TLP **7** (6): 697–744, 2007. © 2007 Cambridge University Press
doi:10.1017/S1471068407003031 First published online 25 May 2007 Printed in the United Kingdom

Automated Verification of Weak Equivalence within the SMODELS *System**

TOMI JANHUNEN and EMILIA OIKARINEN
Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory for Theoretical Computer Science, P.O. Box 5400, FI-02015 TKK, Finland
(*e-mail:* `Tomi.Janhunen@tkk.fi, Emilia.Oikarinen@tkk.fi`)

submitted 7 December 2005; revised 30 June 2006; accepted 23 August 2006

Abstract

In answer set programming (ASP), a problem at hand is solved by (i) writing a logic program whose answer sets correspond to the solutions of the problem, and by (ii) computing the answer sets of the program using an *answer set solver* as a search engine. Typically, a programmer creates a series of gradually improving logic programs for a particular problem when optimizing program length and execution time on a particular solver. This leads the programmer to a meta-level problem of ensuring that the programs are equivalent, i.e., they give rise to the same answer sets. To ease answer set programming at methodological level, we propose a translation-based method for verifying the equivalence of logic programs. The basic idea is to translate logic programs P and Q under consideration into a single logic program EQT(P, Q) whose answer sets (if such exist) yield counter-examples to the equivalence of P and Q. The method is developed here in a slightly more general setting by taking the *visibility* of atoms properly into account when comparing answer sets. The translation-based approach presented in the paper has been implemented as a translator called LPEQ that enables the verification of weak equivalence within the SMODELS system using the same search engine as for the search of models. Our experiments with LPEQ and SMODELS suggest that establishing the equivalence of logic programs in this way is in certain cases much faster than naive cross-checking of answer sets.

KEYWORDS: Answer set programming, weak equivalence, programming methodology, program optimization

1 Introduction

Answer set programming (ASP) has recently been proposed and promoted as a self-standing logic programming paradigm (Marek and Truszczyński 1999; Niemelä 1999; Gelfond and Leone 2002). Indeed, the paradigm has received increasing attention since efficient implementations such as DLV (Leone et al. 2006) and SMODELS (Simons et al. 2002) became available in the late nineties. There are numerous

* This is an extended version of a paper (Janhunen and Oikarinen 2002) presented at the 8th European Workshop on Logics in Artificial Intelligence in Cosenza, Italy.

applications of ASP ranging, e.g., from product configuration (Soininen et al. 2001) to a decision support system of the space shuttle (Balduccini et al. 2001). The variety of answer set solvers is also rapidly growing as new solvers are being developed constantly for the sake of efficiency. The reader is referred elsewhere (Janhunen et al. 2000; Lin and Zhao 2002; Lierler and Maratea 2004; Janhunen 2004; Anger et al. 2005; Gressmann et al. 2005; Liu and Truszczyński 2005) in this respect.

Despite the declarative nature of ASP, the development of programs resembles that of programs in conventional programming. That is, a programmer often develops a series of gradually improving programs for a particular problem, e.g., when optimizing execution time and space. As a consequence, the programmer needs to ensure that subsequent programs which differ in performance yield the same output. This setting leads us to the problem of verifying whether given two logic programs P and Q have exactly the same answer sets, i.e., are *weakly equivalent* (denoted $P \equiv Q$). Looking at this from the ASP perspective, weakly equivalent programs produce the same solutions for the problem that they formalize.

There are also other notions of equivalence that have been proposed for logic programs. Lifschitz et al. (2001) consider P and Q *strongly equivalent*, denoted $P \equiv_{\mathrm{s}} Q$, if and only if $P \cup R \equiv Q \cup R$ for all programs R each of which acts as a potential context for P and Q. By setting $R = \emptyset$ in the definition of $\equiv_{\mathrm{s}}$, we obtain that $P \equiv_{\mathrm{s}} Q$ implies $P \equiv Q$ but the converse does not hold in general. Consequently, the question whether $P \equiv Q$ holds remains open whenever $P \not\equiv_{\mathrm{s}} Q$ turns out to be the case. This implies that verifying $P \equiv Q$ remains as a problem of its own, which cannot be fully compensated by verifying $P \equiv_{\mathrm{s}} Q$. As suggested by its name, $\equiv_{\mathrm{s}}$ is a much stronger relation than $\equiv$ in the sense that the former relates far fewer programs than the latter. This makes $\equiv_{\mathrm{s}}$ better applicable to *subprograms* or *program modules* constituting larger programs rather than complete programs for which $\equiv$ is more natural. Moreover, there is a number of characterizations of strong equivalence (Lifschitz et al. 2001; Pearce et al. 2001; Lin 2002; Turner 2003), which among other things indicate that strongly equivalent programs are classically equivalent, but not necessarily vice versa as to be demonstrated in Example 4.4. Thus, strong equivalence permits only classical *program transformations*, i.e., substitutions of a program module (a set of rules) by another. In contrast to this, weak equivalence is more liberal as regards program transformations some of which are not classical but still used in practice; the reader may consult Example 4.3 for an instance.

For the reasons discussed above, we concentrate on the case of complete programs and weak equivalence in this article. We develop a method that extends Janhunen and Oikarinen (2002) and Janhunen and Oikarinen (2004), and hence fully covers the class of *weight constraint programs* supported by the front-end LPARSE (Syrjänen 2001) used with the SMODELS system (Simons et al. 2002). The key idea in our approach is to *translate* logic programs P and Q under consideration into a single logic program $\mathrm{EQT}(P, Q)$ which has an answer set if and only if P has an answer set that is not an answer set of Q. Such answer sets, if found, act as *counter-examples* to the equivalence of P and Q. Consequently, the equivalence of P and Q

can be established by showing that $\mathrm{EQT}(P, Q)$ and $\mathrm{EQT}(Q, P)$ have no answer sets.[1] Thus the existing search engine of the SMODELS system can be used for the search of counter-examples and there is no need to develop a special purpose search engine for the verification task. Moreover, we are obliged to develop the underlying theory in a more general setting where programs may involve *invisible* atoms, e.g., generated by LPARSE when compiling weight constraints. The basic idea is that such atoms should be neglected by equivalence relations but this is not the case for $\equiv$ and $\equiv_{\mathrm{s}}$. To this end, we apply yet another equivalence relation, namely *visible equivalence* denoted by $\equiv_{\mathrm{v}}$ (Janhunen 2003; Janhunen 2006). This relation is compatible with $\equiv$ in the sense that these equivalence relations coincide in the absence of invisible atoms. In fact, we develop a translation-based verification method for $\equiv_{\mathrm{v}}$ and characterize the class of SMODELS programs for which the method is guaranteed to work by constraining the use of invisible atoms. This class is identified as the class of programs possessing *enough visible atoms*. Most importantly, this property is shared by weight constraint programs produced by the front-end LPARSE during grounding.

The rest of this paper is organized as follows. The rule-based syntax of logic programs supported by the current SMODELS system is described in Section 2. It is then explained in Section 3 how the semantics of such rules is covered by the stable model semantics proposed by Gelfond and Lifschitz (1988). Section 4 introduces the notion of *visible equivalence* mentioned above. We perform a preliminary complexity analysis of the problem of verifying $P \equiv_{\mathrm{v}} Q$ for P and Q given as input. Unfortunately, recent complexity results (Eiter et al. 2005) suggest discouraging rises of complexity in the presence of invisible atoms. Thus we need to impose additional constraints in order to keep the verification problem in **coNP**; thus enabling the use of SMODELS as search engine in a feasible way. In Section 5, we present our translation-based method for verifying the visible equivalence of SMODELS programs. The correctness of the method is also addressed. The resulting complexity classifications are then concluded in Section 5.1. Section 6 concentrates on the case of weight constraint programs supported by the front-end LPARSE of the SMODELS system and shows how programs in the extended language are covered by the translation-based method. Section 7 is devoted to experiments that we have performed with an implementation of the translation-based method, a translator called LPEQ, and the SMODELS system. The results indicate that in certain cases verifying the equivalence of SMODELS programs using LPEQ is one or two orders of magnitude faster than naive cross-checking of stable models. Finally, the paper is finished by a brief conclusion in Section 8.

2 Programs in the smodels Language

The goal of this section is to make the reader acquainted with the rule-based language supported by the current SMODELS system (Simons et al. 2002). Definition 2.1 lists

[1] Turner (Turner 2003) develops an analogous transformation for weight constraint programs and *strong equivalence*. Moreover, Eiter et al. (2004) cover the case of disjunctive programs under strong and *uniform equivalence* and present the respective transformations.

five forms of rules which constitute the knowledge representation primitives of the system. Besides *basic rules* (1) of conventional *normal logic programs*, there are also other expressions such as *constraint rules* (2), *choice rules* (3), *weight rules* (4), and *compute statements* (5). These extensions have been carefully chosen to be directly and efficiently implementable in the search engine of the SMODELS system (Simons et al. 2002). It should be stressed that the front-end of the system, LPARSE (Syrjänen 2001), admits a more liberal use of constraint and weight rules Syrjänen (2004) but we postpone the discussion of such features until Section 6.

Definition 2.1
Rules are expressions of the forms

$$h \leftarrow a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m \tag{1}$$

$$h \leftarrow c\ \{a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m\} \tag{2}$$

$$\{h_1, \ldots, h_l\} \leftarrow a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m \tag{3}$$

$$h \leftarrow w \leqslant \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, {\sim}b_1 = w_{b_1}, \ldots, {\sim}b_m = w_{b_m}\} \tag{4}$$

$$\mathsf{compute}\ \{a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m\} \tag{5}$$

where $n \geqslant 0$, $m \geqslant 0$, and $l > 0$, and where h, each a_i, each b_j, and each h_k are atoms and c, each w_{a_i}, each w_{b_j}, as well as w, are natural numbers.

The symbol $\sim$ occurring in Definition 2.1 denotes *default negation* or *negation as failure to prove* which differs from classical negation in an important way (Gelfond and Lifschitz 1990). We define positive and negative *default literals* in the standard way as atoms a or their negations ${\sim}a$, respectively. The exact model-theoretic semantics of rules is deferred until Section 3, but – informally speaking – the rules listed above are used to draw conclusions as follows.

- The head h of a basic rule (1) can be inferred if the atoms $a_1, \ldots, a_n$ are inferable by other rules whereas the atoms $b_1, \ldots, b_m$ are *not*.
- The head h of a constraint rule (2) can be inferred if the number of inferable atoms among $a_1, \ldots, a_n$ plus the number of non-inferable atoms among $b_1, \ldots, b_m$ is at least c.
- A choice rule (3) is similar to a basic rule except that any subset of the non-empty set of head atoms $\{h_1, \ldots, h_l\}$ can be inferred instead of a single head atom h. Note that it is not necessary to infer any of the head atoms.
- A weight rule (4) involves summing as follows: the weight w_{a_i} (resp. w_{b_j}) is one of the summands if and only if a_i is inferable (resp. b_j is not inferable). The head h can be inferred if such a sum of weights is at least w.
- The default literals involved in a compute statement (5) act as direct constraints saying that the atoms $a_1, \ldots, a_n$ should be inferable by some rules whereas the atoms $b_1, \ldots, b_m$ should not.

A couple of observations follows. A constraint rule (2) becomes equivalent to a basic rule (1) given that $c = n + m$. A weight rule (4) reduces to a constraint rule (2) when all weights are equal to 1 and $w = c$. Moreover, default literals may be assigned different weights in different weight rules, i.e., weights are local in this sense. The types of rules defined above are already well-suited for a variety of knowledge representation and reasoning tasks in a number of domains. Example 2.2 demonstrates the use of rules in a practical setting. The reader is referred elsewhere (Niemelä 1999; Marek and Truszczyński 1999; Simons et al. 2002; Gelfond and Leone 2002) for more examples how to represent knowledge in terms of rules.

Example 2.2
Consider the task of describing coffee orders using rules[2] introduced in Definition 2.1. The nine rules given below form our formalization of this domain which should be self-explanatory. The compute statement in the end identifies the orders of interest to be those for which "acceptable" can be inferred.

$$\begin{array}{l}
\{\mathsf{coffee}, \mathsf{tea}, \mathsf{biscuit}, \mathsf{cake}, \mathsf{cognac}\}.\\
\{\mathsf{cream}, \mathsf{sugar}\} \leftarrow \mathsf{coffee}.\\
\mathsf{cognac} \leftarrow \mathsf{coffee}.\\
\{\mathsf{milk}, \mathsf{lemon}, \mathsf{sugar}\} \leftarrow \mathsf{tea}.\\
\mathsf{mess} \leftarrow \mathsf{milk}, \mathsf{lemon}.\\
\mathsf{happy} \leftarrow 1\ \{\mathsf{biscuit}, \mathsf{cake}, \mathsf{cognac}\}.\\
\mathsf{bankrupt} \leftarrow 6 \leqslant \{\mathsf{coffee} = 1, \mathsf{tea} = 1, \mathsf{biscuit} = 1, \mathsf{cake} = 2, \mathsf{cognac} = 4\}.\\
\mathsf{acceptable} \leftarrow \mathsf{happy}, {\sim}\mathsf{bankrupt}, {\sim}\mathsf{mess}.\\
\mathsf{compute}\ \{\mathsf{acceptable}\}.
\end{array}$$

We define a *logic program* P as a finite[3] set of ground rules of the forms (1)–(5) given in Definition 2.1. It follows that programs under consideration are fully instantiated and thus consist of ground atoms which are parallel to *propositional atoms*, or *atoms* for short in the sequel. The *Herbrand base* of a logic program P can be any fixed set of atoms $\mathrm{Hb}(P)$ containing all atoms that actually appear in the rules of P. Furthermore, we view $\mathrm{Hb}(P)$ as a part of the program which corresponds to defining a logic program as pair $\langle P, \mathrm{Hb}(P)\rangle$ where $\mathrm{Hb}(P)$ acts as the symbol table of P. The flexibility of this definition has important consequences. First, the *length* $||P||$ of the program, i.e., the number of symbols needed to represent P as a string, becomes dependent on $|\mathrm{Hb}(P)|$. This aspect becomes relevant in the analysis of translation functions (Janhunen 2006). Second, the explicit representation of $\mathrm{Hb}(P)$ enables one to keep track of atoms whose occurrences have been removed from a program, e.g., due to program optimization. For instance, the program $\langle\{a \leftarrow {\sim}b.\ \}, \{a, b\}\rangle$ can be rewritten as $\langle\{a.\ \}, \{a, b\}\rangle$ under stable model semantics.

There is a further aspect of atoms that affects the way we treat Herbrand bases, namely the *visibility* of atoms. It is typical in answer set programming that only certain atoms appearing in a program are relevant for representing the solutions of the problem being solved. Others act as auxiliary concepts that might not appear in other programs written for the same problem. As a side effect, the models/interpretations assigned to two programs may differ already on the basis

[2] Rules are separated with full stops and the symbol "←" is dropped from a basic rule (1) or a choice rule (3) if the *body* of the rule is empty ($n = 0$ and $m = 0$).

[3] This reflects the fact that the theory being presented/developed here is closely related to an actual implementation, the SMODELS engine, which admits only finite sets of ground rules.

of auxiliary atoms. Rather than introducing an explicit hiding mechanism in the language itself, we let the programmer decide the visible part of $\mathrm{Hb}(P)$, i.e., $\mathrm{Hb}_{\mathrm{v}}(P) \subseteq \mathrm{Hb}(P)$ which determines the set of *hidden* atoms $\mathrm{Hb}_{\mathrm{h}}(P) = \mathrm{Hb}(P) - \mathrm{Hb}_{\mathrm{v}}(P)$. The ideas presented so far are combined as follows.

Definition 2.3
A logic program in the SMODELS system (or an SMODELS program for short) is a triple $\langle P, \mathrm{Hb}_{\mathrm{v}}(P), \mathrm{Hb}_{\mathrm{h}}(P)\rangle$ where

1. P is a finite set of rules of the forms (1)–(5);
2. $\mathrm{Hb}_{\mathrm{v}}(P)$ and $\mathrm{Hb}_{\mathrm{h}}(P)$ are finite and disjoint sets of atoms and determine the visible and hidden Herbrand bases of the program, respectively; and
3. all atoms occurring in P are contained in $\mathrm{Hb}(P) = \mathrm{Hb}_{\mathrm{v}}(P) \cup \mathrm{Hb}_{\mathrm{h}}(P)$.

Finally, we define $\mathrm{Hb}_{\mathrm{a}}(P)$ as the set of atoms of $\mathrm{Hb}(P)$ not occurring in P.[4]

Note that the atoms of $\mathrm{Hb}_{\mathrm{a}}(P)$ can be viewed as *additional* atoms that just extend $\mathrm{Hb}(P)$. By a slight abuse of notation, we often use P rather than the whole triple when referring to a program $\langle P, \mathrm{Hb}_{\mathrm{v}}(P), \mathrm{Hb}_{\mathrm{h}}(P)\rangle$. To ease the treatment of programs, we make some default assumptions regarding the sets $\mathrm{Hb}(P)$ and $\mathrm{Hb}_{\mathrm{v}}(P)$. Unless otherwise stated, we assume that $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P)$, $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$, and $\mathrm{Hb}_{\mathrm{a}}(P) = \emptyset$, i.e., $\mathrm{Hb}(P)$ contains only atoms that actually appear in P.

Example 2.4
Given $P = \{a \leftarrow {\sim}b.\ \}$, the default interpretation is that $\mathrm{Hb}(P) = \{a, b\}$, $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P) = \{a, b\}$, and $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$. To make an exception in this respect, we have to add explicitly that, e.g., $\mathrm{Hb}_{\mathrm{v}}(P) = \{a, c\}$ and $\mathrm{Hb}_{\mathrm{h}}(P) = \{b\}$. Together with P these declarations imply that $\mathrm{Hb}_{\mathrm{a}}(P)$ is implicitly assigned to $\{c\}$.

Generally speaking, the set $\mathrm{Hb}_{\mathrm{v}}(P)$ can be understood as a *program interface* of P and it gives the basis for comparing the program P with other programs of interest. The atoms in $\mathrm{Hb}_{\mathrm{h}}(P)$ are to be hidden in any such comparisons.

3 Stable model semantics

In this section, we review the details of *stable model semantics* proposed by Gelfond and Lifschitz (1988). Stable models were first introduced in the context of *normal logic programs*, i.e., logic programs that solely consist of basic rules (1), but soon they were generalized for other classes involving syntactic extensions. In addition to recalling the case of normal programs, it is also important for us to understand how the semantic principles underlying stable models can be applied to the full syntax of SMODELS programs introduced in Section 2. Yet another generalization will be presented in Section 6 where the class of weight constraint programs is addressed.

The class of normal programs includes *positive programs* that are free of default negation, i.e., $m = 0$ for all rules (1) of such programs. The standard way to determine

[4] The atoms in $\mathrm{Hb}_{\mathrm{a}}(P)$ are made false by stable semantics to be introduced in Section 3.

the semantics of any positive program P is to take the *least model* of P, denoted by $\mathrm{LM}(P)$, as the semantical basis (Lloyd 1987). This is a particular classical model of P which is minimal with respect to subset[5] inclusion and also unique with this property. Moreover, the least model $\mathrm{LM}(P)$ coincides with the intersection of all classical models of P. Consequently, an atom $a \in \mathrm{Hb}(P)$ is a logical consequence of P in the classical sense if and only if $a \in \mathrm{LM}(P)$. It is also important to realize that the semantic operator $\mathrm{LM}(\cdot)$ is inherently monotonic: $P \subseteq Q$ implies $\mathrm{LM}(P) \subseteq \mathrm{LM}(Q)$ for any positive normal logic programs P and Q.

Gelfond and Lifschitz (1988) show how the least model semantics can be generalized to cover normal logic programs. The idea is to reduce a normal logic program P with respect to a model candidate M by pre-interpreting negative literals that appear in the rules of P. The resulting program P^M – also known as the Gelfond-Lifschitz *reduct* of P – contains a reduced rule $h \leftarrow a_1, \dots, a_n$ if and only if there is a rule (1) in P so that the negative literals $\sim b_1, \dots, \sim b_m$ in the body are satisfied in M. This makes P^M a positive program whose semantics is determined in the standard way, i.e., using its least model (Lloyd 1987).

Definition 3.1 (*Gelfond and Lifschitz (1988)*)
For a normal logic program P, an interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of P if and only if $M = \mathrm{LM}(P^M)$.

For a positive program P, the reduct $P^M = P$ for any $M \subseteq \mathrm{Hb}(P)$ implying that $\mathrm{LM}(P)$ coincides with the unique stable model of P. Unlike this, stable models need not be unique in general: a normal logic program P may possess several stable models or no stable model at all. However, this is not considered as a problem in answer set programming, since the aim is to capture solutions to the problem at hand with the stable models of a program that is constructed to formalize the problem. In particular, if there are no solutions for the problem, then the logic programming representation is not supposed to possess any stable models.

Simons (1999) shows how the stable model semantics can be generalized for the other kinds of rules presented in Section 2. However, the reduced program is not explicitly present in the semantical definitions given by him. This is why we resort to an alternative definition, which appears as Definition 3.4 below. It will be explained in Section 6 how the forthcoming definition can be understood as a special case of that given by Simons et al. (2002) for more general classes of rules. In contrast to their definitions that involve *deductive closures* of sets of rules, we define stable models purely in model-theoretic terms using the least model concept.

Given a logic program P, an *interpretation* I is simply a subset of $\mathrm{Hb}(P)$ defining which atoms a are considered to be *true* ($a \in I$) and which *false* ($a \notin I$). By the following definition, we extend the satisfaction relation $I \models r$ for the types of rules r under consideration. In particular, let us point out that negative default literals are treated classically at this point.

[5] It is assumed that interpretations are represented as sets of atoms evaluating to true.

Definition 3.2
Given an interpretation $I \subseteq \text{Hb}(P)$ for an SMODELS program P,

1. A positive default literal a is satisfied in I (denoted $I \models a$) $\iff$ $a \in I$.
2. A negative default literal $\sim a$ is satisfied in I (denoted $I \models \sim a$) $\iff$ $I \not\models a$.
3. A set of default literals L is satisfied in I (denoted $I \models L$) $\iff$ $I \models l$ for every $l \in L$.
4. A basic rule r of the form (1) is satisfied in I (denoted $I \models r$) $\iff$ $I \models \{a_1, \ldots, a_n, \sim b_1, \ldots, \sim b_m\}$ implies $I \models h$.
5. A constraint rule r of the form (2) is satisfied in I (denoted $I \models r$) $\iff$ $c \leqslant |\{a_i \mid I \models a_i\} \cup \{\sim b_j \mid I \models \sim b_j\}|$ implies $I \models h$.
6. A choice rule r of the form (3) is always satisfied in I.
7. A weight rule r of the form (4) is satisfied in I (denoted $I \models r$) $\iff$

$$\begin{aligned} w & \leqslant \text{WS}_I(a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \sim b_1 = w_{b_1}, \ldots, \sim b_m = w_{b_m}) \\ & = \sum_{I \models a_i} w_{a_i} + \sum_{I \models \sim b_j} w_{b_j} \end{aligned} \tag{6}$$

   implies $I \models h$.
8. A compute statement s of the form (5) is satisfied in I (denoted $I \models s$) $\iff$ $I \models \{a_1, \ldots, a_n, \sim b_1, \ldots, \sim b_m\}$.
9. A program P is satisfied in I ($I \models P$) $\iff$ $I \models r$ for every $r \in P$.

