I N F S Y S

R E S E A R C H

R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

POLYNOMIAL TRANSLATIONS OF NESTED LOGIC PROGRAMS INTO DISJUNCTIVE LOGIC PROGRAMS

David PEARCE Vladimir SARSAKOV
Torsten SCHAUB Hans TOMPITS Stefan WOLTRAN

INFSYS RESEARCH REPORT 1843-02-15

NOVEMBER 2002

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT

INFSYS RESEARCH REPORT 1843-02-15, NOVEMBER 2002

POLYNOMIAL TRANSLATIONS OF NESTED LOGIC PROGRAMS INTO DISJUNCTIVE LOGIC PROGRAMS

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

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. Nested expressions can be formed using conjunction, disjunction, as well as the negation as failure operator in an unrestricted fashion. This provides a very flexible framework for knowledge representation and reasoning. 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 polynomial translations of logic programs with nested expressions into disjunctive logic programs. Moreover, we show that these translations are modular and (strongly) faithful. We have implemented both the straightforward as well as our advanced transformations; the resulting compiler serves as a front-end to `DLV` and is publicly available on the Web. Finally, we also report experimental results, comparing the different translations on several classes of benchmark problems.

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

[2]Institut für Informatik, Universität Potsdam Postfach 90 03 27, D–14439 Potsdam, Germany. E-mail: [sarsakov, torsten]@cs.uni-potsdam.de

[3]Institut für Informationssysteme, Abteilung Wissensbasierte Systeme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A–1040 Wien, Austria. E-mail: [tompits, stefan]@kr.tuwien.ac.at

Acknowledgements: This work was partially supported by the German Science Foundation (DFG) under grant FOR 375/1-1, TP C, as well as by the Austrian Science Fund (FWF) under grant P15068-INF. The authors would like to thank Agata Ciabattoni for pointing out some relevant references.

Preliminary versions of this paper appeared in the Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR'2002), pages 57–66, 2002, as well as in the Proceedings of the 19th International Conference on Logic Programming (ICLP'2002), pages 405–420, 2002.

Contents

1 Introduction

Lifschitz, Tang, and Turner [23] 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 provide well-recognised languages 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., [21, 3, 10, 15]), which in turn is largely fostered by the availability of efficient solvers for the answer set semantics, most notably `DLV` [7, 8] 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 translating nested logic programs into disjunctive logic programs in a semantics-preserving way. In contrast to previous work, our reduction method is guaranteed to be constructible in polynomial 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 the answer sets of $\sigma(\Pi)$ such that the former are given by 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'.

Besides the basic translation σ, we also introduce an alternative one, τ, which satisfies the same set of postulates as σ, but where negation is treated differently. Moreover, we have implemented both translations as a front-end for the logic programming system `DLV`.

The construction of our translations 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 criterion, each of our translations actually satisfies 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 the answer sets of $\rho(\Pi) \cup \Pi'$, for $\rho \in \{\sigma, \tau\}$, such that the former answer sets are given by sets of form $I \cap \mathcal{A}_1$, where I is an answer set of $\rho(\Pi) \cup \Pi'$. 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 $\rho(\Pi) \cup \Pi'$, for each $\rho \in \{\sigma, \tau\}$; 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* [22] (see below).

Let us illustrate the practical relevance of this property by considering the well-known graph problem of computing *Hamiltonian paths*: Given a directed graph (V, E) and a vertex a of this graph, determine whether there exists a path in (V, E) starting at a and passing through each vertex in V exactly once. The following "guess and check" disjunctive logic program, taken from [7], solves this problem:

```
inPath(X,Y) v outPath(X,Y) :- arc(X,Y).

:- inPath(X,Y), inPath(X,Y1), Y <> Y1.
:- inPath(X,Y), inPath(X1,Y), X <> X1.
:- node(X), not reached(X).
reached(X) :- start(X).
reached(X) :- reached(Y), inPath(Y,X).
```

The first rule guesses a subset of all given arcs, while the rest of the program checks whether it is a Hamiltonian Path. In fact, this can be expressed without the "auxiliary" predicate `outPath/2` by means of the following nested rule:

```
inPath(X,Y) v not inPath(X,Y) :- arc(X,Y).
```

Now, given a strongly faithful translation, we may only apply it for translating the first rule, whilst leaving the remaining rules unaffected.

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 and intuitionistic logic.[1] As shown in [29], 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 also been addressed in [32], where encodings of the basic reasoning tasks into quantified Boolean formulas are described. These encodings allow for a straightforward implementation for nested logic programs by appeal to off-the-shelf solvers for 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, appropriate logic programming solvers, like, e.g., `DLV` or `Smodels`,[2] 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 [23] (together with transformation rules given in [18]) provide a method to translate nested logic programs into disjunctive ones, this 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 (provided that 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 [14]; there,

[1] The logic of here-and-there is also known as *Gödel's three-valued logic* in view of [13].

[2] To be precise, together with `GnT`, the extension of `Smodels` to disjunctive logic programs [19].

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.

The paper is organised as follows. We start by introducing the necessary formal foundations, such as answer sets semantics and equilibrium logic. In Section 3, we describe existing translations that can be used for implementing nested logic programs. Once we have introduced the necessary formal means for capturing faithful translations in Section 4, we present in Section 5 our provably polynomial, modular, and strongly faithful translations of nested logic programs to disjunctive ones. In Section 6, we describe a resulting implementation and discuss experimental results, and Section 7 concludes with some general remarks. Proofs of the main results are relegated to an appendix.

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*. As usual, *literals* are formulas of form v or $\neg v$, where v is some variable. The set of variables occurring in a formula ϕ is denoted by $var(\phi)$. For a set of formulas S, we write $var(S)$ as a shorthand for $\bigcup_{s \in S} var(s)$.

We use the concepts of *proper subformula*, *immediate subformula*, and *subformula* in the usual way. In particular, a formula ψ is a proper subformula of ϕ iff it is a subformula of ϕ and $\psi \neq \phi$, and ψ is an immediate subformula of ϕ iff $\phi = \neg\psi$, $\phi = \psi \circ \varphi$, or $\phi = \varphi \circ \psi$ for $\circ \in \{\vee, \wedge, \rightarrow\}$.

The logical complexity of a formula ϕ, symbolically $lc(\phi)$, is defined as the number of the logical symbols $\neg$, $\vee$, $\wedge$, and $\rightarrow$ in ϕ. The logical complexity of a finite set S of formulas is defined as $lc(S) = lc(\bigwedge_{\phi \in S} \phi)$. Furthermore, $|S|$ refers to the cardinality of a set S. For formulas ϕ, ψ, and φ, the expression $\phi[\psi/\varphi]$ denotes the formula resulting from ϕ by uniformly replacing each occurrence of ψ in ϕ by φ.

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 object of our investigation are logic programs with nested expressions [23]. This class of programs generalises the class of 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 can be done in the usual way.

We start with some basic notation. A formula whose sentential connectives comprise only occurrences of $\wedge$, $\vee$, 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. An example of a rule is given by

$$p \leftarrow \neg(q \wedge \neg t) \wedge (q \vee \neg s). \tag{1}$$

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 contains no negated literal at all, then r is *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; 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 2.2.

In what follows, we associate with each rule r a corresponding formula $\hat{r} = B(r) \rightarrow H(r)$ and, accordingly, to each program Π a corresponding set of formulas $\hat{\Pi} = \{\hat{r} \mid r \in \Pi\}$. The logical complexity of a program Π, symbolically $lc(\Pi)$, is given by $lc(\hat{\Pi})$. Moreover, we apply the replacement notion also to programs: An expression $\Pi[\psi/\varphi]$ denotes the program resulting from Π by uniformly replacing each occurrence of an expression ψ in Π by an expression φ.

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 $AS_{\mathcal{A}}(\Pi)$.

As an example, consider the program Π consisting of the single rule (1). Let us check whether $I = \{p\}$ is an answer set of this program.

