A Polynomial Translation of Logic Programs with Nested Expressions into Disjunctive Logic Programs: Preliminary Report

David Pearce[1], Vladimir Sarsakov[2], Torsten Schaub[2⋆],
Hans Tompits[3], and Stefan Woltran[3]

[1] European Commission, DG Information Society – F1
BU33 3/58, Rue de le Loi 200, B-1049 Brussels
David.Pearce@cec.eu.int
[2] Institut für Informatik, Universität Potsdam,
Postfach 60 15 53, D–14415 Potsdam, Germany
sarsakov@rz.uni-potsdam.de,
torsten@cs.uni-potsdam.de
[3] Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Wien, Austria
[tompits,stefan]@kr.tuwien.ac.at

Abstract. Nested logic programs have recently been introduced in order to allow for arbitrarily nested formulas in the heads and the bodies of logic program rules under the answer sets semantics. Previous results show that nested logic programs can be transformed into standard (unnested) disjunctive logic programs in an elementary way, applying the negation-as-failure operator to body literals only. This is of great practical relevance since it allows us to evaluate nested logic programs by means of off-the-shelf disjunctive logic programming systems, like DLV. However, it turns out that this straightforward transformation results in an exponential blow-up in the worst-case, despite the fact that complexity results indicate that there is a polynomial translation among both formalisms. In this paper, we take up this challenge and provide a polynomial translation of logic programs with nested expressions into disjunctive logic programs. Moreover, we show that this translation is modular and (strongly) faithful. We have implemented both the straightforward as well as our advanced transformation; the resulting compiler serves as a front-end to DLV and is publicly available on the Web.

1 Introduction

Lifschitz, Tang, and Turner [24] recently extended the answer set semantics [12] to a class of logic programs in which arbitrarily nested formulas, formed from literals using negation as failure, conjunction, and disjunction, constitute the heads and bodies of rules. These so-called *nested logic programs* generalise the well-known classes of *normal*, *generalised*, *extended*, and *disjunctive logic programs*, respectively. Despite their syntactically much more restricted format, the latter classes are well recognised

⋆ Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

as important tools for knowledge representation and reasoning. This is reflected by the fact that several practically relevant applications have been developed recently using these types of programs (cf., e.g., [22, 3, 11, 16]), which in turn is largely fostered by the availability of efficient solvers for the answer set semantics, most notably DLV [8, 9] and Smodels [27].

In this paper, we are interested in utilising these highly performant solvers for interpreting nested logic programs. We address this problem by providing a translation of nested logic programs into disjunctive logic programs. In contrast to previous work, our translation is guaranteed to be polynomial in time and space, as suggested by related complexity results [32]. More specifically, we provide a translation, σ, from nested logic programs into disjunctive logic programs possessing the following properties:

- σ maps nested logic programs over an alphabet $\mathcal{A}_1$ into disjunctive logic programs over an alphabet $\mathcal{A}_2$, where $\mathcal{A}_1 \subseteq \mathcal{A}_2$;
- the size of $\sigma(\Pi)$ is polynomial in the size of Π;
- σ is *faithful*, i.e., for each program Π over alphabet $\mathcal{A}_1$, there is a one-to-one correspondence between the answer sets of Π and sets of form $I \cap \mathcal{A}_1$, where I is an answer set of $\sigma(\Pi)$; and
- σ is *modular*, i.e., $\sigma(\Pi \cup \Pi') = \sigma(\Pi) \cup \sigma(\Pi')$, for each program Π, Π'.

Moreover, we have implemented translation σ, serving as a front-end for the logic programming system DLV.

The construction of σ relies on the introduction of new *labels*, abbreviating subformula occurrences. This technique is derived from *structure-preserving normal form translations* [36, 33], frequently employed in the context of automated deduction (cf. [1] for an overview). We use here a method adapted from a structure-preserving translation for intuitionistic logic as described in [26].

Regarding the faithfulness of σ, we actually provide a somewhat stronger condition, referred to as *strong faithfulness*, expressing that, for any programs Π and Π' over alphabet $\mathcal{A}_1$, there is a one-to-one correspondence between the answer sets of $\Pi \cup \Pi'$ and sets of form $I \cap \mathcal{A}_1$, where I is an answer set of $\sigma(\Pi) \cup \Pi'$. This condition means that we can add to a given program Π *any nested program* Π' and still recover the answer sets of the combined program $\Pi \cup \Pi'$ from $\sigma(\Pi) \cup \Pi'$; in particular, for any nested logic program Π, we may choose to translate, in a semantics-preserving way, only an arbitrary *program part* $\Pi_0 \subseteq \Pi$ and leave the remaining part $\Pi \setminus \Pi_0$ unchanged. For instance, if Π_0 is already a disjunctive logic program, we do not need to translate it again into another (equivalent) disjunctive logic program. Strong faithfulness is closely related to the concept of *strong equivalence* [23] (see below).

In order to have a sufficiently general setting for our purposes, we base our investigation on *equilibrium logic* [28], a generalisation of the answer set semantics for nested logic programs. Equilibrium logic is a form of minimal-model reasoning in the *logic of here-and-there*, which is intermediate between classical logic and intuitionistic logic (the logic of here-and-there is also known as *Gödel's three-valued logic* in view of [14]). As shown in [28, 29, 23], logic programs can be viewed as a special class of formulas in the logic of here-and-there such that, for each program Π, the answer sets of Π are given by the equilibrium models of Π, where the latter Π is viewed as a set of formulas in the logic of here-and-there.

The problem of implementing nested logic programs has already been addressed in [32], where (linear-time constructible) encodings of the basic reasoning tasks associated with this language into quantified Boolean formulas are described. These encodings provide a straightforward implementation for nested logic programs by appeal to off-the-shelf solvers for quantified Boolean formulas (like, e.g., the systems proposed in [4, 10, 13, 20, 21, 34]). Besides the encodings into quantified Boolean formulas, a further result of [32] is that nested logic programs possess the same worst-case complexity as disjunctive logic programs, i.e., the main reasoning tasks associated with nested logic programs lie at the second level of the polynomial hierarchy. From this result it follows that nested logic programs can in turn be efficiently reduced to disjunctive logic programs. Hence, given such a reduction, solvers for the latter kinds of programs, like, e.g., `DLV` or `Smodels`, can be used to compute the answer sets of nested logic programs. The main goal of this paper is to construct a reduction of this type.

