D. Pearce[1], T. Schaub[2], V. Sarsakov[2],
H. Tompits[3] und S. Woltran[3]

On Implementing Nested Logic Programms: Overview and Comparisons

[1]ESCET, Universidad Rey Juan Carlos
Campus de Mostoles, DI-236, Calle Tulipan s/n, E-28933 Madrid, Spain
email: `D.Pearce@escet.urjc.es`

[2]Institut für Informatik, Universität Potsdam,
Postfach 60 15 53, D–14415 Potsdam, Germany
email: `sarsakov@rz.uni-potsdam.de`
email: `torsten@cs.uni-potsdam.de`

[3]Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Wien, Austria
email: `[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-set semantics. These programs generalise the well-known class of disjunctive logic programs, providing a very flexible and compact framework for knowledge representation and reasoning. In this paper, we discuss several recent approaches from the literature for implementing nested logic programs.

1 Introduction

Lifschitz, Tang, and Turner [24] recently extended the answer-set semantics [14] 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 as important tools for knowledge representation and reasoning. This is reflected by the fact that several practicably relevant applications have been developed recently using these types of programs (cf., e.g., [22, 1, 12]), which in turn is largely fostered by the availability of efficient solvers for the answer-set semantics, most notably `DLV` [7, 8] and `Smodels` [26]. Nested logic programs represent a promising approach to extend these formalisms by means of an enhanced expressibility yet keeping the underlying semantics as simple as the more restricted classes.

In this brief survey, we present three approaches for implementing nested logic programs. Two of the considered techniques deal with translations of nested logic programs into disjunctive logic programs, thus providing front-ends to `DLV` and `Smodels` for computing the answer sets of nested programs. The third approach follows a different strategy, albeit still relying on a certain reduction technique. However, here the outcome is not a logic program, but a formula in *quantified Boolean logic*.

Regarding the first two approaches, translations from nested logic programs into disjunctive logic programs are usually done in at least two steps. First, source programs are reduced to *generalised disjunctive logic programs*. In contrast to ordinary disjunctive programs, generalised disjunctive logic programs are characterised that not only rule bodies may contain negative literals but also rule heads. Secondly, to get rid of these negations in the heads, methods from [18] can be applied.

Concerning the first step, results by Lifschitz, Tang, and Turner [24] provide a technique to translate nested logic programs into generalised disjunctive ones, which is essentially based on distributivity laws. Similar to classical logic, such a translation has the advantage of being "language preserving", i.e., no additional variables have to be introduced. Moreover, the answer sets of the resultant generalised disjunctive logic program, Π_D, exactly match the answer sets of the original nested logic program, Π. In fact, this translation turned out to satisfy an even stronger relation between both programs Π and Π_D, namely the concept of *strong equivalence* [23], expressing that one can add *any nested program* Π' to Π and Π_D, and still get equal answer sets of the combined programs $\Pi \cup \Pi'$ and $\Pi_D \cup \Pi'$. However, this approach suffers from the drawback of an exponential increase of program size of the resultant disjunctive logic programs in the worst case.

In [32] it was shown 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 belong to the same complexity classes as the corresponding problems for disjunctive logic programs, viz. they lie at the second level of the polynomial hierarchy (cf. [6]). Hence, nested logic programs can in turn be efficiently reduced to disjunctive logic programs, and thus avoiding the drawback of an exponential worst-case behavior as faced for the language-preserving translation from [24]. Such a translation is realised in [31], where a reduction of nested logic programs into disjunctive ones is discussed which is guaranteed to be polynomial in time and space. The construction of this translation relies on the introduction of new *labels*, abbreviating subformula occurrences. This technique is derived from *structure-preserving normal form translations* [35, 33], frequently employed in the context of automated deduction.

However, the mentioned complexity results from [32] also motivate a third reduction method. Indeed, [32] considers a reduction of nested logic programs into formulas of quantified Boolean logic, also referred to as *quantified Boolean formulas* (QBFs). More precisely, they deal with QBFs possessing exactly one quantifier alternation. This class of formulas is known as the prototypical problem for the second level of the polynomial hierarchy. Such a method can be seen as a natural generalisation of a similar method successfully applied to problems in NP (see, e.g., [20]). Since there now exist efficient QBF-solvers [10, 15, 21], this reduction technique yields a practicably relevant approach as well. Moreover, the reduction to QBFs enjoys a linear worst-case complexity.

