Minimal logic programs

Pedro Cabalar[1], David Pearce[2] and Agustín Valverde[3]*

[1] Corunna University (Corunna, Spain), cabalar@udc.es
[2] Universidad Rey Juan Carlos (Madrid, Spain), davidandrew.pearce@urjc.es
[3] University of Málaga (Málaga, Spain), a_valverde@ctima.uma.es

Abstract. bb We consider the problem of obtaining a minimal logic program strongly equivalent (under the stable models semantics) to a given arbitrary propositional theory. We propose a method consisting in the generation of the set of prime implicates of the original theory, starting from its set of countermodels (in the logic of Here-and-There), in a similar vein to the Quine-McCluskey method for minimisation of boolean functions. As a side result, we also provide several results about *fundamental* rules (those that are not tautologies and do not contain redundant literals) which are combined to build the minimal programs. In particular, we characterise their form, their corresponding sets of countermodels, as well as necessary and sufficient conditions for entailment and equivalence among them.

Keywords: logic programming, answer set programming, minimisation of boolean and multivalued functions.

1 Introduction

The nonmonotonic formalism of *Equilibrium Logic* [1], based on the nonclassical logic of *here-and-there* (henceforth: HT), yields a logical characterisation for the *answer set* semantics of logic programs [2]. By capturing concepts such as the strong equivalence of programs and providing a means to generalise all previous extensions of answer set semantics, HT and equilibrium logic provide a useful foundation for Answer Set Programming (ASP) (see [3,4]). There has recently been an increasing interest in ASP in the expressiveness of logic programs and the analysis of program rules in a more logical setting. Properties of Equilibrium Logic have allowed, for instance, the extension of the traditional definition of answer set (now equivalent to the concept of equilibrium model) to the most general syntax of arbitrary propositional [5] and first order [6] theories.

In the case of propositional theories (of crucial importance for current ASP solvers), in [7] it was actually shown that any arbitrary theory in Equilibrium Logic is strongly equivalent to (that is, can always be replaced by) a disjunctive logic program possibly allowing default negation in the head. In this way,

* This research was partially supported by Spanish MEC coordinated project TIN-2006-15455-C03, subprojects 01, 02, 03.

this type of rule constitutes a *normal form* for general ASP when dealing with arbitrary theories. Aside from its interest as an expressiveness result, as [5] has shown, the important concept of *aggregate* in ASP [8] can be captured by formulas with nested implications, making such syntactic extensions of practical value. In addition, the problem of generating a program from an arbitrary theory, or merely from another logic program, immediately raises the question of how good is this generation. A study of the complexity and efficiency of translation methods from arbitrary theories to logic programs was already carried out in [9]. A different and perhaps more practically relevant topic has to do with the quality or simplicity of the resulting program, rather than the effort required to obtain it. In ASP various methods, including equilibrium logic, have been used to analyse program simplification, both for ground programs [10,11,12,13] and more recently for programs with variables [14,15].

In this paper we consider the problem of generating a minimal or simpler logic program strongly equivalent to some initial propositional theory (what includes, of course, the case of a logic program). We propose a method that follows steps similar to the Quine-McCluskey algorithm [16,17], well-known from the problem of minimising boolean functions. Our algorithm computes the set of prime implicates of the original theory starting from its set of countermodels in HT. In a second step, a minimal set of prime implicates is selected to cover the whole set of countermodels. Obviously, these two steps mean a considerable computational cost whose reward is the guarantee of syntactic minimality of the obtained program, something not achieved before, to the best of our knowledge.

As we will discuss later in Section 7, the interest of such a minimisation method lies in its two main potential application areas: (i) it may become a useful tool for theoretical research, helping to find minimal logic program patterns obtained from translations of other constructions; and (ii) what possibly has a greater practical impact, it allows an offline minimisation of a ground logic program, which can be perhaps later (re-)used in several different contexts. Apart from the method itself, the paper also provides side results of additional interest from the viewpoint of expressiveness. We identify the normal form of *fundamental* rules, used to conform the minimal programs. In this way, these rules are non-trivial in the sense that they are not tautological and do not contain redundant literals. The paper characterises the set of countermodels of any fundamental rule and provides necessary and sufficient conditions (see Section 6) for capturing entailment and equivalence among them.

The paper is organised as follows. Section 2 recalls the Quine-McCluskey algorithm for minimising boolean functions. Next, Section 3 contains a brief overview of Equilibrium Logic, including some basic definitions about logic programs. Section 4 presents the algorithm for obtaining prime implicates and the next section contains a small example. Then, Section 6 presents an alternative definition of minimal program (based on semantic entailment) and finally, Section 7 concludes the paper. Proofs have been included in the Appendix of an extended version of this document [18].

2 Quine-McCluskey algorithm

The *Quine-McCluskey algorithm* [16,17] allows obtaining a minimal normal form equivalent to any classical propositional theory Γ. The algorithm is dual in the sense that it can be equally used to obtain a minimal DNF from the set of models of Γ or to obtain a minimal CNF from its set of countermodels. Although the former is the most popular version, we will actually focus on the latter for the sake of comparison. After all, as shown in [7], logic programs constitute a CNF for HT. The Quine-McCluskey algorithm computes the prime implicates of a theory Γ starting from its countermodels. To get a minimal CNF, a second algorithm (typically, *Petrick's method* [19]) must be used afterwards to select a minimal subset of prime implicates that suffice to cover all countermodels of Γ. In what follows, we skip many definitions from classical propositional logic assuming the reader's familiarity.

Let At be a set of atoms (called the propositional *signature*). We represent a classical propositional interpretation as a set of atoms $I \subseteq At$ selecting those assigned truth value 1 (true). As usual, a *literal* is an atom p (*positive* literal) or its negation $\neg p$ (*negative* literal). A *clause* is a disjunction of literals and a *formula in CNF* is a conjunction of clauses. The empty disjunction and conjunction respectively correspond to $\bot$ and $\top$. We will use letters $C, D, \ldots$ to denote clauses. Satisfaction of formulas is defined in the usual way. A clause is said to be *fundamental* if it contains no repeated atom occurrences. Non-fundamental clauses are irrelevant: any repeated literal can be just removed, and any clause containing $p \vee \neg p$ too, since it is a tautology. We say that a clause C *subsumes* a clause D, written $C \subseteq D$, iff all literals of C occur in D.