Although results by Lifschitz, Tang, and Turner [24] (together with transformation rules given in [19]) provide a method to translate nested logic programs into disjunctive ones, that approach suffers from the drawback of an exponential blow-up of the resulting disjunctive logic programs in the worst case. This is due to the fact that this translation relies on distributivity laws yielding an exponential increase of program size whenever the given program contains rules whose heads are in disjunctive normal form or whose bodies are in conjunctive normal form, and the respective expressions are not simple disjunctions or conjunctions of literals. Our translation, on the other hand, is always polynomial in the size of its input program.

Finally, we mention that structure-preserving normal form translations in the logic of here-and-there are also studied, yet in much more general settings, by Baaz and Fermüller [2] as well as by Hähnle [15]; there, whole classes of finite-valued Gödel logics are investigated. Unfortunately, these normal form translations are not suitable for our purposes, because they do not enjoy the particular form of programs required here.

2 Preliminaries

We deal with propositional languages and use the logical symbols $\top$, $\bot$, $\neg$, $\vee$, $\wedge$, and $\rightarrow$ to construct formulas in the standard way. We write $\mathcal{L}_{\mathcal{A}}$ to denote a language over an alphabet $\mathcal{A}$ of *propositional variables* or *atoms*. Formulas are denoted by Greek lowercase letters (possibly with subscripts). As usual, *literals* are formulas of form v or $\neg v$, where v is some variable or one of $\top, \bot$.

Besides the semantical concepts introduced below, we also make use of the semantics of classical propositional logic. By a (*classical*) *interpretation*, I, we understand a set of variables. Informally, a variable v is true under I iff $v \in I$. The truth value of a formula ϕ under interpretation I, in the sense of classical propositional logic, is determined in the usual way.

2.1 Logic Programs

The central objects of our investigation are logic programs with nested expressions, introduced by Lifschitz *et al*. [24]. These kinds of programs generalise normal logic

programs by allowing bodies and heads of rules to contain arbitrary Boolean formulas. For reasons of simplicity, we deal here only with languages containing one kind of negation, however, corresponding to default negation. The extension to the general case where strong negation is also permitted is straightforward and proceeds in the usual way.

We start with some basic notation. A formula whose sentential connectives comprise only $\wedge$, $\vee$, or $\neg$ is called an *expression*. A rule, r, is an ordered pair of form

$$H(r) \leftarrow B(r),$$

where $B(r)$ and $H(r)$ are expressions. $B(r)$ is called the *body* of r and $H(r)$ is the *head* of r. We say that r is a *generalised disjunctive rule* if $B(r)$ is a conjunction of literals and $H(r)$ is a disjunction of literals; r is a *disjunctive rule* iff it is a generalised disjunctive rule containing no negated atom in its head; finally, if r is a rule containing no negation at all, then r is called *basic*. A *nested logic program*, or simply a *program*, Π, is a finite set of rules. Π is a *generalised disjunctive logic program* iff it contains only generalised disjunctive rules. Likewise, Π is a *disjunctive logic program* iff Π contains only disjunctive rules, and Π is *basic* iff each rule in Π is basic. We say that Π is a program *over* alphabet $\mathcal{A}$ iff all atoms occurring in Π are from $\mathcal{A}$. The set of all atoms occurring in program Π is denoted by $var(\Pi)$. We use $NLP_{\mathcal{A}}$ to denote the class of all nested logic programs over alphabet $\mathcal{A}$; furthermore, $DLP_{\mathcal{A}}$ stands for the subclass of $NLP_{\mathcal{A}}$ containing all disjunctive logic programs over $\mathcal{A}$; and $GDLP_{\mathcal{A}}$ is the class of all generalised disjunctive logic programs over $\mathcal{A}$. Further classes of programs are introduced in Section 4.

In what follows, we associate to each rule r a corresponding formula $\hat{r} = B(r) \rightarrow H(r)$ and, accordingly, to each program Π a corresponding set of formulas $\hat{\Pi} = \{\hat{r} \mid r \in \Pi\}$.

Let Π be a basic program over $\mathcal{A}$ and $I \subseteq \mathcal{A}$ a (classical) interpretation. We say that I is a *model* of Π iff it is a model of the associated set $\hat{\Pi}$ of formulas. Furthermore, given an (arbitrary) program Π over $\mathcal{A}$, the *reduct*, Π^I, of Π with respect to I is the basic program obtained from Π by replacing every occurrence of an expression $\neg\psi$ in Π which is not in the scope of any other negation by $\bot$ if ψ is true under I, and by $\top$ otherwise. I is an *answer set* (or *stable model*) of Π iff it is a minimal model (with respect to set inclusion) of the reduct Π^I. The collection of all answer sets of Π is denoted by $AS_{\mathcal{A}}(\Pi)$.

Two logic programs, Π_1 and Π_2, are *equivalent* iff they possess the same answer sets. Following Lifschitz *et al*. [23], we call Π_1 and Π_2 *strongly equivalent* iff, for every program Π, $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ are equivalent.

2.2 Equilibrium Logic

Equilibrium logic is an approach to nonmonotonic reasoning that generalises the answer set semantics for logic programs. We use this particular formalism because it offers a convenient logical language for dealing with logic programs under the answer set semantics. It is defined in terms of the logic of here-and-there, which is intermediate between classical logic and intuitionistic logic. Equilibrium logic was introduced in [28]

and further investigated in [29]; proof theoretic studies of the logic can be found in [31, 30].

Generally speaking, the logic of here-and-there is an important tool for analysing various properties of logic programs. For instance, as shown in [23], the problem of checking whether two logic programs are strongly equivalent can be expressed in terms of the logic of here-and-there (cf. Proposition 2 below).

The semantics of the logic of here-and-there is defined by means of two worlds, H and T, called "here" and "there". It is assumed that there is a total order, $\leq$, defined between these worlds such that $\leq$ is reflexive and $H \leq T$. As in ordinary Kripke semantics for intuitionistic logic, we can imagine that in each world a set of atoms is verified and that, once verified "here", an atom remains verified "there".

Formally, by an *HT-interpretation*, $\mathcal{I}$, we understand an ordered pair $\langle I_H, I_T \rangle$ of sets of atoms such that $I_H \subseteq I_T$. We say that $\mathcal{I}$ is an HT-interpretation *over* $\mathcal{A}$ if $I_T \subseteq \mathcal{A}$. The set of all HT-interpretations over $\mathcal{A}$ is denoted by $INT_{\mathcal{A}}$. An HT-interpretation $\langle I_H, I_T \rangle$ is *total* if $I_H = I_T$.