The remainder of the paper is laid out as follows. First, we review the basic definitions and notation about logic programs under the answer-set semantics. In Section 3, we discuss reduction techniques for deriving disjunctive logic programs from nested programs. Section 4 briefly introduces the notion of QBFs and reviews the reduction to QBFs from [32]. In Section 5, we describe implementational issues and give a pointer to experimental results. Section 6 supplies some concluding remarks.

2 Preliminaries

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. 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 $\mathcal{A}$. The set of all Boolean expression over an alphabet $\mathcal{A}$ is given by $\mathcal{L}_{\mathcal{A}}$.

In what follows, we associate to each rule r a corresponding formula $\hat{r} = B(r) \supset H(r)$ 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 alphabet $\mathcal{A}$ 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 $I \subseteq \mathcal{A}$ 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 set of all answer sets of program Π over $\mathcal{A}$ is denoted by $AS_{\mathcal{A}}(\Pi)$.

Example 9. Consider a logic program Π whose only rule is given by

$$p \leftarrow (\neg q \vee r) \wedge (q \vee \neg s).$$

Let us check whether a candidate $I = \{p\}$ is a stable model of Π. Observe that Π^I consists of the rule

$$p \ \leftarrow (\top \vee r) \wedge (q \wedge \top),$$

and therefore Π^I is equivalent to $p \leftarrow \top$, which possesses $\{p\}$ as its unique minimal model. Thus, I is indeed a stable model of Π. In fact, there are no other stable models of Π.

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.

Recently, it was shown that the basic decision problems associated with logic programs under the stable-model semantics have the same complexity for nested logic programs as well as for disjunctive logic programs. For the latter, in [6] it was shown that the corresponding problems are located at the second level of the polynomial hierarchy. These results have been extended to nested logic programs in [32] where it was shown that deciding whether a given nested logic program Π has at least one stable model is complete for Σ_2^P (readers not familiar with the basic concepts of complexity theory are referred to, e.g., [27] for a comprehensive treatise). Additionally, in [32] it is also shown that brave and skeptical entailment from nested programs are complete for complexity classes belonging to the second level of the polynomial hierarchy, respectively. Moreover, emphasising another relevant result from [32], testing whether two nested logic programs are strongly equivalent is in co-NP, and thus easier[15] as the basic reasoning tasks.

3 Translations into Disjunctive Logic Programs

In this section, we present different methods for reducing nested programs into disjunctive ones. We start with some basic conditions, following [31].

[15] Under the assumption that the polynomial hierarchy does not collapse.

Definition 13. *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.*[16] *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 $\Pi \in S_1$, there is a one-to-one correspondence between the answer sets of Π and the answer sets of $\rho(\Pi)$ such that*
$$AS_{\mathcal{A}_1}(\Pi) = \{J \cap \mathcal{A}_1 \mid J \in AS_{\mathcal{A}_2}(\rho(\Pi))\};$$
3. strongly faithful *iff, for all programs $\Pi \in S_1$ and all programs $\Pi' \in NLP_{\mathcal{A}_1}$, there is a one-to-one correspondence between the answer sets of Π and the answer sets of $\rho(\Pi) \cup \Pi'$ such that*
$$AS_{\mathcal{A}_1}(\Pi \cup \Pi') = \{J \cap \mathcal{A}_1 \mid J \in AS_{\mathcal{A}_2}(\rho(\Pi) \cup \Pi')\};$$
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, 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 [17, 18], we say that a function ρ as in Definition 13 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.

As mentioned in Section 1, translating nested logic programs into disjunctive logic programs may be executed by composing two faithful functions, σ and ρ, where σ translates a nested logic program into a generalised disjunctive program and ρ takes the resultant program and outputs an ordinary disjunctive program. Hence, $\rho(\sigma(\Pi))$ yields the desired output.

The next two subsections present two alternative realisations for σ. Afterwards, we briefly recall some results from the literature for obtaining ρ.

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

3.1 Language-Preserving Translation

First of all, we introduce the following notation for substitutions in programs.

Let Π a logic program, let φ be an occurrence of an expression or a rule in Π, and let ψ be an expression, a rule, or a set of rules. Then, Π^{φ}_{ψ} denotes the program resulting from Π by replacing φ in Π by ψ.