Proposition 1. *Let C, D be fundamental clauses. Then, $C \subseteq D$ iff $\models C \rightarrow D$.*

Proposition 2. *An interpretation I is a countermodel of a fundamental clause C iff $p \notin I$ for all positive literals p occurring in C and $p \in I$ for all negative literals $\neg p$ occurring in C.*

Proposition 2 provides a compact way to represent a fundamental clause C (and its set of countermodels). We can just define a *labelling*, let us write it $\overrightarrow{C}$, that for each atom $p \in At$ contains a pair:

- $(p, 1) \in \overrightarrow{C}$ when $\neg p$ occurs in C (meaning p true in all countermodels),
- $(p, 0) \in \overrightarrow{C}$ when p occurs as positive literal in C (meaning p false in all countermodels) or
- $(p, \text{-}) \in \overrightarrow{C}$ if p does not occur in C (meaning that the value of p is indifferent).

Note that it is also possible to retrieve the fundamental clause C corresponding to an arbitrary labelling $\overrightarrow{C}$, so the relation is one-to-one. Typically, we further omit the atom names (assuming alphabetical ordering) and we just write $\overrightarrow{C}$ as a vector of labels. As an example, given the ordered signature $\{p, q, r, s\}$ and the clause $C = \neg p \vee q \vee \neg s$, we would have $\overrightarrow{C} = \{(p, 1), (q, 0), (r, \text{-}), (s, 1)\}$ or

simply $\overrightarrow{C}$ = 10-1. When all atoms in the signature occur in a fundamental clause, the latter is said to be *developed*. Proposition 2 implies that a developed clause C has a single countermodel – note that in this case $\overrightarrow{C}$ does not contain indifference symbols. By abuse of notation, we identify a countermodel I with its corresponding developed clause $\{(p,1) \mid p \in I\} \cup \{(p,0) \mid p \in At \setminus I\}$.

An *implicate* C of a theory Γ is any fundamental clause satisfying $\Gamma \models C$. Clearly, countermodels of C are countermodels of Γ too. As any CNF formula Π equivalent to Γ will consist of implicates of Γ, the task of finding Π can be seen as a search of a set of implicates whose countermodels suffice to *comprise* the whole set of countermodels of Γ. However, if we want Π to be as simple as possible, we will be interested in implicates with minimal size. An implicate of Γ is said to be *prime* iff it is not subsumed by another implicate of Γ. The Quine-McCluskey algorithm computes all implicates of Γ by collecting their labelling vectors in successive sets S_i. In fact, at each step, S_i collects all the vectors with i indifference labels, starting with $i = 0$ indifferences (i.e. with S_0 equal to the set of countermodels of Γ). To compute the next set S_{i+1} the algorithm groups pairs of implicate vectors $\overrightarrow{C}, \overrightarrow{D}$ in S_i that just differ in one atom p, say, for instance $(p,1) \in \overrightarrow{C}$ and $(p,0) \in \overrightarrow{D}$. Then, it inserts a new vector $\overrightarrow{E} = \left(\overrightarrow{C} \setminus \{(p,1)\}\right) \cup \{(p,\text{-})\}$ in S_{i+1} and marks both $\overrightarrow{C}$ and $\overrightarrow{D}$ as non-prime (it is easy to see that clauses C, D are both subsumed by the new implicate E). When no new vector is formed, the algorithm finishes, returning the set of non-marked (i.e., prime) implicates.

Example 1. The table in Figure 2 schematically shows the result of applying the algorithm for a signature $At = \{p, q, r, s\}$ and starting from a set of countermodels shown in column S_0. Numbers between parentheses represent the countermodels of a given implicate (using the decimal representation of their vector of labels). The horizontal lines separate the implicates in each S_i by their number of 1's. Finally, '*' marks mean that the implicate has been subsumed by another one. The prime (i.e., non-marked) implicates are therefore: -100 ($\neg q \vee r \vee s$), 10-- ($\neg p \vee q$), 1--0 ($\neg p \vee s$) and 1-1- ($\neg p \vee \neg r$).

3 Equilibrium Logic

We begin defining the (monotonic) logic of *here-and-there* (HT). A *formula* is defined in the usual way as a well-formed combination of the operators $\bot, \wedge, \vee, \rightarrow$ with atoms in a propositional signature At. We define $\neg\varphi \stackrel{\text{def}}{=} \varphi \rightarrow \bot$ and $\top \stackrel{\text{def}}{=} \neg\bot$. As usual, by *theory* we mean a set of formulas.

An *HT-interpretation* is a pair $\langle H, T\rangle$ of sets of atoms with $H \subseteq T$. When $H = T$ the HT-interpretation is said to be *total*. We recursively define when $\langle H, T\rangle$ *satisfies* a formula φ, written $\langle H, T\rangle \models \varphi$ as follows:

- $\langle H, T\rangle \not\models \bot$

S_0		S_1		S_2	
		-100	$(4,12)$		
$*$ 0100	(4)	$*$ 100-	$(8,9)$		
$*$ 1000	(8)	$*$ 10-0	$(8,10)$		
$*$ 1001	(9)	$*$ 1-00	$(8,12)$		
$*$ 1010	(10)	$*$ 10-1	$(9,11)$	10--	$(8,9,10,11)$
$*$ 1100	(12)	$*$ 101-	$(10,11)$	1--0	$(8,10,12,14)$
$*$ 1011	(11)	$*$ 1-10	$(10,14)$	1-1-	$(10,11,14,15)$
$*$ 1110	(14)	$*$ 11-0	$(12,14)$		
$*$ 1111	(15)	$*$ 1-11	$(11,15)$		
		$*$ 111-	$(14,15)$		

Fig. 1. Example of computation of Quine McCluskey algorithm.