The truth value, $\nu_{\mathcal{I}}(w, \phi)$, of a formula ϕ at a world $w \in \{H, T\}$ in an HT-interpretation $\mathcal{I} = \langle I_H, I_T \rangle$ is recursively defined as follows:

1. if $\phi = \top$, then $\nu_{\mathcal{I}}(w, \phi) = 1$;
2. if $\phi = \bot$, then $\nu_{\mathcal{I}}(w, \phi) = 0$;
3. if $\phi = v$ is an atom, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if $v \in I_w$, otherwise $\nu_{\mathcal{I}}(w, \phi) = 0$;
4. if $\phi = \neg\psi$, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if, for every world u with $w \leq u$, $\nu_{\mathcal{I}}(u, \psi) = 0$, otherwise $\nu_{\mathcal{I}}(w, \phi) = 0$;
5. if $\phi = (\phi_1 \wedge \phi_2)$, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if $\nu_{\mathcal{I}}(w, \phi_1) = 1$ and $\nu_{\mathcal{I}}(w, \phi_2) = 1$, otherwise $\nu_{\mathcal{I}}(w, \phi) = 0$;
6. if $\phi = (\phi_1 \vee \phi_2)$, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if $\nu_{\mathcal{I}}(w, \phi_1) = 1$ or $\nu_{\mathcal{I}}(w, \phi_2) = 1$, otherwise $\nu_{\mathcal{I}}(w, \phi) = 0$;
7. if $\phi = (\phi_1 \rightarrow \phi_2)$, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if for every world u with $w \leq u$, $\nu_{\mathcal{I}}(u, \phi_1) = 0$ or $\nu_{\mathcal{I}}(u, \phi_2) = 1$, otherwise $\nu_{\mathcal{I}}(w, \phi) = 0$.

We say that ϕ is *true under* $\mathcal{I}$ *in* w iff $\nu_{\mathcal{I}}(w, \phi) = 1$, otherwise ϕ is *false under* $\mathcal{I}$ *in* w. An HT-interpretation $\mathcal{I} = \langle I_H, I_T \rangle$ *satisfies* ϕ, or $\mathcal{I}$ is an *HT-model* of ϕ, iff $\nu_{\mathcal{I}}(H, \phi) = 1$. If ϕ is true under any HT-interpretation, then ϕ is *valid in the logic of here-and-there*, or simply *HT-valid*.

Let S be a set of formulas. An HT-interpretation $\mathcal{I}$ is an HT-model of S iff $\mathcal{I}$ is an HT-model of each element of S. We say that $\mathcal{I}$ is an HT-model of a *program* Π iff $\mathcal{I}$ is an HT-model of $\hat{\Pi} = \{B(r) \rightarrow H(r) \mid r \in \Pi\}$.

Two sets of formulas are *equivalent in the logic of here-and-there*, or *HT-equivalent*, iff they possess the same HT-models. Two formulas, ϕ and ψ, are HT-equivalent iff the sets $\{\phi\}$ and $\{\psi\}$ are HT-equivalent.

It is easily seen that any HT-valid formula is valid in classical logic, but the converse does not always hold. For instance, $p \vee \neg p$ and $\neg\neg p \rightarrow p$ are valid in classical logic but not in the logic of here-and-there as the pair $\langle \emptyset, \{p\} \rangle$ is not an HT-model for either of these formulas.

Equilibrium logic can be seen as a particular type of reasoning with minimal HT-models. Formally, an *equilibrium model* of a formula ϕ is a total HT-interpretation $\langle I, I \rangle$

such that (i) $\langle I, I\rangle$ is an HT-model of ϕ, and (ii) for every proper subset J of I, $\langle J, I\rangle$ is not an HT-model of ϕ.

The following result establishes the close connection between equilibrium models and answer sets, showing that answer sets are actually a special case of equilibrium models:

Proposition 1 ([28, 23]). *For any program Π, I is an answer set of Π iff $\langle I, I\rangle$ is an equilibrium model of $\hat{\Pi}$.*

Moreover, HT-equivalence was shown to capture the notion of strong equivalence between logic programs:

Proposition 2 ([23]). *Let Π_1 and Π_2 be programs, and let $\hat{\Pi}_i = \{B(r) \rightarrow H(r) \mid r \in \Pi_i\}$, for $i = 1, 2$. Then, Π_1 and Π_2 are strongly equivalent iff $\hat{\Pi}_1$ and $\hat{\Pi}_2$ are equivalent in the logic of here-and-there.*

Recently, de Jongh and Hendriks [5] have extended Proposition 2 by showing that for nested programs strong equivalence is characterised precisely by equivalence in all intermediate logics lying between here-and-there (upper bound) and the logic KC of weak excluded middle (lower bound) which is axiomatised by intuitionistic logic together with the schema $\neg\varphi \vee \neg\neg\varphi$. Also, in [32] a (polynomial-time constructible) translation is given which reduces the problem of deciding whether two nested programs are strongly equivalent into the validity problem of classical propositional logic (a similar result was independently shown in [25] for disjunctive programs). As a consequence, checking whether two programs are strongly equivalent has co-NP complexity.

We require the following additional concepts. By an *HT-literal*, l, we understand a formula of form v, $\neg v$, or $\neg\neg v$, where v is a propositional atom or one of $\top$, $\bot$. Furthermore, a formula is in *here-and-there negational normal form*, or *HT-NNF*, if it is made up of HT-literals, conjunctions and disjunctions. Likewise, we say that a *program* is in HT-NNF iff all heads and bodies of rules in the program are in HT-NNF.

Following [24], every expression ϕ can effectively be transformed into an expression ψ in HT-NNF possessing the same HT-models as ϕ. In fact, we have the following property:

Proposition 3. *Every expression ϕ is HT-equivalent to an expression $\nu(\phi)$ in HT-NNF, where $\nu(\phi)$ is constructible in polynomial time from ϕ, satisfying the following conditions, for each expression φ, ψ:*

1. $\nu(\varphi) = \varphi$, *if φ is an HT-literal;*
2. $\nu(\neg\neg\neg\varphi) = \nu(\neg\varphi)$;
3. $\nu(\varphi \circ \psi) = \nu(\varphi) \circ \nu(\psi)$, *for* $\circ \in \{\wedge, \vee\}$;
4. $\nu(\neg(\varphi \wedge \psi)) = \nu(\neg\varphi) \vee \nu(\neg\psi)$;
5. $\nu(\neg(\varphi \vee \psi)) = \nu(\neg\varphi) \wedge \nu(\neg\psi)$.