Here, the expressions of form $\neg\psi$ which are not in the scope of any other negation are given by $\neg(q \wedge \neg t)$ and $\neg s$. Since $(q \wedge \neg t)$ and s are false under I, we replace $\neg(q \wedge \neg t)$ and $\neg s$ in Π by $\top$, yielding

$$\Pi^I = \{p \leftarrow \top \wedge (q \vee \top)\}.$$

Obviously, $\{p\}$ is the unique minimal model of Π^I. So, I is indeed an answer set for Π. In fact, there are no other answer sets.

The following concepts are central: Two logic programs, Π_1 and Π_2, are *equivalent* iff they possess the same answer sets. Following Lifschitz *et al.* [22], 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. We introduce more general equivalence notions in Section 4.

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 [22], the problem of checking whether two logic programs are strongly equivalent can be expressed in terms of the logic of here-and-there.

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, $\leq$, 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 $I_H \subseteq I_T$. I_H is called the *here part* of $\mathcal{I}$, and I_T is the *there part* of $\mathcal{I}$. 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 $INT_{\mathcal{A}}$.

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 $v \in I_w$, and $\nu_{\mathcal{I}}(w, \phi) = 0$ otherwise;
4. if $\phi = \neg\psi$, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if for every world u such that $w \leq u$, $\nu_{\mathcal{I}}(u, \psi) = 0$, and $\nu_{\mathcal{I}}(w, \phi) = 0$ otherwise;
5. if $\phi = (\phi_1 \wedge \phi_2)$, then $\nu_{\mathcal{I}}(w, \phi) = min(\{\nu_{\mathcal{I}}(w, \phi_1), \nu_{\mathcal{I}}(w, \phi_2)\})$;
6. if $\phi = (\phi_1 \vee \phi_2)$, then $\nu_{\mathcal{I}}(w, \phi) = max(\{\nu_{\mathcal{I}}(w, \phi_1), \nu_{\mathcal{I}}(w, \phi_2)\})$;
7. if $\phi = (\phi_1 \rightarrow \phi_2)$, then $\nu_{\mathcal{I}}(w, \phi) = 1$ if for every world u such that $w \leq u$, $\nu_{\mathcal{I}}(u, \phi_1) \leq \nu_{\mathcal{I}}(u, \phi_2)$, and $\nu_{\mathcal{I}}(w, \phi) = 0$ otherwise.

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*. An HT-interpretation $\mathcal{I} = \langle I_H, I_T \rangle$ is *total* if $I_H = I_T$.

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, because $\langle \emptyset, \{p\} \rangle$ is not an HT-model for either of these formulas.

An HT-interpretation $\mathcal{I}$ is an HT-model of set of fomulas S iff $\mathcal{I}$ is an HT-model of each element of S. In particular, 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. We write $\alpha \equiv_{HT} \beta$ to express that α and β are HT-equivalent, where α and β are either formulas or sets of formulas. In particular, this includes the case where one of α, β is a formula and the other is a set of 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. It is easily seen that for any equilibrium model $\langle I, I \rangle$ of ϕ, $I \subseteq var(\phi)$ always holds. The collection of all equilibrium models of a formula ϕ over an alphabet $\mathcal{A}$ is denoted by $EM_{\mathcal{A}}(\phi)$. Likewise, for a logic program Π over $\mathcal{A}$, $EM_{\mathcal{A}}(\Pi)$ denotes the collection of all equilibrium models of $\hat{\Pi}$.

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

In other words, for any $\Pi \in NLP_{\mathcal{A}}$, we have that

$$AS_{\mathcal{A}}(\Pi) = \{I \mid \langle I, I\rangle \in EM_{\mathcal{A}}(\Pi)\}.$$

Moreover, HT-equivalence captures the notion of strong equivalence between logic programs:

Proposition 2 ([22]) *Let Π_1 and Π_2 be programs, and let $\hat{\Pi}_i = \{B(r) \to H(r) \mid r \in \Pi_i\}$, 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 [4] 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 [20] (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 [24] for disjunctive programs). As a consequence, checking whether two programs are strongly equivalent has co-NP complexity.

Next, we summarise some important HT-equivalences, originally formulated in [23] for strong equivalence.

Proposition 3 ([23]) *Let ϕ, ψ, and φ be expressions, and let p be an atom. We have the following equivalence results in the logic of here-and-there:*

(A_1) $(\phi \circ \psi) \equiv_{HT} (\psi \circ \phi)$, *for* $\circ \in \{\vee, \wedge\}$;

(A_2) $((\phi \circ \psi) \circ \varphi) \equiv_{HT} (\phi \circ (\psi \circ \varphi))$, *for* $\circ \in \{\vee, \wedge\}$;

(A_3) $(\phi \vee (\psi \wedge \varphi)) \equiv_{HT} ((\phi \vee \psi) \wedge (\phi \vee \varphi))$ *and* $(\phi \wedge (\psi \vee \varphi)) \equiv_{HT} ((\phi \wedge \psi) \vee (\phi \wedge \varphi))$;

(A_4) $\neg(\phi \vee \psi) \equiv_{HT} (\neg\phi \wedge \neg\psi)$ *and* $\neg(\phi \wedge \psi) \equiv_{HT} (\neg\phi \vee \neg\psi)$;

(A_5) $\neg\neg\neg\phi \equiv_{HT} \neg\phi$;

(A_6) $((\psi \vee \varphi) \to \phi) \equiv_{HT} \{\psi \to \phi, \varphi \to \phi\}$ *and* $(\phi \to (\psi \wedge \varphi)) \equiv_{HT} \{\phi \to \psi, \phi \to \varphi\}$;

(A_7) $((\neg\neg p \wedge \psi) \to \phi) \equiv_{HT} (\psi \to (\neg p \vee \phi))$ *and* $(\psi \to (\phi \vee \neg\neg p)) \equiv_{HT} ((\neg p \wedge \psi) \to \phi)$.

Finally, we require the following 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 $\top$, $\bot$. Accordingly, this notion allows us to define a further class of logic programs, viz. the set $GDLP^{ht}_{\mathcal{A}}$, consisting of all programs which are defined like generalised logic programs over $\mathcal{A}$, except that HT-literals may occur in rules instead of ordinary literals.

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* Π is in HT-NNF iff all heads and bodies in Π are in HT-NNF. $NLP^{nnf}_{\mathcal{A}}$ denotes the class of all nested logic programs over $\mathcal{A}$ which are in HT-NNF. A formula is in *here-and-there conjunctive normal form*, or *HT-CNF*, if it is a conjunction of disjunctions of HT-literals; and, likewise, a formula is in *here-and-there disjunctive normal form*, or *HT-DNF*, if it is a disjunction of conjunctions of HT-literals.

Applying the HT-equivalences (A_1)–(A_5) from Proposition 3, every expression can effectively be transformed into HT-NNF, HT-CNF, or HT-DNF, retaining HT-equivalence. We assume to have suitable operators

$\eta_N(\cdot)$, $\eta_C(\cdot)$, and $\eta_D(\cdot)$ performing the corresponding translations into HT-NNF, HT-CNF, and HT-DNF, respectively. We note that, for each expression ϕ, $\eta_N(\phi)$ is obviously constructible in polynomial time. However, the size of both $\eta_C(\phi)$ and $\eta_D(\phi)$ is in general exponential in the size of ϕ—indeed, such an exponential increase is due to iterated applications of the distributivity laws (A_3) and happens whenever the input of $\eta_C(\phi)$ is in HT-DNF or the input of $\eta_D(\phi)$ is in HT-CNF, and the respective input formulas are not simple disjunctions or conjunctions of HT-literals.

3 Previous Translation Results

Prior to our later elaboration in Section 5 of efficiently reducing nested programs into disjunctive ones, let us briefly review here a straightforward method to translate nested programs into disjunctive programs. This method consists of two steps: First, following the discussion in [23], we translate nested programs into generalised disjunctive logic programs using the HT-equivalences of Proposition 3, and afterwards we eliminate negations in the heads of rules by applying a simple procedure due to Janhunen [18]. Given that the transformation steps in the first part of this translation yield an exponential increase of program size in the worst case, the overall translation schema is thus not efficiently constructible in general. However, this part of the translation is “language preserving” in the sense that no new variables are introduced, whilst the second part of the translation—although constructible in polynomial time—introduces new variables (similar attempts can be found, e.g., in [16]). The details of the construction follow.