The equality in (6) determines how weighted literal sets are evaluated. Given an interpretation I and an assignment of weights to default literals as in the body of a weight rule (4), the respective *weight sum* in (6) includes the weight of each literal true in I. This primitive will be needed a lot in the sequel to deal with weight rules.

Example 3.3
The third but last rule of Example 2.2 is satisfied in an interpretation $I_1 = \{\mathsf{tea}, \mathsf{biscuit}\}$, but not in $I_2 = \{\mathsf{coffee}, \mathsf{cake}, \mathsf{cognac}\}$.

An interpretation I is a (classical) *model* of a logic program P if and only if $I \models P$. However, stable models are not arbitrary models of logic programs. As discussed in the beginning of this section, they involve a reduction of logic programs which is based on a pre-interpretation of negative literals.

Definition 3.4
For an SMODELS program P and an interpretation $I \subseteq \text{Hb}(P)$ of P, the reduct P^I contains

1. a basic rule $h \leftarrow a_1, \ldots, a_n$ $\iff$ there is a basic rule (1) in P such that $I \models \{\sim b_1, \ldots, \sim b_m\}$ **or** there is a choice rule (3) in P such that $h \in \{h_1, \ldots, h_l\}$, $I \models h$, and $I \models \{\sim b_1, \ldots, \sim b_m\}$;
2. a constraint rule $h \leftarrow c' \ \{a_1, \ldots, a_n\}$ $\iff$ there is a constraint rule (2) in P and $c' = \max(0, c - |\{\sim b_i \mid I \models \sim b_i\}|)$;
3. a weight rule $h \leftarrow w' \leqslant \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}\}$ $\iff$ there is a weight rule (4) in P and $w' = \max(0, w - \text{WS}_I(\sim b_1 = w_{b_1}, \ldots, \sim b_m = w_{b_m}))$; and
4. no compute statements.

Note that in addition to evaluating negative literals in the bodies of rules, the head atoms $h \in \{h_1, \ldots, h_l\}$ of choice rules (3) are subject to a special treatment: an essential prerequisite for including $h \leftarrow a_1, \ldots, a_n$ in the reduct P^M is that $M \models h$, i.e., $h \in M$. This is the way in which the choice regarding h takes place. Moreover, it is clear by Definition 3.4 that the reduct P^M is free of default negation and it contains only basic rules, constraint rules, and weight rules, but no compute statements. Thus we call an SMODELS program P *positive* if each rule $r \in P$ is of the forms (1), (2) and (4) restricted to the case $m = 0$. The least model semantics can be generalized for positive programs by distinguishing their minimal models.

Definition 3.5
A model $M \models P$ of a (positive) SMODELS program P is *minimal* if and only if there is no $M' \models P$ such that $M' \subset M$.

Positive programs share many important properties of positive *normal programs* and the straightforward semantics based on minimal models and the least model is easily generalized for positive programs.

Definition 3.6
For a positive SMODELS program P, we define an operator $\mathrm{T}_P : \mathbf{2}^{\mathrm{Hb}(P)} \rightarrow \mathbf{2}^{\mathrm{Hb}(P)}$ as follows. Given any interpretation $I \subseteq \mathrm{Hb}(P)$, the result of applying T_P to I, i.e., $\mathrm{T}_P(I) \subseteq \mathrm{Hb}(P)$, contains an atom $a \in \mathrm{Hb}(P)$ if and only if

1. there is a basic rule $a \leftarrow a_1, \ldots, a_n \in P$ and $I \models \{a_1, \ldots, a_n\}$; or
2. there is a constraint rule $a \leftarrow c\ \{a_1, \ldots, a_n\} \in P$ and $c \leqslant |\{a_i \mid I \models a_i\}|$; or
3. there is a weight rule $a \leftarrow w \leqslant \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}\} \in P$ and

$$w \leqslant \mathrm{WS}_I(a_1 = w_{a_1}, \ldots, a_n = w_{a_n}).$$

Intuitively, the operator T_P gives atoms that are necessarily true by the rules of P if the atoms in I are assumed to be true. It follows that $\mathrm{T}_P(I) \subseteq I$ implies $I \models P$ in general. We are now ready to state a number of properties of positive programs.

Proposition 3.7
Let P be a positive SMODELS program.

1. For any collection C of models of P, the intersection $\bigcap C$ is also a model of P.
2. The program P has a unique minimal model M, i.e., the *least model* $\mathrm{LM}(P)$ of P.
3. The least model $\mathrm{LM}(P) = \bigcap\{I \subseteq \mathrm{Hb}(P) \mid I \models P\}$ and $\mathrm{LM}(P) = \mathrm{lfp}(\mathrm{T}_P)$.

Moreover, positive programs are *monotonic* in the sense that $P_1 \subseteq P_2$ implies $\mathrm{LM}(P_1) \subseteq \mathrm{LM}(P_2)$. Given the least model semantics for positive programs, it becomes straightforward to generalize the stable model semantics (Gelfond and Lifschitz 1988) for programs involving default negation. The key idea is to use the reduction from Definition 3.4, but the effect of compute statements must also be taken into account as they are dropped out by Definition 3.4. To this end, we define $\mathrm{CompS}(P)$ as the union of literals appearing in the compute statements (5) of P.

Definition 3.8
An interpretation $M \subseteq \text{Hb}(P)$ is a stable model of an SMODELS program P if and only if $M = \text{LM}(P^M)$ and $M \models \text{CompS}(P)$.

Definition 3.8 reveals the purpose of compute statements: they are used to select particular models among those satisfying the conventional fixed point condition from Definition 3.1. Given any logic program P, we define the set

$$\text{SM}(P) = \{M \subseteq \text{Hb}(P) \mid M = \text{LM}(P^M) \text{ and } M \models \text{CompS}(P)\}. \tag{7}$$

In analogy to the case of normal logic programs, the number of stable models may vary in general. A positive program P has a unique stable model $\text{LM}(P)$ as $P^M = P$ holds; recall that compute statements are not allowed in positive programs. It is also worth noting that $M = \text{LM}(P^M)$ and $M \models \text{CompS}(P)$ imply $M \models P$, i.e., stable models are also classical models in the sense of Definition 3.2. However, the converse does not hold in general, i.e., $M \models P$ need not imply $M = \text{LM}(P^M)$ although it certainly implies $M \models \text{CompS}(P)$. For example, interpretations $M_1 = \{a\}$ and $M_2 = \{a, b\}$ are models of the program $P = \{a \leftarrow 1\ \{\sim a, \sim b\}.\ \}$, but only M_1 is stable. To verify this, note that $P^{M_1} = \{a \leftarrow 0\ \{\}.\ \}$ and $P^{M_2} = \{a \leftarrow 1\ \{\}.\ \}$.

Example 3.9
Recall the program P from Example 2.2. According to SMODELS there are 33 acceptable orders that are captured by the stable models of P. One of them is $M_7 = \{\text{acceptable}, \text{happy}, \text{lemon}, \text{tea}, \text{biscuit}\}$. The reader is kindly asked to verify $M_7 = \text{LM}(P^{M_7})$ and $M_7 \models \text{CompS}(P)$ using the reduct P^{M_7} listed below.

```
tea. biscuit.
cognac ← coffee.
lemon ← tea.
mess ← milk, lemon.
happy ← 1 {biscuit, cake, cognac}.
bankrupt ← 6 ⩽ {coffee = 1, tea = 1, biscuit = 1, cake = 2, cognac = 4}.
acceptable ← happy.
```

4 Notions of equivalence

We begin this section by reviewing two fundamental notions of equivalence that have been proposed for logic programs, namely *weak* and *strong* equivalence, and point out some of their limitations. This is why we resort to another notion of equivalence in Section 4.1: *visible* equivalence is a variant of weak equivalence which takes the visibility of atoms better into account. Then we are ready to identify the respective verification problem in Section 4.2 and discuss in which way invisible atoms render the verification problem more difficult. This serves as a starting point for characterizing a subclass of programs for which visible equivalence can be verified using a translation-based technique in analogy to Janhunen and Oikarinen (2002).

Lifschitz et al. (2001) address two major notions of equivalence for logic programs. The first one arises naturally from the stable model semantics.

Definition 4.1
Logic programs P and Q are *weakly equivalent*, denoted $P \equiv Q$, if and only if $\mathrm{SM}(P) = \mathrm{SM}(Q)$, i.e., P and Q have the same stable models.

The second notion is definable in terms of the first and the definition is given relative to a class of logic programs which is represented by R below. Of course, a natural choice for us would be the class of SMODELS programs but that is not made explicit in the following definition.

Definition 4.2
Logic programs P and Q are *strongly equivalent*, denoted $P \equiv_s Q$, if and only if $P \cup R \equiv Q \cup R$ for any logic program R.

Here the program R can be understood as an arbitrary context in which the other two programs P and Q being compared could be placed. This is how strongly equivalent logic programs can be used as semantics preserving substitutes of each other. This feature makes $\equiv_s$ a *congruence relation* over the class of logic programs under consideration: if $P \equiv_s Q$ holds, then also $P \cup R \equiv_s Q \cup R$ holds for any R. Moreover, it is easy to see that $P \equiv_s Q$ implies $P \equiv Q$, but not necessarily vice versa: $\equiv_s$ relates far fewer programs than $\equiv$ as demonstrated in Example 4.3. This explains why we call $\equiv$ the *weak equivalence* relation for the class of logic programs introduced in Sections 2 and 3. It is worth pointing out that whereas $\equiv$ is an equivalence relation it does not permit substitutions ($P \equiv Q$ does not imply $P \cup R \equiv Q \cup R$ in general) and hence it does not qualify as a congruence relation.

Example 4.3
Consider $P = \{a \leftarrow {\sim}b.\}$ and $Q = \{a.\}$. It is easy to see that $\mathrm{SM}(P) = \mathrm{SM}(Q) = \{\{a\}\}$ and $P \equiv Q$. However, when joined with $R = \{b.\}$, we note that $\mathrm{SM}(P \cup R) \neq \mathrm{SM}(Q \cup R)$ holds so that $P \not\equiv_s Q$. The programs P and Q are not classically equivalent either as $M \models P$ and $M \not\models Q$ hold for $M = \{b\}$.

Although the relation $\equiv_s$ appears attractive at first glance, a drawback is that it is quite restrictive, allowing only rather straightforward semantics-preserving transformations of (sets of) rules. In fact, Lifschitz et al. (2001) characterize $\equiv_s$ in Heyting's logic *here-and-there* (HT) which is an intermediary logic between intuitionistic and classical propositional logics. This result implies that each program transformation admitted by $\equiv_s$ is based on a classical equivalence of the part being replaced (say P) and its substitute (say Q), i.e., $P \equiv_s Q$ implies that P and Q are classically equivalent. However, the converse is not true in general as there are classically equivalent programs that are not strongly equivalent.

Example 4.4
The propositional sentence $a \leftrightarrow (\neg a \rightarrow a)$ is classically valid – suggesting a program transformation that replaces $P = \{a.\}$ by $Q = \{a \leftarrow {\sim}a.\}$. However, since $\mathrm{SM}(P) = \{\{a\}\}$ and $\mathrm{SM}(Q) = \emptyset$, we have $P \not\equiv Q$ and $P \not\equiv_s Q$ although P and Q are classically equivalent.

Since $\equiv_s$ is a congruence relation, it is better applicable to *subprograms* or *program modules* constituting larger programs rather than complete programs. In contrast to this, weak equivalence is mainly targeted to the comparison of complete programs in terms of their stable models. Due to the nature of ASP, this is often the ultimate question confronted by a programmer when optimizing and debugging programs. For this reason, we concentrate on the problem of verifying weak equivalence in this paper and we leave the modularization aspects of weak and visible equivalence to be addressed elsewhere (Oikarinen and Janhunen 2006).

4.1 Visible equivalence

We do not find the notion of weak equivalence totally satisfactory either. For $P \equiv Q$ to hold, the stable models in $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ have to be identical subsets of $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$, respectively. This makes $\equiv$ less useful if $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$ differ by some (in)visible atoms which are not trivially false in all stable models. As already discussed in Section 2, such atoms are needed when some auxiliary concepts are formalized using rules. The use of such atoms/concepts may lead to more concise encodings of problems as demonstrated by our next example.

Example 4.5
Consider the following programs consisting of basic rules, choice rules, and compute statements. The parameter n below is an odd natural number.

Program P_n: $\{\mathsf{bit}_1, \mathsf{bit}_2, \ldots, \mathsf{bit}_n\}.$
$\mathsf{odd} \leftarrow \mathsf{bit}_1, {\sim}\mathsf{bit}_2, \ldots, {\sim}\mathsf{bit}_n.$
$\mathsf{odd} \leftarrow {\sim}\mathsf{bit}_1, \mathsf{bit}_2, {\sim}\mathsf{bit}_3 \ldots, {\sim}\mathsf{bit}_n.$
$\vdots$
$\mathsf{odd} \leftarrow \mathsf{bit}_1, \mathsf{bit}_2, \mathsf{bit}_3, {\sim}\mathsf{bit}_4 \ldots, {\sim}\mathsf{bit}_n.$
$\mathsf{odd} \leftarrow {\sim}\mathsf{bit}_1, \mathsf{bit}_2, \mathsf{bit}_3, \mathsf{bit}_4, {\sim}\mathsf{bit}_5 \ldots, {\sim}\mathsf{bit}_n.$
$\vdots$
$\mathsf{odd} \leftarrow \mathsf{bit}_1, \ldots, \mathsf{bit}_n.$
$\mathsf{compute}\ \{{\sim}\mathsf{odd}\}.$

Program Q_n: $\{\mathsf{bit}_1, \mathsf{bit}_2, \ldots, \mathsf{bit}_n\}.$
$\mathsf{odd}_1 \leftarrow \mathsf{bit}_1.$
$\mathsf{odd}_2 \leftarrow \mathsf{bit}_2, {\sim}\mathsf{odd}_1.\ \mathsf{odd}_2 \leftarrow {\sim}\mathsf{bit}_2, \mathsf{odd}_1.$
$\vdots$
$\mathsf{odd}_n \leftarrow \mathsf{bit}_n, {\sim}\mathsf{odd}_{n-1}.\ \mathsf{odd}_n \leftarrow {\sim}\mathsf{bit}_n, \mathsf{odd}_{n-1}.$
$\mathsf{odd} \leftarrow \mathsf{odd}_n.$
$\mathsf{compute}\ \{{\sim}\mathsf{odd}\}.$

The first program generates all subsets B of $BIT_n = \{\mathsf{bit}_1, \mathsf{bit}_2, \ldots, \mathsf{bit}_n\}$, analyzes when $|B|$ is odd, and accepts only subsets with non-odd (even) cardinality. Thus P_n has 2^{n-1} stable models $M \subseteq BIT_n$ with $|M|$ even but also 2^{n-1} basic rules capturing subsets with odd cardinality. In contrast, huge savings can be achieved by introducing new atoms $\mathsf{odd}_1, \ldots, \mathsf{odd}_n$ so that each odd_i is supposed to be true if and only if $|B \cap \{\mathsf{bit}_1, \ldots, \mathsf{bit}_i\}|$ is odd. Using these, the oddness of $|B|$ can be formalized in terms of $2n$ basic rules. The resulting program Q_n has 2^{n-1} stable models, but they are not identical with the stable models of Q_n due to new atoms involved. Thus we have $\mathrm{SM}(Q_n) \neq \mathrm{SM}(P_n)$ and $Q_n \not\equiv P_n$ for every odd natural number n.

From the programmer's point of view, the programs P_n and Q_n solve the same problem and should be considered equivalent if one neglects the interpretations of $\mathsf{odd}_1, \ldots, \mathsf{odd}_n$ in the stable models of Q_n. To this end, we adopt a slightly more general notion of equivalence (Janhunen 2003; Janhunen 2006) which takes the visibility of atoms properly into account. The key idea is that when two programs P and Q are compared, the hidden atoms in $\mathrm{Hb_h}(P)$ and $\mathrm{Hb_h}(Q)$ are considered to be local to P and Q and thus negligible as far as the equivalence of the programs is concerned. In addition to this feature, a very strict (bijective) correspondence of stable models is necessitated by the notion of *visible equivalence*.

Definition 4.6
Logic programs P and Q are *visibly equivalent*, denoted $P \equiv_{\mathrm{v}} Q$, if and only if $\mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q)$ and there is a bijection $f : \mathrm{SM}(P) \to \mathrm{SM}(Q)$ such that for every $M \in \mathrm{SM}(P)$, $M \cap \mathrm{Hb_v}(P) = f(M) \cap \mathrm{Hb_v}(Q)$.

Proposition 4.7
The relation $\equiv_{\mathrm{v}}$ is an equivalence relation.

By defining $\mathrm{Hb_v}(P_n) = \mathrm{Hb_v}(Q_n) BIT_n$ for the programs P_n and Q_n defined in Example 4.5 we obtain an intuitive relationship $Q_n \equiv_{\mathrm{v}} P_n$. The bijection f involved in this relationship maps a stable model $M \in \mathrm{SM}(Q_n)$ to another $f(M) = M \cap BIT_n \in \mathrm{SM}(P_n)$. Our following example demonstrates the case in which both $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ have stable models that cannot be distinguished if projected to $\mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q)$, i.e., there are stable models $M, N \in \mathrm{SM}(P)$ such that $M \cap \mathrm{Hb_v}(P) = N \cap \mathrm{Hb_v}(P)$ and analogously for Q. However, this does not necessarily exclude the possibility for a bijection in the sense of Definition 4.6.

Example 4.8
Consider logic programs $P = \{a \leftarrow b.\ a \leftarrow c.\ b \leftarrow {\sim}c.\ c \leftarrow {\sim}b.\ \}$ and $Q = \{\{b, c\}.\ a \leftarrow b, c.\ a \leftarrow {\sim}b, {\sim}c.\ b \leftarrow c, {\sim}b.\ c \leftarrow b, {\sim}c.\ \}$ with $\mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q) = \{a\}$ and $\mathrm{Hb_h}(P) = \mathrm{Hb_h}(Q) = \{b, c\}$. The stable models of P are $M_1 = \{a, b\}$ and $M_2 = \{a, c\}$ whereas for Q they are $N_1 = \{a\}$ and $N_2 = \{a, b, c\}$. Thus $P \not\equiv Q$ is clearly the case, but we have a bijection $f : \mathrm{SM}(P) \to \mathrm{SM}(Q)$, which maps M_i to N_i for $i \in \{1, 2\}$, such that $M \cap \mathrm{Hb_v}(P) = f(M) \cap \mathrm{Hb_v}(Q)$. Thus $P \equiv_{\mathrm{v}} Q$ holds.

A brief comparison of $\equiv_{\mathrm{v}}$ and $\equiv$ follows.

Proposition 4.9
If $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ and $\mathrm{Hb_h}(P) = \mathrm{Hb_h}(Q) = \emptyset$, then $P \equiv Q \iff P \equiv_{\mathrm{v}} Q$.

In words, the two relations coincide when all atoms are visible. There is only a slight difference: $\equiv_{\mathrm{v}}$ insists on $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ whereas $\equiv$ does not. Nevertheless, it follows by Definition 2.3 that such a difference is of little account: Herbrand bases are always extendible to meet $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. The value of these observations is that by implementing $\equiv_{\mathrm{v}}$ we obtain an implementation for $\equiv$ as well. We will follow this strategy in Section 5. Moreover, it is also clear by Proposition 4.9 that $\equiv_{\mathrm{v}}$ is not a congruence for $\cup$ and thus it does not support program substitutions like $\equiv_{\mathrm{s}}$.

Visible equivalence has its roots in the study of translation functions (Janhunen 2003, 2006) and it was proposed as a *faithfulness* criterion for a translation function Tr between classes of programs, i.e., $P \equiv_{\mathrm{v}} \mathrm{Tr}(P)$ should hold for all programs P. The bijective relationship of stable models ensures that a faithful translation (see Theorem 6.10 for an instance) preserves the number of stable models. This is highly desirable in ASP where stable models correspond to solutions of problems and the ability to count solutions correctly after potential program transformations is of interest. However, this is not guaranteed, if we consider weaker alternatives of $\equiv_{\mathrm{v}}$ obtained in a general framework based on *equivalence frames* (Eiter et al. 2005). Visible equivalence does not really fit into equivalence frames based on *projected answer sets*. A projective variant of Definition 4.6 would simply impose $\{M \cap \mathrm{Hb}_{\mathrm{v}}(P) \mid M \in \mathrm{SM}(P)\} = \{N \cap \mathrm{Hb}_{\mathrm{v}}(Q) \mid N \in \mathrm{SM}(Q)\}$ on P and Q for $P \equiv_{\mathrm{vp}} Q$ to hold, which is clearly implied by $P \equiv_{\mathrm{v}} Q$ but not vice versa. The key observation is that a weakly faithful translation function Tr, i.e., Tr satisfies $P \equiv_{\mathrm{vp}} \mathrm{Tr}(P)$ for all P, does not necessarily preserve the number of stable models – contradicting the general nature of ASP. As an illustration of these ideas, let us consider $P = \{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\}$ and $Q_n = \mathrm{Tr}_{\mathrm{exp}}(P) = P \cup \{c_i \leftarrow {\sim}d_i.\ d_i \leftarrow {\sim}c_i. \mid 0 < i \leqslant n\}$ where $n > 0$ is a parameter of $\mathrm{Tr}_{\mathrm{exp}}$ and $\mathrm{Hb}_{\mathrm{v}}(Q_n) = \mathrm{Hb}_{\mathrm{v}}(P) = \{a, b\}$ by definition. It follows that $\mathrm{SM}(P) = \{\{a\}, \{b\}\}$ and Q_n has 2^{n+1} stable models so that $M \cap \{a, b\} \in \mathrm{SM}(P)$ holds for each $M \in \mathrm{SM}(Q_n)$. Therefore $P \equiv_{\mathrm{vp}} Q_n$ but $P \not\equiv_{\mathrm{v}} Q_n$ hold for every $n > 0$, i.e., $\mathrm{Tr}_{\mathrm{exp}}$ would be faithful only in the weaker sense. A drawback of translation functions like $\mathrm{Tr}_{\mathrm{exp}}$ is that for sufficiently large values of n, it is no longer feasible to count the number of stable models of P using its translation Q_n which is only polynomially longer than P.