- $\langle H,T\rangle \models p$ if $p \in H$, for any atom $p \in At$
- $\langle H,T\rangle \models \varphi \wedge \psi$ if $\langle H,T\rangle \models \varphi$ and $\langle H,T\rangle \models \psi$
- $\langle H,T\rangle \models \varphi \vee \psi$ if $\langle H,T\rangle \models \varphi$ or $\langle H,T\rangle \models \psi$
- $\langle H,T\rangle \models \varphi \rightarrow \psi$ if both (i) $\langle H,T\rangle \not\models \varphi$ or $\langle H,T\rangle \models \psi$; and (ii) $T \models \varphi \rightarrow \psi$ (in classical logic).

An HT-interpretation is a *model* of a theory Γ if it satisfies all formulas in Γ. As usual, we say that a theory Γ *entails* a formula φ, written $\Gamma \models \varphi$, when any model of Γ is a model of φ. A formula true in all models is said to be *valid* or a *tautology*. An *equilibrium model* of a theory Γ is any total model $\langle T,T\rangle$ of Γ such that no $\langle H,T\rangle$ with $H \subset T$ is model of Γ. *Equilibrium Logic* is the logic induced by equilibrium models.

An alternative definition of HT can be given in terms of a three-valued semantics (Gödel's three-valued logic). Under this viewpoint, an HT-interpretation $M = \langle H,T\rangle$ can be seen as a three-valued mapping $M : At \rightarrow \{0,1,2\}$ where, for any atom p, $M(p) = 2$ if $p \in H$ (p is *true*), $M(p) = 0$ if $p \in At \setminus T$ (p is *false*) and $M(p) = 1$ if $p \in T \setminus H$ (p is *undefined*). The valuation of formulas is defined so that the valuation of conjunction (resp. disjunction) is the minimum (resp. maximum) value, $M(\bot) = 0$ and $M(\varphi \rightarrow \psi) = 2$ if $M(\varphi) \leq M(\psi)$ or $M(\varphi \rightarrow \psi) = M(\psi)$ otherwise. Finally, models of a theory Γ are captured by those HT-interpretations M such that $M(\varphi) = 2$ for all $\varphi \in \Gamma$.

Lemma 1. *In HT, the following formulas are valid:*

$$\alpha \wedge \varphi \wedge \neg\varphi \rightarrow \beta \tag{1}$$

$$\alpha \wedge \varphi \rightarrow \beta \vee \varphi \tag{2}$$

$$(\alpha \wedge \varphi \rightarrow \beta \vee \neg\varphi) \leftrightarrow (\alpha \wedge \varphi \rightarrow \beta) \tag{3}$$

$$(\alpha \wedge \neg\varphi \rightarrow \beta \vee \varphi) \leftrightarrow (\alpha \wedge \neg\varphi \rightarrow \beta) \tag{4}$$

$$(\alpha \rightarrow \beta) \rightarrow (\alpha \wedge \gamma \rightarrow \beta) \tag{5}$$

$$(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta \vee \gamma) \tag{6}$$

The following property of HT will have important consequences for adapting the Quine-McCluskey method to this logic:

Property 1. For any theory Γ, $\langle H, T\rangle \models \Gamma$ implies $\langle T, T\rangle \models \Gamma$.

We can rephrase the property in terms of countermodels: if $\langle T, T\rangle$ is a countermodel of Γ then any $\langle H, T\rangle$ will also be a countermodel. As a result, contrarily to what happened in classical logic (or in Łukasiewicz three-valued logic, for instance), we cannot use an *arbitrary* set S of HT-interpretations to represent the countermodels of some theory. Let us define the *total-closure* (*t-closure*) of a set of interpretations S as:

$$S^t \stackrel{\text{def}}{=} S \cup \{\ \langle H, T\rangle \mid \langle T, T\rangle \in S,\ H \subseteq T\ \}$$

We say that S is *t-closed* if $S^t = S$.

Property 2 (Theorem 2 in [7]). Each t-closed set S of interpretations is the set of countermodels of a logic program.

This fact was used in [7] to compute the number of possible HT-theories (modulo semantic equivalence) for a finite number n of atoms by counting the possible t-closed sets of interpretations. The resulting amount happens to be considerably smaller than 2^{3^n}, which corresponds, for instance, to the number of possible theories we can form in Łukasiewicz logic.

3.1 Logic programs

A *logic program* is a conjunction of clauses (also called *rules*) of the form:

$$a_1 \wedge \cdots \wedge a_n \wedge \neg b_1 \wedge \cdots \wedge \neg b_m \rightarrow c_1 \vee \cdots \vee c_h \vee \neg d_1 \vee \cdots \vee \neg d_k \qquad (7)$$

with $n, m, h, k \geq 0$ and all subindexed letters representing atoms. As before, empty disjunctions stand for $\bot$ while empty conjunctions stand for $\top$. The *empty rule* would therefore correspond to $\top \rightarrow \bot$ or simply $\bot$. We will use letters r, r', ... to denote rules. We will sometimes abbreviate a rule r like (7) using the expression:

$$B_r^+ \wedge \neg B_r^- \rightarrow Hd_r^+ \vee \neg Hd_r^-$$

where B_r^+ (resp. B_r^-) denotes the sets of atoms occurring positively (resp. negatively) in the antecedent or body of r, while Hd_r^+ (resp. Hd_r^-) denotes the sets of atoms occurring positively (resp. negatively) in the consequent or head of r.

For sets of rules of form (7), equilibrium models and answer sets coincide, [3]. Two rules r, r' are said to be *strongly equivalent*, in symbols $r \equiv_s r'$, if for any set Π of rules, $\Pi \cup \{r\}$ and $\Pi \cup \{r'\}$ have the same equilibrium models; strong equivalence for sets of rules is defined analogously. Rules (and sets of rules) are strongly equivalent iff they are logically equivalent in HT, i.e. they have the same HT-models, [3]. Hence, strongly equivalent rules can be interchanged without loss in any context, and deduction in HT provides a key instrument for replacing rules by equivalent, simpler ones.