For translating nested programs into generalised disjunctive programs, we introduce a function λ : $NLP_{\mathcal{A}} \rightarrow GDLP_{\mathcal{A}}$ as follows. First, for a given $\Pi \in NLP_{\mathcal{A}}$, each rule in Π is translated into a rule whose body is in HT-DNF and whose head is in HT-CNF, i.e., we transform Π into

$$\{\eta_C(H(r)) \leftarrow \eta_D(B(r)) \mid r \in \Pi\}. \tag{2}$$

Afterwards, we apply the following manipulations, exploiting the HT-equivalences (A_6) and (A_7) of Proposition 3:

1. Recursively replace each rule in (2) of form

   $$\phi \wedge \psi \leftarrow \varphi \quad \text{by the two rules} \quad \phi \leftarrow \varphi \quad \text{and} \quad \psi \leftarrow \varphi,$$

   as well as each rule of form

   $$\phi \leftarrow \psi \vee \varphi \quad \text{by the two rules} \quad \phi \leftarrow \psi \quad \text{and} \quad \phi \leftarrow \varphi;$$

   and

2. recursively replace in the resulting program each rule of form

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

   as well as each rule of form

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

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

Note that λ does not introduce any new variables, but, due to the usage of $\eta_C(\cdot)$ and $\eta_D(\cdot)$, yields an exponential blow-up of program size whenever the input 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.

Let us illustrate the behavior of λ using our running example

$$\Pi \quad = \quad \{\, p \leftarrow \neg(q \wedge \neg t) \wedge (q \vee \neg s) \,\}. \tag{3}$$

First, we translate the body of the single rule r in Π into HT-DNF. This yields

$$\eta_D(B(r)) = (\neg q \wedge q) \vee (\neg\neg t \wedge q) \vee (\neg q \wedge \neg s) \vee (\neg\neg t \wedge \neg s).$$

Then we split the resulting rule $p \leftarrow \eta_D(B(r))$ according to the first schema in (A_6), producing four rules:

$$p \leftarrow \neg q \wedge q; \tag{4}$$
$$p \leftarrow \neg\neg t \wedge q; \tag{5}$$
$$p \leftarrow \neg q \wedge \neg s; \tag{6}$$
$$p \leftarrow \neg\neg t \wedge \neg s. \tag{7}$$

Observe the inherent exponential nature of this step, as adding a further conjunct to $B(r)$ would return eight rules, adding two conjuncts would result in 16 rules, and so on.

Next, we eliminate double negations in Rules (5) and (7), arriving at the final generalised disjunctive program $\lambda(\Pi)$:

$$p \quad \leftarrow \quad \neg q \wedge q, \tag{8}$$
$$p \vee \neg t \quad \leftarrow \quad q, \tag{9}$$
$$p \quad \leftarrow \quad \neg q \wedge \neg s, \tag{10}$$
$$p \vee \neg t \quad \leftarrow \quad \neg s. \tag{11}$$

It is easy to check that the only answer set of $\lambda(\Pi)$ is given by $\{p\}$, which coincides with the single answer set of Π. In fact, this reflects the following general property:

Proposition 4 ([23]) *For any $\Pi \in NLP_{\mathcal{A}}$, we have that Π and $\lambda(\Pi)$ are strongly equivalent.*

We proceed with translating generalised disjunctive logic programs into disjunctive ones, following Janhunen [18]. To this end, for a given alphabet $\mathcal{A}$, we introduce the following new and disjoint alphabet

$$\bar{\mathcal{A}} = \{\overline{p} \mid p \in \mathcal{A}\},$$

representing negated atoms.

Then, the function $\zeta : GDLP_{\mathcal{A}} \rightarrow DLP_{\mathcal{A} \cup \bar{\mathcal{A}}}$ is defined by setting

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

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

Applying function ζ to the translated program $\lambda(\Pi)$ of our running example (3), we have to replace $\neg t$ in rules (9) and (11) by $\overline{t}$, as $\neg t$ is the only negated head atom occurring in $\lambda(\Pi)$. The resulting disjunctive program $\zeta(\lambda(\Pi))$ is then given by the following rules:

$$
\begin{array}{rcl}
p & \leftarrow & \neg q \wedge q, \\
p \vee \overline{t} & \leftarrow & q, \\
p & \leftarrow & \neg q \wedge \neg s, \\
p \vee \overline{t} & \leftarrow & \neg s, \\
\bot & \leftarrow & t \wedge \overline{t} \\
\overline{t} & \leftarrow & \neg t.
\end{array}
$$

It is easily seen that the only answer set of above program is given by $\{p, \overline{t}\}$.

Proposition 5 ([18]) *Let $\Pi \in GDLP_{\mathcal{A}}$ and $I \subseteq \mathcal{A}$. Then, I is an answer set of Π iff*

$$I \cup \{\overline{p} \mid \neg p \text{ occurs in some head of } \Pi \text{ and } p \notin I\}$$

is an answer set of $\zeta(\Pi)$.

Observe that, by construction, the composed function $\zeta \circ \lambda$ translates any $\Pi \in NLP_{\mathcal{A}}$ into a standard disjunctive logic program over $\mathcal{A} \cup \bar{\mathcal{A}}$. Moreover, in view of Propositions 4 and 5, there is a one-to-one correspondence between the answer sets of Π and the answer sets of $\zeta(\lambda(\Pi))$. However, as we argued before, $\zeta(\lambda(\Pi))$ is exponential in the size of Π in the worst case, which is caused by the first part λ. In Section 5, we present methods to circumvent this particular problem. Also, we are interested in translations which realise a firmer notion of equivalence, generalising strong equivalence between programs. The general desiderata for this purpose are addressed next.

4 Characterisations

In this section, we introduce the general requirements that we impose on our desired translations from nested logic programs into disjunctive logic programs, and we discuss some relevant properties.

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 some class of nested logic programs closed under union.*[3] *Then, a function $\rho : S_1 \rightarrow S_2$ 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] A class S of sets is closed under union providing $A, B \in S$ implies $A \cup B \in S$.

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, 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 [17, 18], 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 polynomial functions is again polynomial, and likewise for functions which are faithful, strongly faithful, or modular. Hence, the composition of PFM-functions is PFM, and the composition of PSM-functions is PSM. Furthermore, since any PSM-function is also PFM, we focus on PSM-functions in the following. In fact, in the next section, we construct two functions, σ and τ, mapping nested programs over a given alphabet $\mathcal{A}_1$ into disjunctive programs over some alphabet $\mathcal{A}_2 \supset \mathcal{A}_1$ which are 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* $\mathcal{A}_i$ $(i = 1, 2)$.

We call $\alpha_\rho : INT_{\mathcal{A}_1} \times NLP_{\mathcal{A}_1} \to INT_{\mathcal{A}_2}$ *a* ρ-associated HT-embedding *iff the following two conditions hold, for each HT-interpretation* $\mathcal{I} = \langle I_H, I_T \rangle$ *over* $\mathcal{A}_1$*, each* $\Pi \in NLP_{\mathcal{A}_1}$*, each* $w \in \{H, T\}$*, and* $\alpha_\rho(\mathcal{I}, \Pi) = \langle J_H, J_T \rangle$*:*

(E_1) $J_w \cap \mathcal{A}_1 = I_w$*; and*

(E_2) $J_w \setminus \mathcal{A}_1 \subseteq var(\rho(\Pi))$.