Equivalence relations play also a role in *forgetting*. Given a logic program P and a set of atoms $F \subseteq \mathrm{Hb}(P)$, the goal is to remove all instances of atoms of F from P but preserve the semantics of P as far as possible. Eiter and Wang (2006) provide an account of forgetting in the case of *disjunctive* logic programs. The result of forgetting $\mathrm{fg}(P, F)$ is not syntactically unique but its stable models are defined as the $\subseteq$-minimal elements of $\mathrm{SM}(P) \setminus F = \{M \setminus F \mid M \in \mathrm{SM}(P)\}$. For instance, the program P_n in Example 4.5 is a valid result of forgetting if we remove $F = \{\mathsf{odd}_1, \ldots, \mathsf{odd}_n\}$ from Q_n. We note that forgetting a set of atoms F is somewhat analogous to hiding F in P, i.e., setting $\mathrm{Hb}_{\mathrm{h}}(P) = F$, but obvious differences are that $\mathrm{Hb}(\mathrm{fg}(P, F)) \cap F = \emptyset$ by definition and forgetting can affect the number of stable models in contrast to hiding. Nevertheless, Eiter and Wang (2006) show that forgetting preserves weak equivalence in the sense that $P \equiv Q$ implies $\mathrm{fg}(P, F) \equiv \mathrm{fg}(Q, F)$. This property is shared by $\equiv_{\mathrm{v}}$ in the fully visible case as addressed in Proposition 4.9. In general, we can establish the following.

Proposition 4.10
If $P \equiv_{\mathrm{v}} Q$, then $\mathrm{fg}(P, \mathrm{Hb}_{\mathrm{h}}(P)) \equiv \mathrm{fg}(Q, \mathrm{Hb}_{\mathrm{h}}(Q))$.

Proof
Let us assume $P \equiv_{\mathrm{v}} Q$ which implies both $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and the existence of a bijection f in the sense of Definition 4.6. Assuming $\mathrm{fg}(P, \mathrm{Hb}_{\mathrm{h}}(P)) \not\equiv$

```
1: function EqNaive($P, Q$): Boolean;
2: begin
3:   for $M \in \mathrm{SM}(P)$ do
4:     if $M \neq \mathrm{LM}(Q^M)$
5:       then return false;
6:   for $N \in \mathrm{SM}(Q)$ do
7:     if $N \neq \mathrm{LM}(P^N)$
8:       then return false;
9:   return true
10: end
```

(a)

```
1: algorithm NotEq($P, Q$);
2: begin
3:   choose $M \subseteq \mathrm{Hb}(P)$ and $N \subseteq \mathrm{Hb}(Q)$;
4:   if $M = \mathrm{LM}(P^M)$ and $M \neq \mathrm{LM}(Q^M)$
5:     then accept;
6:   if $N = \mathrm{LM}(Q^N)$ and $N \neq \mathrm{LM}(P^N)$
7:     then accept;
8:   reject
9: end
```

(b)

Fig. 1. A naive deterministic and a nondeterministic algorithm for verifying $P \equiv Q$ and $P \not\equiv Q$, respectively, when $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ and all atoms of P and Q are visible.

$\mathrm{fg}(Q, \mathrm{Hb_h}(Q))$, we derive without loss of generality the existence of a stable model $M \in \mathrm{SM}(\mathrm{fg}(P, \mathrm{Hb_h}(P)))$ such that $M \notin \mathrm{SM}(\mathrm{fg}(Q, \mathrm{Hb_h}(Q)))$. Note that M is a subset of $\mathrm{Hb}(\mathrm{fg}(P, \mathrm{Hb_h}(P))) = \mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q) = \mathrm{Hb}(\mathrm{fg}(Q, \mathrm{Hb_h}(Q)))$ and a $\subseteq$-minimal element in $\mathrm{SM}(P) \setminus \mathrm{Hb_h}(P)$ defined in the preceding discussion.

Then consider any $M' \in \mathrm{SM}(P)$ such that $M = M' \setminus \mathrm{Hb_h}(P)$. It follows by the properties of f that $N' = f(M') \in \mathrm{SM}(Q)$ and $N' \cap \mathrm{Hb_v}(Q) = M' \cap \mathrm{Hb_v}(P) = M$. Thus $M = N' \setminus \mathrm{Hb_h}(Q)$ belongs to $\mathrm{SM}(Q) \setminus \mathrm{Hb_h}(Q)$. Let us then assume that M is not $\subseteq$-minimal in this set, i.e., there is $N \in \mathrm{SM}(Q) \setminus \mathrm{Hb_h}(Q)$ such that $N \subset M$. Using the properties of f and the same line of reasoning as above for M' and N' but in the other direction, we learn that $N \in \mathrm{SM}(P) \setminus \mathrm{Hb_h}(P)$ holds for $N \subset M$. A contradiction, since M is $\subseteq$-minimal in this set.

It follows that M is also a $\subseteq$-minimal element in $\mathrm{SM}(Q) \setminus \mathrm{Hb_h}(Q)$ so that $M \in \mathrm{SM}(\mathrm{fg}(Q, \mathrm{Hb_h}(Q)))$, a contradiction. □

4.2 Preliminary Analysis of the equivalence verification problem

The definition of stable models is based on the whole Herbrand base $\mathrm{Hb}(P)$ and hence it neglects which atoms are visible and which not. The weak equivalence relation $\equiv$ is based on the same line of thinking and in (Janhunen and Oikarinen 2002), we presented methods for verifying the weak equivalence of two programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. A relatively naive approach is to cross-check the stable models in $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ in order to establish $\mathrm{SM}(P) = \mathrm{SM}(Q)$ and thus $P \equiv Q$. The respective deterministic algorithm EqNaive is described in Figure 1 (a).[6] The algorithm may use any algorithm such as the one given by Simons et al. (2002) for enumerating the stable models of P and Q one at a time. Due to **FNP**-completeness of the respective function problem (Simons et al. 2002), the computation of each model may require time exponential in

[6] For the sake of brevity, compute statements (5) are not covered by EqNaive.

the length of input, i.e., $||P||$ or $||Q||$. The number of stable models to be cross-checked by a function call $\mathsf{EqNaive}(P, Q)$ can also be exponential. However, the tests for instability on lines 4 and 7 can be clearly accomplished in polynomial time. This is because the least model $\mathrm{LM}(R)$ of any positive set of rules R can be computed in time linear in $||R||$ using a generalization of the procedure developed by Dowling and Gallier (1984).

On the other hand, we get an upper limit for the computational time complexity of the equivalence verification problem by inspecting the nondeterministic algorithm presented in Figure 1 (b). The idea is to select an interpretation M for P (line 3) and to verify that M is a counter-example to $P \equiv Q$ (lines 4–7). Both tasks can be completed in time linear in $||P|| + ||Q||$. Since $\mathsf{NotEq}(P, Q)$ accepts its input in the nondeterministic sense if and only if $P \not\equiv Q$, we see that the equivalence verifying problem is a problem in **coNP**. On the other hand, checking the existence of a stable model for a given logic program forms an **NP**-complete decision problem (Simons et al. 2002). Thus one can establish the **coNP**-completeness of the verification problem by reducing the complement of the latter problem, i.e., checking that a logic program does *not* have stable models, to the problem of verifying that P is equivalent with $\{a \leftarrow {\sim}a\}$ – a program having no stable models. These observations on computational complexity suggest an alternative computational strategy for solving the equivalence verification problem (Janhunen and Oikarinen 2002; Oikarinen and Janhunen 2004). The idea is that counter-examples for $P \equiv Q$ are explicitly specified in terms of rules and then proved non-existent using the same search algorithm as what is used for the computation of stable models.

Unfortunately, further sources of complexity arise if we allow the use of hidden atoms in SMODELS programs and consider $\equiv_{\mathrm{v}}$ rather than $\equiv$. To see this, let us analyze how the operation of $\mathsf{EqNaive}$ should be modified in order to deal with invisible atoms. In fact, each cross-checking step has to be refined. It is no longer enough to compute a stable model M for P. In addition to this, we have to *count* how many stable models of P coincide with M up to $\mathrm{Hb}_{\mathrm{v}}(P)$, i.e., determine the number $n = |\{N \in \mathrm{SM}(P) \mid N \cap \mathrm{Hb}_{\mathrm{v}}(P) = M \cap \mathrm{Hb}_{\mathrm{v}}(P)\}|$. Then it is sufficient to check that Q has equally many stable models that coincide with M up to $\mathrm{Hb}_{\mathrm{v}}(P)$. This line of thinking applies directly to the pair of programs given in Example 4.8.

By numbering stable models in the order they are encountered, we obtain the basis for a bijective relationship as insisted by Definition 4.6. The bad news is that the computational complexity of counting models appears to be much higher than finding a model; see Roth (1996) for the case of propositional satisfiability. Since classical models are easily captured with stable models (Niemelä 1999), counting stable models of a logic program cannot be easier than counting satisfying assignments for a set of propositional clauses. Thus the complexity of verifying $\equiv_{\mathrm{v}}$ appears to be very high in general and restrictions on visible atoms do not seem to provide us a way to circumvent the counting problem: If $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q) = \emptyset$ is assumed, then $P \equiv_{\mathrm{v}} Q$ if and only if P and Q have the same number of stable models.

To avoid model counting as discussed above, we should restrict ourselves to logic programs P, for which the set $\{N \in \mathrm{SM}(P) \mid N \cap \mathrm{Hb}_{\mathrm{v}}(P) = M \cap \mathrm{Hb}_{\mathrm{v}}(P)\}$ contains exactly one element for each $M \in \mathrm{SM}(P)$. Then stable models $M, N \in \mathrm{SM}(P)$ can be distinguished in terms of visible atoms:

$$M \neq N \text{ implies } M \cap \mathrm{Hb_v}(P) \neq N \cap \mathrm{Hb_v}(P). \tag{8}$$

Definition 4.11
Given an SMODELS program P, a set of interpretations $C \subseteq \mathbf{2}^{\mathrm{Hb}(P)}$ is *separable* with $\mathrm{Hb_v}(P)$ if (8) holds for all $M, N \in C$, and we say that P has separable stable models if $\mathrm{SM}(P)$ is separable with $\mathrm{Hb_v}(P)$.

Unfortunately, the separability of P and Q does not imply that $\mathsf{EqNaive}(P, Q)$ and $\mathsf{NotEq}(P, Q)$ work correctly as $\mathrm{Hb_h}(P)$ and $\mathrm{Hb_h}(Q)$ differ and may lead to unnecessary disqualification of models by the polynomial time tests $M \neq \mathrm{LM}(Q^M)$ and $N \neq \mathrm{LM}(P^N)$. These tests capture correctly conditions $M \notin \mathrm{SM}(Q)$ and $M \notin \mathrm{SM}(P)$, respectively, but when all atoms are visible. However, a higher computational complexity is involved in the presence of invisible atoms. For example, the former test would have to be replaced by a computation verifying that there is no $N \in \mathrm{SM}(Q)$ such that $M \cap \mathrm{Hb_v}(P) = N \cap \mathrm{Hb_v}(Q)$ holds. This tends to push the worst case time complexity of the equivalence verification problem to the second level of polynomial time hierarchy Stockmeyer (1976). Thus it seems that we need a stronger restriction than separability in order to keep the problem of verifying $P \equiv_{\mathrm{v}} Q$ as a decision problem in **coNP** – an obvious prerequisite for the translation-based verification technique in Janhunen and Oikarinen (2002).

4.3 Programs having enough visible atoms

In the fully visible case, the complexity of the verification problem is alleviated by the computation of least models in algorithm $\mathsf{NotEq}(P, Q)$. Those models are *unique* models associated with the respective Gelfond-Lifschitz reductions of programs and they provide the basis for detecting the (in)stability of model candidates. Having such a unique model for each reduct is the key property that we would like to carry over to the case of programs involving invisible atoms. To achieve this, we propose a semantical restriction for the class of logic programs as follows. Given a logic program P and a set of atoms $A \subseteq \mathrm{Hb}(P)$, we write A_{v} and A_{h} for $A \cap \mathrm{Hb_v}(P)$ and $A \cap \mathrm{Hb_h}(P)$, respectively. Moreover, we are going to use shorthands A, B, and H for the respective sets of atoms $\{a_1, \ldots, a_n\}$, $\{b_1, \ldots, b_m\}$, and $\{h_1, \ldots, h_l\}$ appearing in rules (1) – (5). Analogously, the notations $A = W_A$ and $\sim B = W_B$ capture the sets of weights associated with A and B in the body of (4). The goal of Definition 4.12 is to extract the hidden part $P_{\mathrm{h}}/I_{\mathrm{v}}$ of an SMODELS program P by partially evaluating it with respect to an interpretation $I_{\mathrm{v}} \subseteq \mathrm{Hb_v}(P)$ for its visible part.

Definition 4.12
For an SMODELS program P and an interpretation $I_{\mathrm{v}} \subseteq \mathrm{Hb_v}(P)$ for the visible part of P, the hidden part of P relative I_{v}, denoted $P_{\mathrm{h}}/I_{\mathrm{v}}$, contains

1. a basic rule $h \leftarrow A_{\mathrm{h}}, \sim B_{\mathrm{h}} \iff$ there is a basic rule $h \leftarrow A, \sim B$ in P such that $h \in \mathrm{Hb_h}(P)$ and $I_{\mathrm{v}} \models A_{\mathrm{v}} \cup \sim B_{\mathrm{v}}$;

2. a choice rule $\{H_{\mathrm{h}}\} \leftarrow A_{\mathrm{h}}, \sim B_{\mathrm{h}}$ $\iff$ there is a choice rule $\{H\} \leftarrow A, \sim B$ in P such that $H_h \neq \emptyset$ and $I_{\mathrm{v}} \models A_{\mathrm{v}} \cup \sim B_{\mathrm{v}}$;
3. a constraint rule $h \leftarrow c'\ \{A_{\mathrm{h}}, \sim B_{\mathrm{h}}\}$ $\iff$ there is a constraint rule $h \leftarrow c\ \{A, \sim B\}$ in P such that $h \in \mathrm{Hb}_{\mathrm{h}}(P)$ and $c' = \max(0, c - |\{l \in A_{\mathrm{v}} \cup \sim B_{\mathrm{v}} \mid I_{\mathrm{v}} \models l\}|)$;
4. a weight rule $h \leftarrow w' \leqslant \{A_{\mathrm{h}} = W_{A_{\mathrm{h}}}, \sim B_{\mathrm{h}} = W_{B_{\mathrm{h}}}\}$ $\iff$ there is a weight rule $h \leftarrow w \leqslant \{A = W_A, \sim B = W_B\}$ in P such that $h \in \mathrm{Hb}_{\mathrm{h}}(P)$ and
$$w' = \max(0, w - \mathrm{WS}_{I_{\mathrm{v}}}(A_{\mathrm{v}} = W_{A_{\mathrm{v}}}, \sim B_{\mathrm{v}} = W_{B_{\mathrm{v}}})); \tag{9}$$
5. and no compute statements.

This construction can be viewed as a generalization of the simplification operation $\mathrm{simp}(P, T, F)$ proposed by Cholewinski and Truszczyński (1999) to the case of SMODELS programs, but restricted in the sense that T and F are subsets of $\mathrm{Hb}_{\mathrm{v}}(P)$ rather than $\mathrm{Hb}(P)$. More precisely put, we have $P_{\mathrm{h}}/I_{\mathrm{v}} = \mathrm{simp}(P, I_{\mathrm{v}}, \mathrm{Hb}_{\mathrm{v}}(P) - I_{\mathrm{v}})$ for a normal program, i.e., a set of basic rules P.

Roughly speaking, our idea is to allow the use of invisible atoms as long as they do not interfere with the number of stable models obtained for the visible part. We consider the invisible part of a program "well-behaving" in this sense if and only if $M = \mathrm{LM}((P_{\mathrm{h}}/I_{\mathrm{v}})^M)$ has a unique fixpoint M for every $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$. In particular, it should be pointed out that Definition 4.12 excludes compute statements which are not supposed to affect this property (in perfect analogy to Definition 3.8).

Definition 4.13
An SMODELS program P has enough visible atoms if and only if $P_{\mathrm{h}}/I_{\mathrm{v}}$ has a unique stable model for every $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$.

This property can be achieved for any program by making sufficiently many atoms visible. To see this, consider Definition 4.12 when $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P)$ and $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$: it follows that $P_{\mathrm{h}}/I_{\mathrm{v}} = \emptyset$ for which the existence of a unique stable model is immediate. Generally speaking, verifying the property of having enough visible atoms can be computationally quite hard in the worst case, but this property favorably trades off the complexity of verifying $\equiv_{\mathrm{v}}$ as we shall see in Section 5.

Proposition 4.14
Checking whether an SMODELS program P has enough visible atoms forms a **coNP**-hard decision problem **EVA** that belongs to $\mathbf{\Pi}_2^{\mathrm{p}}$.

Here the language **EVA** consists of SMODELS programs that have enough visible atoms. For the proof, we introduce two further languages $\mathbf{EVA}^{\leqslant 1}$ and $\mathbf{EVA}^{\geqslant 1}$. The former includes those SMODELS programs P for which $P_{\mathrm{h}}/I_{\mathrm{v}}$ has *at most* one stable model for every $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$. The latter is defined analogously, but *at least* one stable model for each $P_{\mathrm{h}}/I_{\mathrm{v}}$ is demanded. It should be now clear that $\mathbf{EVA} = \mathbf{EVA}^{\leqslant 1} \cap \mathbf{EVA}^{\geqslant 1}$ which provides us a basis for complexity analysis.

Proof of Proposition 4.14
To show that $\mathbf{EVA}_{\leqslant 1} \in \mathbf{coNP}$, we describe a nondeterministic Turing machine $M_{>1}$ that accepts the complement of $\mathbf{EVA}_{\leqslant 1}$. Given a finite SMODELS program P as input, the machine $M_{>1}$ chooses nondeterministically two interpretations $I, J \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$

and computes $Q = P_{\mathrm{h}}/I_{\mathrm{v}}$. Then $M_{>1}$ checks in polynomial time that $I_{\mathrm{v}} = J_{\mathrm{v}}$ and $I_{\mathrm{h}} \neq J_{\mathrm{h}}$ as well as that both I_{h} and J_{h} are stable models of Q. If not, it rejects the input and accepts it otherwise. It follows that $M_{>1}$ accepts P in the nondeterministic sense if and only if $P \notin \mathbf{EVA}_{\leqslant 1}$. Hence $\mathbf{EVA}_{\leqslant 1} \in \mathbf{coNP}$.

The case of $\mathbf{EVA}_{\geqslant 1}$ is handled by presenting a nondeterministic machine M_0 which uses an **NP** oracle and which accepts the complement of $\mathbf{EVA}_{\geqslant 1}$. The machine M_0 chooses nondeterministically an interpretation $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$ for the input P. Then it computes $P_{\mathrm{h}}/I_{\mathrm{v}}$ and consults an **NP**-oracle Simons et al. (2002) to check whether $P_{\mathrm{h}}/I_{\mathrm{v}}$ has a stable model. If not, it accepts the input and rejects it otherwise. Given the oracle, these computations can be accomplished in polynomial time. Now M_0 accepts P in the nondeterministic sense if and only if $P \notin \mathbf{EVA}_{\geqslant 1}$. Thus we have established that $\mathbf{EVA}_{\geqslant 1} \in \mathbf{\Pi}^{\mathrm{p}}_{\mathbf{2}}$.

We may now combine $M_{>1}$ and M_0 into one oracle machine M that accepts an SMODELS program $P \iff M_{>1}$ accepts P or M_0 accepts P. Equivalently, we have $P \notin \mathbf{EVA}_{\leqslant 1}$ or $P \notin \mathbf{EVA}_{\geqslant 1}$, i.e., $P \notin (\mathbf{EVA}_{\leqslant 1} \cap \mathbf{EVA}_{\geqslant 1}) = \mathbf{EVA}$. Since M is an oracle machine with an **NP** oracle, we have actually shown that $\mathbf{EVA} \in \mathbf{\Pi}^{\mathrm{p}}_{\mathbf{2}}$.

To establish **coNP**-hardness, we present a reduction from **3SAT** to **EVA**. So let us consider an instance of **3SAT**, i.e., a finite set $S = \{C_1, \ldots, C_n\}$ of three-literal clauses C_i of the form $l_1 \vee l_2 \vee l_3$ where each l_i is either an atom a or its classical negation $\neg a$. Each clause C_i is translated into a rule $u \leftarrow f_1, f_2, f_3$ where $f_i = a$ for $l_i = \neg a$ and $f_i = {\sim}a$ for $l_i = a$. The outcome is an SMODELS program P_S which consists of clauses of S translated in this way plus two additional rules $s \leftarrow {\sim}u$ and $x \leftarrow s, {\sim}x$. Moreover, we define $\mathrm{Hb}_{\mathrm{v}}(P_S) = \mathrm{Hb}(S)$ and $\mathrm{Hb}_{\mathrm{h}}(P_S) = \{u, s, x\}$ so that either $(P_S)_{\mathrm{h}}/I_{\mathrm{v}} = \{u.\ s \leftarrow {\sim}u.\ x \leftarrow s, {\sim}x\}$ or $(P_S)_{\mathrm{h}}/I_{\mathrm{v}} = \{s \leftarrow {\sim}u.\ x \leftarrow s, {\sim}x\}$ depending on $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P_S)$. It follows that $S \in \mathbf{3SAT} \iff$ there is an interpretation $J \subseteq \mathrm{Hb}(S)$ such that $J \models S \iff$ there is an interpretation $I_{\mathrm{v}} = J \subseteq \mathrm{Hb}_{\mathrm{v}}(P_S)$ such that u does not appear as a fact in $(P_S)_{\mathrm{h}}/I_{\mathrm{v}} \iff$ there is an interpretation $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P_S)$ such that $(P_S)_{\mathrm{h}}/I_{\mathrm{v}}$ has no stable models $\iff P_S$ has not enough visible atoms, since $(P_S)_{\mathrm{h}}/I_{\mathrm{v}}$ cannot have several stable models. Thus we may conclude **EVA** to be **coNP**-hard. □

Although the classification of **EVA** given in Proposition 4.14 is not exact, exponential worst case time complexity should be clear. However, there are certain syntactic classes of logic programs which are guaranteed to have enough visible atoms and no computational efforts are needed to verify this. For instance, programs P for which $P_{\mathrm{h}}/I_{\mathrm{v}}$ is always positive or *stratified* (Apt et al. 1988) in some sense. Note that such syntactic restrictions need not be imposed on the visible part of P which may then fully utilize expressiveness of rules.

Example 4.15

Consider logic programs $P = \{a \leftarrow b.\ \}$, $Q = \{a \leftarrow c.\ c \leftarrow b.\ \}$, and $R = \{a \leftarrow {\sim}c.\ c \leftarrow {\sim}d.\ d \leftarrow b.\ \}$ with $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q) = \mathrm{Hb}_{\mathrm{v}}(R) = \{a, b\}$.

Given $I_{\mathrm{v}} = \emptyset$, the hidden parts are $P_{\mathrm{h}}/I_{\mathrm{v}} = \emptyset$, $Q_{\mathrm{h}}/I_{\mathrm{v}} = \emptyset$, and $R_{\mathrm{h}}/I_{\mathrm{v}} = \{c \leftarrow {\sim}d.\ \}$ for which we obtain unique stable models $M_P = M_Q = \emptyset$ and $M_R = \{c\}$. On the other hand, we obtain $P_{\mathrm{h}}/J_{\mathrm{v}} = \emptyset$, $Q_{\mathrm{h}}/J_{\mathrm{v}} = \{c.\ \}$, and $R_{\mathrm{h}}/J_{\mathrm{v}} = \{c \leftarrow {\sim}d.\ d.\ \}$ for

$J_\mathrm{v} = \{a, b\}$. Thus the respective unique stable models of the hidden parts are $N_P = \emptyset$ and $N_Q = \{c\}$, and $N_R = \{d\}$.

