TLP **7** (6): 745–759, 2007. © 2007 Cambridge University Press
doi:10.1017/S1471068407003110 First published online 25 May 2007 Printed in the United Kingdom

Propositional theories are strongly equivalent to logic programs

PEDRO CABALAR*
Department of Computer Science, University of Corunna, Spain
(*e-mail:* cabalar@udc.es)

PAOLO FERRARIS
Department of Computer Sciences, University of Texas, Austin, USA
(*e-mail:* otto@cs.utexas.edu)

submitted 10 August 2005; revised 5 May 2006; accepted 1 December 2006

Abstract

This paper presents a property of propositional theories under the answer sets semantics (called *Equilibrium Logic* for this general syntax): any theory can always be reexpressed as a strongly equivalent disjunctive logic program, possibly with negation in the head. We provide two different proofs for this result: one involving a syntactic transformation, and one that constructs a program starting from the countermodels of the theory in the intermediate logic of here-and-there.

KEYWORDS: answer sets, equilibrium logic, logic of here-and-there, strong equivalence

1 Introduction

One of the most interesting consequences from research in declarative semantics for Logic Programming (LP) has probably been the progressive increase in expressiveness that the field has experienced over the years. A clear example of this trend is the case of the *stable models* (or *answer sets*) semantics (Gelfond and Lifschitz 1988), which meant the first satisfactory treatment of default negation, and whose simplicity directly allowed new extensions like the use of disjunction (Gelfond and Lifschitz 1991) and negation (Lifschitz and Woo 1992; Inoue and Sakama 1998) in the rule heads. In one of the most general formalizations (Lifschitz *et al.* 1999) of answer sets, logic programs consist of rules whose head and body are *nested expressions*, that is, propositional formulas without the connectives for implication and equivalence. The intuitive behavior of these nested expressions is well understood: in fact, it is quite analogous to the way in which we nest operators in Prolog. To put an example

* The first author is partially supported by the Spanish MEC project TIN-2006-15455-C03-02. The second author is partially supported by the National Science Foundation under Grant IIS-0412907.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

of their utility, a set of rules like

$$\begin{array}{rr} a \leftarrow b & a \leftarrow c \wedge \neg d \\ e \vee p \leftarrow b & e \vee p \leftarrow c \wedge \neg d \end{array}$$

can be "packed" into the single rule with nested expressions:

$$a \wedge (e \vee p) \leftarrow b \vee (c \wedge \neg d) \tag{1}$$

In Lifschitz *et al.* (1999), it was also shown that these rules with nested expressions can always be unfolded back into usual program rules (which allow disjunction and negation in the head) and, moreover, that this replacement can be done locally, regardless of the context (this last property of the transformation has the name of *strong equivalence*).

In view of a commonly used abbreviation (see Section 2.2), nested logic programs allow a free use of all the usual connectives for logic programs, except for implication, the only one that cannot be nested. This last syntactic restriction has been eventually removed, thanks to the result established in Lifschitz *et al.* (2001). In that work, it was shown that the answer sets semantics is further generalized by *Equilibrium Logic* (Pearce 1997), which just deals with arbitrary propositional theories, imposing a selection criterion on their models under the intermediate logic of *here-and-there* (Heyting 1930). As a result, theories containing, for instance, a rule in the scope of a disjunction:

$$(p \leftarrow q) \vee r \tag{2}$$

or a rule as body of other rule:

$$p \leftarrow (q \leftarrow r) \tag{3}$$

have now an interpretation in terms of answer sets. Although from a purely logical point of view, ruling out syntactic restrictions is a clear advantage, from the LP viewpoint one may wonder, however, what the practical interest of dealing with arbitrary formulas is. A recent example of their utility has been provided in Ferraris (2005), where nonmonotone aggregates are expressed using rules as part of a body rule. To give an example, according to Ferraris (2005), the following rule with an aggregate in the body[1]

$$p \leftarrow 0 \leqslant \{q = 1, r = -1\} \tag{4}$$

is translated as (3).

Despite their interest, the intuition on how arbitrary theories may behave as "programs" was still far from obvious. Until now, we missed a formal connection with the usual rule format of pairs of head and body. Furthermore, we did not even know whether they provide a real increase in expressiveness, or perhaps they can always be translated instead into a strongly equivalent program (as in the case of nested expressions). In this paper, we show that, in fact, the latter happens to be

[1] Although aggregates are out of the scope of this note, the intuitive meaning of this rule would be to make p true whenever the addition of the numeric weights in (4) for those atoms in $\{q, r\}$ that are true is greater than, or equal to, 0.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

the case. In other words, the main result states that every propositional theory is strongly equivalent to a logic program.