As happened with non-fundamental clauses in classical logic, some rules may contain irrelevant or redundant information. To avoid this, we define:

Definition 1 (fundamental rule). *A rule r is said to be* fundamental *when all pairwise intersections of sets $Hd_r^+, Hd_r^-, B_r^+, B_r^-$ are empty, with the possible exception of $Hd_r^+ \cap Hd_r^-$.*

Lemma 2. *For any rule r:*

i) If one of the intersections $B_r^+ \cap B_r^-$, $B_r^+ \cap Hd_r^+$ and $B_r^- \cap Hd_r^-$ is not empty, then r is a tautology.

ii) Otherwise, r is strongly equivalent to the fundamental rule:

$$r' : B_r^+ \wedge \neg B_r^- \rightarrow (Hd_r^+ \setminus B_r^-) \vee \neg(Hd_r^- \setminus B_r^+).$$

The proof follows from (1)-(4). Lemma 2 plus the main result in [7] implies

Theorem 1. *Fundamental rules constitute a normal form for the logic of HT.*

4 Generation of prime implicates

As can be imagined, our goal of obtaining minimal programs will depend in a crucial way on how we define the relation *simpler than* between two programs, or more specifically, between two rules. To this aim, we can either rely on a syntactic or on a semantic definition. For classical logic, Proposition 1 shows that both choices are interchangeable: we can either use the inclusion of literals among fundamental clauses $C \subseteq D$ (a syntactic criterion) or just use entailment $C \models D$ instead. In the case of HT, however, we will see later that this correspondence is not preserved in a direct way. For this reason, we maintain two different concepts of *smaller rule* in HT: *entailment*, written $r \models r'$ with its usual meaning; and *subsumption*, written $r \subseteq r'$ and defined below.

Definition 2 (subsumption). *A rule r* subsumes *another rule r', written $r \subseteq r'$, when $Hd_r^+ \subseteq Hd_{r'}^+$, $Hd_r^- \subseteq Hd_{r'}^-$, $B_r^+ \subseteq B_{r'}^+$ and $B_r^- \subseteq B_{r'}^-$.*

Rule subsumption $r \subseteq r'$ captures the idea that r results from removing some literals in r'. It follows that the empty rule $\bot$ subsumes all rules. As usual, we handle the strict versions of subsumption, written $r \subset r'$ and meaning both $r \subseteq r'$ and $r \neq r'$, and of entailment, written $r \models\!\!\!\not\, r'$ and meaning that $r \models r'$ but $r' \not\models r$. From (5),(6) we conclude that subsumption is stronger than entailment:

Theorem 2. *For any pair of rules r, r', if $r \subseteq r'$ then $r \models r'$.*

However, in contrast to Proposition 1 for classical logic, in HT the converse direction does not hold. As a counterexample:

Example 2. Given rules $r : p \rightarrow q$ and $r' : p \wedge \neg q \rightarrow \bot$ we have $r \models r'$ but $r \not\subseteq r'$.

In the rest of this section we will focus on syntactic subsumption, providing later (Section 6) a variation that uses semantic entailment instead.

Definition 3 (Syntactically simpler program). *We say that program Π is* syntactically simpler *than program Π', written $\Pi \preceq \Pi'$, if there exists some $\Gamma \subseteq \Pi'$ such that Π results from replacing each rule $r' \in \Gamma$ by some r, $r \subseteq r'$.*

In other words, we might remove some rules from Π' and, from the remaining rules, we might remove some literals at any position. Note that $\Pi \subseteq \Pi'$ implies $\Pi \preceq \Pi'$ but the converse implication does not hold, and that, of course, syntactically simpler does not generally entail any kind of semantic relation.

Theorem 3. *Let Π be a $\preceq$-minimal logic program among those strongly equivalent to some theory Γ. Then Π consists of fundamental rules.*

Definition 4 (implicate/prime implicate). *A fundamental rule r is an* implicate *of a theory Γ iff $\Gamma \models r$. Moreover, r is said to be* prime *iff it is not strictly subsumed by another implicate of Γ.*

To sum up, we face again the same setting as in classical logic: to find a $\preceq$-minimal program Π equivalent to some theory Γ means to obtain a set of prime implicates that cover all countermodels of Γ. Therefore, it is crucial that we are able to characterise the countermodels of fundamental rules, as we did with Proposition 2 in the classical case. Given a fundamental rule r, we define the set of HT-interpretations $CMs(r) \stackrel{\text{def}}{=}$

$$\left\{ \langle H, T\rangle \mid B_r^+ \subseteq H,\ B_r^- \cap T = \emptyset,\ Hd_r^+ \cap H = \emptyset,\ Hd_r^- \subseteq T \right\}$$

Theorem 4. *Given a fundamental rule r and an HT-interpretation M: $M \not\models r$ iff $M \in CMs(r)^t$.*

Theorem 4 and the fact that $CMs(r)$ is never empty leads to:

Observation 1 *Fundamental rules always have countermodels.*

Given two fundamental rules r, r' we say that r *covers* r' when $CMs(r') \subseteq CMs(r)$. Notice that this definition of *covering* is stronger than entailment: if $CMs(r') \subseteq CMs(r)$ then the same still holds for $CMs(r')^t \subseteq CMs(r)^t$ and from Theorem 4 we conclude $r \models r'$. On the other hand, it is very easy to see that covering is a weaker condition than subsumption:

Proposition 3. *If $r \subseteq r'$ then $CMs(r') \subseteq CMs(r)$.*

As with classical clauses, the countermodels in $CMs(r)$ can also be compactly described by a mapping of the set of atoms into a set of labels (which contains more elements now). To do so, note first that the definition of $CMs(r)$ can be directly rephrased in terms of three-valued interpretations as follows: $M \in CMs(r)$ iff the following conditions hold for any atom p: (i) $p \in B_r^+ \Rightarrow M(p) = 2$; (ii) $p \in B_r^- \Rightarrow M(p) = 0$; (iii) $p \in Hd_r^+ \Rightarrow M(p) \neq 2$; and (iv) $p \in Hd_r^- \Rightarrow M(p) \neq 0$. As r is a fundamental rule, the last two conditions are not mutually exclusive. Thus, when $p \in Hd_r^+ \cap Hd_r^-$ we would just have $M(p) = 1$.