Proposition 3 ([24]). *NLPs Π and Π^{φ}_{ψ} are strongly equivalent whenever*

(1) $\varphi = \phi_1 \vee (\phi_2 \wedge \phi_3)$, $\psi = (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$;
(2) $\varphi = \phi_1 \wedge (\phi_2 \vee \phi_3)$, $\psi = (\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)$;
(3) $\varphi = \neg(\phi_1 \wedge \phi_2)$, $\psi = \neg\phi_1 \vee \neg\phi_2$;
(4) $\varphi = \neg(\phi_1 \vee \phi_2)$, $\psi = \neg\phi_1 \wedge \neg\phi_2$;
(5) $\varphi = \neg\neg\neg\phi$, $\psi = \neg\phi$;
(6) $\varphi = (\phi_1 \wedge \phi_2) \leftarrow \phi_3$, $\psi = \{\phi_1 \leftarrow \phi_3, \phi_2 \leftarrow \phi_3\}$;
(7) $\varphi = \phi_1 \leftarrow (\phi_2 \vee \phi_3)$, $\psi = \{\phi_1 \leftarrow \phi_2, \phi_1 \leftarrow \phi_3\}$;
(8) $\varphi = \phi_1 \leftarrow (\phi_2 \wedge \neg\neg\phi_3)$, $\psi = (\phi_1 \vee \neg\phi_3) \leftarrow \phi_2$;
(9) $\varphi = (\phi_1 \vee \neg\neg\phi_3) \leftarrow \phi_2$, $\psi = \phi_1 \leftarrow (\phi_2 \wedge \neg\phi_3)$.

Note that replacements (1)–(5) substitute expressions within a rule, (6) and (7) replace a rule by two new rules, and (8), as well as (9), exchange a rule.

The following properties follow from results given in [24]:

Proposition 4. *There exists a function $\sigma_1 : NLP_{\mathcal{A}} \to GDLP_{\mathcal{A}}$, defined as a deterministic sequence of above substitutions, such that the following properties hold, for any $\Pi \in NLP_{\mathcal{A}}$:*

1. *$Var(\Pi) = Var(\sigma_1[\Pi])$;*
2. *Π and $\sigma_1[\Pi]$ are strongly equivalent;*
3. *σ_1 is modular; and*
4. *there is a program Π' such that $\sigma_1[\Pi']$ is exponential in the size of Π', i.e., σ_1 is not polynomial.*

Note that since the alphabet is not extended when applying σ_1, we can call σ_1 strongly faithful, as well. We proceed by illustrating the functioning of translation σ_1 on Example 9.

Example 10. Without giving an exact definition of σ_1, the basic idea of this transformation is to apply rules from Proposition 3 in such a way that first we have, on the one hand, conjunctions of disjunctions of literals in the heads of the rules, and, on the other hand, disjunctions of conjunctions of literals in the bodies of the rules. Under a literal we understand an atom preceeded by none, one, or two negations.[17] Secondly, these rules are then splitted, in view of Rules

[17] Observe that Rule (5) allows reducing a triple negation to a single negation, but we do not have such rule for double negation. This stems from the logical underpinning of the stable-model semantics which is based on the sub-classical logic of here-and-there (cf. [28, 29, 23] for further details).

(6) and (7) from Proposition 3, and finally, double negated literals are eliminated by Rules (8) and (9).

Hence, in our example program $\Pi = \{p \leftarrow (\neg q \vee r) \wedge (q \vee \neg s)\}$, we have to transform the body to get the desired disjunction of conjunctions. Using Rule (2) iteratively, we first transform the body $(\neg q \vee r) \wedge (q \vee \neg s)$ to

$$\big((\neg q \vee r) \wedge q\big) \vee \big((\neg q \vee r) \wedge \neg s\big)$$

and then to

$$\phi = (\neg q \wedge q) \vee (r \wedge q) \vee (\neg q \wedge \neg s) \vee (r \wedge \neg s).$$

Finally, applying (7) to $p \leftarrow \phi$ yields the program

$$\begin{aligned}\sigma_1[\Pi] = \{\, & p \leftarrow \neg q \wedge q,\\ & p \leftarrow r \wedge q,\\ & p \leftarrow \neg q \wedge \neg s,\\ & p \leftarrow r \wedge \neg s\}.\end{aligned}$$

Here, $\sigma_1[\Pi]$ is already a disjunctive logic program, but this does of course not hold in general. It is easy to verify that $\{p\}$ is the only stable model of $\sigma_1[\Pi]$, since the only rule applicable at the first step is $p \leftarrow \neg q \wedge \neg s$, and p does not occur in any body of a rule in $\sigma_1[\Pi]$.

3.2 Structure-Preserving Translation

Following [30], we give here a slightly improved version of the translation discussed in [31]. The basic idea is to abbreviate subformula occurrences by new *labels*, and then to build disjunctive rules by these labels in a faithful manner. To this end, we first define the following new and disjoint alphabet $\mathcal{A}_{\mathbf{L}} = \{\mathbf{L}_\phi \mid \phi \in \mathcal{L}_\mathcal{A}\}$ of labels.