3 Faithful Translations

Next, we introduce the general requirements we impose on our desired translation from nested logic programs into disjunctive logic programs. The following definition is central:

Definition 1. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two alphabets such that $\mathcal{A}_1 \subseteq \mathcal{A}_2$, and, for $i = 1, 2$, let $S_i \subseteq NLP_{\mathcal{A}_i}$ be a class of nested logic programs closed under unions.*[1] *Then, a function $\rho : S_1 \to S_2$ is*

1. polynomial *iff, for all programs $\Pi \in S_1$, the time required to compute $\rho(\Pi)$ is polynomial in the size of Π;*
2. faithful *iff, for all programs $\Pi \in S_1$,*

$$AS_{\mathcal{A}_1}(\Pi) = \{I \cap \mathcal{A}_1 \mid I \in AS_{\mathcal{A}_2}(\rho(\Pi))\};$$

3. strongly faithful *iff, for all programs $\Pi \in S_1$ and all programs $\Pi' \in NLP_{\mathcal{A}_1}$,*

$$AS_{\mathcal{A}_1}(\Pi \cup \Pi') = \{I \cap \mathcal{A}_1 \mid I \in AS_{\mathcal{A}_2}(\rho(\Pi) \cup \Pi')\}; \quad \textit{and}$$

4. modular *iff, for all programs $\Pi_1, \Pi_2 \in S_1$,*

$$\rho(\Pi_1 \cup \Pi_2) = \rho(\Pi_1) \cup \rho(\Pi_2).$$

In view of the requirement that $\mathcal{A}_1 \subseteq \mathcal{A}_2$, the general functions considered here may introduce new atoms. Clearly, if the given function is polynomial, the number of newly introduced atoms is also polynomial. Faithfulness guarantees that we can recover the stable models of the input program from the translated program. Strong faithfulness, on the other hand, states that we can add to a given program Π *any nested logic program* Π' and still retain, up to the original language, the semantics of the combined program $\Pi \cup \Pi'$ from $\rho(\Pi) \cup \Pi'$. Finally, modularity enforces that we can translate programs rule by rule.

It is quite obvious that any strongly faithful function is also faithful. Furthermore, strong faithfulness of function ρ implies that, for a given program Π, we can translate *any program part* Π_0 of Π whilst leaving the remaining part $\Pi \setminus \Pi_0$ unchanged, and determine the semantics of Π from $\rho(\Pi_0) \cup (\Pi \setminus \Pi_0)$. As well, for any function of form $\rho : NLP_{\mathcal{A}} \to NLP_{\mathcal{A}}$, strong faithfulness of ρ is equivalent to the condition that Π and $\rho(\Pi)$ are strongly equivalent, for any $\Pi \in NLP_{\mathcal{A}}$. Hence, strong faithfulness generalises strong equivalence.

Following [18, 19], we say that a function ρ as in Definition 1 is *PFM*, or that ρ is a *PFM-function*, iff it is polynomial, faithful, and modular. Analogously, we call ρ *PSM*, or a *PSM-function*, iff it is polynomial, strongly faithful, and modular.

It is easy to see that the composition of two PFM-functions is again a PFM-function; and likewise for PSM-functions. Furthermore, since any PSM-function is also PFM, in the following we focus on PSM-functions. In fact, in the next section, we construct a function $\sigma : NLP_{\mathcal{A}_1} \to DLP_{\mathcal{A}_2}$ (where $\mathcal{A}_2$ is a suitable extension of $\mathcal{A}_1$) which is PSM.

Next, we discuss some sufficient conditions guaranteeing that certain classes of functions are strongly faithful. We start with the following concept.

Definition 2. *Let $\rho : NLP_{\mathcal{A}_1} \to NLP_{\mathcal{A}_2}$ be a function such that $\mathcal{A}_1 \subseteq \mathcal{A}_2$, and let $INT_{\mathcal{A}_i}$ be the class of all HT-interpretations over $\mathcal{A}_i$ $(i = 1, 2)$.*

[1] A class S of sets is closed under unions providing $A, B \in S$ implies $A \cup B \in S$.

Then, the function $\alpha_\rho : INT_{\mathcal{A}_1} \times NLP_{\mathcal{A}_1} \to INT_{\mathcal{A}_2}$ *is called a* ρ-associated HT-embedding *iff, for each HT-interpretation* $\mathcal{I} = \langle I_H, I_T \rangle$ *over* $\mathcal{A}_1$*, each* $\Pi \in NLP_{\mathcal{A}_1}$*, and each* $w \in \{H, T\}$*,* $J_w \cap \mathcal{A}_1 = I_w$ *and* $J_w \setminus \mathcal{A}_1 \subseteq var(\rho(\Pi))$*, where* $\alpha_\rho(\mathcal{I}, \Pi) = \langle J_H, J_T \rangle$*.*

Furthermore, for any $G \subseteq INT_{\mathcal{A}_1}$ *and any* $\Pi \in NLP_{\mathcal{A}_1}$*, we define* $\alpha_\rho(G, \Pi) = \{\alpha_\rho(\mathcal{I}, \Pi) \mid \mathcal{I} \in G\}$*.*

Intuitively, a ρ-associated HT-embedding transforms HT-interpretations over the input alphabet $\mathcal{A}_1$ of ρ into HT-interpretations over the output alphabet $\mathcal{A}_2$ of ρ such that the truth values of the atoms in $\mathcal{A}_1$ are retained. The following definition strengthens these kinds of mappings:

Definition 3. *Let* ρ *be as in Definition 2, and let* α_ρ *be a* ρ*-associated HT-embedding. We say that* α_ρ *is a* ρ-associated HT-homomorphism *if, for any* $\mathcal{I}, \mathcal{I}' \in INT_{\mathcal{A}_1}$ *and any* $\Pi \in NLP_{\mathcal{A}_1}$*, the following conditions hold:*