From an LP reading, this result is pointing out that any generic propositional theory can be seen as "shorthand" for a logic program. Furthermore, the transformation from a propositional theory to a logic program is modular; i.e., it can be done formula by formula. For instance, formula (3) inside any theory can be simply replaced by the rules:

$$\begin{aligned} p &\leftarrow \neg r \\ p &\leftarrow q \\ p \vee \neg q \vee r &\leftarrow \end{aligned}$$

and similarly (2) can be replaced by

$$\begin{aligned} p \vee r &\leftarrow q \\ \neg q \vee r &\leftarrow \neg p \end{aligned}$$

Another possible reading of the main result is that the form of logic programs is a kind of *normal form* for arbitrary formulas from the point of view of Equilibrium Logic.

The main result is independently proven in two ways: (1) with a syntactic recursive transformation and (2) with a method that builds a strongly equivalent program starting from the countermodels (in the logic of here-and-there) of the propositional theory. Presenting the two proofs is interesting for several reasons. First, we get in this way an interesting analogy with the two ways of transforming a formula into CNF in classical logic: using a syntatic transformation or building the formula from its countermodels. Second, both proofs may provide technical tools for achieving other interesting results, as we explain later in the conclusions.

The rest of the paper is organized as follows. Section 2 contains the preliminaries, with brief reviews of the logic of here-and-there and of answer sets semantics. In Section 3, we present the first proof of the main result, based on the recursive syntactic transformation, whereas Section 4 contains the second proof, which builds a program from the here-and-there countermodels of the original theory. In Section 5, we further explore the analogies to classical logic, and obtain a disjunctive normal form based on the here-and-there models.

2 Preliminaries

Before introducing the logic of here-and-there, there is a small remark about notation. We assume that we handle a set of atoms Σ called the *propositional signature*. We will use letters $X, Y, \ldots$, to denote propositional interpretations in classical logic, represented here as sets of atoms, i.e., subsets of Σ. For any propositional formula F, we adopt the usual notation $X \models F$ to stand for X *satisfies* F under classical logic.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

2.1 Logic of here-and-there

The logic of here-and-there was originally defined by Heyting (1930). A *propositional formula* is any combination of atoms in Σ with connectives $\bot$ (false), $\vee$, $\wedge$, and $\rightarrow$. We also define the abbreviations:

$$\begin{array}{rcl} \neg F & \stackrel{\text{def}}{=} & F \rightarrow \bot \\ \top & \stackrel{\text{def}}{=} & \bot \rightarrow \bot \\ F \equiv G & \stackrel{\text{def}}{=} & (F \rightarrow G) \wedge (G \rightarrow F) \end{array}$$

As usual, a *propositional theory* is a (possibly infinite) set of propositional formulas.

The semantics of the logic of here-and-there is defined as follows. An *interpretation* is a pair (X, Y) of sets of atoms (respectively called "here" and "there") such that $X \subseteq Y$. Intuitively, atoms in X are considered to be true, atoms not in Y are considered to be false, and the rest $(Y - X)$ are thought to be undefined. We say that an interpretation (X, Y) is *total* when $X = Y$ (no undefined atoms).

Definition 1 (*satisfaction of formulas*)
We recursively define[2] when an interpretation (X, Y) *satisfies* a formula F, written $(X, Y) \models F$, as follows:

- for any atom a, $(X, Y) \models a$ if $a \in X$,
- $(X, Y) \not\models \bot$,
- $(X, Y) \models F \wedge G$ if $(X, Y) \models F$ and $(X, Y) \models G$,
- $(X, Y) \models F \vee G$ if $(X, Y) \models F$ or $(X, Y) \models G$,
- $(X, Y) \models F \rightarrow G$ if $(X, Y) \models F$ implies $(X, Y) \models G$, and $Y \models F \rightarrow G$.

Although we use the same symbol '$\models$' for satisfaction in classical logic and the logic of here-and-there, ambiguity is avoided in view of the difference in the form of the interpretation on the left. In this way, the last line of Definition 1 is referring to classical satisfaction for $Y \models F \rightarrow G$. As usual, an interpretation is a *model* of a theory T if it satisfies all the formulas in T. Two formulas (theories) are *equivalent* if they have the same models.

As a first immediate observation about Definition 1, note that when the interpretation is total ($X{=}Y$), $(Y, Y) \models F$ simply collapses into classical satisfaction $Y \models F$. Another interesting property we will use later is that truth in the "here" component implies truth in the "there" component:

Property 1
For any interpretation (X, Y) and any propositional theory T:
if $(X, Y) \models T$ then $(Y, Y) \models T$ (i.e., $Y \models T$).

When dealing with countermodels, this property can be rephrased as follows: if (Y, Y) is a total countermodel of T, then any (X, Y) is also a countermodel of T. Besides, using the definition of $\neg F$ as $F \rightarrow \bot$, it also allows us to obtain the following characterization of satisfaction for negated formulas.

[2] We have slightly simplified the definition in comparison with the usual definition of satisfaction in the logic of here-and-there which is typically provided in terms of a Kripke structure as in intuitionistic logic, but just handling two worlds in this case. It can be easily seen that both definitions are equivalent.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

Property 2
For any interpretation (X, Y) and any formula F:
$(X, Y) \models \neg F$ iff $Y \models \neg F$.

