, an HT-interpretation $\langle I^H, I^T \rangle$ is a model of Π in the sense of HT iff both I^H and I^T are models of Π in the sense of classical logic.*

Proof. Let Π be a program without negation as failure. An HT-interpretation $\langle I^H, I^T \rangle$ is a model of Π iff, for every rule $Head \leftarrow Body$ in Π,

$$\langle I^H, I^T, H \rangle \models Body \text{ implies } \langle I^H, I^T, H \rangle \models Head$$

and

$$\langle I^H, I^T, T \rangle \models Body \text{ implies } \langle I^H, I^T, T \rangle \models Head.$$

Since $Head$ and $Body$ do not contain negation as failure, these conditions can be simplified as follows:

$$I^H \models Body \text{ implies } I^H \models Head,$$

$$I^T \models Body \text{ implies } I^T \models Head.$$

To require this to hold for every rule $Head \leftarrow Body$ in Π means to require that I^H and I^T be models of Π in the sense of classical logic.

Lemma 2 *An HT-interpretation $\langle I^H, I^T \rangle$ is a model of a program Π iff it is a model of the reduct $\Pi^{(I^T)}$.*

The proof of the lemma uses two facts that can be easily verified by structural induction. One is a "monotonicity property" of HT-interpretations:

Fact 1 *For any HT-interpretation $\langle I^H, I^T \rangle$ and any propositional formula F, if $\langle I^H, I^T, H \rangle \models F$ then $\langle I^H, I^T, T \rangle \models F$.*

The other relates the satisfaction relation of HT to the satisfaction relation of classical logic:

Fact 2 *For any HT-interpretation $\langle I^H, I^T\rangle$ and any propositional formula F, $\langle I^H, I^T, T\rangle \models F$ iff $I^T \models F$.*

Proof of Lemma 2. According to the definition of the reduct (Section 3), $\Pi^{(I^T)}$ is the program obtained from Π by the simultaneous replacement of some maximal subformulas of the form *not* F with $\top$, and of all other maximal subformulas of this form with $\bot$. It is sufficient to check that, for any maximal subformula *not* F and for the formula G that replaces it,

$$\langle I^H, I^T\rangle \models \mathit{not}\ F \text{ iff } \langle I^H, I^T\rangle \models G.$$

Since G is either $\top$ or $\bot$, this claim can be rewritten as

$$\langle I^H, I^T\rangle \models \mathit{not}\ F \text{ iff } G = \top.$$

The left-hand side of this equivalence holds iff

$$\langle I^H, I^T, H\rangle \not\models F \text{ and } \langle I^H, I^T, T\rangle \not\models F.$$

By Fact 1, the first conjunctive term is a consequence of the second one, and can be dropped. By Fact 2, the second term can be rewritten as $I^T \not\models F$, which is the condition characterizing the case $G = \top$ in the definition of the reduct.

Lemma 3 *For any program Π and any set I of atoms, the HT-interpretation $\langle I, I\rangle$ is an equilibrium model of Π iff I is an answer set for Π.*

This lemma generalises Proposition 10 from [Pearce, 1997] to programs with nested expressions. Its proof uses the following fact:

Fact 3 *A set of atoms is closed under a program Π iff it is a model of Π in the sense of classical logic.*

To prove this assertion, note that a set of atoms satisfies a formula F in the sense of Section 3 iff it satisfies F in the sense of classical logic.

Proof of Lemma 3. By the definition of an answer set, I is an answer set for Π iff I is an answer set for the reduct Π^I, that is, iff

- I is closed under Π^I, and
- for every proper subset J of I, J is not closed under Π^I.

In view of Fact 3, these conditions can be restated as follows:

- I is a model of Π^I, and
- for every proper subset J of I, J is not a model of Π^I.

By Lemma 1, this is equivalent to saying that

- $\langle I, I\rangle$ is a model of Π^I, and
- for every proper subset J of I, $\langle J, I\rangle$ is not a model of Π^I.

By Lemma 2, Π^I in both clauses can be replaced with Π, which turns these conditions into the definition of an equilibrium model of Π.

4.5 Proof of Main Theorem

The definition of a unary program (Section 4.1) is extended to propositional theories in a natural way: A theory Γ is *unary* if every formula in Γ is either an atom or an implication whose antecedent and consequent are atoms. In view of Lemma 3, Theorem 1 is a special case of the following assertion:

Lemma 4 *For any theories Γ_1 and Γ_2, the following conditions are equivalent:*

(a) *for every theory Γ, theories $\Gamma_1 \cup \Gamma$ and $\Gamma_2 \cup \Gamma$ have the same equilibrium models,*

(b) *for every unary theory Γ, theories $\Gamma_1 \cup \Gamma$ and $\Gamma_2 \cup \Gamma$ have the same equilibrium models,*

(c) *Γ_1 is equivalent to Γ_2 in the logic of here-and-there.*

Proof. Obviously (a) implies (b). To see that (c) implies (a), observe that if Γ_1 and Γ_2 are equivalent in logic HT then $\Gamma_1 \cup \Gamma$ and $\Gamma_2 \cup \Gamma$ are equivalent in HT also, so that both theories have the same equilibrium models. It remains to check that (b) implies (c).

