Verifying the Equivalence of Logic Programs in the Disjunctive Case*

Emilia Oikarinen and Tomi Janhunen

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O. Box 5400, FIN-02015 HUT, Finland
`Emilia.Oikarinen@hut.fi, Tomi.Janhunen@hut.fi`

Abstract. In this paper, we address the problem of checking whether two disjunctive logic programs possess exactly the same stable models. An existing translation-based method [14], which was designed for weight constraint programs supported by the SMODELS system, is generalized to the disjunctive case. Moreover, we report on our preliminary experiments with an implementation of the method, a translator called DLPEQ.

1 Introduction

Answer set programming (ASP) is a logic programming paradigm [21, 22] where a problem at hand is solved in a declarative fashion 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 a special purpose search engine. There is a growing interest towards ASP and much of that is due to efficient search engines such as DLV [15], SMODELS [28], and GNT [12, 13] available today. Consequently, a variety of interesting applications of ASP has emerged. Let us just mention a decision support system of the space shuttle [1] as a fine example.

The standard syntax of logic programs is that of *normal* or *general* logic programs [19], which admits the use of Clark's *negation as failure* principle in the bodies of rules. A number of generalizations to the basic syntax have been proposed: classical negation [6], disjunction in the heads of rules [7, 27], default negation in the heads of disjunctive rules [9], nested rules [17], and weight rules [28]. The semantics of the resulting classes of logic programs is determined by the respective generalizations of the *stable model semantics* [5] which has settled as a leading semantics in ASP. According to [6], the *answer sets* mentioned above are consistent sets of classical literals that are stable in the same sense [5] as stable models are. In the sequel, we concentrate on disjunctive logic programs, which cover the kinds of programs mentioned above; either as a proper generalization or via suitable translations [4, 6, 9, 10, 25]. Thus a wide variety of logic programs is implicitly under consideration in this paper.

* The research reported in this paper is partially funded by the Academy of Finland under the project "Applications of Rule-Based Constraint Programming" (#53695).

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 improved programs for a particular problem, e.g., when optimizing the execution time and space. As a consequence, the programmer needs to ensure that subsequent programs which differ in performance yield the same output. This leads us to the problem of checking whether given two logic programs P and Q give rise exactly to the same answer sets. If this is the case, then P and Q are called *(weakly) equivalent*, denoted by $P \equiv Q$. Note that such a notion of equivalence can be considered as a sufficient one from the programming perspective, as answer sets capture the solutions of the problem being solved.

In [14], we develop a method for testing the (weak) equivalence of *weight constraint programs* supported by the SMODELS system [28]. The key idea in this 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 SMODELS can be used for the search of counter-examples and there is no need to develop a special purpose search engine for equivalence testing. Only a translator like LPEQ [11] needs to be implemented. The aim of the current paper is to generalize the translation-based method to the disjunctive case so that search engines dedicated to disjunctive logic programs such as GNT [12] can be used for actual computations.

There are also other notions of equivalence that have been proposed for logic programs. Lifschitz et al. [16] consider P and Q *strongly equivalent*, denoted $P \equiv_{\mathrm{s}} Q$, iff P and Q yield exactly the same answer sets in any *context*, i.e. as parts of another program R. The resulting equivalence relation is fairly strong and better applicable to *subprograms* or *program modules* constituting larger programs rather than complete programs. Note that $P \equiv_{\mathrm{s}} Q$ implies $P \equiv Q$ by setting $R = \emptyset$ above, but the converse implication 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 $P \equiv_{\mathrm{s}} Q$. This view is supported by the complexity results: deciding $\equiv$ for finite propositional disjunctive programs $\Pi_2^{\mathbf{P}}$-hard [29] whereas deciding $\equiv_{\mathrm{s}}$ is only **coNP**-complete [26, 18]. Thus, unless the polynomial hierarchy collapses, it is intractable in general to establish $P \equiv Q$ by choosing certain subprograms P' and Q' of P and Q, respectively, and verifying $P' \equiv_{\mathrm{s}} Q'$. Moreover, the difference in computational time complexity suggests that weak equivalence is better suited for disjunctive solvers, which are able to decide properties in $\Sigma_2^{\mathbf{P}}$, whereas SAT solvers and SMODELS suffice for the verification of strong equivalence [18]. For the reasons above, we concentrate on complete programs and the verification of weak equivalence in this paper. Quite recently, Eiter and Fink [2] address yet another notion of equiv-

[1] Turner [29] develops an analogous transformation for weight constraint programs, but for another notion of equivalence, namely *strong equivalence* discussed below.

alence, namely *uniform equivalence* $\equiv_u$, which is a variant of $\equiv_s$ based on sets of facts as possible contexts. However, the problem of deciding $\equiv_u$ is $\Pi_2^{\mathbf{P}}$-complete, suggesting another generalization of the translation-based method.

We proceed as follows. In Section 2, we briefly review the syntax and semantics of disjunctive logic programs. The formal definitions of the equivalence relations for disjunctive logic programs are given in Section 3. Moreover, we discuss the computational cost of the equivalence testing problem. Then, we are ready to present two alternative translation-based techniques for testing the equivalence of disjunctive programs in Section 4. We also address the correctness of the translations. Section 5 shows some results from our preliminary experiments with a prototype implementation called DLPEQ [23] and a solver GNT [12] for disjunctive programs. Section 6 concludes the paper.