Furthermore, for any $G \subseteq INT_{\mathcal{A}_1}$ *and any program* $\Pi \in NLP_{\mathcal{A}_1}$*, 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 *iff the following conditions hold, for any $\mathcal{I},\mathcal{I}' \in INT_{\mathcal{A}_1}$ and any $\Pi \in NLP_{\mathcal{A}_1}$:*

(H_1) *$\mathcal{I}$ is an HT-model of Π iff $\alpha_\rho(\mathcal{I},\Pi)$ is an HT-model of $\rho(\Pi)$;*

(H_2) *$\mathcal{I}$ is total iff $\alpha_\rho(\mathcal{I},\Pi)$ is total;*

(H_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 the condition that $I_H \subset I'_H$ and $I_T = I'_T$ are jointly satisfied is equivalent to the condition that both $J_H \subset J'_H$ 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*

(H_4) *an HT-interpretation $\mathcal{J}$ over $\mathcal{A}_1 \cup var(\rho(\Pi))$ 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. However, these three conditions alone are insufficient 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 $\mathcal{A}_1 \cup var(\rho(\Pi))$. Hence, 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 Condition (H_4), 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.

We have the following result:

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 central 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.*

Let us apply above concepts and characterisations to function $\zeta \circ \lambda : NLP_{\mathcal{A}} \to DLP_{\mathcal{A}\cup\bar{\mathcal{A}}}$ from Section 3. First of all, according to Propositions 4 and 5, $\lambda : NLP_{\mathcal{A}} \to GDLP_{\mathcal{A}}$ is strongly faithful and $\zeta : GDLP_{\mathcal{A}} \to DLP_{\mathcal{A}\cup\bar{\mathcal{A}}}$ is faithful. Moreover, both functions are modular. Hence, the combined function $\zeta \circ \lambda$ is modular and faithful. However, although ζ is polynomial, λ is *not* polynomial, as discussed in Section 3, so $\zeta \circ \lambda$ is not polynomial as well. In any case, by applying the characterisation of Theorem 1, we show that ζ is strongly faithful, and therefore we get that $\zeta \circ \lambda$ is strongly faithful too. The key result for this purpose is the following lemma:

Lemma 2 *Let ζ be the translation defined in Section 3, and let $\zeta' : NLP_{\mathcal{A}} \to NLP_{\mathcal{A}\cup\bar{\mathcal{A}}}$ result from ζ by setting $\zeta'(\Pi) = \zeta(\Pi)$ if $\Pi \in GDLP_{\mathcal{A}}$ and $\zeta'(\Pi) = \Pi$ if $\Pi \in NLP_{\mathcal{A}} \setminus GDLP_{\mathcal{A}}$.*

Then, the function $\alpha_{\zeta'} : INT_{\mathcal{A}} \times NLP_{\mathcal{A}} \to INT_{\mathcal{A}\cup\bar{\mathcal{A}}}$, defined as

$$\alpha_{\zeta'}(\langle I_H, I_T\rangle,\Pi) = \langle I_H \cup \kappa(I_T,\Pi), I_T \cup \kappa(I_T,\Pi)\rangle,$$

is a ζ'-associated HT-homomorphism, where

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

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

Hence, applying Theorem 1, we get that ζ' is strongly faithful, and thus ζ is strongly faithful as well. Consequently, ζ is PSM.

Proposition 6 *Let* $\zeta \circ \lambda : NLP_{\mathcal{A}} \to DLP_{\mathcal{A} \cup \bar{\mathcal{A}}}$ *be the function defined in Section 3. Then,* $\zeta \circ \lambda$ *is modular and strongly faithful.*

In the next section, we improve on this result by providing translations from nested programs into disjunctive programs which are PSM. The key technique for showing strong faithfulness of these functions is again the application of Theorem 1.

5 Translations

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. In order to avoid the exponential blow-up of program size in the worst case, as happening in the translation based on distributivity laws using the technique of Lifschitz *et al.* [23], we propose a method employing the introduction of new atoms, called *labels*, abbreviating subformula occurrences. This approach is derived from structure-preserving normal form translations [36, 33], which are frequently applied in the context of automated reasoning (cf., e.g., [2, 14] for general investigations about structure-preserving normal form translation in finite-valued Gödel logics, and [5, 6] for proof-theoretical issues of such translations in 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, because our goal is to *reconstruct the answer sets* of the original programs from the translated ones, which involves also a suitable handling of additional minimality criteria.

5.1 Main Construction

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

$$\mathcal{A}_{\mathbf{L}} = \{\mathbf{L}_\phi \mid \phi \in \mathcal{L}_{\mathcal{A}}\}$$

whose elements are called *labels*. Furthermore, for any program Π, let $\Sigma(\Pi)$ be the set of all subformulas occurring in expressions in Π.

Our central definition is as follows.

Definition 4 *The function* $\sigma : NLP_{\mathcal{A}} \to DLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}$ *is defined by setting, for any* $\Pi \in NLP_{\mathcal{A}}$*,*

$$\sigma(\Pi) = \{\mathbf{L}_{H(r)} \leftarrow \mathbf{L}_{B(r)} \mid r \in \Pi\} \cup \bigcup_{\phi \in \Sigma(\Pi)} \gamma(\phi)$$

where, for each $\phi \in \Sigma(\Pi)$*, the set* $\gamma(\phi)$ *is defined as follows:*

1. *if $\phi = p$, where p is an atom, then $\gamma(\phi)$ consists of the two rules*

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

2. *if $\phi = \neg\psi$, then $\gamma(\phi)$ consists of the two rules*

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

3. *if $\phi = (\psi_1 \wedge \psi_2)$, then $\gamma(\phi)$ consists of the three rules*

$$\mathbf{L}_{\psi_1 \wedge \psi_2} \leftarrow \mathbf{L}_{\psi_1} \wedge \mathbf{L}_{\psi_2}, \quad \mathbf{L}_{\psi_1} \leftarrow \mathbf{L}_{\psi_1 \wedge \psi_2}, \quad \mathbf{L}_{\psi_2} \leftarrow \mathbf{L}_{\psi_1 \wedge \psi_2}; \quad \textit{and}$$

4. *if $\phi = (\psi_1 \vee \psi_2)$, then $\gamma(\phi)$ consists of the three rules*

$$\mathbf{L}_{\psi_1} \vee \mathbf{L}_{\psi_2} \leftarrow \mathbf{L}_{\psi_1 \vee \psi_2}, \quad \mathbf{L}_{\psi_1 \vee \psi_2} \leftarrow \mathbf{L}_{\psi_1}, \quad \mathbf{L}_{\psi_1 \vee \psi_2} \leftarrow \mathbf{L}_{\psi_2}.$$

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

We note at this point that σ is clearly modular. Furthermore σ is also polynomial. For the latter property, we provide the following estimations, which can be shown by straightforward induction on the logical complexity of a program.

Proposition 7 *Let Π be a nested logic program with $n = lc(\Pi)$ and $m = |var(\Pi)|$. Then,*

$$lc(\sigma(\Pi)) \leq 6n + 4m \quad \textit{and} \quad |var(\sigma(\Pi))| \leq n + m.$$

For illustration of translation σ, consider the program Π from Section 2, consisting of the single rule (1):

$$\Pi = \{p \leftarrow \neg(q \wedge \neg t) \wedge (q \vee \neg s)\}. \tag{12}$$

For brevity, let us write $\varphi = q \wedge \neg t$ and $\psi = q \vee \neg s$. Here, we have that $\Sigma(\Pi)$ is given as follows:

$$\Sigma(\Pi) = \{p, q, t, s, \neg t, \neg s, \varphi, \psi, \neg\varphi, \neg\varphi \wedge \psi\}.$$

From this, $\sigma(\Pi)$ comprises the following rules:

$$\begin{array}{rcl} \sigma(\Pi) & = & \{\mathbf{L}_p \leftarrow \mathbf{L}_{\neg\varphi \wedge \psi}\} \\ & & \cup\{\mathbf{L}_{\neg\varphi \wedge \psi} \leftarrow \mathbf{L}_{\neg\varphi} \wedge \mathbf{L}_\psi, \quad \mathbf{L}_{\neg\varphi} \leftarrow \mathbf{L}_{\neg\varphi \wedge \psi}, \quad \mathbf{L}_\psi \leftarrow \mathbf{L}_{\neg\varphi \wedge \psi}\} \\ & & \cup\{\mathbf{L}_{\neg\varphi} \leftarrow \neg\, \mathbf{L}_\varphi, \quad \bot \leftarrow \mathbf{L}_\varphi \wedge \mathbf{L}_{\neg\varphi}\} \\ & & \cup\{\mathbf{L}_\varphi \leftarrow \mathbf{L}_q \wedge \mathbf{L}_{\neg t}, \quad \mathbf{L}_q \leftarrow \mathbf{L}_\varphi, \quad \mathbf{L}_{\neg t} \leftarrow \mathbf{L}_\varphi\} \\ & & \cup\{\mathbf{L}_q \vee \mathbf{L}_{\neg s} \leftarrow \mathbf{L}_\psi, \quad \mathbf{L}_\psi \leftarrow \mathbf{L}_q, \quad \mathbf{L}_\psi \leftarrow \mathbf{L}_{\neg s}\} \\ & & \cup\{\mathbf{L}_{\neg v} \leftarrow \neg\, \mathbf{L}_v, \quad \bot \leftarrow \mathbf{L}_v \wedge \mathbf{L}_{\neg v} \mid v \in \{t, s\}\} \\ & & \cup\{\mathbf{L}_v \leftarrow v, \quad v \leftarrow \mathbf{L}_v \mid v \in \{p, q, t, s\}\}. \end{array}$$

In the following, we show that σ is strongly faithful. To this end, we introduce the following concepts.