Suppose that Γ_1 has a model $\langle I^H, I^T\rangle$ which is not a model of Γ_2. We'll show how to find a unary theory Γ such that $\langle I^T, I^T\rangle$ is an equilibrium model of one of the theories $\Gamma_1 \cup \Gamma$, $\Gamma_2 \cup \Gamma$ but not an equilibrium model of the other.

Case 1: $\langle I^T, I^T\rangle$ is not a model of Γ_2. It is easy to see that it is a model of Γ_1. Indeed, from the assumption that $\langle I^H, I^T\rangle$ is a model of Γ_1 we can conclude by Fact 1 that $\langle I^H, I^T, T\rangle$ satisfies every formula in Γ_1; consequently $\langle I^T, I^T\rangle$ satisfies every formula in Γ_1 as well. We take $\Gamma = I^T$. It is clear that $\langle I^T, I^T\rangle$ is a model of $\Gamma_1 \cup I^T$; by inspection, it is an equilibrium model of this theory. On the other hand, it is not a model of Γ_2, so that it cannot be a model of $\Gamma_2 \cup I^T$.

Case 2: $\langle I^T, I^T \rangle$ is a model of Γ_2. Define

$$\Gamma = I^H \cup \{A \rightarrow B \ : \ A, B \in I^T \setminus I^H, \ A \neq B\}.$$

Since $\langle I^T, I^T \rangle$ satisfies every formula in Γ, it is a model of $\Gamma_2 \cup \Gamma$. To see that it is in equilibrium, consider any model $\langle J, I^T \rangle$ of $\Gamma_2 \cup \Gamma$ such that J is a proper subset of I^T. Clearly J must contain I^H. But it cannot be equal to I^H, since by assumption $\langle I^H, I^T \rangle$ is not a model of Γ_2. Thus $I^H \subset J \subset I^T$. Take an atom $A \in J \setminus I^H$ and an atom $B \in I^T \setminus J$. For these atoms, $A \rightarrow B$ belongs to Γ. But $\langle J, I^T \rangle$ does not satisfy this implication, contrary to the assumption that it is a model of $\Gamma_2 \cup \Gamma$. Finally, we'll check that $\langle I^T, I^T \rangle$ is not an equilibrium model of $\Gamma_1 \cup \Gamma$. To see this, consider the model $\langle I^H, I^T \rangle$ of Γ_1. Clearly it is a model of I^H. Moreover, this model satisfies each implication $A \rightarrow B$ in Γ: $\langle I^H, I^T, H \rangle$ does not satisfy A because $A \notin I^H$, and $\langle I^H, I^T, T \rangle$ satisfies B because $B \in I^T$. We see that $\langle I^H, I^T \rangle$ satisfies all formulas in Γ, so that this is a model of $\Gamma_1 \cup \Gamma$. On the other hand, I^H is different from I^T, because $\langle I^T, I^T \rangle$ is a model of Γ_2, and $\langle I^H, I^T \rangle$ is not. Consequently, I^H is a proper subset of I^T, so that $\langle I^T, I^T \rangle$ is not an equilibrium model of $\Gamma_1 \cup \Gamma$.

5 Adding a Second Negation

Answer sets are usually defined for logic programs possessing a second kind of negation, which expresses the direct or explicit falsity of an atom. In [Gelfond and Lifschitz, 1991] and [Lifschitz *et al.*, 1999], this second negation is called "classical" and denoted by $\neg$.

A *literal* is understood to be a propositional atom or an atom prefixed by $\neg$. We define *extended formulas*, *extended rules* and *extended programs* exactly as we defined formulas, rules and programs in Section 3 above, except that arbitrary literals are allowed in them in place of atoms. The semantics of extended programs defines when a consistent set X of literals is an answer set for an extended program Π; see [Lifschitz *et al.*, 1999] for details.