Next we relate the property of having enough visible atoms with the model separation property. The proof of Lemma 4.16 takes place in Appendix A.

Lemma 4.16
Let P be an SMODELS program. If $M \subseteq \mathrm{Hb}(P)$ is a stable model of P, then M_h is a stable model of $P_\mathrm{h}/M_\mathrm{v}$ as given in Definition 4.12.

Proposition 4.17
Let P be an SMODELS program. If P has enough visible atoms, then P has separable stable models.

Proof
Suppose that P is an SMODELS program which has enough visible atoms but $\mathrm{SM}(P)$ is not separable with $\mathrm{Hb}_\mathrm{v}(P)$. Then there are two stable models $N, M \in \mathrm{SM}(P)$ such that $M_\mathrm{v} = N_\mathrm{v}$ but $M_\mathrm{h} \neq N_\mathrm{h}$. Thus M_h and N_h are stable models of $P_\mathrm{h}/M_\mathrm{v} = P_\mathrm{h}/N_\mathrm{v}$ by Lemma 4.16. A contradiction as P has enough visible atoms. $\square$

The converse of Proposition 4.17 does not hold in general. Consider, for instance $P = \{a \leftarrow \sim a.\ b \leftarrow a, \sim b.\ \}$ with $\mathrm{Hb}_\mathrm{v}(P) = \{a\}$. Since $\mathrm{SM}(P) = \emptyset$, it is trivially separable with $\mathrm{Hb}_\mathrm{v}(P)$. But for $I_\mathrm{v} = \{a\} \subseteq \mathrm{Hb}_\mathrm{v}(P)$, the hidden part $P_\mathrm{h}/I_\mathrm{v} = \{b \leftarrow \sim b.\ \}$ has no stable models and thus P does not have enough visible atoms.

5 Translation-based verification

In this section, we concentrate on developing a translation-based verification technique for visible equivalence, i.e., the relation $\equiv_\mathrm{v}$ introduced in Section 4. Roughly speaking, our idea is to translate given SMODELS programs P and Q into a single SMODELS program $\mathrm{EQT}(P, Q)$ which has a stable model if and only if P has a stable model M for which Q does *not* have a stable model N such that $M_\mathrm{v} = N_\mathrm{v}$. We aim to use such a translation for finding a *counter-example* for $P \equiv_\mathrm{v} Q$ when $\mathrm{Hb}_\mathrm{v}(P) = \mathrm{Hb}_\mathrm{v}(Q)$ and both P and Q have enough visible atoms. Note that if $\mathrm{Hb}_\mathrm{v}(P) \neq \mathrm{Hb}_\mathrm{v}(Q)$, then $P \not\equiv_\mathrm{v} Q$ follows directly by Definition 4.6. As already discussed in Section 4, we need the property of having enough visible atoms to trade off computational complexity so that a polynomial translation is achievable. The good news is that the programs produced by the front-end LPARSE have this property by default unless too many atoms are explicitly hidden by the programmer. Our strategy for finding a counter-examples M is based on the following four steps.

1. Find a stable model $M \in \mathrm{SM}(P)$ for P.
2. Find the unique stable model N_h for $Q_\mathrm{h}/M_\mathrm{v}$.
3. Form an interpretation $N = M_\mathrm{v} \cup N_\mathrm{h}$.
4. Check that $N \notin \mathrm{SM}(Q)$, i.e., $N \neq \mathrm{LM}(Q^N)$ or $N \not\models \mathrm{CompS}(Q)$.

Here the idea is that the uniqueness of N_h with respect to M_v excludes the possibility that Q could possess another stable model $N' \neq N$ such that $M_\mathrm{v} = N'_\mathrm{v}$.

$$
\begin{array}{rl}
P: & a \leftarrow {\sim}b. \quad b \leftarrow {\sim}a. \\
\mathrm{Hidden}^{\circ}(Q): & b^{\circ} \leftarrow {\sim}a. \\
\mathrm{Least}^{\bullet}(Q): & a^{\bullet} \leftarrow b^{\bullet}, {\sim}a. \quad b^{\bullet} \leftarrow {\sim}a. \\
\mathrm{UnStable}(Q): & d \leftarrow a, {\sim}a^{\bullet}. \quad d \leftarrow a^{\bullet}, {\sim}a. \\
& d \leftarrow b^{\circ}, {\sim}b^{\bullet}. \quad d \leftarrow b^{\bullet}, {\sim}b^{\circ}. \\
& e \leftarrow c. \quad e \leftarrow d. \\
& \mathsf{compute}\ \{e\}.
\end{array}
$$

Fig. 2. The rules of the translation $\mathrm{EQT}(P,Q)$ for $P = \{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\ \}$ and $Q = \{a \leftarrow b, {\sim}a.\ b \leftarrow {\sim}a.\ \}$ where a is visible and b is hidden.

This follows essentially by Lemma 4.16: if $N' \in \mathrm{SM}(Q)$ were the case, then N'_{h} would be unique for $N'_{\mathrm{v}} = M_{\mathrm{v}} = N_{\mathrm{v}}$, i.e., $N'_{\mathrm{h}} = N_{\mathrm{h}}$ and $N = N'$.

In the sequel, we present a translation function EQT that effectively captures the four steps listed above within a single SMODELS program $\mathrm{EQT}(P,Q)$. In order to combine several programs in one, we have to rename atoms and thus introduce *new* atoms outside $\mathrm{Hb}(P) \cup \mathrm{Hb}(Q)$:

- a new atom a° for each atom $a \in \mathrm{Hb}_{\mathrm{h}}(Q)$ and
- a new atom $a^{\bullet}$ for each atom $a \in \mathrm{Hb}(Q)$.

The former atoms will be used in the representation of $Q_{\mathrm{h}}/M_{\mathrm{v}}$ while the latter are to appear in the translation of Q^N. The intuitive readings of a° and $a^{\bullet}$ are that $a \in N_{\mathrm{h}}$ and $a \in \mathrm{LM}(Q^N)$ hold, respectively. For notational convenience, we extend the notations a° and $a^{\bullet}$ for sets of atoms A as well as sets of positive rules R in the obvious way. For instance, A° denotes $\{a^{\circ} \mid a \in A\}$ for any $A \subseteq \mathrm{Hb}_{\mathrm{h}}(Q)$.

Definition 5.1
Let P and Q be SMODELS programs such that $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$. The translation $\mathrm{EQT}(P,Q) = P \cup \mathrm{Hidden}^{\circ}(Q) \cup \mathrm{Least}^{\bullet}(Q) \cup \mathrm{UnStable}(Q)$ extends P with three sets of rules to be made precise by Definitions 5.2–5.4. Atoms c, d, and e introduced in Definition 5.4 are assumed to be new.

As regards our strategy for representing counter-examples, the rules in the translation $\mathrm{EQT}(P,Q)$ play the following roles. The rules of P capture a stable model M for P while the rest of the translation ensures that Q does not have a stable model N such that $M_{\mathrm{v}} = N_{\mathrm{v}}$. To make the forthcoming definitions more accessible for the reader, we use simple normal programs $P = \{a \leftarrow {\sim}b.\ \ b \leftarrow {\sim}a.\ \}$ and $Q = \{a \leftarrow b, {\sim}a.\ \ b \leftarrow {\sim}a.\ \}$ with $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q) = \{a\}$ as our running example. The rules contributed by Definitions 5.1–5.4 are collected together in Fig. 2.

Definition 5.2
The translation $\mathrm{Hidden}^{\circ}(Q)$ of an SMODELS program Q contains

1. a basic rule $h^{\circ} \leftarrow A^{\circ}_{\mathrm{h}}, A_{\mathrm{v}}, {\sim}B^{\circ}_{\mathrm{h}}, {\sim}B_{\mathrm{v}}$ for each basic rule $h \leftarrow A, {\sim}B$ in Q with $h \in \mathrm{Hb}_{\mathrm{h}}(Q)$;
2. a constraint rule $h^{\circ} \leftarrow c\ \{A^{\circ}_{\mathrm{h}}, A_{\mathrm{v}}, {\sim}B^{\circ}_{\mathrm{h}}, {\sim}B_{\mathrm{v}}\}$ for each constraint rule $h \leftarrow c\ \{A, {\sim}B\}$ in Q with $h \in \mathrm{Hb}_{\mathrm{h}}(Q)$;

3. a choice rule $\{H_{\mathrm{h}}^{\circ}\} \leftarrow A_{\mathrm{h}}^{\circ}, A_{\mathrm{v}}, {\sim}B_{\mathrm{h}}^{\circ}, {\sim}B_{\mathrm{v}}$ for each choice rule $\{H\} \leftarrow A, {\sim}B$ in Q with $H_{\mathrm{h}} \neq \emptyset$; and
4. a weight rule $h^{\circ} \leftarrow w \leqslant \{A_{\mathrm{h}}^{\circ} = W_{A_{\mathrm{h}}^{\circ}}, A_{\mathrm{v}} = W_{A_{\mathrm{v}}}, {\sim}B_{\mathrm{h}}^{\circ} = W_{B_{\mathrm{h}}^{\circ}}, {\sim}B_{\mathrm{v}} = W_{B_{\mathrm{v}}}\}$ for each weight rule $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ in Q with $h \in \mathrm{Hb}_{\mathrm{h}}(Q)$.

The translation $\mathrm{Hidden}^{\circ}(Q)$ includes rules which provide a representation for the hidden part $Q_{\mathrm{h}}/M_{\mathrm{v}}$ which depends dynamically on M_{v}. This is achieved by leaving the visible atoms from $\mathrm{Hb}_{\mathrm{v}}(Q) = \mathrm{Hb}_{\mathrm{v}}(P)$ untouched. However, the hidden parts of rules are renamed systematically using atoms from $\mathrm{Hb}_{\mathrm{h}}(Q)^{\circ}$. This is to capture the unique stable model N_{h} of $Q_{\mathrm{h}}/M_{\mathrm{v}}$ but renamed as N_{h}°.[7] In our running example, the program Q has only one rule with a hidden atom b in its head and that rule gets translated into $b^{\circ} \leftarrow {\sim}a$ due to the visibility of a.

Definition 5.3
The translation $\mathrm{Least}^{\bullet}(Q)$ of an SMODELS program Q consists of

1. a rule $h^{\bullet} \leftarrow A^{\bullet}, {\sim}B_{\mathrm{v}}, {\sim}B_{\mathrm{h}}^{\circ}$ for each basic rule $h \leftarrow A, {\sim}B$ in Q;
2. a rule $h^{\bullet} \leftarrow c\ \{A^{\bullet}, {\sim}B_{\mathrm{v}}, {\sim}B_{\mathrm{h}}^{\circ}\}$ for each constraint rule $h \leftarrow c\ \{A, {\sim}B\}$ in Q;
3. a rule $h^{\bullet} \leftarrow A^{\bullet} \cup \{h\}, {\sim}B_{\mathrm{v}}, {\sim}B_{\mathrm{h}}^{\circ}$ (resp. $h^{\bullet} \leftarrow A^{\bullet} \cup \{h^{\circ}\}, {\sim}B_{\mathrm{v}}, {\sim}B_{\mathrm{h}}^{\circ}$) for each choice rule $\{H\} \leftarrow A, {\sim}B$ in Q and head atom $h \in H_{\mathrm{v}}$ (resp. $h \in H_{\mathrm{h}}$); and
4. a rule $h^{\bullet} \leftarrow w \leqslant \{A^{\bullet} = W_{A^{\bullet}}, {\sim}B_{\mathrm{v}} = W_{B_{\mathrm{v}}}, {\sim}B_{\mathrm{h}}^{\circ} = W_{B_{\mathrm{h}}^{\circ}}\}$ for each weight rule $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ in Q.

The rules in $\mathrm{Least}^{\bullet}(Q)$ catch the least model $\mathrm{LM}(Q^N)$ for $N = M_{\mathrm{v}} \cup N_{\mathrm{h}}$ but expressed in $\mathrm{Hb}(Q)^{\bullet}$ rather than $\mathrm{Hb}(Q)$. Note that N is represented as $M_{\mathrm{v}} \cup N_{\mathrm{h}}^{\circ}$ which explains the treatment of negative body literals on the basis of visibility in these rules. Two rules result for our running example. The negative literal ${\sim}a$ appearing in the bodies of both rules is not subject to renaming because a is visible.

Definition 5.4
The translation $\mathrm{UnStable}(Q)$ of an SMODELS program Q includes

1. rules $d \leftarrow a, {\sim}a^{\bullet}$ and $d \leftarrow a^{\bullet}, {\sim}a$ for each $a \in \mathrm{Hb}_{\mathrm{v}}(Q)$;
2. rules $d \leftarrow a^{\circ}, {\sim}a^{\bullet}$ and $d \leftarrow a^{\bullet}, {\sim}a^{\circ}$ for each $a \in \mathrm{Hb}_{\mathrm{h}}(Q)$;
3. a rule $c \leftarrow {\sim}a^{\bullet}, {\sim}d$ for each positive literal $a \in \mathrm{CompS}(Q)$;
4. a rule $c \leftarrow b^{\bullet}, {\sim}d$ for each negative literal ${\sim}b \in \mathrm{CompS}(Q)$;
5. rules $e \leftarrow c$ and $e \leftarrow d$; and
6. a compute statement compute $\{e\}$.

The purpose of $\mathrm{UnStable}(Q)$ is to disqualify N as a stable model of Q. The rules in Items 1 and 2 check if N and $\mathrm{LM}(Q^N)$ differ. If not, then the rules in Items 3 and 4 check if $\mathrm{LM}(Q^N)$ violates some compute statement of Q. The rules in Item 5 summarize the two possible reasons why Q does not have a stable model N such that $M_{\mathrm{v}} = N_{\mathrm{v}}$. This is then insisted by the compute statement in Item 6. In our running example, the program Q is free of compute statements and hence only rules for d and e are included in the translation.

[7] For the sake of simplicity, it is assumed that Q does not involve compute statements referring to invisible atoms in order to achieve the property of having enough visible atoms.

Example 5.5
The translation $\mathrm{EQT}(P, Q)$ given in Fig. 2 has two stable models $\{a, d, e\}$ and $\{b, b^\circ, a^\bullet, b^\bullet, d, e\}$ from which we can read off counter-examples $M_1 = \{a\}$ and $M_2 = \{b\}$ for $P \equiv_{\mathrm{v}} Q$ and the respective disqualified interpretations for Q, i.e., $N_1 = \{a\}$ and $N_2 = \{b\}$. The models $\mathrm{LM}(Q^{N_1}) = \emptyset$ and $\mathrm{LM}(Q^{N_2}) = \{a, b\}$ are also easy to extract by projecting the stable models of $\mathrm{EQT}(P, Q)$ with $\{a^\bullet, b^\bullet\}$.

As regards the translation $\mathrm{EQT}(P, Q)$ as whole, we note that $\mathrm{Hb}(\mathrm{EQT}(P, Q))$ equals to $\mathrm{Hb}(P) \cup \mathrm{Hb}_{\mathrm{h}}(Q)^\circ \cup \mathrm{Hb}(Q)^\bullet \cup \{c, d, e\}$. Moreover, the translation is close to being linear. Item 3 in Definition 5.1 makes an exception in this respect, but linearity can be achieved in practise by introducing a new atom b_r for each choice rule r. Then the rules in the fourth item can be replaced by $h^\bullet \leftarrow h, b_r$ (resp. $h^\bullet \leftarrow h^\circ, b_r$) and $b_r \leftarrow A^\bullet, {\sim}B_{\mathrm{v}}, {\sim}B_{\mathrm{h}}^\circ$. However, we use the current definition to avoid the introduction of further new atoms.

Let us then address the correctness of the translation $\mathrm{EQT}(P, Q)$. We begin by computing the Gelfond-Lifschitz reduct of the translation $\mathrm{EQT}(P, Q)$.

Lemma 5.6
Let P and Q be two SMODELS programs such that $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and $I \subseteq \mathrm{Hb}(P) \cup \mathrm{Hb}_{\mathrm{h}}(Q)^\circ \cup \mathrm{Hb}(Q)^\bullet \cup \{c, d, e\}$ an interpretation of $\mathrm{EQT}(P, Q)$. Moreover, define $M = I \cap \mathrm{Hb}(P)$, $N_{\mathrm{h}} = \{a \in \mathrm{Hb}_{\mathrm{h}}(Q) \mid a^\circ \in I\}$, $N = M_{\mathrm{v}} \cup N_{\mathrm{h}}$, and $L = \{a \in \mathrm{Hb}(Q) \mid a^\bullet \in I\}$ so that $N_{\mathrm{h}}^\circ = I \cap \mathrm{Hb}_{\mathrm{h}}(Q)^\circ$ and $L^\bullet = I \cap \mathrm{Hb}(Q)^\bullet$.

The Gelfond-Lifschitz reduct $\mathrm{EQT}(P, Q)^I$ consists of P^M extended by reducts $\mathrm{Hidden}^\circ(Q)^I$, $\mathrm{Least}^\bullet(Q)^I$, and $\mathrm{UnStable}(Q)^I$ specified as follows.

First, the reduct $\mathrm{Hidden}^\circ(Q)^I$ includes

1. a rule $h^\circ \leftarrow A_{\mathrm{h}}^\circ, A_{\mathrm{v}} \iff$ there is a basic rule $h \leftarrow A, {\sim}B$ in Q such that $h \in \mathrm{Hb}_{\mathrm{h}}(Q)$, and $N \models {\sim}B$;
2. a rule $h^\circ \leftarrow c'\ \{A_{\mathrm{h}}^\circ, A_{\mathrm{v}}\}$ where $c' = \max(0, c - |\{b \in B \mid N \models {\sim}b\}|) \iff$ there is a constraint rule $h \leftarrow c\ \{A, {\sim}B\}$ in Q such that $h \in \mathrm{Hb}_{\mathrm{h}}(Q)$;
3. a rule $h^\circ \leftarrow A_{\mathrm{h}}^\circ, A_{\mathrm{v}} \iff$ there is a choice rule $\{H\} \leftarrow A, {\sim}B$ in Q such that $h \in H_{\mathrm{h}} \neq \emptyset$, $N_{\mathrm{h}} \models h$, and $N \models {\sim}B$; and
4. a rule $h^\circ \leftarrow w' \leqslant \{A_{\mathrm{h}}^\circ = W_{A_{\mathrm{h}}^\circ}, A_{\mathrm{v}} = W_{A_{\mathrm{v}}}\}$ where $w' = \max(0, w - \mathrm{WS}_N({\sim}B = W_B))$ $\iff$ there is a weight rule $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ in Q such that $h \in \mathrm{Hb}_{\mathrm{h}}(Q)$.

Second, the reduct $\mathrm{Least}^\bullet(Q)^I$ consists of

5. a rule $h^\bullet \leftarrow A^\bullet \iff$ there is a basic rule $h \leftarrow A, {\sim}B$ in Q such that $N \models {\sim}B$;
6. a rule $h^\bullet \leftarrow c'\ \{A^\bullet\}$ where $c' = \max(0, c - |\{b \in B \mid N \models {\sim}b\}|) \iff$ there is a constraint rule $h \leftarrow c\ \{A, {\sim}B\}$ in Q;
7. a rule $h^\bullet \leftarrow A^\bullet \cup \{h\}$ (resp. $h^\bullet \leftarrow A^\bullet \cup \{h^\circ\}$) $\iff$ there is a choice rule $\{H\} \leftarrow A, {\sim}B$ in Q with $h \in H_{\mathrm{v}}$ (resp. $h \in H_{\mathrm{h}}$) such that $N \models {\sim}B$; and
8. a rule $h^\bullet \leftarrow w' \leqslant \{A^\bullet = W_{A^\bullet}\}$ where $w' = \max(0, w - \mathrm{WS}_N({\sim}B = W_B)) \iff$ there is a weight rule $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ in Q.

Third, the set $\mathrm{UnStable}(Q)^I$ contains

9. a rule $d \leftarrow a \iff$ there is $a \in \mathrm{Hb}_{\mathrm{v}}(Q)$ such that $L \not\models a$;

10. a rule $d \leftarrow a^\circ \iff$ there is $a \in \mathrm{Hb_h}(Q)$ such that $L \not\models a$;
11. a rule $d \leftarrow a^\bullet \iff$ there is $a \in \mathrm{Hb}(Q)$ such that $N \not\models a$;
12. the fact $c \leftarrow \iff$ there is $a \in \mathrm{CompS}(Q)$ such that $L \not\models a$ and $I \not\models d$;
13. a rule $c \leftarrow b^\bullet \iff$ there is $\sim b \in \mathrm{CompS}(Q)$ and $I \not\models d$; and
14. the rules $e \leftarrow c$ and $e \leftarrow d$.

Lemma 5.6 can be easily verified by inspecting the definition of the translation $\mathrm{EQT}(P,Q)$ (Definitions 5.1–5.4) rule by rule and using the definitions of M, N, and L as well as the generalization of Gelfond-Lifschitz reduct for the various rule types (Definition 3.4). The following proposition summarizes a number properties of $\mathrm{LM}(\mathrm{EQT}(P,Q)^I)$ which are used in the sequel to prove our main theorem.

Proposition 5.7
Let P, Q, I, M, N, and L be defined as in Lemma 5.6. Define conditions (i) $M = \mathrm{LM}(P^M)$, (ii) $N_\mathrm{h} = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$, and (iii) $L = \mathrm{LM}(Q^N)$.

1. $\mathrm{LM}(\mathrm{EQT}(P,Q)^I) \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M)$.
2. If (i), then $\mathrm{LM}(\mathrm{EQT}(P,Q)^I) \cap \mathrm{Hb_h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$.
3. If (i) and (ii), then $\mathrm{LM}(\mathrm{EQT}(P,Q)^I) \cap \mathrm{Hb}(Q)^\bullet = \mathrm{LM}(Q^N)^\bullet$.
4. If (i), (ii), and (iii), then the set of atoms $A = \mathrm{LM}(\mathrm{EQT}(P,Q)^I) \cap \{c,d,e\}$ satisfies
   (a) $d \in A \iff N \neq L$,
   (b) $c \in A \iff d \notin I$ and $L \not\models \mathrm{CompS}(Q)$, and
   (c) $e \in A \iff c \in A$ or $d \in A$.

Theorem 5.8
Let P and Q be two SMODELS programs such that $\mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q)$ and Q has enough visible atoms. Then the translation $\mathrm{EQT}(P,Q)$ has a stable model if and only if there is $M \in \mathrm{SM}(P)$ such that for all $N \in \mathrm{SM}(Q)$, $N_\mathrm{v} \neq M_\mathrm{v}$.

The proofs of Proposition 5.7 and Theorem 5.8 are given in Appendix A. As a corollary of Theorem 5.8, we obtain a new method for verifying the visible equivalence of P and Q. Weak equivalence $\equiv$ is covered by making all atoms of P and Q visible which implies that the programs in question have enough visible atoms.

Corollary 5.9
Let P and Q be two SMODELS programs so that $\mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q)$ and both P and Q have enough visible atoms. Then $P \equiv_\mathrm{v} Q$ if and only if the translations $\mathrm{EQT}(P,Q)$ and $\mathrm{EQT}(Q,P)$ have no stable models.