The following definition is basically an adaption of a structure-preserving normal form translation for intuitionistic logic, as described in [25].

Definition 14. *The function $\mathcal{T}_2 : NLP_\mathcal{A} \to DLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$ is defined by setting, for any $\Pi \in NLP_\mathcal{A}$,*

$$\mathcal{T}_2[\Pi] = \{\mathbf{L}_{H(r)} \leftarrow \mathbf{L}_{B(r)} \mid r \in \Pi\} \cup \gamma(\Pi),$$

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

1. *for each atom p occurring in Π, add the two rules*

$$\mathbf{L}_p \leftarrow p \quad \textit{and} \quad p \leftarrow \mathbf{L}_p;$$

2. *for each expression $\phi = \neg\phi_1$ occurring in Π, add the two rules*

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

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

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

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

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

Theorem 12. *The function $\mathcal{T}_2$ is PSM.*

For illustration, we apply $\mathcal{T}_2$ on our example program.

Example 11. Recall program Π from Example 9, which can be written as $\{p \leftarrow \phi\}$ with $\phi = (\neg q \vee r) \wedge (q \vee \neg s)$. We have,

$$\begin{aligned}
\mathcal{T}_2[\Pi] = {} & \{\mathbf{L}_p \leftarrow \mathbf{L}_\phi\} \cup & (1)\\
& \{\mathbf{L}_\phi \leftarrow \mathbf{L}_{\neg q \vee r} \wedge \mathbf{L}_{q \vee \neg s}, \quad \mathbf{L}_{\neg q \vee r} \leftarrow \mathbf{L}_\phi, \quad \mathbf{L}_{q \vee \neg s} \leftarrow \mathbf{L}_\phi\} \cup & (2)\\
& \{\mathbf{L}_{\neg q} \vee \mathbf{L}_r \leftarrow \mathbf{L}_{\neg q \vee r}, \quad \mathbf{L}_{\neg q \vee r} \leftarrow \mathbf{L}_{\neg q}, \quad \mathbf{L}_{\neg q \vee r} \leftarrow \mathbf{L}_r\} \cup & (3)\\
& \{\mathbf{L}_q \vee \mathbf{L}_{\neg s} \leftarrow \mathbf{L}_{q \vee \neg s}, \quad \mathbf{L}_{q \vee \neg s} \leftarrow \mathbf{L}_q, \quad \mathbf{L}_{q \vee \neg s} \leftarrow \mathbf{L}_{\neg s}\} \cup & (4)\\
& \{\mathbf{L}_{\neg v} \leftarrow \neg \mathbf{L}_v, \quad \bot \leftarrow \mathbf{L}_v \wedge \mathbf{L}_{\neg v} \mid v \in \{q, s\}\} \cup & (5)\\
& \{\mathbf{L}_v \leftarrow v, v \leftarrow \mathbf{L}_v \mid v \in \{p, q, r, s\}\}. & (6)
\end{aligned}$$

Observe that Rule (1) results from labeling the original rule in Π; (2) contains the rules for the conjunction in ϕ; (3) and (4) deal with subformulas which are disjunctions; (5) provides the rules for negation; and, finally, (6) contains all rules for labeling the atoms occurring in Π. Omitting further details, we claim that the only stable model of $\mathcal{T}_2[\Pi]$ is given by

$$I = \{p, \mathbf{L}_p, \mathbf{L}_{\neg q}, \mathbf{L}_{\neg s}, \mathbf{L}_{\neg q \vee r}, \mathbf{L}_{q \vee \neg s}, \mathbf{L}_\phi\}.$$

Since $I \cap \mathcal{A} = \{p\}$, we have that faithfulness holds, i.e., $AS_{\mathcal{A}}(\Pi) = \{I \cap \mathcal{A} \mid I \in AS_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}(\mathcal{T}_2[\Pi])\}$.

Observe, that $\mathcal{T}_2$ already yields disjunctive programs without negations in their heads. However, one can consider slightly different variants of structure-preserving translations (as in [31]), or invoke some optimisations, like, e.g., refraining from labeling atoms and literals. Then, the function given in the next section has to be invoked additionally in order to yield disjunctive logic programs.

3.3 Eliminating Negations in the Heads

So far, we have shown two methods to faithfully translate nested logic programs into syntactically simpler ones. For the translation in Section 3.1, it remains to reduce the resultant programs to (ordinary) disjunctive ones, i.e., to get rid of the remaining negated atoms in the heads of the programs. We proceed by recalling the translation due to Janhunen [18] for this purpose (for an alternative method, cf. [16]). To this end, we define the following new alphabet $\bar{\mathcal{A}} = \{\overline{p} \mid p \in \mathcal{A}\}$ of atoms representing negated atoms.