Axiomatically, the logic of here-and-there is intermediate between intuitionistic and classical logic. Recall that a natural deduction system for intuitionistic logic can be obtained from the corresponding classical system (Bibel and Eder 1993, Table 3) by dropping the law of the excluded middle

$$F \vee \neg F$$

from the list of postulates. The logic of here-and-there, on the other hand, is the result of replacing the excluded middle in the classical system with the weaker axiom schema (Jongh and Hendriks 2003):

$$F \vee (F \rightarrow G) \vee \neg G. \tag{5}$$

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

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

and De Morgan's law

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

(the dual law can be proved even intuitionistically).

The logic of here-and-there differs from intuitionistic logic also as far as minimal adequate sets of connectives are concerned. In fact, a disjunction

$$F \vee G$$

is equivalent (Lukasiewicz 1941), in the logic of here-and-there, to

$$((F \rightarrow G) \rightarrow G) \wedge ((G \rightarrow F) \rightarrow F).$$

2.2 Logic programs

The set of logic programs can be defined as a subset of propositional formulas as follows. A *nested expression* is any propositional formula not containing implications of the form $F \rightarrow G$ with $G \neq \bot$ (i.e., negations and $\top$ are allowed). A *rule* r is a formula of the form $F \rightarrow G$, where F and G are nested expressions.[3] The *head* and the *body* of r are, respectively, defined as $head(r) = G$ and $body(r) = F$. When convenient, we will implicitly consider any nested expression G as the rule $\top \rightarrow G$.

As usual, a *literal* is any atom a or its negation $\neg a$. A rule is said to be *nonnested* when it has the form:

$$(l_1 \wedge \ldots \wedge l_m) \rightarrow (l_{m+1} \vee \ldots \vee l_n)$$

$(0 \leqslant m \leqslant n)$ where $l_1, \ldots, l_n$ are literals. An empty conjunction ($m = 0$) is understood

[3] Traditionally, rules $F \rightarrow G$ are written in the form $G \leftarrow F$ we used in the Introduction. Besides, $\neg$ is usually written as *not*, $\wedge$ as comma, and $\vee$ as semicolon.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

as the formula $\top$ and analogously, an empty disjunction ($n = m$) is represented as $\bot$.

A *(logic) program* is a propositional theory consisting of rules; a program is *nonnested* when all its rules are nonnested.

The usual definition of the *answer sets* (*stable models*) of a logic program Π is given in terms of the minimal models of a reduct program Π^X constructed from Π and a given interpretation X. The original answer sets semantics was defined in Gelfond and Lifschitz (1988) and (1991) for special cases of nonnested programs, and extended in Lifschitz *et al.* (1999) for programs with nested expressions. However, these definitions were later subsumed by a general logical encoding called *Equilibrium Logic* (Pearce 1997), which allows considering answer sets (or *equilibrium models*) for arbitrary propositional theories as a particular type of selected models in the logic of here-and-there.

Definition 2 (*Answer set/equilibrium model*)
A set Y of atoms is an *answer set* (or *equilibrium model*) for a theory T, if for every subset X of Y, $(X, Y) \models T$ iff $X = Y$.

An alternative, reduct-based description of equilibrium models of arbitrary propositional theory was recently obtained in Ferraris (2005).

Apart from constituting a monotonic framework in which answer sets can be defined, the logic of here-and-there further satisfies an interesting property: it allows capturing the concept of strong equivalence of theories. Two theories T_1 and T_2 are *strongly equivalent* if, for any theory T, $T_1 \cup T$ and $T_2 \cup T$ have the same answer sets. The main result in Lifschitz *et al.* (2001) asserts that this condition holds iff T_1 and T_2 are equivalent in the logic of here-and-there.

Our main result can be enunciated now as the following theorem.

Theorem 1 (*Main result*)
Every propositional theory (of Equilibrium Logic) is strongly equivalent to a logic program.

3 Proving Theorem 1 by syntactic transformation

In this proof of Theorem 1, we identify every finite program with the conjunction of its rules. We also need a few lemmas.

Lemma 1
For any formulas F, G, and K,

$$(F \to G) \to K \tag{6}$$

is equivalent to

$$\begin{aligned} &(G \vee \neg F) \to K, \\ &K \vee F \vee \neg G. \end{aligned} \tag{7}$$

in the logic of here-and-there.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

Proof
In the proof of the implication from (7) to (6), we assume $F \to G$, and we want to prove K. We consider the three cases K, F, and $\neg G$ from the last formula of (7). The first case is trivial. If F then G, and if $\neg G$ then $\neg F$ from the hypothesis $F \to G$. In both cases, the antecedent of the implication in (7) is true, so we can conclude K as well.

Now we assume (6) and we want to derive each formula of (7). For the first one, we also assume $G \vee \neg F$, and we want to derive K. It is sufficient to notice that both G and $\neg F$ make the antecedent of (6)—and consequently K—true. To prove the second formula of (7), we consider the axiom (5). The claim is clearly proven in the cases F and $\neg G$. In the remaining case, we obtain K by modus ponens on $F \to G$ and (6). □

Lemma 2
The implication of two finite programs is equivalent to a finite program.