Fix some alphabet $\mathcal{A}$. For $\phi, \psi \in \mathcal{L}_\mathcal{A}$, define $\phi \prec \psi$ iff ϕ is a proper subformula of ψ. Furthermore, let T be a set of formulas over $\mathcal{A}$, and consider a sequence $s = (\phi_1, \phi_2, \ldots, \phi_n)$ containing all elements from

T. We say that s is a *$\prec$-preserving sequence for T* iff there are no elements ϕ_i, ϕ_j occurring in s such that $i < j$ but $\phi_j \prec \phi_i$.

For our purposes, we are mainly interested in $\prec$-preserving sequences for $\Sigma(\Pi)$. More specifically, we call υ a *Σ-selector for S*, where S is a class of programs, if υ is a function assigning to each program $\Pi \in S$ some $\prec$-preserving sequence $\upsilon(\Pi)$ for $\Sigma(\Pi)$.

In order to access the different labeling steps, we decompose translation σ into a sequence of functions $(\sigma_i^\upsilon)_{i \geq 0}$, for some fixed Σ-selector υ for $NLP_{\mathcal{A}}$, such that each σ_i^υ represents the labeling of a single subformula of Π, and $\sigma(\Pi) = \sigma_n^\upsilon(\Pi)$ with $n = |\Sigma(\Pi)|$, for every $\Pi \in NLP_{\mathcal{A}}$. The details follow.

Definition 5 *Let υ be a Σ-selector for $NLP_{\mathcal{A}}$. Then, the sequence $(\sigma_i^\upsilon)_{i \geq 0}$ of functions $\sigma_i^\upsilon : NLP_{\mathcal{A}} \rightarrow NLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}$ is recursively defined as follows:*

1. $\sigma_0^\upsilon(\Pi) = \Pi$, *for every* $\Pi \in NLP_{\mathcal{A}}$*;*

2. *for* $i > 0$, σ_i^υ *is given by*

$$\sigma_i^\upsilon(\Pi) = \Big((\sigma_{i-1}(\Pi))[\phi_i^*/\mathbf{L}_{\phi_i}]\Big) \cup \gamma(\phi_i)$$

if $i \leq |\Sigma(\Pi)|$, *where*

- ϕ_i *is the i-th element of* $\upsilon(\Pi)$,
- ϕ_i^* *results from* ϕ_i *by replacing each immediate subformula* ψ *of* ϕ_i *by* $\mathbf{L}_\psi$, *and*
- $\gamma(\cdot)$ *is as in Definition 4;*

otherwise $\sigma_i^\upsilon(\Pi) = \sigma_{i-1}^\upsilon(\Pi)$, *for every* $\Pi \in NLP_{\mathcal{A}}$.

As an example, consider the nested program Π from (12). Fix some Σ-selector υ such that $\upsilon(\Pi)$ is given by

$$\upsilon(\Pi) = (p,\, q,\, t,\, s,\, \neg t,\, \neg s,\, q \wedge \neg t,\, q \vee \neg s,\, \neg(q \wedge \neg t),\, \neg(q \wedge \neg t) \wedge (q \vee \neg s)).$$

Accordingly, replacing each term ϕ_i, $1 \leq i \leq 10$, in $\upsilon(\Pi)$ by ϕ_i^*, we get the following sequence of expressions:

$$(p,\, q,\, t,\, s,\, \neg\mathbf{L}_t,\, \neg\mathbf{L}_s,\, \mathbf{L}_q \wedge \mathbf{L}_{\neg t},\, \mathbf{L}_q \vee \mathbf{L}_{\neg s},\, \neg\mathbf{L}_\varphi,\, \mathbf{L}_{\neg\varphi} \wedge \mathbf{L}_\psi),$$

where $\varphi = q \wedge \neg t$ and $\psi = q \vee \neg s$.

Now we translate Π step by step, using the functions σ_i^υ, $i \geq 0$. For convenience, in the following we write $\gamma(\phi_1, \ldots, \phi_n)$ to denote the set $\gamma(\phi_1) \cup \ldots \cup \gamma(\phi_n)$, with $\phi_1, \ldots, \phi_n$ expressions.

Clearly, $\sigma_0^\upsilon(\Pi) = \Pi = \{p \leftarrow \neg\varphi \wedge \psi\}$. Otherwise, for $1 \leq i \leq 10$, $\sigma_i^\upsilon(\Pi)$ is given as follows:

$$
\begin{array}{rcl}
\sigma_1^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg\varphi \wedge \psi\} \cup \gamma(p);\\
\sigma_2^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg(\mathbf{L}_q \wedge \neg t) \wedge (\mathbf{L}_q \vee \neg s)\} \cup \gamma(p,q);\\
\sigma_3^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg(\mathbf{L}_q \wedge \neg \mathbf{L}_t) \wedge (\mathbf{L}_q \vee \neg s)\} \cup \gamma(p,q,t);\\
\sigma_4^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg(\mathbf{L}_q \wedge \neg \mathbf{L}_t) \wedge (\mathbf{L}_q \vee \neg \mathbf{L}_s)\} \cup \gamma(p,q,t,s);\\
\sigma_5^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg(\mathbf{L}_q \wedge \mathbf{L}_{\neg t}) \wedge (\mathbf{L}_q \vee \neg \mathbf{L}_s)\} \cup \gamma(p,q,t,s,\neg t);\\
\sigma_6^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg(\mathbf{L}_q \vee \mathbf{L}_{\neg t}) \wedge (\mathbf{L}_q \vee \mathbf{L}_{\neg s})\} \cup \gamma(p,q,t,s,\neg t,\neg s);\\
\sigma_7^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg\mathbf{L}_\varphi \wedge (\mathbf{L}_q \vee \mathbf{L}_{\neg s})\} \cup \gamma(p,q,t,s,\neg t,\neg s,\varphi);\\
\sigma_8^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \neg\mathbf{L}_\varphi \wedge \mathbf{L}_\psi\} \cup \gamma(p,q,t,s,\neg t,\neg s,\varphi,\psi);\\
\sigma_9^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \mathbf{L}_{\neg\varphi} \wedge \mathbf{L}_\psi\} \cup \gamma(p,q,t,s,\neg t,\neg s,\varphi,\psi,\neg\varphi);\\
\sigma_{10}^\upsilon(\Pi) &=& \{\mathbf{L}_p \leftarrow \mathbf{L}_{\neg\varphi \wedge \psi}\} \cup \gamma(p,q,t,s,\neg t,\neg s,\varphi,\psi,\neg\varphi,\neg\varphi \wedge \psi).
\end{array}
$$

Moreover, for each $i > 10$, $\sigma_i(\Pi) = \sigma_{10}(\Pi)$. In fact, for $r = p \leftarrow \neg\varphi \wedge \psi$, we have that

$$\sigma_{10}^\upsilon(\Pi) = \{\mathbf{L}_{H(r)} \leftarrow \mathbf{L}_{B(r)}\} \cup \bigcup_{\phi \in \Sigma(\Pi)} \gamma(\phi) = \sigma(\Pi).$$

In general, the following result is easily shown:

Lemma 3 *Let υ be a Σ-selector for $NLP_\mathcal{A}$. Then, for any nested logic program $\Pi \in NLP_\mathcal{A}$, we have that $\sigma_n^\upsilon(\Pi) = \sigma(\Pi)$, for every $n \geq |\Sigma(\Pi)|$.*

Finally, preparatory for applying Theorem 1, we define the following functions.