2 Disjunctive Logic Programs

We use the symbol "$\sim$" to denote *negation as failure to prove* or *default negation* to distinguish it from classical negation $\neg$. *Default literals* are either atoms a or their default negations $\sim a$. Given a set of atoms A, we define a set of negative literals $\sim A = \{\sim a \mid a \in A\}$. A propositional *disjunctive logic program* (DLP)[2] P is a set of *rules* which are expressions of the form

$$a_1 \mid \ldots \mid a_n \leftarrow b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_k, \tag{1}$$

where $a_1, \ldots, a_n$, $b_1, \ldots, b_m$, and $c_1, \ldots, c_k$ are propositional atoms and n, m and k are nonnegative integers. The *head* of the rule $a_1 \mid \ldots \mid a_n$ is interpreted disjunctively while the rest forming the *body* of the rule is interpreted conjunctively. Intuitively, the rules of the form (1) are used as inference rules: any of the head atoms $a_1, \ldots, a_n$ can be inferred given that the positive body atoms $b_1, \ldots, b_m$ can be inferred and the negative body atoms $c_1, \ldots, c_k$ cannot. Since the order of atoms is not significant in a rule (1) we use a shorthand $A \leftarrow B, \sim C$ where A, B and C are the respective sets of atoms. Rules are separated with full stops and we drop the symbol "$\leftarrow$" if $m = k = 0$. An empty head is denoted by "$\bot$" and a rule with an empty head is called an *integrity constraint*. A program P is *disjunction-free* or a *normal program* if $n = 1$ for each rule of P. Similarly, each rule (1) of a *positive program* P is supposed to satisfy $k = 0$.

Let us then turn our attention to the semantics of a DLP P. The *Herbrand base* $\mathrm{Hb}(P)$ is defined as the set of atoms appearing in P. An *interpretation* $I \subseteq \mathrm{Hb}(P)$ of P defines which atoms $a \in \mathrm{Hb}(P)$ are true ($a \in I$) and which are false ($a \notin I$). An interpretation I is a (classical) *model* of P, denoted by $I \models P$, if and only if for every rule $A \leftarrow B, \sim C$ of P, $B \subseteq I$ and $C \cap I = \emptyset$ imply $A \cap I \neq \emptyset$. For a *minimal model* M of P, there is no other interpretation $M' \subset M$ such that $M' \models P$. The set of minimal models of P is denoted by $\mathbf{MM}(P)$. If P is positive, then $\mathbf{MM}(P)$ gives the standard minimal model semantics of P. If P is also normal, then P has a unique minimal model, i.e. $|\mathbf{MM}(P)| = 1$ [19].

[2] DLPs with variables can be covered by performing a Herbrand instantiation.

However, minimal models do not properly capture the intuitions behind DLPs involving default negation. E.g., $\mathbf{MM}(P_1) = \mathbf{MM}(P_2) = \{\{a\},\{b\}\}$ for $P_1 = \{a \mid b\}$ and $P_2 = \{a \leftarrow \sim b\}$, although only the first model seems intuitive for P_2 lacking a rule for b. The problem can be resolved by the notion of a *stable model* first proposed for normal programs [5] and then generalized for DLPs [27, 7].

Definition 1. *Given a DLP P and an interpretation $M \subseteq \mathrm{Hb}(P)$, the Gelfond-Lifschitz reduct of P is $P_M = \{A \leftarrow B \mid A \leftarrow B, \sim C \in P$ and $M \cap C = \emptyset\}$ and M is a stable model of P if and only if $M \in \mathbf{MM}(P_M)$.*

3 Equivalence Testing

Lifschitz et al. [16] recently introduced two notions of equivalence for *nested logic programs* that form a proper generalization of DLPs. In this paper, we define these notions of equivalence for DLPs. We denote the set of stable models of a DLP P by $\mathbf{SM}(P)$. The first notion of equivalence follows naturally from the stable model semantics. DLPs P and Q are equivalent ($P \equiv Q$) iff $\mathbf{SM}(P) = \mathbf{SM}(Q)$. The second notion is defined in terms of the first. DLPs P and Q are strongly equivalent ($P \equiv_s Q$) iff $P \cup R \equiv Q \cup R$ for all DLPs R. The DLP R in definition of $\equiv_s$ can be seen as an arbitrary context for P and Q. Thus P and Q are strongly equivalent iff they have the same stable models in every context in which they can be placed. Clearly, $P \equiv_s Q$ implies $P \equiv Q$, but not vice versa.

Example 1. Consider programs $P = \{a \leftarrow \sim c\}$ and $Q = \{a \leftarrow \sim b\}$. P and Q have the same stable models, i.e. $\mathbf{SM}(P) = \{\{a\}\} = \mathbf{SM}(Q)$, and thus $P \equiv Q$. However, if we choose a program $R = \{b \leftarrow \sim a\}$ as the context in which P and Q are placed, we notice that P and Q are not strongly equivalent, since $\mathbf{SM}(P \cup R) = \{\{a\}\} \neq \{\{a\},\{b\}\} = \mathbf{SM}(Q \cup R)$. Thus $P \not\equiv_s Q$.

As discussed in the introduction, we concentrate on weak equivalence in this paper. In order to verify $P \equiv Q$, one needs to check that all stable models of P are also stable models of Q, and that all stable models of Q are stable models of P. In this way, the verification of $P \equiv Q$ involves two directions. In a naive approach, one explicitly enumerates all stable models of P and verifies them as stable models of Q, and vice versa. In the worst case, i.e. when the programs are equivalent, one has to find and check all stable models of both programs. This becomes easily infeasible if the number of stable models is high.

A counter-example for the equivalence of programs P and Q (in one direction) is an interpretation M such that $M \in \mathbf{SM}(P)$ and $M \notin \mathbf{SM}(Q)$. There are two possible reasons for M not being a stable model of Q: either $M \not\models Q_M$ ($\iff M \not\models Q$ [13]) or $M \models Q_M$ but M is not a minimal model of Q_M ($M \notin \mathbf{MM}(Q_M)$). Further on, we shall distinguish two types of counter-examples and present them as pairs. A counter-example of type T1 is a pair $\langle M, M \rangle$ such that $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$; and a counter example of type T2 is a pair $\langle M, M' \rangle$ such that $M \in \mathbf{SM}(P)$, $M \models Q_M$, $M' \subset M$, and $M' \models Q_M$.