Proof
We shall prove that the implication of two programs $\Pi_1 \to \Pi_2$ is equivalent to a program, by strong induction on the number of rules of Π_1. If Π_1 is empty (i.e., it is $\top$) then $\Pi_1 \to \Pi_2$ is clearly equivalent to Π_2. If Π_1 consists of a single rule $F \to G$ then

$$\begin{aligned}\Pi_1 \to \Pi_2 \;&=\; (F \to G) \to \left(\bigwedge_{(H \to K) \in \Pi_2} (H \to K) \right) \\ &\Leftrightarrow \bigwedge_{(H \to K) \in \Pi_2} ((F \to G) \to (H \to K)).\end{aligned}$$

It remains to notice that each conjunctive term $(F \to G) \to (H \to K)$ is equivalent to the conjunction of two rules. Indeed, by Lemma 1,

$$\begin{aligned}(F \to G) \to (H \to K) &\Leftrightarrow H \to ((F \to G) \to K) \\ &\Leftrightarrow H \to (((G \vee \neg F) \to K) \wedge (K \vee F \vee \neg G)) \\ &\Leftrightarrow (H \to ((G \vee \neg F) \to K)) \wedge (H \to (K \vee F \vee \neg G)) \\ &\Leftrightarrow ((H \wedge (G \vee \neg F)) \to K) \wedge (H \to (K \vee F \vee \neg G))\end{aligned}$$

It remains to consider the case when Π_1 contains more than one rule. In this case, Π_1 can be broken into two programs Π_1' and Π_2' with $|\Pi_1'| \leqslant |\Pi_1''| < |\Pi_1|$. Then, since

$$\Pi_1 \to \Pi_2 \Leftrightarrow (\Pi_1' \wedge \Pi_1'') \to \Pi_2 \Leftrightarrow \Pi_1' \to (\Pi_1'' \to \Pi_2)$$

the assertion follows by applying the induction hypothesis twice. □

Proof of Theorem 1
It is sufficient to prove that any theory consisting of a single formula is equivalent to a finite program, by structural induction. We also assume that such a formula contains atoms and connectives $\bot$, $\to$, and $\wedge$ only (all the other connectives can be

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

eliminated). Clearly, every atom and the connective $\bot$ are programs. For formulas of the form $F \wedge G$ and $F \to G$, we assume, as induction hypothesis, that there are two finite programs equivalent to F and G, respectively. Clearly, $F \wedge G$ is equivalent to the union of such two programs. The existence of a finite program equivalent to $F \to G$ follows from the induction hypothesis and Lemma 2. □

To understand how an arbitrary propositional theory is converted into a logic program, consider formula (2) as a logic program. First of all, we need to remove the disjunction, so we write it as

$$((r \to (q \to p)) \to (q \to p)) \wedge (((q \to p) \to r) \to r). \tag{8}$$

Then we convert each subformula that is not a logic program into a logic program bottom-up. Lemma 2 shows how each subformula can be converted, but simplifications or other transformations can also be applied. For instance, by Lemma 1, the subformula $r \to (q \to p)$ is converted into

$$((q \wedge (r \vee \neg\bot)) \to p) \wedge (q \to (p \vee \bot \vee \neg r)),$$

which can be equivalently rewritten as

$$(q \wedge r) \to p.$$

Consequently, the first conjunctive term of (8) becomes

$$((q \wedge r) \to p) \to (q \to p);$$

this formula, by Lemma 2 again, can be rewritten as

$$((q \wedge (p \vee \neg(q \wedge r))) \to p) \wedge (q \to (p \vee (q \wedge r) \vee \neg p)),$$

which can be simplified into

$$((q \wedge \neg r) \to p) \wedge (q \to (p \vee r \vee \neg p)).$$

With similar steps, we can rewrite the second conjunctive term of (8) as a logic program. As (8) becomes the conjunction of two programs, it becomes a program itself.

4 Proving Theorem 1 by using countermodels

The main idea of this technique is to start from the countermodels (under the logic of here-and-there) of some propositional theory T with a finite signature and construct a logic program, call it $\Pi(T)$, which has exactly the same set of countermodels. We can think about the construction of $\Pi(T)$ as a process where we start with an empty program, which would have as models all the possible interpretations, and go adding a rule per each model we want to remove. Note that, since there can only be a finite number of models and countermodels for a finite signature, $\Pi(T)$ is finite even if T is infinite. In the case of an infinite signature for T, we can define $\Pi(T)$ as the union of $\Pi(\{F\})$ (computed on the finite set of atoms that occur in F) for each $F \in T$. The form of each rule in $\Pi(T)$ is defined as follows.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

Definition 3 ($r_{X,Y}$)
Given an interpretation (X, Y) under some propositional signature Σ, we define $r_{X,Y}$ as the (nonnested) rule:

$$\left(\bigwedge_{b\in X} b\right) \wedge \left(\bigwedge_{c\in\Sigma-Y} \neg c\right) \rightarrow \bigvee_{a\in Y-X} (a \vee \neg a)$$