Definition 6 *Let υ be a Σ-selector for $NLP_\mathcal{A}$. Then, the sequence $(\alpha_i^\upsilon)_{i\geq 0}$ of functions $\alpha_i^\upsilon : INT_\mathcal{A} \times NLP_\mathcal{A} \to INT_{\mathcal{A}\cup\mathcal{A}_\mathbf{L}}$ is recursively defined as follows:*

1. *$\alpha_0^\upsilon(\mathcal{I},\Pi) = \mathcal{I}$, for every $\mathcal{I} \in INT_\mathcal{A}$ and every $\Pi \in NLP_\mathcal{A}$;*
2. *for $i > 0$, α_i^υ is given by*

$$\alpha_i^\upsilon(\mathcal{I},\Pi) = \langle J'_H, J'_T\rangle$$

with

$$J'_w = \begin{cases} J_w \cup \mathbf{L}_{\phi_i} & \text{if } \phi_i^* \text{ is true in } w \text{ under } \langle J_H, J_T\rangle = \alpha_{i-1}^\upsilon(\mathcal{I},\Pi);\\ J_w & \text{otherwise,}\end{cases}$$

for $w \in \{H,T\}$, if $i \leq |\Sigma(\Pi)|$, where

- *ϕ_i is the i-th element of $\upsilon(\Pi)$, and*
- *ϕ_i^* results from ϕ_i by replacing each immediate subformula ψ of ϕ_i by $\mathbf{L}_\psi$,*

otherwise $\alpha_i^\upsilon(\mathcal{I},\Pi) = \alpha_{i-1}^\upsilon(\mathcal{I},\Pi)$, for every $\mathcal{I} \in INT_\mathcal{A}$ and every $\Pi \in NLP_\mathcal{A}$.

The key property of the sequence $(\alpha_i^\upsilon)_{i\geq 0}$ is the following result:

Lemma 4 *Let υ be a Σ-selector for $NLP_{\mathcal{A}}$. Then, for every $i \geq 0$, α_i^υ is a σ_i^υ-associated HT-homomorphism.*

Hence, by Theorem 1, every σ_i^υ is strongly faithful, for $i \geq 0$. But this in turn implies, in view of Lemma 3, that σ must be strongly faithful as well.

Summarising, we can state our main result as follows:

Theorem 2 *The function σ, mapping nested logic programs over alphabet $\mathcal{A}$ into disjunctive logic programs over alphabet $\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}$, is polynomial, strongly faithful, and modular.*

Since strong faithfulness implies faithfulness, we immediately 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:*

$$AS_{\mathcal{A}}(\Pi) \quad = \quad \{J \cap \mathcal{A} \mid J \in AS_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}(\sigma(\Pi))\}.$$

Remark. As mentioned previously, the mapping σ 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 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.

5.2 An Alternative Translation

In the following, we discuss a slightly different translation, τ, from nested programs into disjunctive programs in which negation is treated differently. To wit, similar to some applications in automated theorem proving, programs are first preprocessed by transforming them in here-and-there negational normal form. Afterwards, a somewhat simpler labeling step, corresponding to the original transformation σ, can then be applied.

For a given input alphabet $\mathcal{A}$, translation τ maps nested programs over $\mathcal{A}$ into disjunctive programs over $\mathcal{A}^* = \mathcal{A} \cup \mathcal{A}_{\mathbf{L}} \cup \bar{\mathcal{A}}$, where $\mathcal{A}_{\mathbf{L}}$ and $\bar{\mathcal{A}}$ are the alphabets introduced above (cf. Sections 3 and 5.1, respectively). It is assumed that $\mathcal{A}$, $\mathcal{A}_{\mathbf{L}}$, and $\bar{\mathcal{A}}$ are pairwise distinct.

The overall construction of τ proceeds in four steps, represented by the following functions:

- $\tau_1 : NLP_{\mathcal{A}} \to NLP_{\mathcal{A}}^{nnf}$ transforms the heads and bodies of the rules in a given program into HT-NNF;
- $\tau_2 : NLP_{\mathcal{A}}^{nnf} \to GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}^{ht}$ translates programs in HT-NNF into programs containing only rules with conjunctions of HT-literals in their bodies and disjunctions of HT-literals in their heads;
- $\tau_3 : GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}^{ht} \to GDLP_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}$ takes the programs returned by τ_2 and eliminates double negations in bodies and heads of rules; and finally

- $\tau_4 : GDLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}} \to DLP_{\mathcal{A}^*}$ eliminates possible negations in the heads of rules, returning disjunctive logic programs.

Combining these elements, translation τ is then given by the composed function $\tau_4 \circ \tau_3 \circ \tau_2 \circ \tau_1$, 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 programs over $\mathcal{A}^* = \mathcal{A} \cup \mathcal{A}_{\mathbf{L}} \cup \bar{\mathcal{A}}$. Similar to our original translation σ, our goal is to show that τ is PSM. Indeed, as we argue below, each constituting function τ_i $(1 \leq i \leq 4)$ is PSM.

Let us first consider translations τ_1 and τ_3. For transforming programs into HT-NNF, we simply use our normal form operator $\eta_N(\cdot)$, thus

$$\tau_1(\Pi) \quad = \quad \{\eta_N(H(r)) \leftarrow \eta_N(B(r)) \mid r \in \Pi\},$$

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

Similarly, τ_3 is realised by using the HT-equivalences of Item (A_7) in Proposition 3, applied to program rules. More specifically, for each $\Pi \in GDLP^{ht}_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$, $\tau_3(\Pi)$ is the result of replacing each rule $r \in \Pi$ of form

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

as well as each rule of form

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

where ϕ and ψ are expressions, and $p, q \in \mathcal{A}$. Since, both τ_1 and τ_3 are polynomial, modular, and retain HT-equivalence, the following result is immediate.

Lemma 5 *Translations $\tau_1 : NLP_{\mathcal{A}} \to NLP^{nnf}_{\mathcal{A}}$ and $\tau_3 : GDLP^{ht}_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}} \to GDLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$ are PSM.*

Moreover, translation τ_4 coincides with function ζ from Section 3, but defined over the input alphabet $\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}$. In view of Lemma 2, we thus know that τ_4 is PSM.

Lemma 6 *Translation $\tau_4 : GDLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}} \to DLP_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}\cup\bar{\mathcal{A}}}$ is PSM.*

It remains to discuss τ_2. Recall that $\Sigma(\Pi)$ denotes the set of all subformulas occurring in the rules of a given program Π.