We write $\mathrm{CE}(P, Q)$ for the set of counter-examples of types T1 and T2 defined above. A counter-example can also be used to show that programs are not strongly equivalent by forming an SE-model [29]. That is, in case of a type T1 counter-example, the pair $\langle M, M\rangle$ is an SE-model of P but not of Q and in case of type T2, $\langle M', M\rangle$ is an SE-model of Q but not of P. Thus $\mathrm{CE}(P,Q) \neq \emptyset$ implies $P \not\equiv_s Q$. It is also worth pointing out that in case of a counter-example $\langle M, M'\rangle$ of type T2, the model M' is not necessarily a minimal model of Q_M. The number of counter-examples of type T2 can be reduced by insisting on minimality, but then translations (to be presented in Section 4) need to be revised accordingly.

Next, we discuss the computational cost of equivalence testing. We assume the reader to be familiar with basic concepts of computational complexity [24] and introduce the languages corresponding to decision problems of our interest.

Definition 2. *For any finite DLPs P and Q, and $M \subseteq \mathrm{Hb}(P)$,*

- $P \in \mathrm{SM} \iff$ *there exists an interpretation* $M \subseteq \mathrm{Hb}(P)$ *s.t.* $M \in \mathbf{SM}(P)$,
- $\langle P, M\rangle \in \mathrm{NotMin} \iff$ *there exists an interpretation* $M' \subset M$ *s.t.* $M' \models P$,
- $\langle P, Q\rangle \in \mathrm{IMPR} \iff \mathbf{SM}(P) \subseteq \mathbf{SM}(Q)$,
- $\langle P, Q\rangle \in \mathrm{IMPL} \iff \langle Q, P\rangle \in \mathrm{IMPR}$, *and*
- $\langle P, Q\rangle \in \mathrm{EQV} \iff \mathbf{SM}(P) = \mathbf{SM}(Q) \iff P \equiv Q$.

The language SM for disjunctive logic programs is $\Sigma_2^{\mathbf{P}}$-complete [3], thus its complement $\overline{\mathrm{SM}}$ is $\Pi_2^{\mathbf{P}}$-complete. Restricted to normal logic programs, SM is only **NP**-complete [20]. The language NotMin is in **NP**, since we can construct a nondeterministic Turing machine deciding NotMin by selecting an interpretation $M' \subset M$ and checking whether $M' \models P$. Thus an **NP**-oracle can be used to decide whether there exists $M' \subset M$ such that $M' \models P$ for $M \subseteq \mathrm{Hb}(P)$.

Turner [29] shows that deciding the weak equivalence of DLPs is $\Pi_2^{\mathbf{P}}$-hard. It is indeed $\Pi_2^{\mathbf{P}}$-complete, since IMPR, IMPL, EQV are $\Pi_2^{\mathbf{P}}$-complete. The language IMPR is in $\Pi_2^{\mathbf{P}}$, since we can construct a nondeterministic **NP**-oracle Turing machine deciding $\overline{\mathrm{IMPR}}$ by first selecting an interpretation M and checking that $M \models P$, then using an oracle for NotMin to check that there exists no $M' \subset M$ s.t. $M' \models P_M$, thus $M \in \mathbf{SM}(P)$. If $M \not\models Q_M$, we accept $\langle P, Q\rangle$, as $\langle M, M\rangle$ is a counter-example of type T1. Otherwise, an oracle for NotMin is used to check whether there exists $M' \subset M$ s.t. $M' \models Q_M$. If $M \notin \mathbf{MM}(Q_M)$, we accept $\langle P, Q\rangle$, as there is a counter-example of type T2. IMPR is $\Pi_2^{\mathbf{P}}$-hard, since for an arbitrary DLP R, $R \in \overline{\mathrm{SM}} \iff R \notin \mathrm{SM} \iff \mathbf{SM}(R) = \emptyset$. Since $\mathbf{SM}(\{\perp\}) = \emptyset$, $\mathbf{SM}(R) \subseteq \mathbf{SM}(\{\perp\}) \iff \langle R, \{\perp\}\rangle \in \mathrm{IMPR}$. Similarly IMPL is $\Pi_2^{\mathbf{P}}$-complete and EQV is $\Pi_2^{\mathbf{P}}$-complete as $\mathrm{EQV} = \mathrm{IMPR} \cap \mathrm{IMPL}$.

Since EQV is $\Pi_2^{\mathbf{P}}$-complete, testing the weak equivalence of DLPs is as hard as deciding whether a DLP has stable models or not. Thus there is no complexity theoretical obstacle for developing a polynomial transformation for testing the equivalence of DLPs as introduced in [14] for the weight constraint programs. The question remains, however, whether it is possible to find a similar systematic way to obtain the transformation as in [14]. It is worth noting that deciding the equivalence of DLPs is strictly harder than deciding the equivalence of weight constraint programs[3], which forms only a **coNP**-complete decision problem.

[3] Unless the polynomial hierarchy collapses.

4 Translations for Equivalence Testing

In this section we present two translations for equivalence testing, a single translation and a two-phased one. We assume throughout this section that P and Q are two DLPs satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. This is not a significant restriction, since any disjunctive logic program can be extended with rules of the form $a \leftarrow a$ without affecting the stable models of the program, as $\{a \leftarrow a\} \equiv_s \emptyset$.

The idea behind the first translation is to transform DLPs P and Q into one program $\mathrm{TR}(P, Q)$ that has a stable model iff P has a stable model that is not a stable model of Q. There are two possible reasons for $M \notin \mathbf{SM}(Q)$ corresponding to counter-examples of type T1 and T2: either $M \not\models Q_M$ (T1) or $M \models Q_M$, but $M \notin \mathbf{MM}(Q_M)$ (T2). The translation is used to capture these two possibilities. A further objective is that it should be easy to extract a counter-example for $P \equiv Q$ from a stable model N of the translation $\mathrm{TR}(P, Q)$.