Adding a second negation to the syntax of logic programs is important for applications to knowledge representation, but, computationally, this extension is not very essential. In fact, the second negation can be eliminated from a program by a simple syntactic transformation. For every atom A of the underlying language, choose a new atom A'. For any extended program Π, let Π' be the non-extended program obtained from Π by replacing all negative literals $\neg A$ with A'. By *Cons* we denote the set of all constraints of the form $\bot \leftarrow A, A'$. It is easy to show that there is a 1–1 correspondence between the answer sets for Π and the answer sets for $\Pi' \cup Cons$; if X is an answer set for Π then the corresponding answer set X' for $\Pi' \cup Cons$ is obtained from X by replacing every negative literal $\neg A$ with A'. For instance, if Π is

$$\begin{array}{l} p; \neg p\ , \\ q \leftarrow p\ , \\ \neg q \end{array} \tag{15}$$

then $\Pi' \cup Cons$ is

$$\begin{array}{l} p; p'\ , \\ q \leftarrow p\ , \\ q'\ , \\ \bot \leftarrow p, p'\ , \\ \bot \leftarrow q, q' \end{array} \tag{16}$$

(assuming that the underlying language has no atoms other than p and q). The only answer set for (15) is $\{\neg p\}$; accordingly, the only answer set for (16) is $\{p'\}$.

Questions concerning the strong equivalence of programs also arise in this extended setting. For instance, it was shown in [Erdem and Lifschitz, 1999] that, in any extended program, the disjunctive rule

$$p; \neg p \tag{17}$$

can be equivalently replaced by the two nondisjunctive rules

$$\begin{array}{l} p \leftarrow not\ \neg p\ , \\ \neg p \leftarrow not\ p\ . \end{array} \tag{18}$$

The following theorem allows us to reduce the verification of the strong equivalence of extended programs to checking the equivalence of sets of formulas in HT.

Theorem 2 *For any extended programs Π_1 and Π_2, the following conditions are equivalent:*

(a) *for every extended program Π, extended programs $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets,*

(b) *$\Pi'_1 \cup Cons$ is equivalent to $\Pi'_2 \cup Cons$ in the logic of here-and-there.*

To see, for instance, that (17) and (18) are strongly equivalent it suffices to check the equivalence in HT of

$$\begin{array}{l} p; p'\ , \\ \bot \leftarrow p, p' \end{array}$$

and

$$\begin{aligned}
p &\leftarrow not\ p'\ ,\\
p' &\leftarrow not\ p\ ,\\
\bot &\leftarrow p, p'\ .
\end{aligned}$$

But these are exactly programs (4) and (5) discussed in the introduction, with p' substituted for q. Rewritten as sets of propositional formulas, these two programs become (6) and (7), which we have already shown to be equivalent in Section 2.

Proof of Theorem 2. Let Π_1, Π_2 and Π be extended programs such that $\Pi_1' \cup Cons$ is equivalent to $\Pi_2' \cup Cons$ in HT. By Theorem 1, $\Pi_1' \cup Cons \cup \Pi'$ has the same answer sets as $\Pi_2' \cup Cons \cup \Pi'$. Consequently $\Pi_1 \cup \Pi$ has the same answer sets as $\Pi_2 \cup \Pi$. Conversely, assume that Π_1, Π_2 be extended programs such that, for every extended program Π, $\Pi_1 \cup \Pi$ has the same answer sets as $\Pi_2 \cup \Pi$. Then, for every extended program Π, $\Pi_1' \cup \Pi' \cup Cons$ has the same answer sets as $\Pi_2' \cup \Pi' \cup Cons$. Since every non-extended program in the language with the additional atoms A' can be represented in the form Π' for some extended program Π in the original language, Theorem 1 implies that $\Pi_1' \cup Cons$ is equivalent to $\Pi_2' \cup Cons$ in HT.

Alternatively, strong equivalence of extended programs can be studied by extending the language of HT with a second negation $\sim$, known in the logical literature as *strong* negation. Strong negation was originally introduced by Nelson [Nelson, 1949] in order to model a logical concept of constructible falsity. Later Vorob'ev [Vorob'ev, 1952a, Vorob'ev, 1952b] showed how to axiomatise the notion of strong negation. Adding $\sim$ together with the Vorob'ev axioms to the logic HT yields a logic with five truth values, called *here-and-there with strong negation*, which we can denote by *N5*. This logic is studied algebraically in [Kracht, 1998]; proof systems can be found in [Pearce *et al.*, 2000b] and [Pearce *et al.*, 2000a]. *N5*-interpretations are like HT-interpretations $\langle I^H, I^T \rangle$, except that I^H and I^T are now sets of *literals*, as before with $I^H \subseteq I^T$. The concept of an equilibrium model in *N5* is defined analogously to the case of HT [Pearce, 1997].

To identify extended formulas and extended rules with propositional formulas of *N5*, we translate $\neg$ to strong negation $\sim$. The correspondence between the language of logic programs and the language of propositional formulas in the presence of two negations is summarised in the following table:

Extended formulas	,	;	*not*	$\neg$
N5 formulas	$\wedge$	$\vee$	$\neg$	$\sim$

Our main theorem readily generalises to the new setting (the method of proof is exactly the same): we can show that two extended programs are strongly equivalent if and only if they are equivalent in the logic *N5*.