Proposition 5 ([18]). *Let* $\rho : GDLP_{\mathcal{A}} \to DLP_{\mathcal{A}\cup\bar{\mathcal{A}}}$ *be the function defined by setting, for any* $\Pi \in GDLP_{\mathcal{A}}$,

$$\rho[\Pi] = \Pi' \cup \{\bot \leftarrow (p \wedge \overline{p}), \overline{p} \leftarrow \neg p \mid \neg p \textit{ occurs in the head of some rule in } \Pi\},$$

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

Then, ρ *is a PFM-function.*

This result was strengthened in [31] by showing that ρ is actually PSM. Thus, we define the following composed strongly faithful function.

Definition 15. *Let* $\mathcal{T}_1 : NLP_{\mathcal{A}} \to DLP_{\mathcal{A}\cup\bar{\mathcal{A}}}$ *be the function defined by setting* $\mathcal{T}_1[\Pi] = \rho[\sigma_1[\Pi]]$*, for any* $\Pi \in NLP_{\mathcal{A}}$.

Theorem 13. $\mathcal{T}_1$ *is a PSM-function.*

This concludes our discussion of approaches translating nested programs into programs interpretable by existing logic-programming solvers, like `DLV` [7, 8] or `Smodels` [26] (together with its front-end for disjunctions [19]).

4 Translation into Quantified Boolean Logic

In this section we extend the reduction method to other formalisms for capturing nested logic programs. One such suitable candidate is quantified Boolean logic. From a theoretical point of view, this is motivated by the fact that the problem of evaluating quantified Boolean formulas with one quantifier alternation lies at the second level of the polynomial hierarchy. A practical motivation, on the other hand, is the availability of efficient solvers for this logic.

In this section, we present the QBF-encoding given in [32]. In fact, this encoding generalises previous QBF-reductions of disjunctive logic programs given in [5, 4]. We start with a brief introduction to quantified Boolean logic.

4.1 Interlude: Quantified Boolean Logic

Quantified Boolean logic is an extension of classical propositional logic in which formulas are permitted to contain quantifications over propositional variables. More formally, the language of quantified Boolean logic comprises the symbols of propositional logic, together with unary operators of form $\forall p$ (where p is some atom), called *universal quantifiers*. Similar to first-order logic, $\exists p$ is defined as the operator $\neg\forall p\neg$ and is called an *existential quantifier*.

The precise semantical meaning of QBFs is defined as follows. First, some ancillary notation. An occurrence of a propositional variable p in a QBF Φ is *free* iff it does not appear in the scope of a quantifier $\mathsf{Q}p$ ($\mathsf{Q} \in \{\forall, \exists\}$), otherwise the occurrence of p is *bound*. If Φ contains no free variable occurrences, then Φ is *closed*, otherwise Φ is *open*. Furthermore, $\Phi[p/\phi]$ denotes the result of substituting in Φ each free occurrence of a variable p by a formula ϕ.

By an *interpretation*, I, we understand a set of atoms. Informally, an atom p is true under I iff $p \in I$. In general, the truth value, $\nu_I(\Phi)$, of a QBF Φ under an interpretation I is recursively defined as follows:

1. if $\Phi = \top$, then $\nu_I(\Phi) = 1$;
2. if $\Phi = p$ is an atom, then $\nu_I(\Phi) = 1$ if $p \in I$, otherwise $\nu_I(\Phi) = 0$;
3. if $\Phi = \neg\Psi$, then $\nu_I(\Phi) = 1$ if $\nu_I(\Psi) = 0$, otherwise $\nu_I(\Phi) = 0$;
4. if $\Phi = (\Phi_1 \wedge \Phi_2)$, then $\nu_I(\Phi) = 1$ if $\nu_I(\Phi_1) = \nu_I(\Phi_2) = 1$, otherwise $\nu_I(\Phi) = 0$;
5. if $\Phi = \forall p \Psi$, then $\nu_I(\Phi) = 1$ if $\nu_I(\Psi[p/\top]) = \nu_I(\Psi[p/\bot]) = 1$, otherwise $\nu_I(\Phi) = 0$.

The truth conditions for $\bot$, $\vee$, $\supset$, $\equiv$, and $\exists$ follow from the above in the usual way. Obviously, we have that

$$\nu_I(\forall p\Psi) = \nu_I(\Psi[p/\top] \wedge \Psi[p/\bot]); \quad \text{and}$$
$$\nu_I(\exists p\Psi) = \nu_I(\Psi[p/\top] \vee \Psi[p/\bot]).$$

We say that Φ is *true under I* iff $\nu_I(\Phi) = 1$, otherwise Φ is *false under I*. If $\nu_I(\Phi) = 1$, then I is a *model* of Φ.