For each atom $a \in \mathrm{Hb}(Q)$ we introduce new atoms $a^\bullet$ and a°. We define $A^\bullet = \{a^\bullet \mid a \in A\}$ and $A^\circ = \{a^\circ \mid a \in A\}$ for any set of atoms A. The Herbrand base of $\mathrm{TR}(P, Q)$ will be $\mathrm{Hb}(P) \cup \mathrm{Hb}(P)^\bullet \cup \mathrm{Hb}(P)^\circ \cup \{\mathrm{diff}, \mathrm{unsat}, \mathrm{unsat}^\bullet, \mathrm{ok}\}$, where diff, unsat, $\mathrm{unsat}^\bullet$ and ok are new atoms. Given $M \in \mathbf{SM}(P)$, the atoms in $\mathrm{Hb}(P)^\bullet$ are used to select a subset M' for M, $M' \subseteq M$. The intended meaning for the new atoms introduced in the translation $\mathrm{TR}(P, Q)$ is as follows.

unsat – indicates that $M \not\models Q_M$ (thus $M \not\models Q$ [13]).
$a^\bullet$ – denotes that atom $a \in M$ is in the sub-model M' searched for Q_M
a° – denotes that atom $a \in M$ is not in the sub-model M'.
$\mathrm{unsat}^\bullet$ – indicates that $M' \not\models Q_M$.
diff – indicates that $M' \neq M$, i.e. $M' \subset M$.
ok – indicates that a counter-example for the equivalence is found.

Definition 3. *Let* diff, unsat, $\mathrm{unsat}^\bullet$ *and* ok *be new atoms not appearing in* P *or in* Q*. The translation* $\mathrm{TR}(P, Q)$ *contains the following rules:*

1. *all the rules of* P *without modifications,*
2. *a rule* $\mathrm{unsat} \leftarrow B, {\sim}(A \cup C)$ *for each rule* $A \leftarrow B, {\sim}C \in Q$,
3. *rules* $a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}\mathrm{unsat}$ *and* $a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}\mathrm{unsat}$ *for each* $a \in \mathrm{Hb}(P)$,
4. *a rule* $\mathrm{unsat}^\bullet \leftarrow B^\bullet, {\sim}(A^\bullet \cup C), {\sim}\mathrm{unsat}$ *for each rule* $A \leftarrow B, {\sim}C \in Q$,
5. *a rule* $\mathrm{diff} \leftarrow a, {\sim}a^\bullet, {\sim}\mathrm{unsat}$ *for each atom* $a \in \mathrm{Hb}(P)$ *and*
6. *rules* $\mathrm{ok} \leftarrow \mathrm{unsat}$, $\mathrm{ok} \leftarrow \mathrm{diff}, {\sim}\mathrm{unsat}, {\sim}\mathrm{unsat}^\bullet$ *and* $\bot \leftarrow {\sim}\mathrm{ok}$.

Now, let us discuss the meaning of each individual part of the translation, the items below corresponding respectively to the items of Definition 3.

1. Capture a stable model M of P.
2. Check whether $M \models Q_M$. Thus, if $M \not\models Q_M$, then unsat is implied.
3. Select an interpretation M' s.t. $M' \subseteq M$. These rules force that if $a \in M$, then either a is in the sub-model candidate M' ($a^\bullet \in N$) or not ($a^\circ \in N$).
4. Check whether $(M')^\bullet \models (Q_M)^\bullet$, where $(M')^\bullet = N \cap \mathrm{Hb}(P)^\bullet$. Thus, if there exists a rule in Q_M that is not satisfied in M', $\mathrm{unsat}^\bullet$ is implied.

5. Check that M' is a proper subset of M. That is, diff is implied if $M' \subset M$.
6. Summarize the reasons for M not being a stable model of Q. Either
   - T1: $M \not\models Q_M$ ($\Rightarrow$ unsat $\in N$), or
   - T2: $M \models Q_M$ ($\Rightarrow$ unsat $\notin N$) and $M \notin \mathbf{MM}(Q_M)$, since $M' \subset M$ ($\Rightarrow$ diff $\in N$) and $M' \models Q_M$ ($\Rightarrow$ unsat$^\bullet \notin N$).

   A stable model for the translation exists iff such a reason exists, thus the integrity constraint ensures that every stable model contains the atom ok.

A counter-example can be easily constructed from a stable model N of $\mathrm{TR}(P,Q)$. Clearly ok $\in N$. If unsat $\in N$, then $M = N \cap \mathrm{Hb}(P) \in \mathbf{SM}(P)$ and $M \not\models Q_M$. Thus $\langle M, M\rangle$ is a counter-example of type T1. On the other hand, if unsat $\notin N$, then also diff $\in N$ and unsat$^\bullet \notin N$. Now, $M = N \cap \mathrm{Hb}(P) \in \mathbf{SM}(P)$ and $M \notin \mathbf{MM}(Q_M)$. Furthermore, $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} \subset M$ and $M' \models Q_M$. Thus $\langle M, M'\rangle$ is a counter-example of type T2.