1. $\mathcal{I}$ *is an HT-model of* Π *iff* $\alpha_\rho(\mathcal{I}, \Pi)$ *is an HT-model of* $\rho(\Pi)$*;*
2. $\mathcal{I}$ *is total iff* $\alpha_\rho(\mathcal{I}, \Pi)$ *is total;*
3. *if* $\mathcal{I} = \langle I_H, I_T \rangle$ *and* $\mathcal{I}' = \langle I'_H, I'_T \rangle$ *are HT-models of* Π*, then* $I_H \subset I'_H$ *and* $I_T = I'_T$ *holds precisely if* $J_H \subset J'_H$ *and* $J_T = J'_T$*, for* $\alpha_\rho(\mathcal{I}, \Pi) = \langle J_H, J_T \rangle$ *and* $\alpha_\rho(\mathcal{I}', \Pi) = \langle J'_H, J'_T \rangle$*; and*
4. *an HT-interpretation* $\mathcal{J}$ *over* $var(\rho(\Pi))$ *is an HT-model of* $\rho(\Pi)$ *only if* $\mathcal{J} \in \alpha_\rho(INT_{\mathcal{A}_1}, \Pi)$*.*

Roughly speaking, ρ-associated HT-homomorphisms retain the relevant properties of HT-interpretations for being equilibrium models with respect to transformation ρ. More specifically, the first three conditions take semantical and set-theoretical properties into account, respectively, whilst the last one expresses a specific "closure condition". The inclusion of the latter requirement is explained by observation that the first three conditions alone are not sufficient to exclude the possibility that there may exist some equilibrium model $\mathcal{I}$ of Π such that $\alpha_\rho(\mathcal{I}, \Pi)$ is not an equilibrium model of $\rho(\Pi)$. The reason for this is that the set $\alpha_\rho(INT_{\mathcal{A}_1}, \Pi)$, comprising the images of all HT-interpretations over $\mathcal{A}_1$ under α_ρ with respect to program Π, does, in general, not cover *all* HT-interpretations over $var(\rho(\Pi))$. Hence, for a general ρ-associated HT-embedding $\alpha_\rho(\cdot, \cdot)$, there may exist some HT-model of $\rho(\Pi)$ which is not included in $\alpha_\rho(INT_{\mathcal{A}_1}, \Pi)$ preventing $\alpha_\rho(\mathcal{I}, \Pi)$ from being an equilibrium model of $\rho(\Pi)$ albeit $\mathcal{I}$ is an equilibrium model of Π. The addition of the last condition in Definition 3, however, excludes this possibility, ensuring that all relevant HT-interpretations required for checking whether $\alpha_\rho(\mathcal{I}, \Pi)$ is an equilibrium model of $\rho(\Pi)$ are indeed considered. The following result can be shown:

Lemma 1. *For any function* $\rho : NLP_{\mathcal{A}_1} \to NLP_{\mathcal{A}_2}$ *with* $\mathcal{A}_1 \subseteq \mathcal{A}_2$*, if there is some* ρ*-associated HT-homomorphism, then* ρ *is faithful.*

From this, we obtain the following property:

Theorem 1. *Under the circumstances of Lemma 1, if* ρ *is modular and there is some* ρ*-associated HT-homomorphism, then* ρ *is strongly faithful.*

We make use of the last result for showing that the translation from nested logic programs into disjunctive logic programs, as discussed next, is PSM.

4 Main Construction

In this section, we show how logic programs with nested expressions can be efficiently mapped to disjunctive logic programs, preserving the semantics of the respective programs. Although results by Lifschitz *et al.* [24] already provide a reduction of nested logic programs into disjunctive ones (by employing additional transformation steps as given in [19]), that method is exponential in the worst case. This is due to the fact that the transformation relies on distributive laws, yielding an exponential increase of program size whenever the given program contains rules whose heads are in disjunctive normal form or whose bodies are in conjunctive normal form, and the respective expressions are not simple disjunctions or conjunctions of HT-literals.

To avoid such an exponential blow-up, our technique is based on the introduction of new atoms, called *labels*, abbreviating subformula occurrences. This method is derived from structure-preserving normal form translations [36, 33], which are frequently applied in the context of automated reasoning (cf., e.g., [2, 15] for general investigations about structure-preserving normal form translation in finite-valued Gödel logics, and [6, 7] for proof-theoretical issues of such translations for classical and intuitionistic logic). In contrast to theorem proving applications, where the main focus is to provide translations which are satisfiability (or, alternatively, validity) equivalent, here we are interested in somewhat stronger equivalence properties, viz. in the *reconstruction of the answer sets* of the original programs from the translated ones, which involves also an adequate handling of additional minimality criteria.

The overall structure of our translation can be described as follows. Given a nested logic program Π, we perform the following steps:

1. For each $r \in \Pi$, transform $H(r)$ and $B(r)$ into HT-NNF;
2. translate the program into a program containing only rules with conjunctions of HT-literals in their bodies and disjunctions of HT-literals in their heads;
3. eliminate double negations in bodies and heads; and
4. transform the resulting program into a disjunctive logic program, i.e., make all heads negation free.

Steps 1 and 3 are realised by using properties of logic programs as described in [24]; Step 2 represents the central part of our construction; and Step 4 exploits a procedure due to Janhunen [19].

In what follows, for any alphabet $\mathcal{A}$, we define the following new and disjoint alphabets:

- a set $\mathcal{A}_{\mathbf{L}} = \{\mathbf{L}_\phi \mid \phi \in \mathcal{L}_\mathcal{A}\}$ of labels; and
- a set $\overline{\mathcal{A}} = \{\overline{p} \mid p \in \mathcal{A}\}$ of atoms representing negated atoms.

Furthermore, $NLP^{nnf}_\mathcal{A}$ is the class of all nested logic programs over $\mathcal{A}$ which are in HT-NNF, and $GDLP^{hf}_\mathcal{A}$ is the class of all programs over $\mathcal{A}$ which are defined like generalised logic programs, except that HT-literals may occur in rules instead of ordinary literals.

We assume that for each of the above construction stages, Step i is realized by a corresponding function $\sigma_i(\cdot)$ ($i = 1, \ldots, 4$). The overall transformation is then described

by the composed function $\sigma = \sigma_4 \circ \sigma_3 \circ \sigma_2 \circ \sigma_1$, which is a mapping from the set $NLP_{\mathcal{A}}$ of all programs over $\mathcal{A}$ into the set $DLP_{\mathcal{A}^*}$ of all disjunctive logic program over $\mathcal{A}^* = \mathcal{A} \cup \mathcal{A}_{\mathbf{L}} \cup \bar{\mathcal{A}}$. More specifically,

$$\sigma_1 : NLP_{\mathcal{A}} \to NLP_{\mathcal{A}}^{nnf}$$

translates any nested logic program over $\mathcal{A}$ into a nested program in HT-NNF. Translation

$$\sigma_2 : NLP_{\mathcal{A}}^{nnf} \to GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}^{ht}$$

takes these programs and transforms their rules into simpler ones as described by Step 2, introducing new labels. These rules are then fed into mapping

$$\sigma_3 : GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}^{ht} \to GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}},$$

yielding generalised disjunctive logic programs. Finally,

$$\sigma_4 : GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}} \to DLP_{\mathcal{A}^*}$$

outputs standard disjunctive logic programs.