For instance, if $\Sigma = \{p, q, r\}$ then $r_{\{q\},\{p,q\}}$ is

$$q \wedge \neg r \rightarrow p \vee \neg p.$$

When the interpretation is total, the head (Y, Y) of $r_{Y,Y}$ is empty, leading to the constraint:

$$\left(\bigwedge_{b\in Y} b\right) \wedge \left(\bigwedge_{c\in\Sigma-Y} \neg c\right) \rightarrow \bot$$

For instance, with $\Sigma = \{p, q, r\}$, $r_{\{q\},\{q\}}$ is

$$q \wedge \neg p \wedge \neg r \rightarrow \bot.$$

From the definition of satisfaction in the logic of here-and-there, it is not difficult to see that (X, Y) is always a countermodel of $r_{X,Y}$. Moreover, when $X \subset Y$, this is indeed the *only* countermodel. When $X = Y$, however, Property 1 may lead to additional countermodels, as reflected by the following proposition.

Proposition 1
Given any interpretation (X, Y), an interpretation (X', Y') is a countermodel of $r_{X,Y}$ iff

i) $Y' = Y$, if $X = Y$, and
ii) $X' = X$, $Y' = Y$ otherwise.

To prove this result, the following lemma will be particularly useful.

Lemma 3
Let (X, Y) and (X', Y') be a pair of interpretations. Then,

$$(X', Y') \models body(r_{X,Y}) \text{ iff } X \subseteq X' \subseteq Y' \subseteq Y.$$

Proof
Note first that $X' \subseteq Y'$ trivially follows from the form of interpretations. We have that $(X', Y') \models body(r_{X,Y})$ iff

$$(X', Y') \models \text{b for all } b \in X, \text{and} (X', Y') \models \neg c \text{for all} \in \Sigma - Y.$$

By Property 2, we can rewrite this condition as

$$b \in X' \text{for all } b \in X, \text{and } c \notin Y' \text{for all } c \notin Y$$

and then as

$$b \in X \Rightarrow b \in X', \text{and } c \notin Y \Rightarrow c \notin Y'.$$

This is clearly equivalent to $X \subseteq X'$ and $Y' \subseteq Y$. □

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

Proof of Proposition 1

i) If $X = Y$ then $head(r_{X,Y}) = \bot$ so that $r_{X,Y}$ can be rewritten as $\neg body(r_{X,Y})$. Consequently, by Property 2, the countermodels of $r_{X,Y}$ are the interpretations (X', Y') such that Y' satisfies $body(r_{X,Y})$ in classical logic. It remains to notice that, in this case, $body(r_{X,Y})$ is the conjunction of all literals classically satisfied by Y, and so, this guarantees $Y' = Y$.

ii) If $X \neq Y$, we must have $X \subset Y$, by the form of interpretations. We show first that (X, Y) is not model of $r_{X,Y}$. To this aim, it suffices to show that $(X, Y) \models body(r_{X,Y})$ but $(X, Y) \not\models head(r_{X,Y})$. The satisfaction of the body trivially follows from Lemma 3 (take $X' = X$ and $Y' = Y$). For the head, take the disjunction $a \vee \neg a$ in $head(r_{X,Y})$ for each $a \in Y - X$. Since $a \notin X$ we get $(X, Y) \not\models a$ whereas since $a \in Y$, $Y \not\models \neg a$ and, by Property 2, $(X, Y) \not\models \neg a$.
Now, it remains to prove that any interpretation (X', Y') different from (X, Y) is a model of $r_{X,Y}$. Note first that as $Y - X$ is not empty, $r_{X,Y}$ is a tautology in classical logic, since its head contains at least a disjunction like $a \vee \neg a$. Therefore, $Y \models r_{X,Y}$ and we just have to show that $(X', Y') \models head(r_{X,Y})$ whenever $(X', Y') \models body(r_{X,Y})$. If we assume the latter, from Lemma 3 we conclude $X \subseteq X' \subseteq Y' \subseteq Y$, but as (X', Y') is different from (X, Y) then either $X \subset X'$ or $Y' \subset Y$. Assume first that there exists some atom $d \in X' - X$. Since $X' \subseteq Y$, we get $d \in Y - X$, and so d is one of the atoms a in the head of $r_{X,Y}$. As $d \in X'$, we have $(X', Y') \models d$ and so $(X', Y') \models head(r_{X,Y})$. On the other hand, if we assume that there exists some $d \in Y - Y'$, as $X \subseteq Y'$, we conclude $d \in Y - X$ and so, d is again one of the atoms in $head(r_{X,Y})$. Now, as $d \notin Y'$, by Property 2, $(X', Y') \models \neg d$ and $(X', Y') \models head(r_{X,Y})$.

□

A set S of interpretations is *total-closed* if given any total $(Y, Y) \in S$ then also $(X, Y) \in S$ for any $X \subseteq Y$. Clearly, a theory has a total-closed set of countermodels by Property 1. Then the main theorem immediately follows from the following theorem.