In the sequel, we use the following abbreviations in the context of QBFs: Let $P = \{p_1, \ldots, p_n\}$ and $Q = \{q_1, \ldots, q_n\}$ be indexed sets of variables. Then, $P \leq Q$ abbreviates $\bigwedge_{i=1}^{n}(p_i \to q_i)$, and $P < Q$ is a shorthand for $(P \leq Q) \wedge \neg(Q \leq P)$. Furthermore, for an indexed set $P = \{p_1, \ldots, p_n\}$ of variables and $\mathsf{Q} \in \{\forall, \exists\}$, we let $\mathsf{Q}P\,\Phi$ stand for the formula $\mathsf{Q}p_1\mathsf{Q}p_2 \cdots \mathsf{Q}p_n\,\Phi$.

The operators $\leq$ and $<$ are fundamental tools for expressing certain tests on sets of atoms. In particular, the following properties hold: Let $P = \{p_1, \ldots, p_n\}$ be a set of indexed atoms, and $P' = \{p'_1, \ldots, p'_n\}$ be a corresponding set disjoint to P. Furthermore, let $I_1, I_2 \subseteq P$ be two interpretations. Then,

(i) $I_1 \subseteq I_2$ iff $I'_1 \cup I_2$ is a model of $P' \leq P$; and
(ii) $I_1 \subset I_2$ iff $I'_1 \cup I_2$ is a model of $P' < P$.

4.2 Reductions of Nested Logic Programs

We proceed by giving the translations to QBFs as presented in [32]. There, the desired translation is obtained as a special case of a more general encoding, viz. of a translation of *equilibrium logic* to QBFs. Observe that the subsequent encoding is *linear* in the size of the nested logic program Π.

Definition 16. *Let Π be a nested logic program over atoms $V = Var(\Pi)$, and let $V' = \{v' \mid v \in V\}$ be a set of disjoint atoms corresponding to V.*

We define a QBF

$$\mathcal{T}_3[\Pi] = \hat{\Pi} \wedge \neg\exists V'\Big((V' < V) \wedge \bigwedge_{r \in \Pi} \big(B^*(r) \supset H^*(r)\big)\Big),$$

where $B^*(r)$ (resp., $H^*(r)$) denotes the formula resulting from $B(r)$ (resp., $H^*(r)$) by replacing each occurrence of an atom p which does not appear in the scope of a negation in $B^*(r)$ (resp., $H^*(r)$) by p'.

Theorem 14. *Let $\Pi \in NLP_{\mathcal{A}}$, $I \subseteq Var(\Pi)$, and $\mathcal{T}_3[\cdot]$ as defined in Definition 16.*
Then, I is a stable model of Π iff I is a model of $\mathcal{T}_3[\Pi]$.

As illustration, let us analyse the functioning of $\mathcal{T}_3[\cdot]$ applied to the program from Example 9.

Example 12. Let $\Pi = \{p \leftarrow (\neg q \vee r) \wedge (q \vee \neg s)\}$. Then,

$$\hat{\Pi} = \Big((\neg q \vee r) \wedge (q \vee \neg s)\Big) \supset p.$$

Hence, $\mathcal{T}_3[\Pi]$ is given by

$$\hat{\Pi} \wedge \neg\exists V'\Big[(V' < V) \wedge \Big(((\neg q \vee r') \wedge (q' \vee \neg s)) \supset p'\Big)\Big],$$

with $V = \{p, q, r, s\}$ and $(V' < V)$ being a shorthand for

$$\begin{aligned}&(p' \supset p) \wedge (q' \supset q) \wedge (r' \supset r) \wedge (s' \supset s) \wedge \\ &\neg\Big((p \supset p') \wedge (q \supset q') \wedge (r \supset r') \wedge (s \supset s')\Big).\end{aligned} \tag{7}$$

First, let us verify that the interpretation $I = \{p\}$, which represents the only stable model of Π, is a model of $\mathcal{T}_3[\Pi]$. Obviously, $\hat{\Pi}$ is true under I, so it remains to check whether

$$\exists V'\Big[(V' < V) \wedge \Big(((\neg q \vee r') \wedge (q' \vee \neg s)) \supset p'\Big)\Big] \tag{8}$$

is false under I. By the properties of $<$ and the semantics of existential quantification, we have that (8) is true iff there there is some $J \subset I$ such that

$$\Big(((\neg q \vee r') \wedge (q' \vee \neg s)) \supset p'\Big) \tag{9}$$

is true under $J' \cup I$. Since $I = \{p\}$, J' must then be the empty set. Observe that given I, $J' = \emptyset$ is in turn the only way to make (7) evaluate to true. But (9) is false under $J' \cup I$, because the antecedent of (9) evaluates to true under $J' \cup I$, whereas p' is false under $J' \cup I$. Hence, (8) is false under I, which eventually yields that I is a model of $\mathcal{T}_3[\Pi]$.