Example 2. Consider DLPs $P = \{a \mid b\}$ and $Q = \{a \leftarrow {\sim}b\}$. Now, $\mathbf{SM}(P) = \{\{a\},\{b\}\}$ and $\mathbf{SM}(Q) = \{\{a\}\}$. The translation $\mathrm{TR}(P,Q)$ is

$$\begin{aligned}\{\ & a \mid b.\ \ \mathrm{unsat} \leftarrow {\sim}a, {\sim}b.\ \ a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}\mathrm{unsat}.\ \ a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}\mathrm{unsat}.\\ & b^\bullet \leftarrow b, {\sim}b^\circ, {\sim}\mathrm{unsat}.\ \ b^\circ \leftarrow b, {\sim}b^\bullet, {\sim}\mathrm{unsat}.\ \ \mathrm{unsat}^\bullet \leftarrow {\sim}a^\bullet, {\sim}b, {\sim}\mathrm{unsat}.\\ & \mathrm{diff} \leftarrow a, {\sim}a^\bullet, {\sim}\mathrm{unsat}.\ \ \mathrm{diff} \leftarrow b, {\sim}b^\bullet, {\sim}\mathrm{unsat}.\ \ \mathrm{ok} \leftarrow \mathrm{unsat}.\\ & \mathrm{ok} \leftarrow \mathrm{diff}, {\sim}\mathrm{unsat}, {\sim}\mathrm{unsat}^\bullet.\ \ \bot \leftarrow {\sim}\mathrm{ok}\ \}.\end{aligned}$$

Consider a model candidate $N = \{b, b^\circ, \mathrm{diff}, \mathrm{ok}\}$. The reduct $\mathrm{TR}(P,Q)_N$ is $\{a \mid b.\ a^\bullet \leftarrow a.\ a^\circ \leftarrow a.\ b^\circ \leftarrow b.\ \mathrm{diff} \leftarrow a.\ \mathrm{diff} \leftarrow b.\ \mathrm{ok} \leftarrow \mathrm{unsat}.\ \mathrm{ok} \leftarrow \mathrm{diff}.\}$. $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$, since $\mathbf{MM}(\mathrm{TR}(P,Q)_N) = \{\{a, a^\bullet, a^\circ, \mathrm{diff}, \mathrm{ok}\}, \{b, b^\circ, \mathrm{diff}, \mathrm{ok}\}\}$. Since the translation has a stable model, we can conclude, that $P \not\equiv Q$, and the counter-example is $\langle M, M'\rangle$ where $M = N \cap \mathrm{Hb}(P) = \{b\}$ and $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} = \emptyset$. Now $M \in \mathbf{SM}(P)$ and $M \notin \mathbf{MM}(Q_M) = \{\emptyset\}$. Also, $M' \subset M$ and $M' \models Q_M$, since $Q_M = \emptyset$. Since unsat $\notin N$, the counter-example is of type T2. As both stable models of P are models of Q, there is no counter-example of type T1. This can be also verified from the translation: the first two rules in $\mathrm{TR}(P,Q)$ ensure that unsat cannot belong to any stable model of $\mathrm{TR}(P,Q)$.

Finding a counter-example for equivalence divides clearly in two cases (types T1 and T2) which suggests to perform testing in two phases. The idea is as follows. In the first phase, we use a translation $\mathrm{TR}_1(P,Q)$ to test whether all the stable models of P are *models* of Q_M. If there exists $M \in \mathbf{SM}(P)$ such that $M \not\models Q_M$, then $\mathrm{TR}_1(P,Q)$ has a stable model and we have found a counter-example of type T1. Otherwise, we will continue to the second phase, where the second translation $\mathrm{TR}_2(P,Q)$ is used to verify whether every $M \in \mathbf{SM}(P)$ is a minimal model of Q_M. The required translations can be obtained rather easily by simplifying $\mathrm{TR}(P,Q)$. The two-phased approach can be motivated by computational arguments. Counter-examples of type T1 can be found using a compact translation, whereas the translation for finding a counter-example of type T2 is more involved. Thus counter-examples of type T2 should be of interest only if counter-examples of type T1 do not exist. In this way, the search space is divided (conditionally) in two parts. Moreover, the translation of the type T2 counter-examples can be simplified, since it is known that every $M \in \mathbf{SM}(P)$ is necessarily a model of Q_M. In the definitions of the translations $\mathrm{TR}_1(P,Q)$ and $\mathrm{TR}_2(P,Q)$ below, we use the same notations and atoms as in Definition 3.

Definition 4. *Let* unsat *be a new atom not appearing in P or in Q. The translation* $\mathrm{TR}_1(P,Q)$ *contains the rules from items 1 and 2 from Definition 3 and in addition a rule* $\bot \leftarrow \sim\mathrm{unsat}$.

We can easily construct a counter-example of type T1 for $P \equiv Q$ from a stable model of the translation $\mathrm{TR}_1(P,Q)$. If $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$, then $M \in \mathbf{SM}(P)$ and $M \not\models Q$ ($\Rightarrow M \not\models Q_M$), where $M = N \cap \mathrm{Hb}(P)$. Thus $\langle M, M\rangle$ is a counter-example of type T1. If $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$ is the case, then we have to perform further testing using another translation $\mathrm{TR}_2(P,Q)$ given below.

Definition 5. *Let* diff *and* unsat$^\bullet$ *be new atoms not appearing in P or in Q. The translation* $\mathrm{TR}_2(P,Q)$ *contains the following rules:*

1. *all the rules of P without modifications,*
2. *rules $a^\bullet \leftarrow a, \sim a^\circ$ and $a^\circ \leftarrow a, \sim a^\bullet$ for each atom $a \in \mathrm{Hb}(P)$,*
3. *a rule* $\mathrm{unsat}^\bullet \leftarrow B^\bullet, \sim(A^\bullet \cup C)$ *for each rule $A \leftarrow B, \sim C \in Q$,*
4. *a rule* $\mathrm{diff} \leftarrow a, \sim a^\bullet$ *for each atom $a \in \mathrm{Hb}(P)$ and*
5. *rules* $\bot \leftarrow \sim\mathrm{diff}$ *and* $\bot \leftarrow \mathrm{unsat}^\bullet$.