5.1 Computational Complexity Revisited

In this section, we review the computational complexity of verifying visible equivalence of SMODELS programs using the reduction involved in Theorem 5.8. First, we will introduce languages corresponding to the decision problems of our interest and analyze their worst-case time complexities. The main goal is to establish that the verification of visible equivalence forms a **coNP**-complete decision problem for SMODELS programs that have enough visible atoms.

Definition 5.10
For any SMODELS programs P and Q,

- $P \in \mathbf{SM} \iff$ there is a stable model $M \in \mathrm{SM}(P)$;
- $\langle P, Q\rangle \in \mathbf{IMPR} \iff \mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and for each $M \in \mathrm{SM}(P)$, there is $N \in \mathrm{SM}(Q)$ such that $N_{\mathrm{v}} = M_{\mathrm{v}}$;
- $\langle P, Q\rangle \in \mathbf{IMPL} \iff \langle Q, P\rangle \in \mathbf{IMPR}$; and
- $\langle P, Q\rangle \in \mathbf{EQV} \iff P \equiv_{\mathrm{v}} Q$.

The computational complexity of **SM** is already well-understood: it forms an **NP**-complete decision problem as shown by Marek and Truszczyński (1991) and Simons et al. (2002), and thus its complement $\overline{\mathbf{SM}}$ is **coNP**-complete.

Theorem 5.11
IMPR, **IMPL**, and **EQV** are **coNP**-complete decision problems for SMODELS programs having enough visible atoms.

Proof
Let us establish that (i) $\mathbf{IMPR} \in \mathbf{coNP}$ and (ii) $\overline{\mathbf{SM}}$ can be reduced to **IMPR**.

(i) Let P and Q be two SMODELS programs having enough visible atoms. Then define a reduction r from **IMPL** to $\overline{\mathbf{SM}}$ by setting $r(P, Q) = \mathrm{EQT}(P, Q)$ if $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and $r(P, Q) = \emptyset$ otherwise. To justify that $r(P, Q)$ can be computed in polynomial time we make the following observations. The condition $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ can be verified in linear time if atoms in $\mathrm{Hb}_{\mathrm{v}}(P)$ and $\mathrm{Hb}_{\mathrm{v}}(Q)$ are ordered, e.g., alphabetically. If not, sorting can be done in time of $\mathcal{O}(n \log n)$ where $n = \max(|\mathrm{Hb}_{\mathrm{v}}(P)|, |\mathrm{Hb}_{\mathrm{v}}(Q)|)$. Moreover, we can identify four subprograms of $\mathrm{EQT}(P, Q)$, i.e., P, $\mathrm{Hidden}^{\circ}(Q)$, $\mathrm{Least}^{\bullet}(Q)$, and $\mathrm{UnStable}(Q)$ in Definition 5.1 whose lengths depend mostly linearly on $||P||$, $||Q||$, and $|\mathrm{Hb}(Q)|$, respectively. The rules of Item 3 make the only exception with a quadratic blow-up.
It follows by Definition 5.10 and Theorem 5.8 that $\langle P, Q\rangle \in \mathbf{IMPR} \iff r(P, Q) \notin \mathbf{SM}$, i.e., $r(P, Q) \in \overline{\mathbf{SM}}$. Since $\overline{\mathbf{SM}} \in \mathbf{coNP}$ (Simons et al. 2002) and r is a polynomial time reduction from **IMPR** to $\overline{\mathbf{SM}}$, $\mathbf{IMPR} \in \mathbf{coNP}$.

(ii) Let R be any SMODELS program. Now $R \in \overline{\mathbf{SM}} \iff R \notin \mathbf{SM} \iff \mathrm{SM}(R) = \emptyset$. Then consider any SMODELS program $\bot$ having no stable models, i.e., $\mathrm{SM}(\bot) = \emptyset$, with a visible Herbrand base $\mathrm{Hb}_{\mathrm{v}}(\bot) = \mathrm{Hb}_{\mathrm{v}}(R)$. It follows that $\mathrm{SM}(R) = \emptyset \iff \langle R, \bot\rangle \in \mathbf{IMPR}$. Thus $R \in \overline{\mathbf{SM}} \iff \langle R, \bot\rangle \in \mathbf{IMPR}$.

Items (i) and (ii) above imply that **IMPR** is **coNP**-complete. The classification of **IMPL** follows by a trivial reduction $\langle P, Q\rangle \in \mathbf{IMPR} \iff \langle Q, P\rangle \in \mathbf{IMPL}$ that works in both directions, i.e., from **IMPR** to **IMPL** and back.

The case of **EQV** follows. Definitions 4.6 and 5.10 imply that $\langle P, Q\rangle \in \mathbf{EQV} \iff \langle P, Q\rangle \in \mathbf{IMPR}$ and $\langle P, Q\rangle \in \mathbf{IMPL}$. Thus $\mathbf{EQV} = \mathbf{IMPR} \cap \mathbf{IMPL}$ and $\mathbf{EQV} \in \mathbf{coNP}$ as **coNP** is closed under intersection. The **coNP**-hardness of **EQV** follows easily as it holds for any SMODELS program R and a trivial SMODELS program $\bot$ with $\mathrm{SM}(\bot) = \emptyset$ and $\mathrm{Hb}_{\mathrm{v}}(\bot) = \mathrm{Hb}_{\mathrm{v}}(R)$ that $R \in \overline{\mathbf{SM}} \iff \langle R, \bot\rangle \in \mathbf{EQV}$. Thus we may conclude that **EQV** is **coNP**-complete. □

6 Weight constraint programs

The verification method presented in Section 5 covers the class of SMODELS programs as defined in Section 2. This class corresponds very closely to the input language of the SMODELS search engine but it excludes *optimization statements* which will not be addressed in this paper. In this section we concentrate on extending our translation-based verification method for the input language supported by the front-end of the SMODELS system, namely LPARSE (Syrjänen 2001; Syrjänen and Niemelä 2001). The class of *weight constraint programs* (Simons et al. 2002) provides a suitable abstraction of this language in the propositional case.[8] Simons et al. (2002) show how weight constraint programs can be transformed into SMODELS programs using a modular translation that introduces new atoms. Our strategy is to use this translation to establish that the weak equivalence of two weight constraint programs reduces to the visible equivalence of the respective translations.

Next we introduce the syntax and semantics of weight constraint programs. Recalling the syntax of weight rules (4), a natural way to extend their expressiveness is to allow more versatile use of weights as well as constraints associated with them. This is achieved by recognizing weight constraints as first-class citizens and using them as basic building blocks of rules instead of plain atoms.

Definition 6.1
A weight constraint C is an expression of the form

$$l \leqslant \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, {\sim}b_1 = w_{b_1}, \ldots, {\sim}b_m = w_{b_m}\} \leqslant u, \tag{10}$$

where a_i's and b_j's are atoms, and l, u, w_{a_i}'s, and w_{b_j}'s are natural numbers.

As before, we use a shorthand $l \leqslant \{A = W_A, {\sim}B = W_B\} \leqslant u$ for a weight constraint (10) where $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_m\}$ are the sets of atoms appearing in the constraint. The numbers l and u give the respective *lower and upper bounds* for the constraint. Definition 6.1 can be extended to the case where integers rather than natural numbers are used as weights. However, negative weights can be translated away (Simons et al. 2002) from weight constraints.

Definition 6.2
A weight constraint rule is an expression of the form

$$C_0 \leftarrow C_1, \ldots, C_r \tag{11}$$

where C_i is a weight constraint for each $i \in \{0, \ldots, r\}$.

A weight constraint program is a program consisting of weight constraint rules. As a weight constraint rule (11) is a generalization of a weight rule (4), we can define the satisfaction relation for weight constraint programs in analogy to Definition 3.2.

Definition 6.3
For a weight constraint program P and an interpretation $I \subseteq \mathrm{Hb}(P)$,

[8] Since LPARSE is responsible for instantiating variables and pre-interpreting certain function symbols the input language is actually much more general.

1. a weight constraint C of the form $l \leqslant \{A = W_A, \sim B = W_B\} \leqslant u$ is satisfied in $I \iff l \leqslant \mathrm{WS}_I(A = W_A, \sim B = W_B) \leqslant u$,
2. a weight constraint rule of the form $C_0 \leftarrow C_1, \dots, C_r$ is satisfied in $I \iff I \models C_0$ is implied by $I \models C_1, \dots,$ and $I \models C_r$, and
3. $I \models P \iff I \models r$ for every weight constraint rule $r \in P$.

The stable model semantics of normal programs (Gelfond and Lifschitz 1988) can be generalized to the case of weight constraint programs using the reduction devised for them by Simons et al. (2002).

Definition 6.4
Given an interpretation I for a weight constraint C of the form $l \leqslant \{A = W_A, \sim B = W_B\} \leqslant u$, the reduct C^I is the constraint $l' \leqslant \{A = W_A\}$ where the lower bound $l' = \max(0, l - \mathrm{WS}_I(\sim B = W_B))$.

Definition 6.5
For a weight constraint program P and an interpretation $I \subseteq \mathrm{Hb}(P)$, the reduct P^I contains a reduced weight constraint rule $h \leftarrow C_1^I, \dots, C_r^I$ for each $C_0 \leftarrow C_1, \dots, C_r \in P$ and $h \in A_0 \cap I$ satisfying for all $i \in \{1, \dots, r\}$, $\mathrm{WS}_I(A_i = W_{A_i}, \sim B_i = W_{B_i}) \leqslant u_i$.

It should be pointed out that P^I consists of *Horn constraint rules* of the form $h \leftarrow C_1, \dots, C_r$, where h is an atom, each constraint C_i contains only positive literals and has only a lower bound condition. We say that a weight constraint program P is positive if all the rules in P are Horn constraint rules. Thus P^I is positive by definition. The properties of minimal models carry over to the case of weight constraint programs, too. Thus a *positive* weight constraint program P has a unique minimal model, the least model, $\mathrm{LM}(P)$, and we can define stable models for weight constraints programs almost in analogy to SMODELS programs.

Definition 6.6
An interpretation $M \subseteq \mathrm{Hb}(P)$ for a weight constraint program P is a stable model of $P \iff$ (i) $M \models P$ and (ii) $M = \mathrm{LM}(P^M)$.

This definition is only slightly different from Definition 3.8 as $M \models P \iff M \models P^M$ does not hold generally for weight constraint programs – making condition (i) in Definition 6.6 necessary. However, if we consider the restricted language described in Section 2 and interpret the rules involved as weight constraint rules (11), then Definitions 3.8 and 6.6 yield the same semantics as stated below. To this end, we consider only choice rules (3) and weight rules (4) without loss of generality. Simons et al. (2002) encode rules of these forms using the following weight constraint rules:

$$0 \leqslant \{h_1 = 1, \dots, h_l = 1\} \leftarrow n + m \leqslant \{a_1 = 1, \dots, a_n = 1, \sim b_1 = 1, \dots, \sim b_m = 1\} \quad (12)$$

$$1 \leqslant \{h = 1\} \leftarrow w \leqslant \{a_1 = w_{a_1}, \dots, a_n = w_{a_n}, \sim b_1 = w_{b_1}, \dots, \sim b_m = w_{b_m}\} \quad (13)$$

Proposition 6.7
Let P be an SMODELS program and P_w its representation as a weight constraint program. Then for any interpretation $M \subseteq \mathrm{Hb}(P) = \mathrm{Hb}(P_w)$,

$$M = \mathrm{LM}(P^M) \iff M \models P_w \text{ and } M = \mathrm{LM}(P_w^M).$$

The proof of this proposition is given in Appendix A. Simons et al. (2002) show how weight constraint programs can be translated into SMODELS *programs* consisting only of basic rules (1), choice rules (3) and weight rules (4). The translation is highly *modular* so that each weight constraint rule can be translated independently of each other. However, in order to keep the length of the translation linear, two new atoms have to be introduced for each weight constraint appearing in a program.

Definition 6.8
The translation $\mathrm{Tr}_{\mathrm{SNS}}(C)$ of a weight constraint C of the form $l \leqslant \{A = W_A, {\sim}B = W_B\} \leqslant u$ is translated into two weight rules

$$\mathrm{sat}(C) \leftarrow l \leqslant \{A = W_A, {\sim}B = W_B\}. \tag{14}$$
$$\mathrm{unsat}(C) \leftarrow u + 1 \leqslant \{A = W_A, {\sim}B = W_B\}. \tag{15}$$

where $\mathrm{sat}(C)$ and $\mathrm{unsat}(C)$ are new atoms specific to C.

Definition 6.9
Let P be a weight constraint program and $f \notin \mathrm{Hb}(P)$ a new atom. The translation of P into an SMODELS program $\mathrm{Tr}_{\mathrm{SNS}}(P)$ consists of

1. $\mathrm{Tr}_{\mathrm{SNS}}(C)$ for each weight constraint C appearing in P and
2. the following SMODELS rules introduced for each $C_0 \leftarrow C_1, \ldots, C_r \in P$:

$$\{A_0\} \leftarrow \mathrm{sat}(C_1), {\sim}\mathrm{unsat}(C_1), \ldots, \mathrm{sat}(C_r), {\sim}\mathrm{unsat}(C_r). \tag{16}$$
$$f \leftarrow {\sim}\mathrm{sat}(C_0), \mathrm{sat}(C_1), {\sim}\mathrm{unsat}(C_1), \ldots, \mathrm{sat}(C_r), {\sim}\mathrm{unsat}(C_r). \tag{17}$$
$$f \leftarrow \mathrm{unsat}(C_0), \mathrm{sat}(C_1), {\sim}\mathrm{unsat}(C_1), \ldots, \mathrm{sat}(C_r), {\sim}\mathrm{unsat}(C_r). \tag{18}$$
$$\texttt{compute } \{{\sim}f\}. \tag{19}$$

where A_0 is the set of positive default literals appearing in C_0.

The visible Herbrand base of $\mathrm{Tr}_{\mathrm{SNS}}(P)$ is defined by $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{SNS}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$.

Let us then briefly explain intuitions underlying $\mathrm{Tr}_{\mathrm{SNS}}$. The rules given in (14) and (15) check whether the lower bound l of the weight constraint C is satisfied the upper bound u of C is *not* satisfied, respectively, and then $\mathrm{sat}(C)$ and $\mathrm{unsat}(C)$ can be inferred by the rules accordingly. The choice rule in (16) makes any subset of A_0 true if the body of the weight constraint rule is satisfied in the sense of Definition 6.3, i.e., all lower bounds and upper bounds are met. Finally, two basic rules in (17) and (18) and the compute statement in (19) ensure the satisfaction of the head constraint C_0 whenever the body $C_1, \ldots, C_r$ is satisfied. A very tight correspondence of stable models is obtained using the translation $\mathrm{Tr}_{\mathrm{SNS}}(P)$.

Theorem 6.10
The translation function $\mathrm{Tr}_{\mathrm{SNS}}$ is faithful, i.e., $P \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{SNS}}(P)$ holds for all weight constraint programs P.

The proof of the theorem can be found in Appendix A. Theorem 6.10 and Definition 6.9 imply together that the visible equivalence of weight constraint programs can be reduced to that of SMODELS programs using $\mathrm{Tr}_{\mathrm{SNS}}$.

Corollary 6.11
For all weight constraint programs P and Q,

$$P \equiv_{\mathrm{v}} Q \iff \mathrm{Tr}_{\mathrm{SNS}}(P) \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{SNS}}(Q).$$

However, we have to address the degree of visibility of atoms in the translation $\mathrm{Tr}_{\mathrm{SNS}}(P)$ in order to apply the translation-based method presented in Section 5. Recalling the limitations of the method, we should establish that $\mathrm{Tr}_{\mathrm{SNS}}(P)$ and $\mathrm{Tr}_{\mathrm{SNS}}(Q)$ have enough visible atoms under some reasonable assumptions about P and Q. For the sake of simplicity, we will only consider a relatively straightforward setting made precise in Proposition 6.12. Nevertheless, it implies the applicability of our verification method to a substantial class of weight constraint programs.

Proposition 6.12
If P is a weight constraint program such that $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$, then $\mathrm{Tr}_{\mathrm{SNS}}(P)$ has enough visible atoms.

Proof
Let P be any weight constraint program such that $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$, i.e., $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P)$. Moreover, let us pick any interpretation $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(P)$. Since $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$ we have $I_{\mathrm{v}} = I$ so that I is actually an interpretation for the whole program.

Let us then consider any rule $C_0 \leftarrow C_1, \ldots, C_r \in P$ and its translation under $\mathrm{Tr}_{\mathrm{SNS}}$ as given in Definition 6.9. Since $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{SNS}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$ by definition and $I = I_{\mathrm{v}}$, the rules involved in the translation contribute to $P_{\mathrm{h}}/I_{\mathrm{v}}$ as follows: (14) is reduced to $\mathrm{sat}(C_i) \leftarrow l_i' \leqslant \{\}$ where $l_i' = \max(0, l_i - w_i)$ for $w_i = \mathrm{WS}_{I_{\mathrm{v}}}(A_i = W_{A_i}, \sim B_i = W_{B_i})$; (15) is reduced to $\mathrm{unsat}(C_i) \leftarrow u_i' \leqslant \{\}$ where $u_i' = \max(0, (u_i + 1) - w_i)$; (16) is dropped altogether as $(A_0)_{\mathrm{h}} = \emptyset$; (17) and (18) remain intact because they involve only hidden atoms; and (19) is dropped by definition.

Let us then verify that $\mathrm{Tr}_{\mathrm{SNS}}(P)_{\mathrm{h}}/I_{\mathrm{v}}$ is a *stratified* program. After inspecting the dependencies in the reduced rules, we note that the hidden atoms in $\mathrm{Hb}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ can be assigned to strata as follows: the atoms $\mathrm{sat}(C)$ and $\mathrm{unsat}(C)$ associated with weight constraints C belong to stratum 0 and f belongs to stratum 1. Thus $\mathrm{Tr}_{\mathrm{SNS}}(P)$ is essentially a stratified normal logic program as the remainders of weight rules act as facts. Then $\mathrm{Tr}_{\mathrm{SNS}}(P)$ has a unique stable model (Apt et al. 1988). □

By denying occurrences of hidden atoms in weight constraint programs, we obtain a translation-based method for verifying weak equivalence. Note that the requirement $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$, i.e., $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ in this case, can be easily met by extending the Herbrand bases of programs as discussed in Section 2.

Corollary 6.13
Let P and Q two weight constraint programs such that $\mathrm{Hb}_{\mathrm{h}}(P) = \mathrm{Hb}_{\mathrm{h}}(Q) = \emptyset$ and $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$. Then $P \equiv Q \iff \mathrm{Tr}_{\mathrm{SNS}}(P) \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{SNS}}(Q) \iff$ the translations $\mathrm{EQT}(\mathrm{Tr}_{\mathrm{SNS}}(P), \mathrm{Tr}_{\mathrm{SNS}}(Q))$ and $\mathrm{EQT}(\mathrm{Tr}_{\mathrm{SNS}}(Q), \mathrm{Tr}_{\mathrm{SNS}}(P))$ have no stable models.

It seems that hidden atoms can be tolerated in weight constraint programs to some degree, but we skip such an extension of Corollary 6.13 for space reasons. Nevertheless, the result established above enables us to implement the verification of weak equivalence for the programs produced by LPARSE.

7 Experiments

The translation function EQT presented in Section 5 has been implemented in C under the Linux operating system. The translator which we have named LPEQ takes two logic programs P and Q as its input and produces the translation $\mathrm{EQT}(P, Q)$ as its output. The implementation assumes the internal file format of SMODELS which enables us to use the front-end LPARSE of SMODELS in conjunction with LPEQ.[9] It is yet important to note that LPEQ checks that the *visible* Herbrand bases of the programs being compared are exactly the same as insisted by $\equiv_{\mathrm{v}}$. In practice, visible atoms are recognized as those having a name in the symbol table of a program. The latest version of LPEQ is also prepared to deal with programs involving *invisible* atoms, e.g., introduced by the front-end LPARSE as discussed in Section 6. Before producing the translation $\mathrm{EQT}(P, Q)$, the translator uses Tarjan's algorithm to find *strongly connected components* for the dependency graph of Q_{h} when its checks that $Q_{\mathrm{h}}/I_{\mathrm{v}}$ is stratifiable for all $I_{\mathrm{v}} \subseteq \mathrm{Hb}_{\mathrm{v}}(Q)$. An overapproximation is used in this respect: all dependencies of invisible atoms are taken into account regardless of I_{v}. A successful test guarantees that Q has enough visible atoms so that $\mathrm{EQT}(P, Q)$ works correctly. Otherwise, an error message is printed for the user.

The current implementation (LPEQ version 1.17) is available[10] in the WWW. The files related with benchmark problems and experiments reported in this section are also provided. To assess the feasibility of LPEQ in practice we performed a number of tests with different test cases. The running times of the LPEQ approach were compared with those of a fictitious approach, i.e., the NAIVE one:

1. Compute one stable model M of P not computed so far.
2. Check whether Q has a stable model N such that $M_{\mathrm{v}} = N_{\mathrm{v}}$. Stop if not.
3. Continue from step 1 until all stable models of P have been enumerated.

It is obvious that a similar check has to be carried out in the other direction to establish $P \equiv_{\mathrm{v}} Q$ in analogy to Corollaries 5.9 and 6.13.

There is still room for optimization in both approaches. If one finds a counter-example in one direction, then $P \not\equiv_{\mathrm{v}} Q$ is known to hold and there is no need to do testing in the other direction except if one wishes to perform a thorough analysis. Since running times seem to scale differently depending on the direction, we count always running times in both directions. However, one should notice that the search for counter-examples in one direction is stopped immediately after finding a counter-example. Since the running times of SMODELS may also depend on the order of rules in programs and literals in rules, we shuffle them randomly.

In both approaches, the SMODELS system (version 2.28) is responsible for the computation of stable models for programs that are instantiated using the front-end LPARSE (version 1.0.13). In the LPEQ approach, the total running time in one direction is the running time needed by SMODELS for trying to compute *one* stable model of the translation produced by LPEQ. The translation time is also taken into account

[9] A textual human-readable output can also be produced on request.
[10] Please consult `http://www.tcs.hut.fi/Software/lpeq/` for binaries and scripts involved.