To verify that no other interpretation is a model of $\mathcal{T}_3[\Pi]$, we first argue why no $\{p\} \subset I \subseteq \{p, q, r, s\}$ is a model of $\mathcal{T}_3[\Pi]$. Obviously, such an I satisfies $\hat{\Pi}$, but it satisfies also (8), since an interpretation $J = \{p\} \subset I$ yields a model $J' \cup I$ for both $(V' < V)$ and (9). Obviously, I is then not a model of $\mathcal{T}_3[\Pi]$.

Finally, observe that any remaining interpretation I with $\emptyset \subseteq I \subseteq \{q, r, s\}$ does not serve as a candidate for being a model of $\mathcal{T}_3[\Pi]$, because such sets already give countermodels for the formula $\hat{\Pi} = \big((\neg q \vee r) \wedge (q \vee \neg s)\big) \supset p$.

5 Implementation and Experimental Results

As described in [31, 30], both translations $\mathcal{T}_1$ and $\mathcal{T}_2$ have been implemented as a front-end to DLV [7, 8]. This system represents a state-of-the-art implementation for disjunctive logic programs under the answer-set semantics [13]. The resulting compiler, called nlp, is implemented in Prolog and is publicly available at

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

The current implementation runs under SICStus Prolog and comprises roughly 900 lines of code.

Furthermore, translation $\mathcal{T}_3$ has been implemented as a special module of the reasoning system QUIP [5, 4, 3, 9], which is a prototype tool for solving several nonmonotonic reasoning tasks based on reductions to QBFs. The current version of QUIP provides interfaces to most of the sequential QBF-solvers available, among them the BDD-package boole[18], and systems based on an extended Davis-Putnam procedure (e.g., [10, 15, 21]).

Moreover, we have implemented a generator for two different benchmark series that produce scalable problem instances of nested logic programs. For further details, we refer the reader to [30]. In what follows, we just outline the basic issues.

The first benchmark series generates nested logic programs whose rule heads and bodies consist of two-level disjunctive and/or conjunctive normal-form formulas. The generation of such nested logic programs is guided by several parameters, like the number of variables, rules, facts, and constraints. Further parameters are used to fix the structure of heads and bodies.

The second series is based on the idea that (single headed) logic programs with cardinality atoms can be transformed into nested logic programs [11]. A cardinality atom has the form $m\{p_1, \ldots, p_n\}n$ and informally means that a resultant stable model must contain at least m and at most n atoms out of $\{p_1, \ldots, p_n\}$ (cf. [34] for a detailed semantics). Such atoms can then be used in the same way as ordinary atoms in the head and the body of rules. As in the first series, we generate parameterised programs, but this time with cardinality atoms, which are then transformed into nested logic programs.

[18] This system is available at http://www.cs.cmu.edu/~modelcheck/bdd.html.

Both test series can be downloaded at URL

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

First results are reported in [30] and at the above URL. Roughly speaking, $\mathcal{T}_2$ outperforms $\mathcal{T}_1$ whenever the size of the resultant disjunctive program is dramatically increased due to the exponential behavior of $\mathcal{T}_1$. Comparisons including the QBF translation are subject to future work.

6 Conclusion

In this paper, we gave a brief overview on possible implementation techniques for a recently proposed extension of the answer-set semantics 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. All three approaches rely on a reduction technique allowing the utilisation of off-the-shelf systems for evaluating nested logic programs. In order to compare the techniques under consideration, we briefly mentioned two different benchmark series and referred to some first experimental results.

Future work includes a more careful experimental evaluation as well as invoking optimisations. To wit, for the method of translating nested logic programs into disjunctive logic programs, we will consider also so-called *hybrid approaches* for our purpose. This technique [2], successfully applied in classical theorem-proving, tries to combine the benefits of language-preserving (no additional atoms) and structure-preserving (no exponential blow-up in formula length) translations. On the other hand, to compute stable models via the QBF approach, we still have to compare different strategies to have an optimal variable-ordering for BDDs, which is known as a crucial parameter for efficient performance.