Stable models of the translation $\mathrm{TR}_2(P,Q)$ yield counter-examples as follows. Since $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$, there exists no counter-example of type T1, i.e. if $M \in \mathbf{SM}(P)$, then $M \models Q_M$. If $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$, then $M = N \cap \mathrm{Hb}(P) \in \mathbf{SM}(P)$, $M \models Q_M$, and $M \notin \mathbf{MM}(Q_M)$. Also, $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} \subset M$ and $M' \models Q_M$. Thus $\langle M, M'\rangle$ is a counter-example of type T2.

In fact, there is a tight correspondence between the set of counter-examples $\mathrm{CE}(P,Q)$ and $\mathbf{SM}(\mathrm{TR}(P,Q))$ to be made explicit as follows. First we define a mapping from $\mathrm{CE}(P,Q)$ to $\mathbf{SM}(\mathrm{TR}(P,Q))$. Given a pair of interpretations $\langle M, M'\rangle$ of P such that $M' \subseteq M$, we define $\mathrm{EXT}_{P,Q}(M,M') = M \cup \{\mathrm{unsat}, \mathrm{ok}\}$, if $M = M'$, and $\mathrm{EXT}_{P,Q}(M,M') = M \cup \{\mathrm{diff}, \mathrm{ok}\} \cup \{a^\bullet \mid a \in M'\} \cup \{a^\circ \mid a \in M \setminus M'\}$, otherwise. The other direction follows. Given an interpretation of the translation $\mathrm{TR}(P,Q)$, let $\mathrm{PROJ}_{P,Q}(N) = \langle N \cap \mathrm{Hb}(P), N \cap \mathrm{Hb}(P)\rangle$, if $\mathrm{unsat} \in N$, and $\mathrm{PROJ}_{P,Q}(N) = \langle N \cap \mathrm{Hb}(P), \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\}\rangle$, otherwise.

Theorem 1. *If $\langle M, M'\rangle \in \mathrm{CE}(P,Q)$, then the set $N = \mathrm{EXT}_{P,Q}(M,M') \in \mathbf{SM}(\mathrm{TR}(P,Q))$ such that $\mathrm{PROJ}_{P,Q}(N) = \langle M, M'\rangle$. If $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$, then the pair $\langle M, M'\rangle = \mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}(P,Q)$ such that $N = \mathrm{EXT}_{P,Q}(M,M')$.*

As a consequence of Theorem 1, the mappings $\mathrm{EXT}_{P,Q}$ and $\mathrm{PROJ}_{P,Q}$ are bijections implying that the counter-examples in $\mathrm{CE}(P,Q)$ and the stable models of $\mathrm{TR}(P,Q)$ are in a one-to-one correspondence. This relationship implies the correctness of our method for verifying the equivalence of DLPs.

Corollary 1. *For DLPs P and Q such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$, $P \equiv Q \iff \mathbf{SM}(\mathrm{TR}(P,Q)) = \emptyset$ and $\mathbf{SM}(\mathrm{TR}(Q,P)) = \emptyset$.*

Let us then turn our attention to the correctness of the two-phased translation $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$. In this case, we partition the set of counter-examples $\mathrm{CE}(P,Q)$ into $\mathrm{CE}_1(P,Q) \cup \mathrm{CE}_2(P,Q)$ by using the type of counter-examples as a criterion. The mappings $\mathrm{EXT}_{P,Q}$ and $\mathrm{PROJ}_{P,Q}$ are revised accordingly by dropping the atom "ok" not appearing in $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$. The correctness of the translation is established phase-wise.

Theorem 2. *If* $\langle M, M\rangle \in \mathrm{CE}_1(P,Q)$*, then there is* $N = \mathrm{EXT}_{P,Q}(M,M) \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$ *such that* $\mathrm{PROJ}_{P,Q}(N) = \langle M, M\rangle$*. If* $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$*, then* $\langle M, M\rangle = \mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}_1(P,Q)$ *such that* $N = \mathrm{EXT}_{P,Q}(M,M)$*.*

Theorem 3. *Suppose that* $\mathrm{CE}_1(P,Q) = \emptyset$*, or equivalently* $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$*. If* $\langle M, M'\rangle \in \mathrm{CE}_2(P,Q)$*, then there is* $N = \mathrm{EXT}_{P,Q}(M,M') \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$ *such that* $\mathrm{PROJ}_{P,Q}(N) = \langle M, M'\rangle$*. If* $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$*, then* $\langle M, M'\rangle = \mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}_2(P,Q)$ *such that* $N = \mathrm{EXT}_{P,Q}(M,M')$*.*

By a slight notational generalization, we define the set of stable models $\mathbf{SM}([\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)])$ as $\mathbf{SM}(\mathrm{TR}_1(P,Q))$, if $\mathbf{SM}(\mathrm{TR}_1(P,Q)) \neq \emptyset$, and as $\mathbf{SM}(\mathrm{TR}_2(P,Q))$, otherwise. Then the correctness of the two-phased translation-based method can be formulated in analogy to Corollary 1.

Corollary 2. *For DLPs* P *and* Q *such that* $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$*,* $P \equiv Q \iff \mathbf{SM}([\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]) = \emptyset$ *and* $\mathbf{SM}([\mathrm{TR}_1(Q,P), \mathrm{TR}_2(Q,P)]) = \emptyset$*.*

5 Experiments

Translation functions TR and $[\mathrm{TR}_1, \mathrm{TR}_2]$ presented in Section 4 have been implemented in C under Linux. The translator DLPEQ [23] takes two DLPs as input and produces by default the translation TR as its output. Translations TR_1 and TR_2 can be produced using command line options. The program assumes the internal format of GNT and thus the front-end LPARSE can be used. Also, the translation is printed in a textual form upon request. It is important to note that DLPEQ checks that the *visible* Herbrand bases of the DLPs being compared are the same (a visible atom has a name in the symbol table of the DLP). The front-end may produce some *invisible* atoms that are neglected in the comparison. To support programs produced by LPARSE, such atoms keep their roles in the respective programs. Also, DLPEQ even enables equivalence testing of DLPs P and Q such that $\mathrm{Hb}(P) \neq \mathrm{Hb}(Q)$: the *visible* Herbrand bases of P and Q are extended to be the same by adding (useless) rules of the form $a \leftarrow a$.