As argued in the following, each of these functions is PSM; hence, the overall function $\sigma = \sigma_4 \circ \sigma_3 \circ \sigma_2 \circ \sigma_1$ is PSM as well.

We continue with the technical details, starting with σ_1.

For the first step, we use the procedure $\nu(\cdot)$ from Proposition 3 to transform heads and bodies of rules into HT-NNF.

Definition 4. *The function $\sigma_1 : NLP_{\mathcal{A}} \to NLP_{\mathcal{A}}^{nnf}$ is defined by setting*

$$\sigma_1(\Pi) = \{\nu(H(r)) \leftarrow \nu(B(r)) \mid r \in \Pi\},$$

for any $\Pi \in NLP_{\mathcal{A}}$.

Since, for each expression ϕ, $\nu(\phi)$ is constructible in polynomial time and ϕ is HT-equivalent to $\nu(\phi)$ (cf. Proposition 3), the following result is immediate:

Lemma 2. *The translation σ_1 is PSM.*

The second step is realised as follows:

Definition 5. *The function $\sigma_2 : NLP_{\mathcal{A}}^{nnf} \to GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}^{ht}$ is defined by setting, for any $\Pi \in NLP_{\mathcal{A}}^{nnf}$,*

$$\sigma_2(\Pi) = \{\mathbf{L}_{H(r)} \leftarrow \mathbf{L}_{B(r)} \mid r \in \Pi\} \cup \gamma(\Pi),$$

where $\gamma(\Pi)$ is constructed as follows:

1. *for each HT-literal l occurring in Π, add the two rules*

$$\mathbf{L}_l \leftarrow l \quad \textit{and} \quad l \leftarrow \mathbf{L}_l;$$

2. *for each expression $\phi = (\phi_1 \wedge \phi_2)$ occurring in Π, add the three rules*

$$\mathbf{L}_\phi \leftarrow \mathbf{L}_{\phi_1} \wedge \mathbf{L}_{\phi_2}, \quad \mathbf{L}_{\phi_1} \leftarrow \mathbf{L}_\phi, \quad \mathbf{L}_{\phi_2} \leftarrow \mathbf{L}_\phi; \quad \textit{and}$$

3. *for each expression* $\phi = (\phi_1 \vee \phi_2)$ *occurring in* Π*, add the three rules*

$$\mathbf{L}_{\phi_1} \vee \mathbf{L}_{\phi_2} \leftarrow \mathbf{L}_{\phi}, \quad \mathbf{L}_{\phi} \leftarrow \mathbf{L}_{\phi_1}, \quad \mathbf{L}_{\phi} \leftarrow \mathbf{L}_{\phi_2}.$$

This definition is basically an adaption of a structure-preserving normal form translation for intuitionistic logic, as described in [26].

It is quite obvious that σ_2 is modular and, for each $\Pi \in NLP^{nnf}_{\mathcal{A}}$, we have that $\sigma_2(\Pi)$ is constructible in polynomial time. In order to show that σ_2 is strongly faithful, we define a suitable HT-homomorphism as follows.

Sublemma 1 *Let* σ_2 *be the translation defined above, and let* $\sigma_2^* : NLP_{\mathcal{A}} \rightarrow NLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$ *result from* σ_2 *by setting* $\sigma_2^*(\Pi) = \sigma_2(\Pi)$ *if* $\Pi \in NLP^{nnf}_{\mathcal{A}}$ *and* $\sigma_2^*(\Pi) = \Pi$ *if* $\Pi \in NLP_{\mathcal{A}} \setminus NLP^{nnf}_{\mathcal{A}}$.

Then, the function $\alpha_{\sigma_2^*} : INT_{\mathcal{A}} \times NLP_{\mathcal{A}} \rightarrow INT_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$*, defined as*

$$\alpha_{\sigma_2^*}(\mathcal{I}, \Pi) = \langle I_H \cup \lambda_H(\mathcal{I}, \Pi), I_T \cup \lambda_T(\mathcal{I}, \Pi)\rangle,$$

is a σ_2^**-associated HT-homomorphism, where*

$$\lambda_w(\mathcal{I}, \Pi) = \{\mathbf{L}_{\phi} \in \mathcal{A}_{\mathbf{L}} \cap var(\sigma_2^*(\Pi)) \mid \nu_{\mathcal{I}}(w, \phi) = 1\}$$

if $\Pi \in NLP^{nnf}_{\mathcal{A}}$*, and* $\lambda_w(\mathcal{I}, \Pi) = \emptyset$ *otherwise, for any* $w \in \{H, T\}$ *and any HT-interpretation* $\mathcal{I} = \langle I_H, I_T\rangle$ *over* $\mathcal{A}$*.*

Hence, according to Theorem 1, σ_2^* is strongly faithful. As a consequence, σ_2 is strongly faithful as well. Thus, the following holds:

Lemma 3. *The function* σ_2 *is PSM.*

For Step 3, we use a method due to Lifschitz *et al*. [24] for eliminating double negations in heads and bodies of rules. The corresponding function σ_3 is defined as follows:

Definition 6. *Let* $\sigma_3 : GDLP^{ht}_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}} \rightarrow GDLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$ *be the function obtained by replacing, for each given program* $\Pi \in GDLP^{ht}_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$*, each rule* $r \in \Pi$ *of form*

$$\phi \vee \neg\neg p \leftarrow \psi \quad \textit{by} \quad \phi \leftarrow \psi \wedge \neg p,$$

as well as each rule of form

$$\phi \leftarrow \psi \wedge \neg\neg q \quad \textit{by} \quad \phi \vee \neg q \leftarrow \psi,$$

where ϕ *and* ψ *are expressions and* $p, q \in \mathcal{A}$*.*

As shown in [24], performing replacements of the above type results in programs which are strongly equivalent to the original programs. In fact, it is easy to see that such replacements yield transformed programs which are strongly faithful to the original ones. Since these transformations are clearly modular and constructible in polynomial time, we obtain that σ_3 is PSM.