Theorem 2

Each total-closed set S of interpretations is the set of countermodels of a nonnested logic program:

$$\Pi(S) \stackrel{\text{def}}{=} \{r_{X,Y} \mid (X, Y) \in S\}$$

Proof

Any interpretation (X, Y) is a countermodel of $r_{X,Y}$ by Proposition 1, so every element of S is a countermodel of $\Pi(S)$. Now take any countermodel (X, Y) of $\Pi(S)$. By construction of $\Pi(S)$ and Proposition 1 this means that either $r_{X,Y}$ or $r_{Y,Y}$ belongs to $\Pi(S)$. Consequently, either (X, Y) or (Y, Y) is element of S. Since S is total-closed, we conclude that $(X, Y) \in S$. □

For instance, consider formula (2). As its six countermodels are

$$(\emptyset, \{q\}), \quad (\{q\}, \{q\}), \quad (\{q\}, \{p, q\}), \quad (\emptyset, \{q, r\}), \quad (\{q\}, \{q, r\}), \quad (\{q\}, \{p, q, r\}),$$

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

it is strongly equivalent to the six rules

$$\begin{aligned} \neg p \wedge \neg r &\rightarrow q \vee \neg q \\ q \wedge \neg p \wedge \neg r &\rightarrow \bot \\ q \wedge \neg r &\rightarrow p \vee \neg p \\ \neg p &\rightarrow q \vee \neg q \vee r \vee \neg r \\ q \wedge \neg p &\rightarrow r \vee \neg r \\ q &\rightarrow p \vee \neg p \vee r \vee \neg r. \end{aligned}$$

The one-to-one correspondence between total-closed sets of intepretations and the "classes" of strongly equivalent programs shown by Theorem 2 can be used to count the number of such classes.

Theorem 3
The number of different logic programs (modulo strong equivalence) that can be built for a finite signature of n atoms is

$$\prod_{i=0}^{n} \left(2^{2^i-1}+1\right)^{\binom{n}{i}}.$$

Proof
We need to count the number of total-closed sets S of interpretations. For any S, let S_Y be the subset of S consisting of the elements of the form (X, Y). Clearly each S_Y is independent from $S_{Y'}$ if $Y' \neq Y$, so the number of values of S is the product of the number of values of S_Y for each $Y \subseteq \Sigma$. If $(Y, Y) \notin S_Y$ then S_Y can independently contain or not each term of the form (X, Y), where X is a proper subset of Y. There are $2^{|Y|}-1$ of such subsets giving $2^{2^{|Y|}-1}$ total combinations. On the other hand, when $(Y, Y) \in S$, S_Y is completely determined as $\{(X, Y) \mid X \subseteq Y\}$, since S is total-closed. Consequently the total number of values for S is

$$\prod_{Y \subseteq \Sigma} \left(2^{2^{|Y|}-1}+1\right)$$

The final result is obtained by grouping all sets Y with the same cardinality, and by noticing that the sets Y of size i are $\binom{n}{i}$. □

5 Normal forms

The reader may have noticed that Theorem 2 seems, in principle, stronger than the original claim: we know now that any theory is strongly equivalent to a *nonnested* logic program. Nevertheless, this was also implicitly asserted by Theorem 1, since as we had seen, a nested program can always be transformed into a nonnested one under strong equivalence (Lifschitz *et al.* 1999). Thus, both proofs actually point out that nonnested logic programs act as a kind of normal form for the logic of here-and-there. Furthermore, they show a strong analogy to CNF in classical logic: we can understand the program as a conjunction of nonnested rules that, in their turn, are seen as clauses. The method in Section 4 is then completely analogous to

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

the construction of the classical CNF of a formula starting from its countermodels. In both cases, we build a rule/clause per each countermodel, and this clause refers to all the atoms of the signature.

In classical logic, however, we know that the construction of CNF from the theory countermodels is completely dual to the construction of DNF from its models. So, the following question arises immediately: is there some kind of *disjunctive normal form* for the logic of here-and-there that can be built starting from the theory models? The answer is affirmative, as we show next.

Definition 4
Given an interpretation (X, Y) for a finite signature Σ, let $c_{X,Y}$ be the formula

$$\left(\bigwedge_{a\in X} a\right) \wedge \left(\bigwedge_{b\in\Sigma-Y} \neg b\right) \wedge \left(\bigwedge_{c\in Y-X} \neg\neg c\right) \wedge \bigwedge_{d,e\in Y-X} (d \rightarrow e).$$

For instance, if $\Sigma = \{p, q, r\}$ then $c_{\{q\},\{p,q\}}$ is

$$q \wedge \neg r \wedge \neg\neg p \wedge (p \rightarrow p).$$

The interesting properties of $c_{X,Y}$ are stated in the next peoposition and theorem.

Proposition 2
The only interpretations that satisfy $c_{X,Y}$ are (X, Y) and (Y, Y).