To assess the feasibility of DLPEQ in practice we ran tests to compare running times of the DLPEQ and DLPEQ-2 (two-phased translation) approach with a NAIVE approach, in which one (i) generates a stable model M of P, (ii) tests whether M is a stable model of Q, (iii) stops if not, and otherwise continues from step (i) until all stable models of P get enumerated. A similar testing has to be carried out in the other direction to establish $P \equiv Q$. There is still room for optimization in all the approaches. If one finds a counter-example in one direction, then $P \not\equiv Q$ holds immediately and there is no need to test the other direction except to perform a thorough analysis. We decided not to use this optimization, since running times turned out to scale differently depending on the direction. Thus we count running times in both directions.

In all the approaches the GnT system is responsible for the computation of stable models. In the DLPEQ approach, the total running time (in one direction) is the running time needed for trying to compute *one* stable model of the translation produced by DLPEQ. In the DLPEQ-2 approach, the total running time (in one direction) is the time needed for trying to compute *one* stable model of the translation TR_1 plus the time needed for trying to compute *one* stable model of the translation TR_2, if there exists no stable models for TR_1. In both approaches the actual translation times are not taken into account, as they are negligible. In the NAIVE approach, we try to exclude possible overhead due to implementing the NAIVE approach as a shell script. Thus the total running time (in one direction) consists only of the running time for finding the necessary (but not necessarily all) stable models of P plus the individual running times for testing that the stable models found are also stable models of Q. These tests are realized in practice by adding M as a compute statement to Q. The tests were run under the Debian GNU/Linux 2.4.18 operating system on a 450MHz Intel Pentium III computer with 256MB of memory.

We use a $\Sigma_2^{\mathbf{P}}$-complete problem of finding a minimal model of a set of clauses containing specified atoms [3] as our first test problem. Each clause $a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_m$ of an instance is translated into a rule $a_1 | \ldots | a_n \leftarrow b_1, \ldots, b_m$ and a rule $\bot \leftarrow \sim c_i$ is included for each specified atom c_i. The resulting DLP has a stable model iff there is a minimal model of the clauses containing all the specified atoms c_i. We generate DLPs that solve an instance of a random 3-sat problem and add the extra rules for random atoms c_i, for $i = 1, \ldots, \lfloor 2v/100 \rfloor$, where v is the number of atoms in the instance. First we keep the clauses-to-variables ratio c/v constant at 3.5. The DPLs generated from such instances typically have several stable models. To simulate a sloppy programmer making mistakes, we drop *one* random rule from each program. Then we test the equivalence of the modified (Drop-1) and the original DLP to see if making a mistake affects the stable models of the program or not. Thus, the tested pairs of DLPs include both equivalent and nonequivalent cases. We vary the number of variables v from 50 to 100 with steps of 5. For each number of variables we repeat the test 100 times – generating each time a new random instance. For the problem sizes used, the percentage of nonequivalent cases varies between 60 and 90. The maximum, average and minimum running times of each approach are plotted in Fig. 1 (left). The DLPEQ approach turns out to be faster than the NAIVE one and the difference in running times increases as instances grow in size. Furthermore, the DLPEQ-2 approach outperforms the DLPEQ approach.

Next we generate programs as in the previous experiment, but keep the number of variables constant at $v = 100$. We vary the ratio c/v from 3.25 to 4.50 with steps of 0.25 to see how the translation-based approaches perform compared to the NAIVE one as the tested DLPs change from ones having many

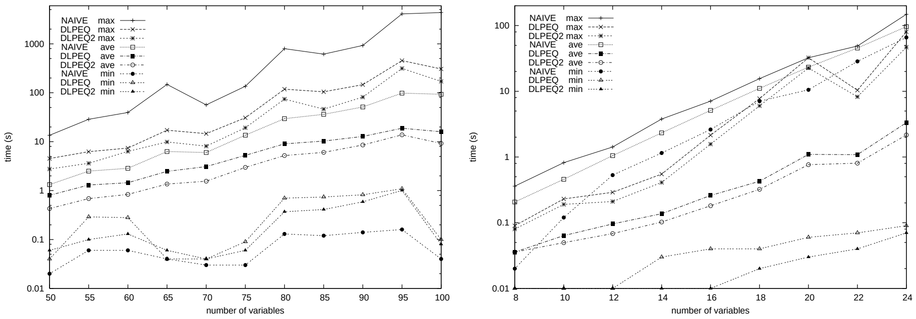

Fig. 1. Equivalence testing of random 3-sat instances with the ratio $c/v = 3.5$ (left) and random 2-qbf instances with the ratio $c/v = 3.5$ (right).

stable models to programs having no stable models. With lower values of c/v the DLPEQ and DLPEQ-2 approaches are clearly superior to the NAIVE one. As the ratio increases the performance of the NAIVE approach gradually improves and finally outperforms the two other approaches.