Lemma 4. *The function σ_3 is PSM.*

Finally, we eliminate remaining negations possibly occurring in the heads of rules. To this end, we employ a procedure due to Janhunen [19] (for an alternative method, cf. [17]).

Definition 7. *Let $\sigma_4 : GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}} \to DLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}\cup\bar{\mathcal{A}}}$ be the function defined by setting, for any program $\Pi \in GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$,*

$$\sigma_4(\Pi) = \bar{\Pi} \cup \{\bot \leftarrow (p \wedge \overline{p}),\ \overline{p} \leftarrow \neg p \mid \neg p \text{ occurs in the head of some rule in } \Pi\},$$

where $\bar{\Pi}$ results from Π by replacing each occurrence of a literal $\neg p$ in the head of a rule in Π by $\overline{p}$.

Janhunen showed that replacements of the above kind lead to a transformation which is PFM. As a matter of fact, since his notion of faithfulness is somewhat stricter than ours, the results in [19] actually imply that

$$AS_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}(\Pi \cup \Pi') = \{I \cap (\mathcal{A} \cup \mathcal{A}_\mathbf{L}) \mid I \in AS_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}\cup\bar{\mathcal{A}}}(\sigma_4(\Pi) \cup \Pi')\},$$

for any $\Pi, \Pi' \in GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$. However, we need a stronger condition here, viz. that the above equation holds for any $\Pi \in GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$ and any $\Pi' \in NLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$. We show this by appeal to Theorem 1.

Sublemma 2 *Let σ_4 be the translation defined above, and let $\sigma_4^* : NLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}} \to NLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}\cup\bar{\mathcal{A}}}$ result from σ_4 by setting $\sigma_4^*(\Pi) = \sigma_4(\Pi)$ if $\Pi \in GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$ and $\sigma_4^*(\Pi) = \Pi$ if $\Pi \in NLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}} \setminus GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$.*

Then, the function $\alpha_{\sigma_4^*} : INT_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}} \times NLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}} \to INT_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}\cup\bar{\mathcal{A}}}$, defined as

$$\alpha_{\sigma_4^*}(\mathcal{I}, \Pi) = \langle I_H \cup \kappa(\mathcal{I}, \Pi), I_T \cup \kappa(\mathcal{I}, \Pi)\rangle,$$

is a σ_4^*-associated HT-homomorphism, where

$$\kappa(\mathcal{I}, \Pi) = \{\overline{p} \mid \neg p \text{ occurs in the head of some rule in } \Pi \text{ and } p \notin I_T\}$$

if $\Pi \in GDLP_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$, and $\kappa(\mathcal{I}, \Pi) = \emptyset$ otherwise, for any HT-interpretation $\mathcal{I} = \langle I_H, I_T\rangle$ over $\mathcal{A} \cup \mathcal{A}_\mathbf{L}$.

Observe that, in contrast to the definition of function $\alpha_{\sigma_2^*}$ from Sublemma 1, here the same set of newly introduced atoms is added to both worlds. As before, we obtain that σ_4^* is strongly faithful, and hence that σ_4 is strongly faithful as well.

Lemma 5. *The function σ_4 is PSM.*

Summarising, we obtain our main result, which is as follows:

Theorem 2. *Let $\sigma_1, \ldots, \sigma_4$ be the functions defined above. Then, the composed function $\sigma = \sigma_4 \circ \sigma_3 \circ \sigma_2 \circ \sigma_1$, mapping nested logic programs over alphabet $\mathcal{A}$ into disjunctive logic programs over alphabet $\mathcal{A} \cup \mathcal{A}_\mathbf{L} \cup \bar{\mathcal{A}}$, is polynomial, strongly faithful, and modular.*

Since strong faithfulness implies faithfulness, we get the following corollary:

Corollary 1. *For any nested logic program Π over $\mathcal{A}$, the answer sets of Π are in a one-to-one correspondence to the answer sets of $\sigma(\Pi)$, determined by the following equation:*

$$AS_{\mathcal{A}}(\Pi) = \{I \cap \mathcal{A} \mid I \in AS_{\mathcal{A}^*}(\sigma(\Pi))\},$$

where $\mathcal{A}^* = \mathcal{A} \cup \mathcal{A}_{\mathbf{L}} \cup \bar{\mathcal{A}}$.

We conclude with a remark concerning the construction of function σ_2. As pointed out previously, this mapping is based on a structure-preserving normal form translation for intuitionistic logic, as described in [26]. Besides the particular type of translation used here, there are also other, slightly improved structure-preserving normal form translations in which fewer rules are introduced, depending on the polarity of the corresponding subformula occurrences. However, although such optimised methods work in monotonic logics, they are not sufficient in the present setting. For instance, in a possible variant of translation σ_2 based on the polarity of subformula occurrences, instead of introducing all three rules for an expression ϕ of form $(\phi_1 \wedge \phi_2)$, only $\mathbf{L}_{\phi} \leftarrow \mathbf{L}_{\phi_1} \wedge \mathbf{L}_{\phi_2}$ is used if ϕ occurs in the body of some rule, or both $\mathbf{L}_{\phi_1} \leftarrow \mathbf{L}_{\phi}$ and $\mathbf{L}_{\phi_2} \leftarrow \mathbf{L}_{\phi}$ are used if ϕ occurs in the head of some rule, and analogous manipulations are performed for atoms and disjunctions. Applying such an encoding to $\Pi = \{p \leftarrow;\ q \leftarrow;\ r \vee (p \wedge q) \leftarrow\}$ over $\mathcal{A}_0 = \{p, q, r\}$ yields a translated program possessing two answer sets, say S_1 and S_2, such that $S_1 \cap \mathcal{A}_0 = \{p, q\}$ and $S_2 \cap \mathcal{A}_0 = \{p, q, r\}$, although only $\{p, q\}$ is an answer set of Π.

5 Conclusion

We have developed a translation of logic programs with nested expressions into disjunctive logic programs. We have proven that our translation is polynomial, strongly faithful, and modular. This allows us to utilise off-the-shelf disjunctive logic programming systems for interpreting nested logic programs. In fact, we have implemented our translation as a front end for the system `DLV` [8, 9]. The corresponding compiler is implemented in Prolog and can be downloaded from the Web at URL

`http://www.cs.uni-potsdam.de/~torsten/nlp`.