Definition 5 (Rule labelling). *Given a fundamental rule r, we define its* labelling $\overrightarrow{r}$ *as a set containing, for each atom $p \in At$, a pair:*

$(p, 2)$ *if* $p \in B_r^+$	$(p, \bar{2})$ *if* $p \in Hd_r^+ \setminus Hd_r^-$
$(p, 0)$ *if* $p \in B_r^-$	$(p, \bar{0})$ *if* $p \in Hd_r^- \setminus Hd_r^+$
$(p, 1)$ *if* $p \in Hd_r^+ \cap Hd_r^-$	$(p, \text{-})$ *if p does not occur in r*

Note that $(p, \bar{2})$ stands for $M(p) \neq 2$ whereas $(p, \bar{0})$ stands for $M(p) \neq 0$. We will sometimes write $\overrightarrow{r}(p) = v$ when $(p, v) \in \overrightarrow{r}$ and we also use the abbreviation $\overrightarrow{r}|_{\neq p}$ to stand for $\{(q, v) \in \overrightarrow{r} \mid q \neq p\}$. As an example of labelling, given the signature $At = \{a, b, c, d, e, p, q\}$, the fundamental clause $r : a \wedge b \wedge \neg d \rightarrow e \vee p \vee \neg p \vee \neg q$ would correspond to $\overrightarrow{r} = 22\text{-}0\bar{2}1\bar{0}$. In fact, there actually exists a one-to-one correspondence between a fundamental rule and its labelling, i.e., we can get back the rule r from $\overrightarrow{r}$. Thus, the set of countermodels $CMs(r)$, or its corresponding labelling $\overrightarrow{r}$, can be used to univocally represent the original rule r. It is important to remember, however, that $CMs(r)$ *is not* the set of countermodels of r – by Theorem 4, the latter actually corresponds to $CMs(r)^t$.

A single countermodel $M = \langle H, T \rangle$ can also be seen as a labelling that assigns a label in $\{0, 1, 2\}$ to each atom in the signature. Using the above definitions, the rule[4] r_M corresponding to M would be: $B^+_{r_M} = H$, $B^-_{r_M} = At \setminus T$ and $Hd^+_{r_M} = Hd^-_{r_M} = T \setminus H$. It is easy to see that rules derived from countermodels are $\preceq$-*maximal* – in fact, there is no way to construct *any* fundamental rule strictly subsumed by r_M. These implicates will constitute the starting set S_0 of the algorithm, which will generate new implicates that always subsume at least one of those previously obtained. The proposed algorithm (Generation of Prime Implicates, GPI) is shown in Table (a) of Figure 4. The algorithm applies three basic matching steps to implicates whose labels just differ in one atom p. Note that the possible matches would be much more than three, but only three cases i), ii) and iii) yield any effect. To understand the purpose behind these three operations, consider the form of the corresponding involved fundamental rules. Using the correspondence between labelling and rule we obtain Table (b) in Figure 4.

Proposition 4. *Let α, β and p be arbitrary propositional formulas. For the three cases i), ii), iii) in Table (b) Figure 4, the equivalence $r'' \leftrightarrow r \wedge r'$ is an HT-valid formula.*

Table (b) in Figure 4 also explains the way in which the algorithm assigns the non-prime marks. For instance, in cases i) and ii) only r is subsumed by the new generated rule r'' and so, GPI leaves r' without being marked in the current step $i + 1$. However, in case iii) the obtained rule r'' subsumes not only r and r' but also $\alpha \wedge \neg p \rightarrow \beta$ and $\alpha \wedge p \rightarrow \beta$ which were left previously unmarked, and must therefore be marked now too.

To prove termination and correctness of the algorithm, we will provide a pair of useful definitions. Given a label $v \in \{0, 1, 2, \bar{2}, \bar{0}, \text{-}\}$ we define its *weight*, $w(v)$, as a value in $\{0, 1, 2\}$ specified as follows: $w(0) = w(1) = w(2) = 0$, $w(\bar{2}) = w(\bar{0}) = 1$ and $w(\text{-}) = 2$. Intuitively, $w(v)$ points out the number of matches like i), ii) and iii) in the algorithm required to obtain label v. The *weight* of a fundamental rule r, $w(r)$, is defined as the sum of the weights $w(v)$ of all labels in $\overrightarrow{r}$, that is, $w(r) \stackrel{\text{def}}{=} \Sigma_{p \in At}\ w(\overrightarrow{r}(p))$. To put an example, given $\overrightarrow{r} = 22\text{-}0\bar{2}1\bar{0}$, $w(r) = 0 + 0 + 2 + 0 + 1 + 0 + 1 = 4$. Clearly, there exists a

[4] In fact, constructing a rule per each countermodel was one of the techniques used in [7] to show that any arbitrary theory is equivalent to a set of rules.