(a) Place a queen on each column

```
negq(X,Y2) :- q(X,Y), d(X), d(Y), d(Y2), Y2 != Y.
q(X,Y) :- not negq(X,Y), not q(X,Y2): d(Y2): Y2 != Y , d(X), d(Y).
hide negq(X,Y).
```

(b) Place a queen on each column using a choice rule

```
1 { q(X,Y):d(Y) } 1 :- d(X).
```

(c) Place a queen on each row

```
negq(X2,Y) :- q(X,Y), d(X), d(Y), d(X2), X2 != X.
q(X,Y) :- not negq(X,Y), not q(X2,Y): d(X2): X2 != X , d(X), d(Y).
hide negq(X,Y).
```

(d) Make sure that queens do not threaten each other (same row or diagonal)

```
:- d(X), d(Y), d(X1), q(X,Y), q(X1,Y), X1 != X.
:- d(X), d(Y), d(X1), d(Y1), q(X,Y), q(X1,Y1), X != X1,  Y != Y1,
   abs(X - X1) == abs(Y - Y1).
d(1..queens).
```

(e) Make sure that queens do not threaten each other (same column or diagonal)

```
:- d(X), d(Y), d(Y1), q(X,Y), q(X,Y1), Y1 != Y.
:- d(X), d(Y), d(X1), d(Y1), q(X,Y), q(X1,Y1), X != X1,  Y != Y1,
   abs(X - X1) == abs(Y - Y1).
d(1..queens).
```

Fig. 3. Encoding the n-queens problem.

although it is negligible. The NAIVE approach has been implemented as a shell script. The running time in one direction consists of the running time of SMODELS for finding the necessary (but not necessarily all) stable models of P plus the running times of SMODELS for testing that the stable models found are also stable models of Q. These tests are realized in practice by adding $M_v \cup \{\sim a \mid a \in \mathrm{Hb}_v(Q) \setminus M_v\}$ as a compute statement to Q. It is worth noting that the NAIVE approach does not test in any way that the stable model N of Q with $M_v = N_v$ is unique. However, the set of benchmarks is selected in such a way that programs have enough visible atoms and the correctness of the NAIVE approach is guaranteed. All the tests reported in this section were run under the Linux 2.6.8 operating system on a 1.7GHz AMD Athlon XP 2000+ CPU with 1 GB of main memory. As regards timings in test results, we report the sum of user and system times.

7.1 The n-Queens Benchmark

Our first experiment was based on the n-queens problem. We verified the visible equivalence of three different formulations which are variants of one proposed by Niemelä (1999, p. 260). The encoding $Q_n^{x_1}$ consists of parts (a) and (d) given in Figure 3 and is designed so that queens are placed column-wise to the board. The second program $Q_n^{x_2}$ consists of parts (b) and (d) given in Figure 3, i.e., it is a variant of $Q_n^{x_1}$ where the choice between placing or not placing a queen in a particular cell of the chessboard is equivalently formulated using a choice rule rather than plain basic

Table 1. *Results for two equivalent logic programs (n-queens)*

n	SM[c]	t_{avg}[a] LPEQ	t_{avg} NAIVE	RAR[d]	CP_{avg}[b] LPEQ	CP_{avg} NAIVE	RI[e]	RO[f]
1	1	0.000	0.080	–	0	0	7	28
2	0	0.000	0.051	–	0	0	38	130
3	0	0.003	0.051	17.000	0	0	124	384
4	2	0.019	0.120	6.316	0	2	300	884
5	10	0.042	0.454	10.810	5	18	600	1718
6	4	0.136	0.259	1.904	16	18	1058	2974
7	40	0.516	2.340	4.535	40	84	1708	4740
8	92	2.967	6.721	2.265	163	253	2584	7104
9	352	17.316	32.032	1.850	615	955	3720	10154
10	724	99.866	90.694	0.908	2613	3127	5150	13978
11	2680	617.579	451.302	0.731	11939	13662	6908	18664

[a] Average running time in seconds.
[b] Average number of choice points during the search.
[c] Number of stable models for $Q_n^{x_1}$ and $Q_n^{x_2}$.
[d] Ratio of average running times.
[e] Number of rules in the input: $|Q_n^{x_1}| + |Q_n^{x_2}|$.
[f] Number of rules in the output: $|\text{EQT}(Q_n^{x_1}, Q_n^{x_2})| + |\text{EQT}(Q_n^{x_2}, Q_n^{x_1})|$.

rules. The third program Q_n^y, i.e., parts (c) and (e) given in Figure 3, is an orthogonal version of $Q_n^{x_1}$ in which queens are placed row-wise rather than column-wise.

First we verified the visible equivalence of $Q_n^{x_1}$ and $Q_n^{x_2}$ and then that of $Q_n^{x_1}$ and Q_n^y using both the LPEQ and the NAIVE approaches. The number of queens n was varied from 1 to 11 and the verification task was repeated 100 times for each number of queens – generating each time new randomly shuffled versions of the programs involved. The results of these experiments are shown in Tables 1 and 2, respectively. It appears that the NAIVE approach becomes superior in the case of two equivalent well-structured logic programs containing hidden atoms (the atoms `negq(X,Y)` are explicitly hidden) as programs grow and the number of stable models increases. Comparing the average running times from Tables 1 and 2, it can be seen that the difference in running times is smaller in the case where the second program does not contain hidden atoms. This can be seen as an indication that it is particularly the translation of the hidden part $\text{Hidden}^{\circ}(\cdot)$ that increases the running time of the LPEQ approach. To investigate this further, we verified the equivalence of $Q_n^{x_1}$ and Q_n^y without declaring the atoms `negq(X,Y)` hidden. The results obtained from this experiment resembled our previous results in Janhunen and Oikarinen (2002), i.e., the LPEQ approach performs somewhat better than the NAIVE one. Moreover, the average running times of NAIVE approach are approximately the same as with hidden atoms, but the average running times for the LPEQ approach are significantly smaller. The reason why the NAIVE approach appears to be immune to changes in the visibility of atoms is the following. In our encodings of the n-queens problem, the interpretation for hidden atoms can be directly determined once the interpretation for visible part is known. However, this is not necessarily the case in general and

Table 2. *Results for two equivalent logic programs (n*-queens*).*

n	SM[c]	t_{avg}[a] LPEQ	t_{avg} NAIVE	RAR[d]	CP_{avg}[b] LPEQ	CP_{avg} NAIVE	RI[e]	RO[f]
1	1	0.000	0.080	–	0	0	4	30
2	0	0.000	0.050	–	0	0	36	146
3	0	0.007	0.052	7.43	0	0	136	478
4	2	0.020	0.124	6.20	0	2	344	1146
5	10	0.052	0.473	9.09	4	18	700	2270
6	4	0.169	0.281	1.66	16	18	1244	3970
7	40	0.815	2.583	3.17	38	84	2016	6366
8	92	5.994	7.531	1.26	176	263	3116	9578
9	352	35.900	36.836	1.03	603	955	4404	13726
10	724	238.726	110.109	0.46	2734	3243	6100	18930
11	2680	1521.730	565.029	0.37	12210	13927	8184	25310

[a] Average running time in seconds.
[b] Average number of choice points during the search.
[c] Number of stable models for Q_n^{x1} and Q_n^y.
[d] Ratio of average running times.
[e] Number of rules in the input: $|Q_n^{x1}| + |Q_n^y|$.
[f] Number of rules in the output: $|\text{EQT}(Q_n^{x1}, Q_n^y)| + |\text{EQT}(Q_n^y, Q_n^{x1})|$.

finding the unique stable model for the hidden part can be more laborious and time consuming as in our last benchmark to be described in Section 7.3.

We chose the pairs of programs $(Q_n^{x_1}, Q_n^{x_2})$ and $(Q_n^{x_1}, Q_n^y)$ for our experiments in order to to see if the two approaches would perform differently depending on whether a *local change* (a choice rule is used instead of basic rules) or a *global change* (an orthogonal encoding is introduced) is made in the encoding. However, our test results show no clear indication in either direction. Furthermore, we decided to test non-equivalent pairs of n-queens programs. To this end, we dropped n random rules from Q_n^y, and verified the equivalence of $Q_n^{x_1}$ and the modified version of Q_n^y by selecting only non-equivalent pairs (both with and without hidden atoms). The results turned out to be very similar to the results that were obtained for equivalent program pairs. In all our n-queens experiments the number of *choice points* (i.e., the number of choices made by SMODELS while searching for stable models for the translation) is slightly smaller in the LPEQ approach than in the NAIVE one. Thus it seems that verifying the equivalence of logic programs using LPEQ leads to smaller search space, but the eventual efficiency can vary as far as time is concerned.

7.2 Random 3-SAT and graph problems

We also performed some tests with randomly generated logic programs. We generated logic programs that solve an instance of a random 3-sat problem with a constant clauses to variables ratio $c/v = 4$. Such instances are typically satisfiable, but so close to the *phase transition point* (approximately 4.3) that finding models is already demanding for SAT solvers. To simulate a sloppy programmer making mistakes, we dropped one random rule from each program. Due to non-existence of

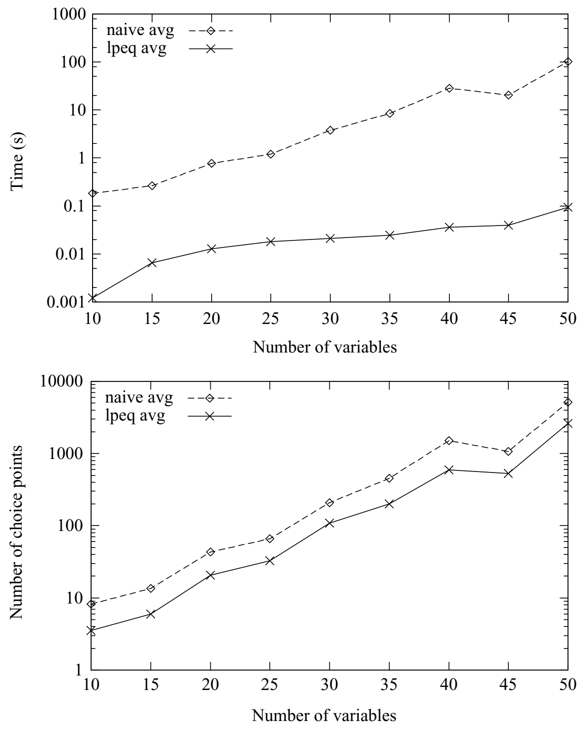

Fig. 4. Average running times and numbers of choice points for random 3-sat instances with the ratio $c/v = 4$.

hidden atoms, we checked the weak equivalence of the modified program and the original program to see if making such a mistake affects stable models or not. As a consequence, the pairs of programs included both equivalent and nonequivalent cases. When $c/v = 4$, approximately 45–60% of the program pairs were equivalent. This does not seem to depend much on the problem size (measured in the number of variables in the problem) within problem sizes used in the experiments. With smaller values of c/v the percentage of equivalent program pairs is lower but for larger values of c/v the percentage grows up to 90%. In the first experiment with random 3-sat programs, we varied the number of variables v from 10 to 50 with steps of 5. For each number of variables we repeated the test 100 times and generated each time a new random instance. The average running times and the average number of choice points for both approaches are shown in Figure 4. These results indicate that the LPEQ approach is significantly faster than the NAIVE one. The difference increases as program instances grow. The number of choice points is also lower in the former approach on an average.

In the second experiment with random 3-sat instances we generated programs as in the previous experiment, but we kept the number of variables constant, $v = 40$, and varied the ratio c/v from 3.75 to 4.75 with steps of 0.125. For each value of the ratio c/v, we repeated the test 100 times generating each time a new random instance. The motivation behind this experiment was to see how the LPEQ approach

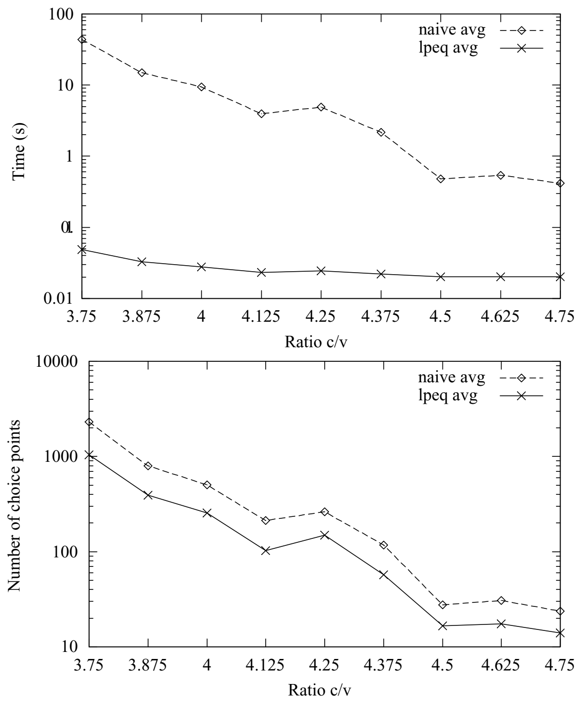

Fig. 5. Average running times and numbers of choice points for random 3-sat instances with fixed $v = 40$ and varying ratio c/v.

performs compared to the NAIVE one as the programs change from almost always satisfiable (many stable models) to almost always unsatisfiable (no stable models). The averages of running times and numbers of choice points are presented in Figure 5 for both approaches. For low values of the ratio c/v, the LPEQ approach is significantly better than the NAIVE one like previously. As the ratio increases, the performance of the NAIVE approach gradually improves, but LPEQ is still better. The average number of choice points is also lower in the LPEQ approach.

We also combined structured logic programs with randomness. We used two graph problems formalized with rules by Niemelä (1999, p. 262): the problems of n-coloring of a graph with n colors and finding a Hamiltonian circuit for a graph. Using the Stanford GraphBase library, we generated random planar graphs with v vertices where v ranges from 10 to 17 and instantiated the respective logic programs for 4-coloring and Hamiltonian circuit by invoking LPARSE. As in the preceding experiments with random 3-sat programs, the second program for equivalence testing was obtained by dropping one random rule from the one instantiated by LPARSE. The tests were repeated 100 times for each value of v using a new random planar graph every time. The average running times and the average number of choice points for both experiments are presented in Figure 6. In both experiments the LPEQ approach is significantly better than the NAIVE approach, though running times differ more in the 4-coloring problem. The numbers of choice points vary as before.

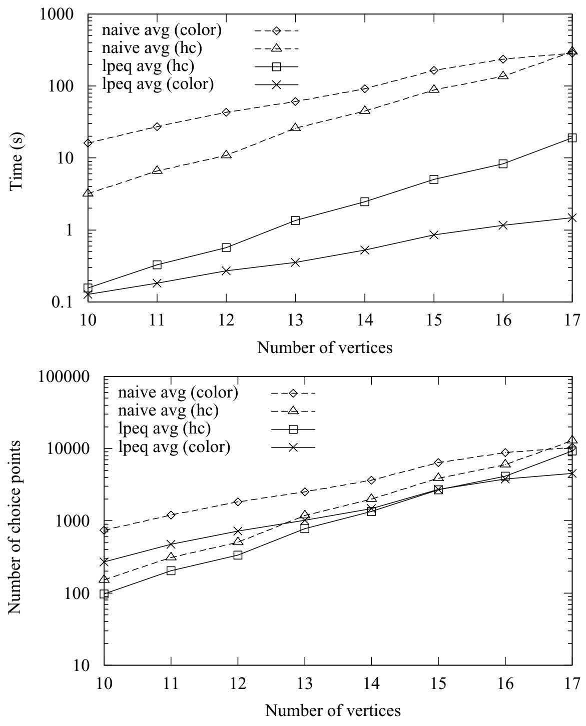

Fig. 6. Average running times and average number of choice points for 4-coloring random planar graphs and finding Hamiltonian circuits.

7.3 Knapsack

Finally, we used the knapsack problem whose encoding involves weight constraints. Here the objective was to test the performance of the translation-based approach when programs involve a substantial number of hidden atoms and the verification of equivalence requires the property of having enough visible atoms as stated in Corollary 6.13. It should be stressed that the previous version of LPEQ (1.13) and the corresponding translation presented in (Janhunen and Oikarinen 2002) do not cover such programs. In the knapsack problem, there are n types of items, each item of type i has size w_i and profit c_i. The goal is to fill the knapsack with X_i items of type i so that the maximum size W is not exceeded and the minimum profit C is gained, i.e.,

$$\sum_{i=1}^{n} X_i \cdot w_i \leqslant W \text{ and } \sum_{i=1}^{n} X_i \cdot c_i \geqslant C.$$

We decided to use an encoding of the knapsack problem proposed by Dovier et al. (2005) using the same weights and costs. An instance of the encoding is denoted by KS(W, C) where the parameters W and C are as above. We considered the visible equivalence of programs KS$(127, C)$ and KS$(127, C-1)$ for the values of C in the sequence $184, 180, \ldots, 104, 100$. The starting value $C = 184$ was selected, since it is the highest possible value for KS$(127, C)$ to have stable models. As C decreases, the number of stable models possessed by KS$(127, C)$ grows. For each value of C

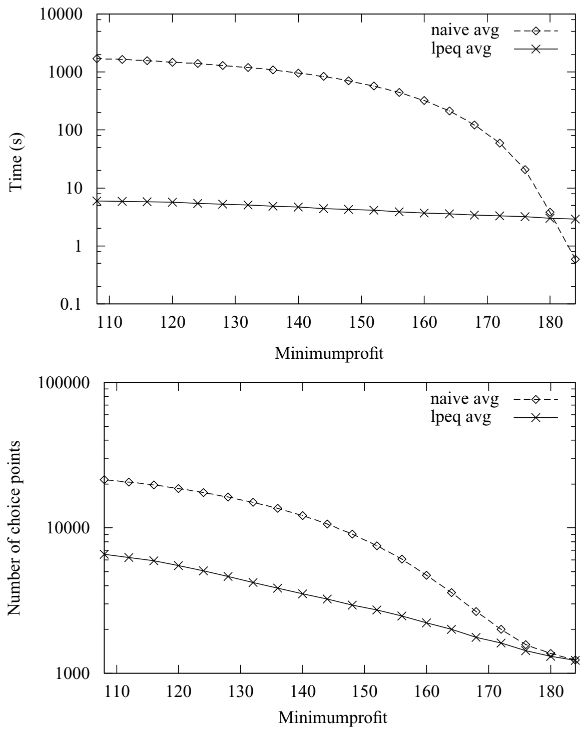

Fig. 7. Average running times and average number of choice points for knapsack.

we generated 10 randomly shuffled versions of KS$(127, C)$ and KS$(127, C-1)$. The programs KS$(127, C)$ and KS$(127, C-1)$ are always visibly non-equivalent as stable models of KS$(127, C)$ are also stable models of KS$(127, C-1)$ up to visible parts, but not vice versa, because of weights used in (Dovier et al. 2005).

The averages of running times and numbers of choice points for the knapsack problem are presented in Figure 7. It is worth noting that the total running time is dominated by the direction that does not yield a counter-example. However, the LPEQ approach is also significantly faster than the NAIVE one in the direction that actually yields counter-examples.

8 Conclusion

In this article, we propose a translation-based approach for verifying the equivalence of logic programs under the stable model semantics. The current translation EQT(P, Q) and its implementation LPEQ cover the types of rules supported by the SMODELS search engine which provide the basic knowledge representation primitives. More general forms of rules implemented in the front-end LPARSE are also covered by LPEQ. This is partially achieved by LPARSE itself as it expresses high-level constructs using the primitives of the engine. However, the task of verifying equivalence is complicated considerably since LPARSE may have to introduce hidden atoms. To this end, the newest version of LPEQ includes a proper support for hidden atoms so that it can be used to verify *visible equivalence* of SMODELS programs (denoted $\equiv_{\mathrm{v}}$) rather than ordinary weak equivalence (denoted $\equiv$). The underlying theory around the property of having enough visible atoms is developed in Section 4 and we consider these ideas as a significant extension to the original translation-based approach presented in Janhunen and Oikarinen (2002).

Our conclusion of the experiments reported in Section 7 is that the translation-based approach can really be useful in practice. In many cases, the number of choice points and time needed for computations is less than in the NAIVE cross-checking approach. To the best of our understanding, this is because the translation $\mathrm{EQT}(P, Q)$ provides an explicit specification of a counter-example that guides the search performed by SMODELS. Such coordination is not possible in the NAIVE approach where the stable models of P and Q are computed separately and cross-checked. However, if the programs being compared are likely to have few stable models or no stable models at all, we expect that the NAIVE approach becomes superior to ours. Recall that P is included in the translation $\mathrm{EQT}(P, Q)$ which has no stable models in the case that P has no stable models. The NAIVE approach may also be better off when programs turn out to be equivalent and the verification task boils down to establishing the correspondence of stable models.

As regards future work, there are several issues to be addressed.

- The current translation and its implementation LPEQ do not cover minimize/maximize statements that are nevertheless supported by the SMODELS search engine. Basically, one can deal with optimization on two levels. The first is to verify the equivalence of programs without optimization statements which should intuitively imply equivalence in the presence of the same optimization statements expressed in terms of visible atoms. The second approach is the fully general one that allows differences in the non-optimal models of the programs being compared and in the formulation of optimization statements as there may be several formulations that are effectively equivalent.
- Other notions of equivalence – such as the stronger notion of equivalence proposed by Lifschitz et al. (2001) – should be covered by devising and implementing suitable translations. Some translations in this respect have already been presented by Turner (2003) and Eiter et al. (2004). However, the visibility aspects of these relations have not been fully analyzed so far.
- The current implementation provides already a reasonably good support for invisible atoms, since those introduced by LPARSE can be dealt with. However, the notion of stratification used by LPEQ is very cautious and we should also pursue other natural classes of programs that have enough visible atoms. One obvious question in this respect is whether the property of having enough visible atoms is *preserved* by LPARSE.
- The case of disjunctive logic programs is also interesting, as efficient implementations are available: DLV (Leone et al. 2006) and GNT (Janhunen et al. 2006). The latter uses SMODELS for actual computations in analogy to the translation-based approach followed by this paper. In (Oikarinen and Janhunen 2004) we extend the translation-based approach to the disjunctive case. The respective implementation for disjunctive programs, namely DLPEQ,

is reported in Janhunen and Oikarinen (2004). For now, invisible atoms are not supported by DLPEQ and it is interesting to see whether the concept of having enough visible atoms lifts to the disjunctive case in a natural way.

Acknowledgments

The authors wish to thank Michael Gelfond, Stefan Woltran, and anonymous referees for their comments and suggestions for improvement. This research has been partially supported by the Academy of Finland (under Projects #53695 *"Applications of Rule-Based Constraint Programming"*, and #211025 *"Applications of Constraint Programming Techniques"*) and the European Commission (under contract IST-FET-2001-37004 *"Working Group on Answer Set Programming"*). The second author gratefully acknowledges financial support from Helsinki Graduate School in Computer Science and Engineering, Nokia Foundation, and Finnish Cultural Foundation.

Appendix A Proofs

Proof of Lemma 4.16

Suppose that $M \in \mathrm{SM}(P)$, i.e. $M = \mathrm{LM}(P^M)$ and $M \models \mathrm{CompS}(P)$. To prove $M_\mathrm{h} \in \mathrm{SM}(P_\mathrm{h}/M_\mathrm{v})$, let us establish first that $M_\mathrm{h} \models (P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$. Assuming the contrary, some rule $r \in (P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$ must be falsified by M_h. Since basic rules and constraint rules are special cases of weight rules (c.f. discussion after Definition 2.1), it is sufficient to consider only rules r of two types: weight rules and choice rules.