It is interesting to note that the method of eliminating the second negation from extended programs discussed at the beginning of this section can be generalised to *N5*. For any formula F in the language of *N5*, there is an equivalent "reduced" formula $r(F)$ in which the strong negation symbol $\sim$ has been "driven-in" so that it stands directly in front of an atom [Vorob'ev, 1964]. Consider the syntactic transformation that eliminates strong negation from a formula F by replacing each part of the form $\sim A$ in $r(F)$ with a new atom A'. Under the assumptions $\neg(A \wedge A')$, this transformation provides a reduction of *N5* to *HT*. This is similar to the reduction of Nelson's logic of strong negation to intuitionistic logic proposed by Gurevich [1977].

6 Conclusion

Facts about the strong equivalence of logic programs tell us how one can simplify a part of a logic program without looking at the other parts. Strong equivalence can be characterized in terms of the logic of here-and-there. The set of theorems of this logic includes all intuitionistically provable propositional formulas, the weak law of the excluded middle, and De Morgan's laws. The fact that two programs are strongly equivalent can be often established by deriving them from each other using these logical means. There is a exponential time algorithm for verifying strong equivalence.

The definition of the answer set semantics for extended nondisjunctive programs in [Gelfond and Lifschitz, 1991] was suggested by the view that such programs are merely a special case of default theories in the sense of [Reiter, 1980]. Defaults are a generalisation of inference rules: besides the premise and the conclusion, a default has "justifications." Accordingly, the symbol $\leftarrow$ in a logic program is similar to the bar separating the conclusion from the premise in an inference rule. The role of negation as failure in a program is similar to the role of justifications in a default theory. An extension for a default theory is a theory in the sense of classical logic; accordingly, the symbol $\neg$ in a logic program is classical negation.

The results of this paper provide additional evidence in support of an alternative view of the answer set semantics, presented in [Pearce, 1997]. Answer sets are a special case of equilibrium models. Rules in a logic program are similar to propositional formulas in the logic of here-and-there, $\leftarrow$ and negation as failure being the counterparts of implication and negation. Introducing the second negation is similar to adding strong negation to that logic. This perspective is useful, in particular, for the study of strong equivalence.

Acknowledgements

We are grateful to Yuliya Babovich, Pedro Cabalar, Jonathan Campbell, Esra Erdem, Selim Erdoğan, Michael Gelfond, Paolo Ferraris, Dick de Jongh, Lex Hendriks, Joohyung Lee, Grigori Mints and Hudson Turner for useful discussions related to the subject of this paper. The work of the first author was partially supported by National Science Foundation under grant IIS-9732744.

References

[Bibel and Eder, 1993] 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.

[Erdem and Lifschitz, 1999] Esra Erdem and Vladimir Lifschitz. 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)*, pages 107–116, 1999.

[Ferraris and Lifschitz, 2000] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. In preparation, 2000.

[Gelfond and Lifschitz, 1988] 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.

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

[Gödel, 1932] Kurt Gödel. Zum intuitionistischen Aussagenkalkül. *Anzeiger der Akademie der Wissenschaften in Wien*, pages 65–66, 1932. Reproduced in: Kurt Gödel, *Collected Works*, Vol. 1, OUP, 1986.

[Gurevich, 1977] Yuri Gurevich. Intuitionistic logic with strong negation. *Studia Logica*, 36:49–59, 1977.

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

[Inoue and Sakama, 1998] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998.

[Kracht, 1998] Marcus Kracht. On extensions of intermediate logics by strong negation. *Journal of Philosophical Logic*, 27, 1998.

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

[Łukasiewicz, 1941] Jan Łukasiewicz. Die Logik und das Grundlagenproblem. In *Les Entretiens de Zürich sue les Fondements et la méthode des sciences mathématiques 1938*, pages 82–100. Zürich, 1941.

[Nelson, 1949] David Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14:16–26, 1949.

[Pearce *et al.*, 2000a] David Pearce, Immaculada de Guzmán, and Agustín Valverde. A tableau calculus for equilibrium entailment. In *Proc. TABLEAUX-2000*, pages 352–367, 2000.

[Pearce *et al.*, 2000b] David Pearce, Immaculada de Guzmán, and Agustin Valverde. Computing equilibrium models using signed formulas. In *Proc. CL-2000*, pages 688–702, 2000.

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

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

[Reiter, 1980] Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Vorob'ev, 1952a] Nikolay Vorob'ev. Constructive propositional calculus with strong negation. *Doklady Akademii Nauk SSSR*, 85:465–468, 1952. In Russian.

[Vorob'ev, 1952b] Nikolay Vorob'ev. The problem of deducibility in constructive propositional calculus with strong negation. *Doklady Akademii Nauk SSSR*, 85:689–692, 1952. In Russian.

[Vorob'ev, 1964] Nikolay Vorob'ev. Constructive propositional calculus with strong negation. *Transactions of Steklov's Institute*, 72:195–227, 1964. In Russian.