Our technique is based on the introduction of new atoms, abbreviating subformula occurrences. This method has its roots in structure-preserving normal form translations [36, 33], which are frequently used in automated deduction. In contrast to theorem proving applications, however, where the main focus is to provide satisfiability (or, alternatively, validity) preserving translations, we are concerned with much stronger equivalence properties, involving additional minimality criteria, since our goal is to *reconstruct* the answer sets of the original programs from the translated ones.

With the particular labeling technique employed here, our translation avoids the risk of an exponential blow-up in the worst-case, as faced by a previous approach of Lifschitz *et al.* [24] due to the usage of distributivity laws. However, this is not to say that our translation is *always* the better choice. As in classical theorem proving, it is

rather a matter of experimental studies under which circumstances which approach is the more appropriate one. To this end, besides the implementation of our structural translation, we have also implemented the distributive translation into disjunctive logic programs in order to conduct experimental results. These experiments are subject to current research.

Also, we have introduced the concept of *strong faithfulness*, as a generalisation of (standard) faithfulness and strong equivalence. This allows us, for instance, to translate, in a semantics-preserving way, arbitrary program parts and leave the remaining program unaffected.

Acknowledgements. This work was partially supported by the Austrian Science Fund (FWF) under grants P15068-INF and N Z29-INF. The authors would like to thank Agata Ciabattoni for pointing out some relevant references.

References

1. M. Baaz, U. Egly, and A. Leitsch. Normal Form Transformations. In *Handbook of Automated Reasoning*, volume I, chapter 5, pages 273–333. Elsevier Science B.V., 2001.
2. M. Baaz and C. G. Fermüller. Resolution-based Theorem Proving for Many-valued Logics. *Journal of Symbolic Computation*, 19(4):353–391, 1995.
3. C. Baral and C. Uyan. Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In *Proc. LPNMR-01*, pages 186–199, 2001.
4. M. Cadoli, A. Giovanardi, and M. Schaerf. An Algorithm to Evaluate Quantified Boolean Formulae. In *Proc. AAAI-98*, pages 262–267, 1998.
5. D. de Jongh and L. Hendriks. Characterization of Strongly Equivalent Logic Programs in Intermediate Logics. Technical report, 2001. Preprint at `http://turing.wins.uva.nl/~lhendrik/`.
6. U. Egly. On Different Structure-Preserving Translations to Normal Form. *Journal of Symbolic Computation*, 22(2):121–142, 1996.
7. U. Egly. On Definitional Transformations to Normal Form for Intuitionistic Logic. *Fundamenta Informaticae*, 29(1,2):165–201, 1997.
8. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Nonmonotonic Reasoning. In *Proc. LPNMR-97*, pages 363–374, 1997.
9. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons and Benchmarks. In *Proc. KR-98*, pages 406–417, 1998.
10. R. Feldmann, B. Monien, and S. Schamberger. A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In *Proc. AAAI-00*, pages 285–290, 2000.
11. M. Gelfond, M. Balduccini, and J. Galloway. Diagnosing Physical Systems in A-Prolog. In *Proc. LPNMR-01*, pages 213–225, 2001.
12. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
13. E. Giunchiglia, M. Narizzano, and A. Tacchella. QUBE: A System for Deciding Quantified Boolean Formulas Satisfiability. In *Proc. IJCAR-01*, pages 364–369, 2001.
14. K. Gödel. Zum intuitionistischen Aussagenkalkül. *Anzeiger der Akademie der Wissenschaften in Wien*, pages 65–66, 1932.
15. R. Hähnle. Short Conjunctive Normal Forms in Finitely Valued Logics. *Journal of Logic and Computation*, 4(6):905–927, 1994.

16. K. Heljanko and I. Niemelä. Bounded LTL Model Checking with Stable Models. In *Proc. LPNMR-01*, pages 200–212, 2001.
17. K. Inoue and C. Sakama. Negation as Failure in the Head. *Journal of Logic Programming*, 35(1):39–78, 1998.
18. T. Janhunen. On the Intertranslatability of Autoepistemic, Default and Priority Logics, and Parallel Circumscription. In *Proc. JELIA-98*, pages 216–232, 1998.
19. T. Janhunen. On the Effect of Default Negation on the Expressiveness of Disjunctive Rules. In *Proc. LPNMR-01*, pages 93–106, 2001.
20. H. Kleine-Büning, M. Karpinski, and A. Flögel. Resolution for Quantified Boolean Formulas. *Information and Computation*, 117(1):12–18, 1995.
21. R. Letz. Advances in Decision Procedures for Quantified Boolean Formulas. In *Proc. IJCAR-01 Workshop on Theory and Applications of Quantified Boolean Formulas*, pages 55–64, 2001.
22. V. Lifschitz. Answer Set Planning. In *Proc. ICLP-99*, pages 23–37, 1999.
23. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
24. V. Lifschitz, L. Tang, and H. Turner. Nested Expressions in Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.
25. F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proc. KR-02*, pages 170–176, 2002.
26. G. Mints. Resolution Strategies for the Intuitionistic Logic. In *Constraint Programming: NATO ASI Series*, pages 282–304. Springer, 1994.
27. I. Niemelä and P. Simons. Smodels: An Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs. In *Proc. LPNMR-97*, pages 420–429, 1997.
28. D. Pearce. A New Logical Characterisation of Stable Models and Answer Sets. In *Non-Monotonic Extensions of Logic Programming*, pages 57–70. Springer, 1997.
29. D. Pearce. From Here to There: Stable Negation in Logic Programming. In *What is Negation?* Kluwer, 1999.
30. D. Pearce, I. de Guzmán, and A. Valverde. A Tableau Calculus for Equilibrium Entailment. In *Proc. TABLEAUX-00*, pages 352–367, 2000.
31. D. Pearce, I. de Guzmán, and A. Valverde. Computing Equilibrium Models Using Signed Formulas. In *Proc. CL-00*, pages 688–702, 2000.
32. D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proc. EPIA-01*, pages 306–320. Springer, 2001.
33. D. A. Plaisted and S. Greenbaum. A Structure Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.
34. J. Rintanen. Improvements to the Evaluation of Quantified Boolean Formulae. In *Proc. IJCAI-99*, pages 1192–1197, 1999.
35. J. Siekmann and G. Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.
36. G. Tseitin. On the Complexity of Proofs in Propositional Logics. *Seminars in Mathematics*, 8, 1970. Reprinted in [35].