Our second test problem is a $\Sigma_2^{\mathbf{P}}$-complete problem of deciding the validity of a 2-qbf (quantified Boolean formula). A 2-qbf instance is of the form $\Phi = \exists X \forall Y \phi$, where $X \cap Y = \emptyset$ and ϕ is a 3-sat formula in DNF over the set of variables $X \cup Y$. In our experiment $|X| = |Y|$ and $v = |X \cup Y|$. We generate random 2-qbf instances according to model A presented by Gent and Walsh [8], i.e. each clause in ϕ contains at least two variables that are universally quantified. We transform a generated random instance Φ to a DLP using a transformation presented by Eiter and Gottlob [3]. We test the equivalence of the program and its modified version (Drop-1). We keep c/v constant at 3.5 and vary the number of variables v from 10 to 24 with steps of 2. For each v we repeat the test 100 times, generating each time a new random instance. For the problem sizes used, the percentage of nonequivalent cases is approximately 50–60. The maximum, average and minimum running times of all the approaches are plotted in Fig. 1 (right). The DLPEQ approach is clearly faster than the NAIVE one. The difference in running times increases as instances grow in size. Furthermore, the DLPEQ-2 approach performs slightly better that the DLPEQ approach.

6 Conclusions

This paper extends the translation-based approach for testing the equivalence of logic programs under the stable model semantics [14] to the case of disjunctive programs. Two systematic transformations are presented in order to reduce the problem of testing whether two logic programs are equivalent to the problem of computing stable models for a disjunctive program. As a consequence, existing search engines such as DLV and GNT can be used for the search of counter-examples, and there is no need to develop a special purpose system for this task.

Our implementation, a translator program called DLPEQ, is compatible with GNT and it enables the equivalence testing of disjunctive programs in practice.

The preliminary experiments reported in Section 5 suggest that the translation-based approach is superior to a naive cross-checking approach when the programs possess many stable models, although the length of the translation exceeds the sum of the lengths of the programs being tested. To our understanding, this is because the translation provides an explicit specification of a counter-example for $P \equiv Q$ which enables GNT to prune search space. This is not possible in the NAIVE approach where the stable models of P and Q are explicitly enumerated. However, if the programs under comparison have few/no stable models, then the naive approach is likely to be faster. A further observation is that the two-phased translation $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$ is more efficient than the approach based on a single translation $\mathrm{TR}(P,Q)$. Thus it seems a good idea to compute the two types of counter-examples in isolation.

Finally, let us sketch future work. It seems that even more general classes of logic programs can be covered via suitable translations. For instance, the weak equivalence of *nested programs* [16] can be addressed by implementing the transformation of nested programs into disjunctive programs [25]. On the other hand, determining the strong equivalence of nested (as well as disjunctive) programs is computationally less complex, which suggests using an NP solver for the task. E.g., Lin [18] gives a transformation that enables the use of SAT solvers for testing strong equivalence. Moreover, DLPEQ should be extended to cover other equivalence relations such as uniform [2] and strong equivalence [16].

References

1. M. Balduccini, M. Gelfond, R. Watson, and M. Nogueira. The USA-advisor: A case study in answer set planning. In *Proc. of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, pages 439–442, Vienna, Austria, September 2001. Springer. LNAI 2173.
2. T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In *Proc. of the 19th International Conference on Logic Programming (ICLP'03)*, 2003. To appear.
3. T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Math. and AI*, 15:289–323, 1995.
4. P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. Available at `http://www.cs.utexas.edu/users/vl/papers.html`, 2003. Unpublished draft.
5. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. MIT Press.
6. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. of the 7th International Conference on Logic Programming*, pages 579–597, Jerusalem, Israel, June 1990. MIT Press.
7. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
8. I. Gent and T. Walsh. Beyond NP: the QSAT phase transition. In *Proc. of the 16th National Conference on Artificial Intelligence and the 11th Conference on*

Innovative Applications of Artificial Intelligence, pages 648–653, Orlando, Florida, USA, July 1999. AAAI, MIT Press.
9. K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.
10. T. Janhunen. On the effect of default negation on the expressiveness of disjunctive rules. In *Logic Programming and Nonmonotonic Reasoning, Proc. of the 6th International Conference*, pages 93–106, Vienna, Austria, September 2001. Springer.
11. T. Janhunen. LPEQ 1.13 — a tool for testing the equivalence of logic programs. `http://www.tcs.hut.fi/Software/lpeq/`, 2002. Computer Program.
12. T. Janhunen. GnT 2 — a tool for computing disjunctive stable models. `http://www.tcs.hut.fi/Software/gnt/`, 2003. Computer Program.
13. T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J.-H. You. Unfolding partiality and disjunctions in stable model semantics. CoRR: cs.AI/0303009, March 2003.
14. T. Janhunen and E. Oikarinen. Testing the equivalence of logic programs under stable model semantics. In *Logics in Artificial Intelligence, Proc. of the 8th European Conference*, pages 493–504, Cosenza, Italy, September 2002. Springer.
15. N. Leone et al. The DLV system for knowledge representation and reasoning. CoRR: cs.AI/0211004, September 2003.
16. V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
17. V. Lifschitz, L.R. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Math. and AI*, 25(3–4):369–389, 1999.
18. F. Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In D. Fensel et al., editors, *Principles of Knowledge Representation and Reasoning: Proc. of the 8th International Conference*, pages 170–176, Tolouse, France, April 2002. Morgan Kaufmann.
19. J.W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1987.
20. W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.
21. W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer, 1999.
22. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Math. and AI*, 25(3–4):241–273, 1999.
23. E. Oikarinen. DLPEQ 1.9 — a tool for testing the equivalence of disjunctive logic programs. `http://www.tcs.hut.fi/Software/lpeq/`, 2003. Computer Program.
24. C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
25. D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs: Preliminary report. In *Proc. of the 18th International Conference on Logic Programming*, pages 405–420, Copenhagen, Denmark, July–August 2002. Springer.
26. D. Pearce, H. Tompits, and S. Woltran. Encodings for equilibirium logic and logic programs with nested expressions. In *Proc. of the 10th Portuguese Conference on Artificial Intelligence*, pages 306–320, Porto, Portugal, December 2001. Springer.
27. T. Przymusinski. Extended stable semantics for normal and disjunctive logic programs. In *Proc. of the 7th International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.
28. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
29. H. Turner. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4–5):609–622, 2003.