Definition 7 *The function $\tau_2 : NLP^{nnf}_{\mathcal{A}} \to GDLP^{ht}_{\mathcal{A}\cup\mathcal{A}_{\mathbf{L}}}$ is defined by setting, for any $\Pi \in NLP^{nnf}_{\mathcal{A}}$,*

$$\tau_2(\Pi) = \{\mathbf{L}_{H(r)} \leftarrow \mathbf{L}_{B(r)} \mid r \in \Pi\} \cup \bigcup_{\phi\in\Sigma(\Pi)} \delta(\phi)$$

where, for each $\phi \in \Sigma(\Pi)$, the set $\delta(\phi)$ is defined as follows:

1. *if ϕ is an HT-literal, then $\delta(\phi)$ consists of the two rules*

$$\mathbf{L}_\phi \leftarrow \phi \quad \text{and} \quad \phi \leftarrow \mathbf{L}_\phi;$$

2. *if $\phi = (\psi_1 \wedge \psi_2)$, then $\delta(\phi)$ consists of the three rules*

$$\mathbf{L}_{\psi_1\wedge\psi_2} \leftarrow \mathbf{L}_{\psi_1} \wedge \mathbf{L}_{\psi_2}, \quad \mathbf{L}_{\psi_1} \leftarrow \mathbf{L}_{\psi_1\wedge\psi_2}, \quad \mathbf{L}_{\psi_2} \leftarrow \mathbf{L}_{\psi_1\wedge\psi_2}; \quad \text{and}$$

3. if $\phi = (\psi_1 \vee \psi_2)$, then $\delta(\phi)$ consists of the three rules

$$\mathbf{L}_{\psi_1} \vee \mathbf{L}_{\psi_2} \leftarrow \mathbf{L}_{\psi_1 \vee \psi_2}, \quad \mathbf{L}_{\psi_1 \vee \psi_2} \leftarrow \mathbf{L}_{\psi_1}, \quad \mathbf{L}_{\psi_1 \vee \psi_2} \leftarrow \mathbf{L}_{\psi_2}.$$

Observe that, since τ_2 is defined only for programs in HT-NNF, there is no need to have an explicit clause for negated subformulas; rather, contrasting Definition 4, all negations are handled like atomic formulas, in terms of HT-literals.

For illustration, let us apply transformation τ_2 to our running example

$$\Pi = \{p \leftarrow \neg(q \wedge \neg t) \wedge (q \vee \neg s)\}.$$

Transforming Π into HT-NNF, we obtain

$$\Pi' = \{p \leftarrow (\neg q \vee \neg\neg t) \wedge (q \vee \neg s)\}. \tag{13}$$

Then, $\tau_2(\Pi')$ is given by the following rules, using the abbreviations $\varrho = (\neg q \vee \neg\neg t)$ and $\chi = (q \vee \neg s)$:

$$\begin{aligned}
\tau_2(\Pi') \quad = \quad & \{\mathbf{L}_p \leftarrow \mathbf{L}_{\varrho \wedge \chi}\} & (14)\\
& \cup\{\mathbf{L}_{\varrho \wedge \chi} \leftarrow \mathbf{L}_\varrho \wedge \mathbf{L}_\chi, \mathbf{L}_\varrho \leftarrow \mathbf{L}_{\varrho \wedge \chi}, \mathbf{L}_\chi \leftarrow \mathbf{L}_{\varrho \wedge \chi}\} & (15)\\
& \cup\{\mathbf{L}_{\neg q} \vee \mathbf{L}_{\neg\neg t} \leftarrow \mathbf{L}_\varrho, \mathbf{L}_\varrho \leftarrow \mathbf{L}_{\neg q}, \mathbf{L}_\varrho \leftarrow \mathbf{L}_{\neg\neg t}\} & (16)\\
& \cup\{\mathbf{L}_q \vee \mathbf{L}_{\neg s} \leftarrow \mathbf{L}_\chi, \mathbf{L}_\chi \leftarrow \mathbf{L}_q, \mathbf{L}_\chi \leftarrow \mathbf{L}_{\neg s}\} & (17)\\
& \cup\{\mathbf{L}_{\neg\neg t} \leftarrow \neg\neg t, \neg\neg t \leftarrow \mathbf{L}_{\neg\neg t}\} & (18)\\
& \cup\{\mathbf{L}_{\neg v} \leftarrow \neg v, \neg v \leftarrow \mathbf{L}_{\neg v} \mid v \in \{q,t,s\}\} & (19)\\
& \cup\{\mathbf{L}_v \leftarrow v, v \leftarrow \mathbf{L}_v \mid v \in \{p,q,t,s\}\}. & (20)
\end{aligned}$$

We proceed with demonstrating that τ_2 is PSM. To begin with, it is obvious that τ_2 is both polynomial and modular. To show that τ_2 is strongly faithful, we show that $\sigma(\Pi)$ and $\tau_2(\Pi)$ are strongly equivalent, for every $\Pi \in NLP^{nnf}_{\mathcal{A}}$. From this, given that $\sigma(\Pi)$ is strongly faithful, strong faithfulness of $\tau_2(\Pi)$ is an immediate consequence. The details follow.

Let $\ell \in \{\gamma, \delta\}$, where $\gamma(\cdot)$ and $\delta(\cdot)$ are the set operators from Definitions 4 and 7, respectively, and consider

$$L^\ell(\Pi) = \{r \mid r \in \ell(\phi), \text{for some HT-literal } \phi \in \Sigma(\Pi)\},$$

for $\Pi \in NLP^{nnf}_{\mathcal{A}}$. Obviously, $\sigma(\Pi) \setminus L^\gamma(\Pi) = \tau_2(\Pi) \setminus L^\delta(\Pi)$ holds. Thus, it remains to show that $L^\gamma(\Pi)$ and $L^\delta(\Pi)$ are strongly equivalent. This can be seen as follows.

Consider an atom p occurring in Π. Then, the construction of $\sigma(\Pi)$ draws upon the following sets:

$$\begin{aligned}
\gamma(\neg\neg p) &= \{\mathbf{L}_{\neg\neg p} \leftarrow \neg\mathbf{L}_{\neg p},\ \bot \leftarrow \mathbf{L}_{\neg p} \wedge \mathbf{L}_{\neg\neg p}\};\\
\gamma(\neg p) &= \{\mathbf{L}_{\neg p} \leftarrow \neg\mathbf{L}_p,\ \bot \leftarrow \mathbf{L}_p \wedge \mathbf{L}_{\neg p}\};\\
\gamma(p) &= \{\mathbf{L}_p \leftarrow p,\ p \leftarrow \mathbf{L}_p\}.
\end{aligned}$$

Correspondingly, $\tau_2(\Pi)$ draws upon the following sets

$$\begin{aligned}
\delta(\neg\neg p) &= \{\mathbf{L}_{\neg\neg p} \leftarrow \neg\neg p,\ \neg\neg p \leftarrow \mathbf{L}_{\neg\neg p}\};\\
\delta(\neg p) &= \{\mathbf{L}_{\neg p} \leftarrow \neg p,\ \neg p \leftarrow \mathbf{L}_{\neg p}\};\\
\delta(p) &= \gamma(p) \quad = \quad \{\mathbf{L}_p \leftarrow p,\ p \leftarrow \mathbf{L}_p\}.
\end{aligned}$$

However, by the semantics of the logic of here-and-there, it is easily seen that $\gamma(\neg p) \cup \gamma(p)$ is strongly equivalent to $\delta(\neg p) \cup \delta(p)$, and, likewise, $\gamma(\neg\neg p) \cup \gamma(\neg p) \cup \gamma(p)$ is strongly equivalent to $\delta(\neg\neg p) \cup \delta(\neg p) \cup \delta(p)$. Moreover, observing that

1. $\neg\neg p \in \Sigma(\Pi)$ implies both $\gamma(\neg\neg p) \cup \gamma(\neg p) \cup \gamma(p) \subseteq \sigma(\Pi)$ and $\delta(\neg\neg p) \cup \delta(\neg p) \cup \delta(p) \subseteq \tau_2(\Pi)$,

2. $\neg p \in \Sigma(\Pi)$ implies both $\gamma(\neg p) \cup \gamma(p) \subseteq \sigma(\Pi)$ and $\delta(\neg p) \cup \delta(p) \subseteq \tau_2(\Pi)$, and

3. $\ell(\phi) \cap \ell(\psi) = \emptyset$ for HT-literals ϕ, ψ with $\phi \neq \psi$, where $\ell \in \{\gamma, \delta\}$,

it readily follows that $L^\gamma(\Pi)$ and $L^\delta(\Pi)$ are indeed strongly equivalent. We therefore obtain the following result:

Lemma 7 *Translation $\tau_2 : NLP^{nnf}_{\mathcal{A}} \to GDLP^{ht}_{\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}}$ is PSM.*

Summarising, we obtain the adequacy of the composed translation $\tau = \tau_4 \circ \tau_3 \circ \tau_2 \circ \tau_1$.

Theorem 3 *Let τ_1, τ_2, τ_3, and τ_4 be the functions defined above, and let $\tau = \tau_4 \circ \tau_3 \circ \tau_2 \circ \tau_1$.*

Then, translation τ, 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.