- If a weight rule $h \leftarrow w_1 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}\}$ in $(P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$ is falsified by M_h, we have $M_\mathrm{h} \not\models h$ and $w_1 \leqslant \mathrm{WS}_{M_\mathrm{h}}(A_\mathrm{h} = W_{A_\mathrm{h}})$ in which $w_1 = \max(0, w_2 - \mathrm{WS}_{M_\mathrm{h}}(\sim B_\mathrm{h} = W_{B_\mathrm{h}}))$ is related with a rule $h \leftarrow w_2 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}, \sim B_\mathrm{h} = W_{B_\mathrm{h}}\}$ included in $P_\mathrm{h}/M_\mathrm{v}$. Thus $w_2 \leqslant \mathrm{WS}_{M_\mathrm{h}}(A_\mathrm{h} = W_{A_\mathrm{h}}, \sim B_\mathrm{h} = W_{B_\mathrm{h}})$. Then the definition of $P_\mathrm{h}/M_\mathrm{v}$ and that of w_1 in terms of the bound w_2 imply that $w_2 = \max(0, w - \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}}, \sim B_\mathrm{v} = W_{B_\mathrm{v}}))$ for some weight rule $h \leftarrow w \leqslant \{A = W_A, \sim B = W_B\}$ of P. By combining weight sums on the basis of $M = M_\mathrm{h} \cup M_\mathrm{v}$, we obtain $w \leqslant \mathrm{WS}_M(A = W_A, \sim B = W_B)$. On the other hand, the reduct P^M contains a weight rule $h \leftarrow w_3 \leqslant \{A = W_A\}$ where $w_3 = \max(0, w - \mathrm{WS}_M(\sim B = W_B))$. It follows that $w_3 \leqslant \mathrm{WS}_M(A = W_A)$ and $M \not\models h$. A contradiction, since $M \models P^M$ holds for M.
- A choice rule $\{H_\mathrm{h}\} \leftarrow A_\mathrm{h}, \sim B_\mathrm{h}$ cannot be falsified by definition, a contradiction.

Hence $M_\mathrm{h} \models (P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$ and it remains to establish the minimality of M_h with respect to this property. Suppose there is $M' \models (P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$ such that $M' \subset M_\mathrm{h}$. Using M' we define an interpretation $N = M_\mathrm{v} \cup M'$ so that $N_\mathrm{v} = M_\mathrm{v}$, $N_\mathrm{h} = M' \subset M_\mathrm{h}$, and $N_\mathrm{h} \models (P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$ by definition. Let us then assume that $N \not\models P^M$, i.e., there is some rule r of the reduct P^M not satisfied by $N \subset M$. As above, it is sufficient to consider the contribution of weight rules and choice rules to P^M.

- If r is a weight rule $h \leftarrow w_2 \leqslant \{A = W_A\}$ in P^M, then $N \not\models h$ and $w_2 \leqslant \mathrm{WS}_N(A = W_A)$ holds for $w_2 = \max(0, w - \mathrm{WS}_M(\sim B = W_B))$ and some weight rule $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ of P. Since M_v and N_v coincide, we obtain

$$\begin{array}{rcl} w & \leqslant & \mathrm{WS}_N(A = W_A) + \mathrm{WS}_M({\sim}B = W_B) \\ & = & \mathrm{WS}_{N_\mathrm{h}}(A_\mathrm{h} = W_{A_\mathrm{h}}) + \mathrm{WS}_{M_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}}) + \\ & & \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}}, {\sim}B_\mathrm{v} = W_{B_\mathrm{v}}). \end{array} \qquad \text{(A 1)}$$

Two cases arise. (i) If $h \in \mathrm{Hb}_\mathrm{h}(P)$, then $P_\mathrm{h}/M_\mathrm{v}$ contains a rule $h \leftarrow w_3 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}, {\sim}B_\mathrm{h} = W_{B_\mathrm{h}}\}$ where $w_3 = \max(0, w - \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}}, {\sim}B_\mathrm{v} = W_{B_\mathrm{v}}))$. It follows by (A 1) that $w_3 \leqslant \mathrm{WS}_{N_\mathrm{h}}(A_\mathrm{h} = W_{A_\mathrm{h}}) + \mathrm{WS}_{M_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}})$. Moreover, the reduct $(P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$ includes a rule $h \leftarrow w_4 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}\}$ where $w_4 = \max(0, w_3 - \mathrm{WS}_{M_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}}))$. Thus $w_4 \leqslant \mathrm{WS}_{N_\mathrm{h}}(A_\mathrm{h} = W_{A_\mathrm{h}})$ holds so that $N \not\models h$ and $h \in \mathrm{Hb}_\mathrm{h}(P)$ imply $N_\mathrm{h} \not\models r$, a contradiction with $N_\mathrm{h} \models (P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$. (ii) If $h \in \mathrm{Hb}_\mathrm{v}(P)$, then the definition of N implies $M \not\models h$. Moreover $N \subseteq M$ implies $\mathrm{WS}_N(A = W_A) \leqslant \mathrm{WS}_M(A = W_A)$ so that $w_2 \leqslant \mathrm{WS}_M(A = W_A)$. Thus $M \not\models r$ and $M \not\models P^M$ which contradicts the fact that $M = \mathrm{LM}(P^M)$.

- If r is a basic rule $h \leftarrow A$ associated with a choice rule $\{H\} \leftarrow A, {\sim}B$ of P, then $h \in H$, $M \models h$, $M \models {\sim}B$, $N \not\models h$, and $N \models A$. Now $h \in \mathrm{Hb}_\mathrm{v}(P)$ is impossible as $M_\mathrm{v} = N_\mathrm{v}$, $M \models h$, and $N \not\models h$. Hence $h \in \mathrm{Hb}_\mathrm{h}(P)$ is necessarily the case and $H_\mathrm{h} \neq \emptyset$. Moreover, $M_\mathrm{v} = N_\mathrm{v}$, $N \models A$, and $M \models {\sim}B$ imply that $M_\mathrm{v} \models A_\mathrm{v} \cup {\sim}B_\mathrm{v}$. Thus $\{H_\mathrm{h}\} \leftarrow A_\mathrm{h}, {\sim}B_\mathrm{h}$ is included in $P_\mathrm{h}/M_\mathrm{v}$. In addition, $M \models {\sim}B$ and $M \models h$ imply $M_\mathrm{h} \models {\sim}B_\mathrm{h}$ and $M_\mathrm{h} \models h$ so that $h \leftarrow A_\mathrm{h}$ is included in $(P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$. Finally, we obtain $N_\mathrm{h} \models A_\mathrm{h}$, $N_\mathrm{h} \not\models h$ and $N_\mathrm{h} \not\models r$ from $N \models A$ and $N \not\models h$. A contradiction.

To conclude the analysis above, it must be the case that $N \models P^M$. Since $N \subset M$, this contradicts the fact that M is a minimal model of P^M. Thus M_h is necessarily a minimal model of $(P_\mathrm{h}/M_\mathrm{v})^{M_\mathrm{h}}$, i.e., a stable model of $P_\mathrm{h}/M_\mathrm{v}$. □

Proof of Proposition 5.7
We prove the given four claims depending on conditions (i) $M = \mathrm{LM}(P^M)$, (ii) $N_\mathrm{h} = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$, and (iii) $L = \mathrm{LM}(Q^N)$. Let us define $J = \mathrm{LM}(\mathrm{EQT}(P,Q)^I)$ for more concise notation. It is clear that $J \models \mathrm{EQT}(P,Q)^I$ holds.

Claim 1: $J \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M)$.

($\supseteq$) Since $P^M \subseteq \mathrm{EQT}(P,Q)^I$ by Lemma 5.6, also $J \models P^M$ holds. Then $J \cap \mathrm{Hb}(P) \models P^M$ as P^M is based on $\mathrm{Hb}(P)$. Thus $\mathrm{LM}(P^M)$ is contained in $J \cap \mathrm{Hb}(P)$.

($\subseteq$) Now $\mathrm{LM}(P^M) \models P^M$ holds. Then define an interpretation $K = \mathrm{LM}(P^M) \cup \mathrm{Hb}_\mathrm{h}(Q)^\circ \cup \mathrm{Hb}(Q)^\bullet \cup \{c, d, e\}$ for which $K \models \mathrm{EQT}(P,Q)^I$ holds trivially by Lemma 5.6. Thus $J \subseteq K$ and $K \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M)$ imply $J \cap \mathrm{Hb}(P) \subseteq \mathrm{LM}(P^M)$.

Claim 2: If (i), then $J \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$.

Assuming (i) we obtain $M = I \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M) = J \cap \mathrm{Hb}(P)$ by Claim 1.

($\supseteq$) Let us assume that $J \not\models ((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$. In this respect, it is sufficient to consider only cases where weight rules and choice rules belong to the reduct.

- Suppose there is a weight rule $h \leftarrow w_1 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}, {\sim}B_\mathrm{h} = W_{B_\mathrm{h}}\} \in Q_\mathrm{h}/M_\mathrm{v}$ where $h \in \mathrm{Hb}_\mathrm{h}(Q)$ and $w_1 = \max(0, w - \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}}, {\sim}B_\mathrm{v} = W_{B_\mathrm{v}}))$ is obtained from $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\} \in Q$. Then the rule $h^\circ \leftarrow w_2 \leqslant \{A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}\}$ is in $((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$ and $w_2 = \max(0, w_1 - \mathrm{WS}_{N_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}}))$. Since this rule is falsified under J, we have $J \not\models h^\circ$ and $w_2 \leqslant \mathrm{WS}_J(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ})$. Using the definitions of w_2 and w_1, we obtain an inequality
$$\begin{aligned} w \leqslant\ & \mathrm{WS}_J(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}) + \mathrm{WS}_{N_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}}) + \\ & \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}}, {\sim}B_\mathrm{v} = W_{B_\mathrm{v}}). \end{aligned} \tag{A 2}$$
On the other hand, there is a rule $r = h^\circ \leftarrow w_3 \leqslant \{A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}, A_\mathrm{v} = W_{A_\mathrm{v}}\}$ with $w_3 = \max(0, w - \mathrm{WS}_N({\sim}B = W_B))$ in $\mathrm{EQT}(P,Q)^I$ by Lemma 5.6. Since $N = M_\mathrm{v} \cup N_\mathrm{h}$ by definition, we obtain $w_3 \leqslant \mathrm{WS}_J(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}) + \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}})$ from the definition of w_3 and (A 2). As $M = J \cap \mathrm{Hb}(P)$, we know that $M_\mathrm{v} = J \cap \mathrm{Hb}_\mathrm{v}(Q)$ and $w_3 \leqslant \mathrm{WS}_J(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}, A_\mathrm{v} = W_{A_\mathrm{v}})$. Thus $J \not\models r$ and $J \not\models \mathrm{EQT}(P,Q)^I$ which contradicts the choice of J in the beginning of this proof.
- Suppose there is a choice rule $\{H_\mathrm{h}\} \leftarrow A_\mathrm{h}, {\sim}B_\mathrm{h} \in Q_\mathrm{h}/M_\mathrm{v}$ so that $H_\mathrm{h} \neq \emptyset$ and $M_\mathrm{v} \models A_\mathrm{v} \cup {\sim}B_\mathrm{v}$ hold for a rule $\{H\} \leftarrow A, {\sim}B \in Q$. Consider any $h \in H_\mathrm{h}$. If $N_\mathrm{h} \models h$, $N_\mathrm{h} \models {\sim}B_\mathrm{h}$, and $M_\mathrm{v} \models {\sim}B_\mathrm{v}$, there is a rule $h^\circ \leftarrow A_\mathrm{h}^\circ$ included in $((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$. Assuming that this rule is falsified by J implies that $J \not\models h^\circ$ and $J \models A_\mathrm{h}^\circ$. Since $N = M_\mathrm{v} \cup N_\mathrm{h}$ by definition, we have $N \models {\sim}B$. Together with $N_\mathrm{h} \models h$, this implies that there is a rule $r = h^\circ \leftarrow A_\mathrm{h}^\circ, A_\mathrm{v}$ in $\mathrm{EQT}(P,Q)^I$ by Lemma 5.6. Since $M_\mathrm{v} = J \cap \mathrm{Hb}_\mathrm{v}(Q)$ as above, we obtain $J \models A_\mathrm{v}$ so that $J \not\models r$ and $J \not\models \mathrm{EQT}(P,Q)^I$. A contradiction regardless of the choice of h.

Thus $J \models ((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$ follows and $\mathrm{LM}(((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$ is necessarily contained in $J \cap \mathrm{Hb}(Q)^\circ$.

($\subseteq$) Define an interpretation $K = M \cup \mathrm{LM}(((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ \cup \mathrm{Hb}(Q)^\bullet \cup \{c,d,e\}$. Since (i) is assumed, it is clear that $K \models P^M$ but the satisfaction of rules addressed in Items 1–4 of Lemma 5.6 must be verified. A case analysis follows.

- Let us assume that there is a weight rule $r = h^\circ \leftarrow w_1 \leqslant \{A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}, A_\mathrm{v} = W_{A_\mathrm{v}}\} \in \mathrm{EQT}(P,Q)^I$ where $w_1 = \max(0, w - \mathrm{WS}_N({\sim}B = W_B))$ is associated with $h \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\} \in Q$ satisfying $h \in \mathrm{Hb}_\mathrm{h}(Q)$. By assuming $K \not\models r$, we obtain $K \not\models h^\circ$ and $w_1 \leqslant \mathrm{WS}_K(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}, A_\mathrm{v} = W_{A_\mathrm{v}})$. It follows that $w \leqslant \mathrm{WS}_K(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}, A_\mathrm{v} = W_{A_\mathrm{v}}) + \mathrm{WS}_N({\sim}B = W_B)$ by the definition of w_1. On the other hand, the hidden part $Q_\mathrm{h}/M_\mathrm{v}$ contains a weight rule $h \leftarrow w_2 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}, {\sim}B_\mathrm{h} = W_{B_\mathrm{h}}\}$ where $w_2 = \max(0, w - \mathrm{WS}_{M_\mathrm{v}}(A_\mathrm{v} = W_{A_\mathrm{v}}, {\sim}B_\mathrm{v} = W_{B_\mathrm{v}}))$. Thus the reduct $(Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$ contains a rule $r' = h \leftarrow w_3 \leqslant \{A_\mathrm{h} = W_{A_\mathrm{h}}\}$ where the limit $w_3 = \max(0, w_2 - \mathrm{WS}_{N_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}}))$. Using the definition of w_2, $N = M_\mathrm{v} \cup N_\mathrm{h}$, and K, we obtain $K \cap \mathrm{Hb}_\mathrm{v}(Q) = M_\mathrm{v}$; and from the preceding inequality concerning w, $w_2 \leqslant \mathrm{WS}_K(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ}) + \mathrm{WS}_{N_\mathrm{h}}({\sim}B_\mathrm{h} = W_{B_\mathrm{h}})$. Similarly, the definition of w_3, yields us $w_3 \leqslant \mathrm{WS}_K(A_\mathrm{h}^\circ = W_{A_\mathrm{h}^\circ})$. But then the definition of K implies that r' is not satisfied by $\mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$, a contradiction.
- Suppose there is a rule $r = h^\circ \leftarrow A_\mathrm{h}^\circ, A_\mathrm{v} \in \mathrm{EQT}(P,Q)^I$ associated with a choice rule $\{H\} \leftarrow A, {\sim}B \in Q$ such that $h \in H_\mathrm{h}$, $N_\mathrm{h} \models h$, and $N \models {\sim}B$. Assuming $K \not\models r$ implies $K \not\models h^\circ$, $K \models A_\mathrm{h}^\circ$, and $K \models A_\mathrm{v}$. Since $K \cap \mathrm{Hb}_\mathrm{v}(Q) = M_\mathrm{v}$ and $N =$

$M_\mathrm{v} \cup N_\mathrm{h}$ by definition, we know that $M_\mathrm{v} \models A_\mathrm{v} \cup {\sim}B_\mathrm{v}$. Since $H_\mathrm{h} \neq \emptyset$, it follows that $\{H_\mathrm{h}\} \leftarrow A_\mathrm{h}, {\sim}B_\mathrm{h}$ is included in $Q_\mathrm{h}/M_\mathrm{v}$. Moreover, the rule $r' = h \leftarrow A_\mathrm{h}$ belongs to $(Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$ as $N_\mathrm{h} \models {\sim}B_\mathrm{h}$ and $N_\mathrm{h} \models h$. But then the definition of K implies that r' is not satisfied by $\mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$, a contradiction.

The other rule types are covered by weight rules. It follows by the structure of $\mathrm{EQT}(P,Q)^I$ described in Lemma 5.6 that $K \models \mathrm{EQT}(P,Q)^I$. In particular the rules in Items 5–14 are trivially satisfied by K as their heads are. It follows that $J \subseteq K$ and $J \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ \subseteq \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$ as $K \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$.

Claim 3: If (i) and (ii), then $J \cap \mathrm{Hb}(Q)^\bullet = \mathrm{LM}(Q^N)^\bullet$.

Let us assume both (i) and (ii). It follows by Claims 1 and 2 that $M = I \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M) = J \cap \mathrm{Hb}(P)$ and $N_\mathrm{h}^\circ = I \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ = J \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ$.

($\supseteq$) Let us first establish $J \models (Q^N)^\bullet$. It is clear by Lemma 5.6 that almost all rules of $(Q^N)^\bullet$ are present in $\mathrm{EQT}(P,Q)^I$. The only exception concerns a rule $r = h^\bullet \leftarrow A^\bullet \cup \{h\}$ (resp. $r = h^\bullet \leftarrow A^\bullet \cup \{h^\circ\}$) included in $\mathrm{EQT}(P,Q)^I$ for a choice rule $\{H\} \leftarrow A, {\sim}B \in Q$ such that $h \in H_\mathrm{v}$ (resp. $h \in H_\mathrm{h}$) and $N \models {\sim}B$. Suppose that $J \not\models r'$ for the corresponding rule $r' = h^\bullet \leftarrow A^\bullet$ included in $(Q^N)^\bullet$ which presumes that $N \models h$. This implies $J \models h$ (resp. $J \models h^\circ$) as $N = M_\mathrm{v} \cup N_\mathrm{h}$ and $M = J \cap \mathrm{Hb}(P)$ (resp. $N_\mathrm{h}^\circ = J \cap \mathrm{Hb}_\mathrm{h}(Q)$). Thus $J \not\models r$, a contradiction. Hence $J \models (Q^N)^\bullet$ and $J \cap \mathrm{Hb}(Q)^\bullet \models (Q^N)^\bullet$.

($\subseteq$) Let us then define an interpretation $K = \mathrm{LM}(P^M) \cup \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ \cup \mathrm{LM}(Q^N)^\bullet \cup \{c,d,e\}$. It can be shown as in Claim 2 that $K \models P^M$ and the rules mentioned in Items 1–4 of Lemma 5.6 are satisfied by K. As noted already, most of the rules of $(Q^N)^\bullet$ are included in $\mathrm{EQT}(P,Q)^I$ as such and thus satisfied by the definition of K as $\mathrm{LM}(Q^N)^\bullet \models (Q^N)^\bullet$. The only exceptions are made by rules r of the forms defined above. Suppose that $K \not\models r$ and define $r' = h^\bullet \leftarrow A^\bullet$. It follows that $K \not\models r'$ and $h \in \mathrm{LM}(P^M)$ (resp. $h \in \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$). Then $M = \mathrm{LM}(P^M)$ (resp. $N_\mathrm{h} = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$) implies $N \models h$ so that $r' \in (Q^N)^\bullet$. Thus $N \models r'$ by the definition of N, a contradiction. Finally, the rules in Items 9–14 of Lemma 5.6 are satisfied by K as $K \models \{c,d,e\}$. Thus $K \models \mathrm{EQT}(P,Q)^I$. Since $K \cap \mathrm{Hb}(Q)^\bullet = \mathrm{LM}(Q^N)^\bullet$, we obtain $J \cap \mathrm{Hb}(Q)^\bullet \subseteq \mathrm{LM}(Q^N)^\bullet$.

Claim 4: If (i), (ii), (iii), and $A = J \cap \{c,d,e\}$, then (a) $d \in A \iff N \neq L$, (b) $c \in A \iff d \notin I$ and $L \not\models \mathrm{CompS}(Q)$, and (c) $e \in A \iff c \in A$ or $d \in A$.

Assume (i), (ii), and (iii). Using Claims 1–3, we obtain $M = I \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M) = J \cap \mathrm{Hb}(P)$, $N_\mathrm{h}^\circ = I \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ = J \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ$, and $L^\bullet = I \cap \mathrm{Hb}(Q)^\bullet = \mathrm{LM}(Q^N)^\bullet = J \cap \mathrm{Hb}(Q)^\bullet$.

(a) The structure of $\mathrm{EQT}(P,Q)^I$ made explicit in Lemma 5.6 and the properties of $\mathrm{LM}(\mathrm{EQT}(P,Q)^I)$ imply that $d \in A \iff$ there is an atom $a \in \mathrm{Hb}(Q)$ such that $L \not\models a$ and $N \models a$; or $N \not\models a$ and $L \models a$. But this is equivalent to $N \neq L$.

(b) The same premises imply that $c \in A \iff c \in J \iff I \not\models d$; and there is $a \in \mathrm{CompS}(Q)$ such that $L \not\models a$ or or there $\sim b \in \mathrm{CompS}(Q)$ such that $L \models b$. Or equivalently, $d \notin I$ and $L \not\models \mathrm{CompS}(Q)$.

(c) Finally, we have $e \in A \iff J \models e \iff J \models c$ or $J \models d \iff c \in A$ or $d \in A$.

□

Proof of Theorem 5.8

($\Longrightarrow$) Suppose that $\mathrm{EQT}(P,Q)$ has a stable model K, i.e. $K = \mathrm{LM}(\mathrm{EQT}(P,Q)^K)$ and $K \models \mathrm{CompS}(\mathrm{EQT}(P,Q))$. Let us then extract three interpretations from K: $M = K \cap \mathrm{Hb}(P)$, $N = M_\mathrm{v} \cup N_\mathrm{h}$ where $N_\mathrm{h} = \{a \in \mathrm{Hb}_\mathrm{h}(Q) \mid a^\circ \in K\}$, and $L = \{a \in \mathrm{Hb}(Q) \mid a^\bullet \in K\}$. It follows that $M = K \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M)$ by Claim 1 in Proposition 5.7. Besides, we have $M \models \mathrm{CompS}(P)$ as $K \models \mathrm{CompS}(\mathrm{EQT}(P,Q))$ and $\mathrm{CompS}(P) \subseteq \mathrm{CompS}(\mathrm{EQT}(P,Q))$. Thus $M \in \mathrm{SM}(P)$.

We may now apply Claim 2 in Proposition 5.7 since condition (i) is satisfied. Thus $N_\mathrm{h}^\circ = K \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ$ which makes condition (ii) true in Proposition 5.7 so that $N_\mathrm{h} \in \mathrm{SM}(Q_\mathrm{h}/M_\mathrm{v})$ is the case.

This enables the use of Claim 3 in Proposition 5.7 to obtain $L^\bullet = K \cap \mathrm{Hb}(Q)^\bullet = \mathrm{LM}(Q^N)^\bullet$. Thus $L = \mathrm{LM}(Q^N)$ and condition (iii) in Proposition 5.7 is satisfied.