$S_0 := \{\overrightarrow{r_M} \mid M \not\models \Gamma\}$;
$i := 0$;
while $S_i \neq \emptyset$ **do**
  $S_{i+1} := \emptyset$;
  for each $\overrightarrow{r}, \overrightarrow{r}' \in S_i$ **such that** $\overrightarrow{r} \neq \overrightarrow{r}'$ and $\overrightarrow{r}|_{\neq p} = \overrightarrow{r}'|_{\neq p}$ for some p **do**
    case i): $(p,1) \in \overrightarrow{r}$ and $(p,0) \in \overrightarrow{r}'$ **then**
      Add $\overrightarrow{r}'' : \overrightarrow{r}|_{\neq p} \cup \{\ (p,\bar{2})\ \}$ to S_{i+1};
      Mark $\overrightarrow{r}$ as non-prime;
    case ii): $(p,1) \in \overrightarrow{r}$ and $(p,2) \in \overrightarrow{r}'$ **then**
      Add $\overrightarrow{r}'' : \overrightarrow{r}|_{\neq p} \cup \{\ (p,\bar{0})\ \}$ to S_{i+1};
      Mark $\overrightarrow{r}$ as non-prime;
    case iii): $(p,\bar{2}) \in \overrightarrow{r}$ and $(p,\bar{0}) \in \overrightarrow{r}'$ **then**
      Add $\overrightarrow{r}'' : \overrightarrow{r}|_{\neq p} \cup \{\ (p,\text{-})\ \}$ to S_{i+1};
      Mark $\overrightarrow{r}$, $\overrightarrow{r}'$, $\overrightarrow{r}|_{\neq p} \cup \{(p,0)\}$ and $\overrightarrow{r}|_{\neq p} \cup \{(p,2)\}$ as non-prime;
    end case
  end for each
  $i := i + 1$;
end while

(a) Pseudocode.

case i)	case ii)	case iii)
$r : \alpha \to \beta \vee p \vee \neg p$	$r : \alpha \to \beta \vee p \vee \neg p$	$r : \alpha \to \beta \vee p$
$r' : \alpha \wedge \neg p \to \beta$	$r' : \alpha \wedge p \to \beta$	$r' : \alpha \to \beta \vee \neg p$
$r'' : \alpha \to \beta \vee p$	$r'' : \alpha \to \beta \vee \neg p$	$r'' : \alpha \to \beta$

(b) Rule form of generated implicates.

Fig. 2. Algorithm for generation of prime-implicates (GPI).

maximum value for a rule weight which corresponds to assigning label '-' to all atoms, i.e., $2 \cdot |At|$.

Lemma 3. *Let $IMP_i(\Gamma)$ denote the set of implicates of Γ of weight i. Then, in the algorithm GPI, $S_i = IMP_i(\Gamma)$.*

Theorem 5. *Algorithm GPI stops after a finite number of steps $i = n$ and $\bigcup_{i=0\ldots n} S_i$ is the set of implicates of Γ.*

Theorem 6. *Let Γ be some arbitrary theory. The set of unmarked rules obtained by GPI is the set of prime implicates of Γ.*

5 Examples

To illustrate algorithm GPI, we begin considering its application to translate the theory $\Gamma = \{(\neg p \to q) \to p\}$ into a strongly equivalent minimal logic program. The set of countermodels is represented by the labels at the first column of the

table below, assuming alphabetical ordering for atoms in $\{p, q\}$. For instance, labelling 10 stands for $M(p) = 1$ and $M(q) = 0$, that is, $M = \langle \emptyset, \{p\} \rangle$ whereas 02 stands for $M'(p) = 0$ and $M'(q) = 2$, i.e., $M' = \langle \{q\}, \{q\} \rangle$. The remaining columns show all matches that take place when forming each remaining S_i. The marks '*' show those implicates that result marked at each match.

S_0	S_1	S_2
10	$10, *11 \longmapsto 1\bar{2}$	
02	$02, *12 \longmapsto \bar{2}2$	$*1\bar{2}, *1\bar{0} \longmapsto 1\text{-}$ (mark $*10, *12$ too)
12	$02, *01 \longmapsto 0\bar{0}$	$\bar{2}2, *\bar{2}1 \longmapsto \bar{2}\bar{0}$
01	$12, *11 \longmapsto 1\bar{0}$	$0\bar{0}, *1\bar{0} \longmapsto \bar{2}\bar{0}$
11	$01, *11 \longmapsto \bar{2}1$	

Notice that implicate 01 in S_0 is not marked until we form -1 in S_2. Observe as well that implicate $\bar{0}\bar{2}$ is obtained in two different ways at that same step. Since no match can be formed at S_2 the algorithm stops at that point. If we carefully observe the previous table, the final unmarked labellings $\overrightarrow{r}$ (i.e., prime implicates r) are shown below:

$\overrightarrow{r}$	r	$CMs(r)$	$CMs(r)^t$
02	$\neg p \wedge q \rightarrow \bot$	02	02, 01
$\bar{2}2$	$q \rightarrow p$	02, 12	02, 12, 01
$0\bar{0}$	$\neg p \rightarrow \neg q$	02, 01	02, 01
1-	$p \vee \neg p$	10, 11, 12	10, 11, 12
$\bar{2}\bar{0}$	$p \vee \neg q$	11, 12, 01, 02	11, 12, 01, 02

Finally, to obtain a minimal program we must select now a minimal set of implicates that covers all the countermodels of Γ. To this aim, any method from minimisation of boolean functions (like for instance, Petrick's method [19]) is still applicable here, as we just have a "coverage" problem and do not depend any more on the underlying logic. Without entering into details, the application of Petrick's method, for instance, would proceed to the construction of a chart:

	01	$\bar{2}1$	$0\bar{0}$	1-	$\bar{2}\bar{0}$
10				×	
02	×	×	×		×
12		×		×	×
01	×	×	×		×
11				×	×

Implicate 1- will be *essential*, as it is the only one covering countermodel 10. If we remove it plus the countermodels it covers we obtain the smaller chart:

	02	$\bar{2}2$	$0\bar{0}$	$\bar{2}\bar{0}$
02	×	×	×	×
01	×	×	×	×

that just points out that any of the remaining implicates can be used to form a minimal program equivalent to Γ. So, we get the final four $\preceq$-minimal programs:

$$\Pi_1 = \{p \vee \neg p,\ \neg p \wedge \neg q \rightarrow \bot\} \qquad \Pi_3 = \{p \vee \neg p,\ \neg p \rightarrow \neg q\}$$
$$\Pi_2 = \{p \vee \neg p,\ q \rightarrow p\} \qquad \Pi_4 = \{p \vee \neg p,\ \neg q \vee p\}$$