Proof
Consider the following formulas $c'_{X,Y}$ and $c''_{X,Y}$:

$$c'_{X,Y} \stackrel{\text{def}}{=} \bigwedge_{c\in Y-X} \neg\neg c \qquad\qquad c''_{X,Y} \stackrel{\text{def}}{=} \bigwedge_{d,e\in Y-X} (d \rightarrow e)$$

Clearly, $c_{X,Y} = body(r_{X,Y}) \wedge c'_{X,Y} \wedge c''_{X,Y}$. We are going to see for which conditions the three terms $body(r_{X,Y})$, $c'_{X,Y}$, and $c''_{X,Y}$ are satisfied by an interpretation (X', Y'). Each formula $\neg\neg c$ in $c'_{X,Y}$ is satisfied by (X', Y') iff $Y' \models c$ by Property 2. That means that $(X', Y') \models c'_{X,Y}$ iff $Y - X \subseteq Y'$, or, equivalently, iff $Y \subseteq Y' \cup X$. On the other hand, by Lemma 3, $(X', Y') \models body(r_{X,Y})$ iff $X \subseteq X' \subseteq Y' \subseteq Y$. Notice that if $X \subseteq Y'$ then $Y' \cup X = Y'$. Consequently, $(X', Y') \models body(r_{X,Y}) \wedge c'_{X,Y}$ iff

$$X \subseteq X' \subseteq Y' = Y.$$

That means that the only models for $c_{X,Y}$ are the ones of the form (X', Y) with $X \subseteq X' \subseteq Y$, which satisfy $c''_{X,Y}$. Since Y contains all the consequences of the implications of $c''_{X,Y}$, then $Y \models c''_{X,Y}$. If we consider that X does not contain any of the antecedents, then $(X, Y) \models c''_{X,Y}$, and also $(Y, Y) \models c''_{X,Y}$ by Property 1. It remains to show that if $X \subset X' \subset Y$ then $(X', Y) \not\models c''_{X,Y}$. This is immediate, since $c''_{X,Y}$ contains an implication $d \rightarrow e$ with $d \in X'$ and $e \notin X'$. □

Theorem 4
Let T be a theory over a finite signature and let $F(T)$ denote the formula constructed from the models of T:

$$F(T) \stackrel{\text{def}}{=} \bigvee_{(X,Y)\ :\ (X,Y)\models T} c_{X,Y}.$$

Then T is equivalent to $F(T)$ under the logic of here-and-there.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

Proof
Take any model (X, Y) of T. Then, $c_{X,Y}$ is a disjunctive term in $F(T)$. Since $(X, Y) \models c_{X,Y}$ by Proposition 2, we can conclude that $(X, Y) \models F(T)$. Now take any model (X, Y) of $F(T)$. That means that, by Proposition 2, (X, Y) satisfies some disjunctive term $c_{X',Y}$ of $F(T)$, where

- $X = X'$, if $X \subset Y$, and
- $X' \subseteq Y$, if $X = Y$.

We know that $(X', Y) \models T$ by construction of $F(T)$. Consequently, if $X \subset Y$ then (X, Y) is a model of T because $(X', Y) = (X, Y)$. For the case $X = Y$, it is sufficient to notice that $(X, Y) = (Y, Y)$, and that $(X', Y) \models T$ implies $(Y, Y) \models T$ by Property 1. □

For instance, consider formula (2). As its 21 models are

$$(\emptyset,\emptyset),\quad (\emptyset,\{p\}),\quad (\{p\},\{p\}),\quad (\emptyset,\{r\}),\quad (\{r\},\{r\}),\quad (\emptyset,\{p,q\}),\quad (\{p\},\{p,q\}),$$
$$(\{p,q\},\{p,q\}),\quad (\{p\},\{q,r\}),\quad (\{p,q\},\{q,r\}),\quad (X,\{p,r\}),\quad (Y,\{p,q,r\}),$$
$$X \subseteq \{p,r\} \text{ and } Y \subseteq \{p,q,r\}, Y \neq \{q\})$$

it is strongly equivalent to the disjunction of

$$\neg p \wedge \neg q \wedge \neg r$$
$$\neg q \wedge \neg r \wedge \neg\neg p \wedge (p \rightarrow p)$$
$$p \wedge \neg q \wedge \neg r$$
$$\neg p \wedge \neg q \wedge \neg\neg r \wedge (r \rightarrow r)$$
$$\neg r \wedge \neg\neg p \wedge \neg\neg q \wedge (p \rightarrow p) \wedge (p \rightarrow q) \wedge (q \rightarrow p) \wedge (q \rightarrow q)$$
$$\vdots$$

As a final remark, note that $F(T)$ is a disjunction of clauses $c_{X,Y}$, which are in their turn conjunctions of expressions of a very restricted type: each one may involve negation, implications, and (at most two) atoms, but no conjunction or disjunction. It must be said, however, that we still ignore whether $F(T)$ may have any practical interest at all for computing answer sets.