On the other hand, $e \in A$ holds for $A = K \cap \{c,d,e\}$ as $K \models \mathrm{CompS}(\mathrm{EQT}(P,Q))$ and $e \in \mathrm{CompS}(\mathrm{EQT}(P,Q))$ by Definition 5.1. It follows by (c) and (b) in Claim 4 of Proposition 5.7 that $c \in A$ or $d \in A$, i.e. $d \notin A$ and $L \not\models \mathrm{CompS}(Q)$; or $d \in A$. Using (a) we obtain $N = L$ and $L \not\models \mathrm{CompS}(Q)$; or $N \neq L$. By substituting $\mathrm{LM}(Q^N)$ for L and N for L, we have $N = \mathrm{LM}(Q^N)$ and $N \not\models \mathrm{CompS}(Q)$; or $N \neq \mathrm{LM}(Q^N)$. Since Q has enough visible atoms, we know that N_h is unique with respect to Q and M_v, and there is no $N \in \mathrm{SM}(Q)$ such that $N_\mathrm{v} = M_\mathrm{v}$.

($\Longleftarrow$) Suppose that P has a stable model $M = \mathrm{LM}(P^M)$ and there is no $N \in \mathrm{SM}(Q)$ such that $N_\mathrm{v} = M_\mathrm{v}$. Since Q has enough visible atoms any such candidate N must be based on the unique stable model $N_\mathrm{h} = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$. So let us define $N = M_\mathrm{v} \cup N_\mathrm{h}$. The instability of N implies either $N \neq \mathrm{LM}(Q^N)$; or $N = \mathrm{LM}(Q^N)$ and $N \not\models \mathrm{CompS}(Q)$. In either case, let $L = \mathrm{LM}(Q^N)$. Moreover, let $A \subseteq \{c,d,e\}$ be a set of atoms so that $d \in A \iff N \neq \mathrm{LM}(Q^N)$, $c \in A \iff N = \mathrm{LM}(Q^N)$ and $N \not\models \mathrm{CompS}(Q)$, and $e \in A$ unconditionally.

Let us then define an interpretation $K = M \cup N_\mathrm{h}^\circ \cup L^\bullet \cup A$. It is easy to see that $K \models \mathrm{CompS}(\mathrm{EQT}(P,Q))$ as $M \models \mathrm{CompS}(P)$ and $K \models e$ by definition. It remains to establish that $K = \mathrm{LM}(\mathrm{EQT}(P,Q)^K)$. First, the definition of K implies that $K \cap \mathrm{Hb}(P) = M$. It follows by Claim 1 in Proposition 5.7 that $\mathrm{LM}(\mathrm{EQT}(P,Q)^K) \cap \mathrm{Hb}(P) = \mathrm{LM}(P^M) = M$. Second, we have $K \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = N_\mathrm{h}^\circ$ by definition. Using Claim 2 in Proposition 5.7 we obtain $\mathrm{LM}(\mathrm{EQT}(P,Q)^K) \cap \mathrm{Hb}_\mathrm{h}(Q)^\circ = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})^\circ = N_\mathrm{h}^\circ$. Third, we defined K so that $K \cap \mathrm{Hb}(Q)^\bullet = L^\bullet$. It follows by Proposition 5.7 (Claim 3) that $\mathrm{LM}(\mathrm{EQT}(P,Q)^K) \cap \mathrm{Hb}(Q)^\bullet = \mathrm{LM}(Q^N)^\bullet = L^\bullet$. Finally, we recall that $K \cap \{c,d,e\} = A$. It follows by Claim 4 in Proposition 5.7 that (a) $d \in \mathrm{LM}(\mathrm{EQT}(P,Q)^K) \iff N \neq L \iff N \neq \mathrm{LM}(Q^N) \iff d \in A$ by the definition of A above; (b) $c \in \mathrm{LM}(\mathrm{EQT}(P,Q)^K) \iff d \notin K$ and $L \not\models \mathrm{CompS}(Q) \iff d \notin A$ and $L \not\models \mathrm{CompS}(Q) \iff N = L$ and $N \not\models \mathrm{CompS}(Q) \iff N = \mathrm{LM}(Q^N)$ and $N \not\models \mathrm{CompS}(Q) \iff$

$c \in A$; and (c) $e \in \mathrm{LM}(\mathrm{EQT}(P,Q)^K)$ holds as the instability of N implies either $d \in A$ or $c \in A$. Thus $\mathrm{LM}(\mathrm{EQT}(P,Q)^K) \cap \{c,d,e\} = A$. To summarize, we have established $\mathrm{LM}(\mathrm{EQT}(P,Q)^K) = M \cup N^{\circ}_{\mathrm{h}} \cup L^{\bullet} \cup A = K$. Thus $K \in \mathrm{SM}(\mathrm{EQT}(P,Q))$. □

Proof of Proposition 6.7

Let $M \subseteq \mathrm{Hb}(P)$ be any interpretation for P and P_w. We rewrite (12) using shorthands as $0 \leqslant \{H = \mathbf{1}\} \leftarrow |A| + |B| \leqslant \{A = \mathbf{1}, {\sim}B = \mathbf{1}\}$ where $\mathbf{1}$s are sets of weights of appropriate sizes consisting of only 1s. As regards the respective choice rule $\{H\} \leftarrow A, {\sim}B$ and any $h \in H$, Definition 3.4 implies that $h \leftarrow A$ belongs to $P^M \iff M \models h$ and $M \models {\sim}B$. On the other hand, Definition 6.5 implies $h \leftarrow (|A| + |B| - \mathrm{WS}_M({\sim}B = \mathbf{1})) \leqslant \{A = \mathbf{1}\} \in P^M_w \iff M \models h$. Quite similarly, we use $1 \leqslant \{h = 1\} \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ as an abbreviation for (13). Then the reduced rule $h \leftarrow w' \leqslant \{A = W_A\}$ where $w' = \max(0, w - \mathrm{WS}_M({\sim}B = W_B))$ belongs to P^M unconditionally and to $P^M_w \iff M \models h$.

($\implies$) Suppose that $M = \mathrm{LM}(P^M)$. It follows immediately that $M \models P^M$ and $M \models P$. Since choice rules and their translations (12) do not interfere with the satisfaction of rules, we conclude $M \models P_w$ by the close relationship of (4) and (13). Moreover, it is easy to see that $\mathrm{LM}(P^M_w) \subseteq M$ as the analysis above shows that the head atom h of every rule included in P^M_w is necessarily true in M, i.e., $h \in M$.

It remains to prove by induction that each interpretation in a sequence defined by $M_0 = \emptyset$ and $M_i = \mathrm{T}_{P^M}(M_{i-1})$ for $i > 0$ is contained in $\mathrm{LM}(P^M_w)$. Note that $M_i \subseteq M$ for each $i \geqslant 0$ and $M = \mathrm{lfp}(\mathrm{T}_{P^M}) = M_i$ for some finite i due to compactness of T_{P^M}. Let us then consider any $h \in M_i$. Note that $h \in M$ holds, i.e., $M \models h$. The definition of M_i implies that (i) there is a rule $h \leftarrow A \in P^M$ such that $M \models {\sim}B$ and $A \subseteq M_{i-1}$; *or* (ii) there is a rule $h \leftarrow w' \leqslant \{A = W_A\} \in P^M$ with $w' \leqslant \mathrm{WS}_{M_{i-1}}(A = W_A)$. If (i) holds, the rule $h \leftarrow |A| \leqslant \{A = \mathbf{1}\}$ belongs to P^M_w as $M \models h$. Moreover, $A \subseteq M_{i-1} \subseteq \mathrm{LM}(P^M_w)$ by induction hypothesis. In case of (ii), $M \models h$ implies that the reduced rule is also in P^M_w. Since $M_{i-1} \subseteq \mathrm{LM}(P^M_w)$, we obtain $w' \leqslant \mathrm{WS}_{\mathrm{LM}(P^M_w)}(A = W_A)$. Thus $h \in \mathrm{LM}(P^M_w)$ results in both cases so that $M_i \subseteq \mathrm{LM}(P^M_w)$ for each M_i and M in particular so that $M = \mathrm{LM}(P^M_w)$.

($\impliedby$) Let us then assume that $M \models P_w$ and $M = \mathrm{LM}(P^M_w)$ as well as $M \not\models P^M$. The last cannot be caused by a choice rule because $h \leftarrow A$ is included in P^M only if $M \models h$. If a weight rule is the reason, then $h \leftarrow w' \leqslant \{A = W_A\}$ with $w' = \max(0, w - \mathrm{WS}_M({\sim}B = W_B))$ belongs to P^M, $w' \leqslant \mathrm{WS}_M(A = W_A)$, and $M \not\models h$. By adding $\mathrm{WS}_M({\sim}B = W_B)$ on both sides of the inequality, we obtain $w \leqslant \mathrm{WS}_M(A = W_A, {\sim}B = W_B)$. Thus a rule $1 \leqslant \{h = 1\} \leftarrow w \leqslant \{A = W_A, {\sim}B = W_B\}$ of P_w is not satisfied by M, a contradiction. Hence $M \models P^M$.

Now $M \models P^M$ implies $\mathrm{LM}(P^M) \subseteq M$ and we need induction to establish inclusion in the other direction. This time we use a sequence defined by $M_0 = \emptyset$ and $M_i = \mathrm{T}_{P^M_w}(M_{i-1})$ for $i > 0$. Then consider any $h \in M_i$. Since M is the limit of the sequence, we obtain $h \in M$ and $M \models h$. Moreover, the definition of M_i implies that (iii) there is a rule $h \leftarrow w'' \leqslant \{A = \mathbf{1}\} \in P^M_w$ where $w'' = |A| + |B| - \mathrm{WS}_M({\sim}B = \mathbf{1}) \leqslant \mathrm{WS}_{M_{i-1}}(A = \mathbf{1})$; or (iv) there is a rule $h \leftarrow w' \leqslant \{A = W_A\} \in P^M_w$ such that $w' \leqslant \mathrm{WS}_{M_{i-1}}(A = W_A)$. In case of (iii), we infer $\mathrm{WS}_M({\sim}B = \mathbf{1}) = |B|$ and $\mathrm{WS}_{M_{i-1}}(A = \mathbf{1}) = |A|$ as necessities so that $M \models {\sim}B$ and $A \subseteq M_{i-1}$ follow. Thus

$h \leftarrow A \in P^M$ as $M \models h$ and $\mathrm{LM}(P^M) \models A$ follows by the induction hypothesis $M_{i-1} \subseteq \mathrm{LM}(P^M)$. If (iv) holds, the reduced rule is also a member of P^M by definition. Using the induction hypothesis again, we obtain $w' \leqslant \mathrm{WS}_{\mathrm{LM}(P^M)}(A = W_A)$. To conclude the preceding case analysis, we have $h \in \mathrm{LM}(P^M)$ for any $h \in M_i$ and thus $M_i \subseteq \mathrm{LM}(P^M)$. Since $M = M_i$ for some i, we obtain $M \subseteq \mathrm{LM}(P^M)$. □

Proof of Theorem 6.10

Consider any weight constraint program P. Now $P \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{SNS}}(P)$ holds by the definition of $\equiv_{\mathrm{v}}$ if and only if $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}(P))$ and there is a bijection $\mathrm{Ext} : \mathrm{SM}(P) \rightarrow \mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ such that for all $M \in \mathrm{SM}(P)$ it holds that $M \cap \mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Ext}(M) \cap \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{SNS}}(P))$. Since $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ holds by Definition 6.9, it remains to to establish such a bijection Ext from $\mathrm{SM}(P)$ to $\mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$.

Given an interpretation $M \subseteq \mathrm{Hb}(P)$, we define $\mathrm{Ext}(M) = M \cup \mathrm{SU}_P(M)$ where $\mathrm{SU}_P(M)$ satisfies for each weight constraint $C = l \leqslant \{A = W_A, {\sim}B = W_B\} \leqslant u$ appearing in P that

1. $\mathrm{sat}(C) \in \mathrm{SU}_P(M) \iff l \leqslant \mathrm{WS}_M(A = W_A, {\sim}B = W_B)$, and
2. $\mathrm{unsat}(C) \in \mathrm{SU}_P(M) \iff u + 1 \leqslant \mathrm{WS}_M(A = W_A, {\sim}B = W_B)$.

Now, if $M \in \mathrm{SM}(P)$, then $N = \mathrm{Ext}(M) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ follows by the results of Simons et al. (2002). Thus Ext is indeed a function from $\mathrm{SM}(P)$ to $\mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ and it remains to establish that Ext is a bijection. It is clearly injective as $M_1 \neq M_2$ implies $\mathrm{Ext}(M_1) \neq \mathrm{Ext}(M_2)$ by the definition of Ext.

To prove that Ext is also a surjection, let us consider any $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ and the respective projection $M = N \cap \mathrm{Hb}(P)$. Since $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$, it holds that $N \models \mathrm{Tr}_{\mathrm{SNS}}(P)$ and moreover $M \in \mathrm{SM}(P)$ holds by the results of Simons et al. (2002). Thus we need to show $N = N'$ for $N' = \mathrm{Ext}(M) = M \cup \mathrm{SU}_P(M)$. Since $\mathrm{Ext} : \mathrm{SM}(P) \rightarrow \mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$ we know that $N' \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{SNS}}(P))$.

Let us show that $\mathrm{SU}_P(M) \subseteq N$. Assuming the opposite there is an atom $a \in \mathrm{SU}_P(M)$ such that $a \notin N$. By the definition of $\mathrm{SU}_P(M)$ either (i) $a = \mathrm{sat}(C)$ or (ii) $a = \mathrm{unsat}(C)$ for some $C = l \leqslant \{A = W_A, {\sim}B = W_B\} \leqslant u$ appearing in P. This leads to a case analysis as follows.

(i) If $a = \mathrm{sat}(C) \in \mathrm{SU}_P(M)$, then there is a rule (14) in $\mathrm{Tr}_{\mathrm{SNS}}(P)$ such that $l \leqslant \mathrm{WS}_M(A = W_A, {\sim}B = W_B) = \mathrm{WS}_N(A = W_A, {\sim}B = W_B)$ where last equality holds by the definition of M as $A \subseteq \mathrm{Hb}(P)$ and $B \subseteq \mathrm{Hb}(P)$. Since $\mathrm{sat}(C) \notin N$, it follows that (14) is not satisfied by N. But this contradicts $N \models \mathrm{Tr}_{\mathrm{SNS}}(P)$.

(ii) Quite similarly, if $a = \mathrm{unsat}(c) \in \mathrm{SU}_P(M)$, then there is a rule (15) such that $u + 1 \leqslant \mathrm{WS}_M(A = W_A, {\sim}B = W_B) = \mathrm{WS}_N(A = W_A, {\sim}B = W_B)$. Then (15) is not satisfied by N as $\mathrm{unsat}(c) \notin N$. A contradiction with $N \models \mathrm{Tr}_{\mathrm{SNS}}(P)$.

Hence $\mathrm{SU}_P(M) \subseteq N$ is necessarily the case. Since $M \subseteq N$ by definition, we have $N' \subseteq N$. How about the converse inclusion $N \subseteq N' = M \cup \mathrm{SU}_P(M)$? It is clear that $N \cap \mathrm{Hb}(P) = M \subseteq N'$. Then a potential difference $N' \setminus N$ (if any) must be caused by new atoms involved in $\mathrm{Tr}_{\mathrm{SNS}}(P)$. There are three kinds of such atoms.

1. Suppose that $\mathrm{sat}(C) \in N$ for some $C = l \leqslant \{A = W_A, {\sim}B = W_B\} \leqslant u$ appearing in P. Since N is a stable model of $\mathrm{Tr}_{\mathrm{SNS}}(P)$ and there is only one rule (14) in

$\mathrm{Tr}_{\mathrm{SNS}}(P)$ having $\mathrm{sat}(C)$ as its head, the body of that rule must be satisfied in N, too, i.e., $l \leqslant \mathrm{WS}_N(A = W_A, {\sim}B = W_B)$. Since $M = N \cap \mathrm{Hb}(P)$, $A \subseteq \mathrm{Hb}(P)$, and $B \subseteq \mathrm{Hb}(P)$, the same holds for M. Thus $\mathrm{sat}(C) \in \mathrm{SU}_P(M)$.

2. Using the same line of reasoning and the rule (15) included in $\mathrm{Tr}_{\mathrm{SNS}}(P)$, we know that $\mathrm{unsat}(C) \in N$ implies $\mathrm{unsat}(C) \in \mathrm{SU}_P(M)$.
3. Now $f \notin N$ must hold as N is a stable model of $\mathrm{Tr}_{\mathrm{SNS}}(P)$ which includes (19).

To conclude, we have established $N \subseteq N'$ which indicates that there is $M \in \mathrm{SM}(P)$ such that $N = \mathrm{Ext}(M)$. Therefore Ext is bijective and $\mathrm{Tr}_{\mathrm{SNS}}$ faithful. □

References

Anger, C., Gebser, M., Linke, T., Neumann, A. and Schaub, T. 2005. The nomore++ system. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. Springer-Verlag, pp. 422–426. LNAI 3662.

Apt, K., Blair, H. and Walker, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, pp. 89–148.

Balduccini, M., Barry, M., Gelfond, M., Nogueira, M. and Watson, R. 2001. An A-Prolog decision support system for the space shuttle. In *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages*. Springer-Verlag, pp. 169–183.

Cholewinski, P. and Truszczyński, M. 1999. Extremal problems in logic programming and stable model computation. *Journal of Logic Programming 38,* 2, 219–242.

Dovier, A., Formisano, A. and Pontelli, E. 2005. A comparison of CLP(FD) and ASP solutions to NP-complete problems. In *Proceedings of Convegno Italiano di Logica Computazionale*. Italian Association for Logic Programming (GULP), Rome, Italy. Available at `http://www.disp.uniroma2.it/CILC2005/`.

Dowling, W. and Gallier, J. 1984. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming 3*, 267–284.

Eiter, T., Fink, M., Tompits, H. and Woltran, S. 2004. Simplifying logic programs under uniform and strong equivalence. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, V. Lifschitz and I. Niemelä, Eds. Springer-Verlag, pp. 87–99. LNAI 2923.

Eiter, T., Tompits, H. and Woltran, S. 2005. On solution correspondences in answer-set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05*. Professional Book Center, Edinburgh, Scotland, UK, pp. 97–102.

Eiter, T. and Wang, K. 2006. Forgetting and conflict resolving in disjunctive logic programming. In *Proceedings of the 11th International Workshop on Nonmonotonic Reasoning*, J. Dix and A. Hunter, Eds. University of Clausthal, Department of Informatics, Technical Report, IfI-06-04, Lake District, UK, pp. 85–91. Available at `http://www.in.tu-clausthal.de/forschung/technical-reports/`.

Gelfond, M. and Leone, N. 2002. Logic programming and knowledge representation – the A-Prolog perspective. *Artificial Intelligence 138*, 3–38.

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

Gelfond, M. and Lifschitz, V. 1990. Logic programs with classical negation. In *Proceedings of the 7th International Conference on Logic Programming*. The MIT Press, pp. 579–597.

Gressmann, J., Janhunen, T., Mercer, R. Schaub, T., Tichy, R. and Thiele, S. 2005. Platypus: A platform for distributed answer set solving. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. Springer-Verlag, pp. 227–239. LNAI 3662.

Janhunen, T. 2003. Translatability and intranslatability results for certain classes of logic programs. Series A: Research report 82, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland. November. Available at `http://www.tcs.hut.fi/Publications/series-a.shtml`.

Janhunen, T. 2004. Representing normal programs with clauses. In *Proceedings of the 16th European Conference on Artificial Intelligence*, R. L. de Mántaras and L. Saitta, Eds. IOS Press, pp. 358–362.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics 16,* 1–2 (June), 35–86.

Janhunen, T., Niemelä, I., Simons, P. and You, J.-H. 2000. Unfolding partiality and disjunctions in stable model semantics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 7th International Conference*, A. Cohn, F. Giunchiglia, and B. Selman, Eds. Morgan Kaufmann, Breckenridge, Colorado, 411–419.

Janhunen, T., Niemelä, I., Seipel, D., Simons, P. and You, J.-H. 2006. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic 7,* 1, 1–37.

Janhunen, T. and Oikarinen, E. 2002. Testing the equivalence of logic programs under stable model semantics. In *Logics in Artificial Intelligence, Proceedings of the 8th European Conference*, S. Flesca et al., Eds. Springer-Verlag, pp. 493–504. LNAI 2424.

Janhunen, T. and Oikarinen, E. 2004. lpeq and dlpeq – translators for automated equivalence testing of logic programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, V. Lifschitz and I. Niemelä, Eds. Springer-Verlag, pp. 336–340. LNAI 2923.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*. 7, 3, 499–562.

Lierler, Y. and Maratea, M. 2004. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer-Verlag, pp. 346–350. LNAI 2923.

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

Lin, F. 2002. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference*, D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, Eds. Morgan Kaufmann, pp. 170–176.

Lin, F. and Zhao, Y. 2002. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of the 18th National Conference on Artificial Intelligence*. AAAI, AAAI Press, pp. 112–117.

Liu, L. and Truszczyński, M. 2005. Pbmodels software to compute stable models by pseudoboolean solvers. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning*, C. Baral, G. Greco, N. Leone, and G. Terracina, Eds. Springer-Verlag, pp. 410–415. LNAI 3662.

Lloyd, J. 1987. *Foundations of Logic Programming*. Springer-Verlag, Berlin.

Marek, W. and Truszczyński, M. 1991. Autoepistemic logic. *Journal of the ACM 38*, 588–619.

Marek, W. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*. Springer-Verlag, Berlin, 375–398.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 241–273.

Oikarinen, E. and Janhunen, T. 2004. Verifying the equivalence of logic programs in the disjunctive case. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, V. Lifschitz and I. Niemelä, Eds. Springer-Verlag, pp. 180–193. LNAI 2923.

Oikarinen, E. and Janhunen, T. 2006. Modular equivalence for normal logic programs. In *Proceedings of the 17th European Conference on Artificial Intelligence*. Riva del Garda, Italy, pp. 412–416.

Pearce, D., Tompits, H. and Woltran, S. 2001. Encodings for equilibirium logic and logic programs with nested expressions. In *Proceedings of the 10th Portuguese Conference on Artificial Intelligence*, P. Brazdil and A. Jorge, Eds. Springer-Verlag, pp. 306–320. LNAI 2258.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial Intelligence 82*, 273–302.

Simons, P. 1999. Extending the stable model semantics with more expressive rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer-Verlag, pp. 305–316. LNAI 1730.

Simons, P., Niemelä, I. and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138,* 1–2, 181–234.

Soininen, T., Niemelä, I., Tiihonen, J. and Sulonen, R. 2001. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming*. AAAI Press, pp. 195–201.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theoretical Computer Science 3*, 1–22.

Syrjänen, T. 2001. Lparse 1.0 user's manual. Available at `http://www.tcs.hut.fi/Software/smodels`.

Syrjänen, T. 2004. Cardinality constraint programs. In *Logics in Artificial Intelligence: 9th European Conference*, J. Alferes and J. Leite, Eds. Springer-Verlag, pp. 187–200. LNAI 3229.

Syrjänen, T. and Niemelä, I. 2001. The Smodels system. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, T. Eiter, W. Faber, and M. Truszczyński, Eds. Springer-Verlag, pp. 434–438. LNAI 2173.

Turner, H. 2003. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming 3,* 4–5, 609–622.