This example can be used to compare to transformations in [9] (for reducing arbitrary theories to logic programs) that applied on Γ yield program Π_2 plus the additional rule $\{p \vee \neg p \vee \neg q\}$ trivially subsumed by $p \vee \neg p$ in Π_2. Nevertheless, not all examples are so trivial. For instance, the theory $\{(\neg p \rightarrow q) \rightarrow \neg(p \rightarrow r)\}$ from Example 1 in [9] yielded in that case a translation consisting of the six rules $\{(q \wedge \neg p \rightarrow \bot), (q \rightarrow \neg r), (\neg p \rightarrow \neg p), (\neg p \vee \neg r), (\neg p \rightarrow \neg p \vee \neg q), (\neg p \vee \neg q \vee \neg r)\}$ that can be trivially reduced (after removing tautologies and subsumed rules) to $\{(q \wedge \neg p \rightarrow \bot), (q \rightarrow \neg r), (\neg p \vee \neg r)\}$ but which is not a minimal program. In fact, the GPI algorithm obtains (among others) the strongly equivalent minimal program $\{(q \wedge \neg p \rightarrow \bot), (\neg p \vee \neg r)\}$. Note that another important advantage is that the GPI method obtains *all* the possible minimal programs when several choices exist.

As an example of the use of GPI on logic programs, consider the case where we must combine two pieces of program from different knowledge sources. If we take, for instance $\Pi = \{(\neg r \wedge q \rightarrow p), (\neg p \rightarrow r \vee s)\}$ and $\Pi' = \{(r \rightarrow p), (\neg r \rightarrow q)\}$, these two programs are minimal when analysed independently, whereas none of the rules in $\Pi \cup \Pi'$ is subsumed by another in that set. After applying GPI to $\Pi \cup \Pi'$ however, we obtain that the unique minimal strongly equivalent program is just $\{(\neg r \rightarrow p), (r \rightarrow p), (\neg r \rightarrow q)\}$.

6 Entailment

Looking at the first example in the previous section, and particularly at the first implicates chart, it may be surprising to find that, although all of them are prime, some implicates entail others. The reason for this was explained before: contrarily to the classical case, (our definition of) syntactically simpler does not mean entailment in HT. Note, for instance, how all implicates excepting 1- are entailed by $\overline{2}\overline{0}$. We claim, however, that our criterion is still reasonable, as there is no clear reason why $\neg p \vee q$ should be syntactically simpler than $p \wedge \neg q \rightarrow \bot$ or $p \rightarrow q$. On the other hand, it seems that most imaginable criteria for syntactic simplicity should be probably stronger than our $\preceq$ relation. In this section we consider an alternative semantic definition of prime implicate that relies on the concept of entailment.

Definition 6 (s-prime implicate). *An implicate r of a theory Γ is said to be* semantically prime *(*s-prime *for short) if there is no implicate r' of Γ such that $r' \nvDash r$.*

The only s-prime implicates in the example would be now 1- and $\overline{2}\overline{0}$. The intuition behind this variant is that, rather than focusing on syntactic simplicity, we are interested now in collecting a minimal set of rules that are as strong as

possible, but considering rule by rule (remember that the program as a whole has to be strongly equivalent to the original theory). Since subsumption implies entailment, finding the s-prime implicates could be done as a postprocessing to the GPI algorithm seen before (we would just remove some prime implicates that are not s-prime). Another, more efficient alternative would be a suitable modification of the algorithm to disregard entailed implicates from the very beginning. For space reasons, we do not enter into the details of this modification. What is perhaps more important is that in any of these two alternatives for computing s-prime implicates we can use the following simple syntactic characterisations of entailment and equivalence of rules.

Theorem 7. *For every rule r, and every fundamental rule r', $r \models r'$ iff the following conditions hold:*

1. $B_r^- \subseteq B_{r'}^-$
2. $Hd_r^- \subseteq Hd_{r'}^- \cup B_{r'}^+$
3. $B_r^+ \subseteq B_{r'}^+ \cup Hd_{r'}^-$
4. $Hd_r^+ \subseteq Hd_{r'}^+ \cup B_{r'}^-$
5. *Either* $B_r^+ \cap Hd_{r'}^- = \varnothing$ *or* $Hd_r^+ \cap Hd_{r'}^+ = \varnothing$.

For instance, it is easy to see that rules r and r' in Example 2 satisfy the above conditions. Theorem 7 leads to a characterisation of equivalence between fundamental rules.

Corollary 1. *If r and r' are fundamental rules, then: $r \equiv_s r'$ iff the following conditions hold:*

1. $B_r^- = B_{r'}^-$
2. $Hd_r^+ = Hd_{r'}^+$
3. $Hd_r^- \cup B_r^+ = Hd_{r'}^- \cup B_{r'}^+$
4. *If* $Hd_r^+ = Hd_{r'}^+ \neq \varnothing$, *then* $B_r^+ = B_{r'}^+$ *and* $Hd_r^- = Hd_{r'}^-$

So we can actually classify fundamental rules into two categories: if their positive head is not empty $Hd_r^+ \neq \varnothing$, then there is no other equivalent fundamental rule. On the other hand, if $Hd_r^+ = \varnothing$, then r can be called a *constraint*, since it is strongly equivalent to $B_r^+ \wedge Hd_r^- \wedge \neg B_r^- \rightarrow \bot$ but also to any rule that results from removing atoms from the constraint's positive body $B_r^+ \cup Hd_r^-$ and adding them negated in the head (this is called *shifting* in [20]). For instance:
$p \wedge q \wedge \neg r \rightarrow \bot \ \equiv_s \ q \wedge \neg r \rightarrow \neg p \ \equiv_s \ p \wedge \neg r \rightarrow \neg q \ \equiv_s \ \neg r \rightarrow \neg p \vee \neg q$

7 Discussion and related work

We provided a method for generating a minimal logic program strongly equivalent to some arbitrary propositional theory in equilibrium logic. We actually considered two alternatives: a syntactic one, exclusively treating programs with a smaller syntax in the sense of rule and literal occurrences; and a semantic one, in the sense of programs that make use of rules that are as deductively strong as possible.