6 Conclusions

We have shown that when we consider answer sets for arbitrary propositional theories (i.e., *Equilibrium Logic*) we actually obtain the same expressiveness than when we just deal with disjunctive programs that allow negation in the head. To this aim, we proved that for any propositional theory, there always exists a *strongly equivalent* logic program, i.e., a program that may safely replace the original theory in terms of answer sets, even in the context of additional information. In fact, to be precise, our result shows that disjunctive logic programs (with negation in the head) constitute a conjunctive normal form for the monotonic basis of Equilibrium Logic, the intermediate logic of *here-and-there*.

We actually provided two different proofs for this result: one consisting in a recursive syntactic transformation, and a second one that deals with countermodels of the propositional theory in the logic of *here-and-there*. The reason for presenting

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

the two proofs is that, in both cases, they provide technical tools for obtaining additional results. For instance, the proof based on the recursive transformation can be adapted to generalize the Completion Lemma for logic programs from Ferraris and Lifschitz (2005) to the case of propositional theories: the new statement of the lemma is almost as general as the statement from Ferraris (2005). On the other hand, the technique based on countermodels has helped us establish for the first time, as far as we know, the number of different programs (modulo strong-equivalence) that can be built with a given number of atoms. Furthermore, this technique has also opened research work under development (Cabalar *et al.* 2006) that studies the generation of a minimal program from a set of countermodels, analogously to the methods (Quine 1952; McCluskey 1956) for minimizing boolean functions in classical logic, well-known in digital circuit design.

The main focus in this technical note was just to prove the existence of a strongly equivalent program for any propositional theory, leaving more detailed related topics to be treated in subsequent work. Consequently, the methods we present to obtain the resulting program are mostly thought to achieve simple proofs of the main result, rather than to obtain an efficient computation or a more compact representation of the final program itself. As an example of the work directly derived from the current result, but perhaps more interesting from a computational viewpoint, Cabalar *et al.* (2005) presented a pair of alternative syntactic transformations to reduce a propositional theory into a strongly equivalent logic program. One of the transformations preserves the vocabulary of the original theory, but is exponential in the obtained program size, whereas the other is polynomial, but with the cost of adding auxiliary atoms. Cabalar *et al.* (2005) further confirm this complexity result for nonnested programs, actually proving that *no polynomial time transformation* to nonnested programs can be obtained if we preserve the original vocabulary. The existence of polynomial space–time strongly equivalent translations to programs with nested expressions is an open question.

Acknowledgments

We thank Vladimir Lifschitz and David Pearce for their invaluable help to improve the paper; in fact, the motivation for the current research was originated by their joint discussions.

References

Bibel, W. and Eder, E. 1993. A survey of logical calculi. In *The Handbook of Logic in AI and Logic Programming*. Vol. 1. Oxford University Press, 67–182.

Cabalar, P., Pearce, D. and Valverde, A. 2005. Reducing propositional theories in equilibrium logic to logic programs. In *Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA'05). Lecture Notes in Computer Science*, vol. 3808. Springer-Verlag, Berlin, Heidelberg, 4–17.

Cabalar, P., Pearce, D. and Valverde, A. 2006. Minimal logic programs. Unpublished draft.

Ferraris, P. 2005. Answer sets for propositional theories. In *Proceedings of the 8th International Conference. on Logic Programming and Nonmonotonic Reasoning*. Springer-Verlag, Berlin, Heidelberg, 119–131.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press

Ferraris, P. and Lifschitz, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5*, 45–74.

Gelfond, M. and Lifschitz, V. 1988. The stable models semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*. The MIT Press, Cambridge, MA, 1070–1080.

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

Heyting, A. 1930. Die formalen Regeln der intuitionistischen Logik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse*, 42–56.

Inoue, K. and Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming 35*(1), 39–78.

Jongh, D. D. and Hendriks, L. 2003. Characterization of strongly equivalent logic programs in intermediate logics. *TPLP 3*(3), 259–270.

Lifschitz, V., Pearce, D. and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2*, 526–541.

Lifschitz, V., Tang, L. R. and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

Lifschitz, V. and Woo, T. 1992. Answer sets in general nonmonotonic reasoning (preliminary report). In *Proceedings of Third International Conference on Principles of Knowledge Representation and Reasoning*, B. Nebel, C. Rich and W. Swartout, Eds. Morgan Kaufmann, San Mateo, CA, 603–614.

Lukasiewicz, J. 1941. Die logik und das grundlagenproblem. *Les Entries de Zürich sur les Fondaments et la Méthode des Sciences Mathématiques 12,* 6–9 (1938), 82–100.

McCluskey, E. J. 1956. Minimization of boolean functions. *Bell System Technical Journal 35*, 1417–1444.

Pearce, D. 1997. A new logical characterisation of stable models and answer sets. In *Non monotonic extensions of logic programming. Proceedings of NMELP'96. (LNAI 1216)*. Springer-Verlag.

Quine, W. V. O. 1952. The problem of simplifying truth functions. *American Mathematical Monthly 59*, 521–531.

https://doi.org/10.1017/S1471068407003110 Published online by Cambridge University Press