References

1. C. Baral and C. Uyan. Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-01*), pages 186–199, 2001.
2. T. Boy de la Tour. An Optimality Result for Clause Form Translation. *Journal of Symbolic Computation*, 14(4):283–302, 1992.
3. J. Delgrande, T. Schaub, H. Tompits, and S. Woltran. On Computing Solutions to Belief Change Scenarios. In S. Benferhat and P. Besnard, editors, *Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty* (*ECSQARU-01*), LNCS, pages 510–521, 2001.
4. U. Egly, T. Eiter, V. Klotz, H. Tompits, and S. Woltran. Computing Stable Models with Quantified Boolean Formulas: Some Experimental Results. In *Proceedings of the AAAI 2001 Spring Symposium on Answer Set Programming*, pages 53–59, 2001.

5. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks Using Quantified Boolean Formulas. In *Proceedings of the 17th National Conference on Artificial Intelligence* (*AAAI-00*), pages 417–422. AAAI Press, 2000.
6. T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3–4):289–323, 1995.
7. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Non-monotonic Reasoning. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-97*), pages 363–374, 1997.
8. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons and Benchmarks. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning* (*KR-98*), pages 406–417, 1998.
9. T. Eiter, V. Klotz, H. Tompits, and S. Woltran. Modal Nonmonotonic Logics Revisited: Efficient Encodings for the Basic Reasoning Tasks. In *Proceedings of the 11th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods* (*TABLEAUX 2002*), pages 100–114, 2002.
10. R. Feldmann, B. Monien, and S. Schamberger. A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In *Proceedings of the 17th National Conference on Artificial Intelligence* (*AAAI-00*), pages 285–290, 2000.
11. P. Ferraris and V. Lifschitz. Weight Constraints as Nested Expressions. Unpublished draft; available at `http://www.cs.utexas.edu/users/vl/`, 2002.
12. M. Gelfond, M. Balduccini, and J. Galloway. Diagnosing Physical Systems in A-Prolog. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-01*), pages 213–225, 2001.
13. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming* (*ICLP-88*), pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.
14. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–385, 1991.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. QuBE: A System for Deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the International Joint Conference on Automated Reasoning* (*IJCAR-01*), pages 364–369, 2001.
16. K. Inoue and C. Sakama. Negation as Failure in the Head. *Journal of Logic Programming*, 35(1):39–78, 1998.
17. T. Janhunen. On the Intertranslatability of Autoepistemic, Default and Priority Logics, and Parallel Circumscription. In *Proceedings of the 6th European Workshop on Logics in Artificial Intelligence* (*JELIA-98*), pages 216–232, 1998.
18. T. Janhunen. On the Effect of Default Negation on the Expressiveness of Disjunctive Rules. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-01*), pages 93–106, 2001.
19. T. Janhunen, I. Niemelä, P. Simons, and J.-H. You. Unfolding Partiality and Disjunctions in Stable Model Semantics. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning* (*KR-00*), pages 411–424. Morgan Kaufmann, 2000.
20. H. Kautz and B. Selman. Planning as Satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

21. R. Letz. Advances in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of the IJCAR 2001 Workshop on Theory and Applications of Quantified Boolean Formulas* (*QBF-01*), pages 55–64, 2001.
22. V. Lifschitz. Answer Set Planning. In *Proceedings of the 16th International Conference on Logic Programming* (*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. G. Mints. Resolution Strategies for the Intuitionistic Logic. In B. Mayoh, E. Tyugu, and J. Penjaam, editors, *Constraint Programming: NATO ASI Series*, pages 282–304. Springer, 1994.
26. I. Niemelä and P. Simons. Smodels: An Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-97*), pages 420–429, 1997.
27. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Mass., 1994.
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, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A Polynomial Translation of Logic Programs with Nested Expressions into Disjunctive Logic Programs. Technical Report INFSYS RR-1843-02-15, Vienna University of Technology, Institute of Information Systems, 2002. Available at `http://www.kr.tuwien.ac.at/research/reports/`.
31. 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 *Proceedings of the 19th International Conference on Logic Programming* (*ICLP-02*), pages 405–420, 2002.
32. D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proceedings of the 10th Portuguese Conference on Artificial Intelligence* (*EPIA-01*), volume 2258 of *LNCS*, pages 306–320. Springer-Verlag, 2001.
33. D. A. Plaisted and S. Greenbaum. A Structure Preserving Clause Form Translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.
34. P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.
35. G. Tseitin. On the Complexity of Proofs in Propositional Logics. *Seminars in Mathematics*, 8, 1970.