Some results in the paper were known before or were previously given for more restrictive cases. For instance, Lemma 2 (in presence of Observation 1) provides a necessary and sufficient condition for tautological rules. This becomes a confirmation using HT logic of Theorem 4.4 in [21]. On the other hand, our Theorem 7 is a generalisation of Theorem 6 in [13] for programs with negation in the head. Finally, the *shifting* operation we derived from Corollary 1 was introduced before in [20] (see Corollary 4.8 there) – in fact, we have been further able to show that it preserves *strong* equivalence.

As commented in the introduction, we outline two main potential applications for our method: as a help for theoretical research and as a tool for simplifying ground logic programs. In the first case, the method may become a valuable support when exploring a particular group or pattern of expressions. Consider for instance the translation into logic programs of other constructions (say aggregates, classes of complex expressions in arbitrary theories, high level action languages, etc) or even think about an hypothetical learning algorithm that must explore a family of rules to cover input examples. In all these cases, we may be reasonably interested in finding the smallest strongly equivalent representation of the set of rules handled. To put an example, the translation of a nested implication $(p \rightarrow q) \rightarrow r$ into the logic program $\{(q \rightarrow r), (\neg p \rightarrow r), (p \vee \neg q \vee r)\}$ found in [7] and of crucial importance in that paper, can be now shown, using our algorithm, to be the most compact (strongly equivalent) possible one.

Regarding the application of our method as a tool for simplifying ground logic programs, we may use the basic methodology proposed in [22] to clarify the situation. Three basic types of simplifications are identified, depending on whether the result is: (i) smaller in size (less atoms or rules, rules with less literals, etc); (ii) belonging to a simpler class (Horn clauses, normal programs, disjunctive programs, etc); and (iii) more efficient for a particular algorithm or solver. Apart from this classification, they also distinguish between online (i.e., embedded in the algorithm for computing answer sets) and offline simplifications. Our orientation in this paper has been completely focused on (offline) simplifications of type (i). This has a practical interest when we deal with program modules to be (re-)used in several different contexts, or for instance, with rules describing a transition in a planning scenario, as this piece of program is then replicated many times depending on the plan length. Adapting the algorithm to simplifications of type (ii) is left for future work: we can force the generation of a particular class of programs (for instance, by just disregarding implicates not belonging to that class), or use the method to show that this generation is not possible. Another topic for future study is the analysis of complexity.

References

1. Pearce, D.: Equilibrium logic. Ann Math Artif Intell **47** (2006) 3–41
2. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K., eds.: Proc. of the Fifth International Conference on Logic Programming, ICLP'88 (Seattle, WA, USA), Cambridge, Massachusetts, The MIT Press (1988) 1070–1080

3. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2**(4) (2001) 526–541
4. Lifschitz, V., Pearce, D., Valverde, A.: A characterization of strong equivalence for logic programs with variables. In: Proc. of the 9th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'07). (2007) to appear.
5. Ferraris, P.: Answer sets for propositional theories. In: Proc. of the 8th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'05). (2005) 119–131
6. Ferraris, P., Lee, J., Lifschitz, V.: A new perspective on stable models. In: Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI'07). (2007)
7. Cabalar, P., Ferraris, P.: Propositional theories are strongly equivalent to logic programs. Theory and Practice of Logic Programming (2007) to appear.
8. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: Proc. of the 9th European Conf. on Artificial Intelligence (JELIA'04). (2004)
9. Cabalar, P., Pearce, D., Valverde, A.: Reducing propositional theories in equilibrium logic to logic programs. In Bento, C., Cardoso, A., Dias, G., eds.: Progress in Artificial Intelligence, Proc. of the 12th Portuguese Conf. on Artificial Intelligence, EPIA'05, Covilhã, Portugal, December 5-8, 2005. Volume 3808 of Lecture Notes in Computer Science., Springer (2005) 4–17
10. Osorio, M., Navarro, J.A., Arrazola, J.: Equivalence in answer set programming. In Pettorossi, A., ed.: Proc. LOPSTR-01, Springer LNCS 2372 (2001) 57–75
11. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying logic programs under uniform and strong equivalence. In: Proc. of LPNMR'04. (2004) 87–99
12. Pearce, D.: Simplifying logic programs under answer set semantics. In: Proc. of the Intl. Conf. on Logic Programming (ICLP'04). (2004) 210–224
13. Lin, F., Chen, Y.: Discovering classes of strongly equivalent logic programs. In: Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI'05). (2005) 516–521
14. Eiter, T., Fink, M., Tompits, H., Traxler, P., Woltran, S.: Replacements in non-ground answer-set programming. In: Proc. of KR'06, AAAI (2006) 340–350
15. Fink, M., Pichler, R., Tompits, H., Woltran, S.: Complexity of rule redundancy in non-ground answer-set programming over finite domains. In: LPNMR2007, Springer (2007)
16. Quine, W.V.O.: The problem of simplifying truth functions. American Mathematical Monthly **59** (1952) 521–531
17. McCluskey, E.J.: Minimization of boolean functions. Bell System Technical Journal **35** (1956) 1417–1444
18. Cabalar, P., Pearce, D., Valverde, A.: Minimal logic programs (extended report) (2007) Technical report available at `http://www.dc.fi.udc.es/~cabalar/minlp-ext.pdf`.
19. Petrick, S.R.: A direct termination of the irredundant forms of a boolean function from the set of prime implicants. Technical Report AFCRC-TR-56-110, Air Force Cambridge Res. Center, Cambridge, MA, USA (1956)
20. Inoue, K., Sakama, C.: Negation as failure in the head. Journal of Logic Programming **35**(1) (1998) 39–78
21. Inoue, K., Sakama, C.: Equivalence of logic programs under updates. Lecture Notes in Artificial Intelligence, Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004) **3229** (2004) 174–186
22. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Formal methods for comparing and optimizing nonmonotonic logic programs. (last updated 2007) Research project web page `http://www.kr.tuwien.ac.at/research/eq.html`.