In concluding, we briefly sketch a slight optimisation of τ_2.

Define the notion of a *quasi-subformula* similar to the concept of a subformula, except that every HT-literal ϕ has only ϕ as its sole quasi-subformula. Accordingly, let $\Sigma^q(\Pi)$ be the collection of all quasi-subformulas occurring in expressions in a given program Π. Then, define τ_2^* like τ_2, but use $\Sigma^q(\Pi)$ instead of $\Sigma(\Pi)$. It can be shown that τ_2^* is PSM.

We illustrate this variant of τ_2 using the overall transformation $\tau^* = \tau_4 \circ \tau_3 \circ \tau_2^* \circ \tau_1$ applied on our running example

$$\Pi = \{p \leftarrow \neg(q \wedge \neg t) \wedge (q \vee \neg s)\}.$$

We know from (13) that $\tau_1(\Pi) = \Pi'$, where

$$\Pi' = \{p \leftarrow (\neg q \vee \neg\neg t) \wedge (q \vee \neg s)\}.$$

Furthermore,

$$\Sigma^q(\Pi') = \{p,\, q,\, \neg q,\, \neg s,\, \neg\neg t,\, \phi, \psi, \phi \vee \psi\},$$

Note that $\Sigma^q(\Pi')$ differs from $\Sigma(\Pi')$ in that the latter set additionally contains $\neg t$, t, and s. Hence, $\tau_2^*(\Pi') = \tau_2(\Pi') \setminus (\delta(\neg t) \cup \delta(t) \cup \delta(s))$.

In total, $\tau^*(\Pi)$ consists of the following rules. First of all, $\tau^*(\Pi)$ contains Rules (14)–(17) as well as $\{v \leftarrow \mathbf{L}_v, \mathbf{L}_v \leftarrow v \mid v \in \{p, q\}\}$. The rules in (18) are translated by τ_3 to $\mathbf{L}_{\neg\neg t} \vee \neg t \leftarrow \top$ and $\bot \leftarrow \mathbf{L}_{\neg\neg t} \wedge \neg t$. Finally, $\tau^*(\Pi)$ contains the rules for negative literals, $\mathbf{L}_{\neg q} \leftarrow \neg q$ and $\mathbf{L}_{\neg s} \leftarrow \neg s$, as well as the results of applying τ_4 to $\neg q \leftarrow \mathbf{L}_{\neg q}$, $\neg s \leftarrow \mathbf{L}_{\neg s}$, and $\mathbf{L}_{\neg\neg t} \vee \neg t \leftarrow \top$, respectively.

Table 1: The syntax of nlp input files.

Meaning	Symbols
$\bot, \top$	`false/0`, `true/0`
$\neg$	`not/1` (prefix)
$\wedge$	`,/1` (infix)
$\vee$	`;/1` (infix)
$\leftarrow$	`:-/1` (infix; in rule)

Thus, $\tau^*(\Pi)$ is given by the following sets of rules:

$$\begin{aligned}
\tau^*(\Pi) \quad = \quad & \{\mathbf{L}_p \leftarrow \mathbf{L}_{\phi\wedge\psi}\} \\
& \cup\{\mathbf{L}_{\phi\wedge\psi} \leftarrow \mathbf{L}_\phi \wedge \mathbf{L}_\psi,\, \mathbf{L}_\phi \leftarrow \mathbf{L}_{\phi\wedge\psi},\, \mathbf{L}_\psi \leftarrow \mathbf{L}_{\phi\wedge\psi}\} \\
& \cup\{\mathbf{L}_{\neg q} \vee \mathbf{L}_{\neg\neg t} \leftarrow \mathbf{L}_\phi,\, \mathbf{L}_\phi \leftarrow \mathbf{L}_{\neg q},\, \mathbf{L}_\phi \leftarrow \mathbf{L}_{\neg\neg t}\} \\
& \cup\{\mathbf{L}_q \vee \mathbf{L}_{\neg s} \leftarrow \mathbf{L}_\psi,\, \mathbf{L}_\psi \leftarrow \mathbf{L}_q,\, \mathbf{L}_\psi \leftarrow \mathbf{L}_{\neg s}\} \\
& \cup\{\mathbf{L}_{\neg\neg t} \vee \overline{t} \leftarrow \top,\, \bot \leftarrow \mathbf{L}_{\neg\neg t} \wedge \neg t\} \\
& \cup\{\mathbf{L}_{\neg v} \leftarrow \neg v,\, \overline{v} \leftarrow \mathbf{L}_{\neg v} \mid v \in \{q,s\}\} \\
& \cup\{\bot \leftarrow v \wedge \overline{v},\, \overline{v} \leftarrow \neg v \mid v \in \{q,t,s\}\} \\
& \cup\{\mathbf{L}_v \leftarrow v,\, v \leftarrow \mathbf{L}_v \mid v \in \{p,q\}\}.
\end{aligned}$$

6 Experimental Results

6.1 The nlp system

We have implemented the translation $\zeta \circ \lambda$ from Section 3 as well the two structural transformations σ and τ as a front-end to the disjunctive logic programming system DLV [7, 8].[4] This system represents a state-of-the-art implementation for disjunctive logic programs under the answer sets semantics [11]. 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 the SICStus Prolog system and comprises roughly 900 lines of code. As mentioned above, it implements the three major translations $\zeta \circ \lambda$, σ, and τ. In addition, it contains a number of optional improvements that allow for more compact code generation.

The syntax of nlp is summarised in Table 1.

For illustration, the code for our running example $\Pi = \{p \leftarrow (\neg q \vee r) \wedge (q \vee \neg s)\}$, which can be saved, e.g., in file `example.nlp`, is given by

```
p :- not (q;  not t), (q; not s).
```

Once the file is read into nlp, it is subject to multiple transformations. Most of these transformations are rule-centered in the sense that they apply in turn to each single rule. While the original file is supposed to have the extension `nlp`, the result of the respective compilation is indicated by the extension `dlp` ($\zeta \circ \lambda$),

[4]With a slight abuse of language, here we will refer to function $\zeta \circ \lambda$ as "language-preserving translation" although, strictly speaking, only λ is language-preserving and ζ adds new symbols.

`str` (σ), and `htl` (τ), respectively. Applying the implementation of the three transformations $\zeta \circ \lambda$, σ, and τ, respectively, to the source file `example.nlp` yields the files given in Figure 1–3.

The first rule in Figure 1 illustrates that we (currently) refrain from any sophisticated logical simplifications. On the other hand, the implemented versions of our structural translations are optionally improvable (via a flag) in two simple yet effective ways: First, no labels are generated for literals and second only a single label is generated for any n-ary disjunctions or conjunctions. All our experiments are conducted using this optimisation. For completeness, we give the non-optimized code in Figure 4. Also, three different schemas for label generation are supported; here, we adhere to a number-oriented version.

6.2 Experimental analysis

In view of the already considerably diverging sizes on our toy example, it is interesting to compare the resulting images of our translations in a systematic way. The remainder of this section is therefore dedicated to an experimental analysis that aims at evaluating the relative performance of the considered translations. To this end, we have implemented two different benchmark series that produce scalable problem instances.

Normal form series: The first one 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 10 parameters: on the one hand, the number of variables, rules, facts, and constraints; on the other hand, the structure of heads and bodies is fixed by three parameters: the number of connectives on the first and second level plus the number of literals connected to the first level. The placement of literals (here: p, $\neg p$, and $\neg\neg p$), respectively, is done randomly. These parameters are also reflected in the name of the source file. For instance, the file `test_50.20_5_0.3_2_0.2_3_1.30.2.nlp` comprises a program over an alphabet of 50 variables, containing twenty rules, such as

```
p42,p4;p11,not p50;p46,p26 :-
          (not not p10;p37;p27),(p20;p34;p39),not not p11.
```

plus five facts and no constraints, expressed by `20_5_0`. While the primary connective of the head is a disjunction, namely '`;`', the one of the body is conjunction '`,`'. Although this can be arbitrarily fixed, it is currently set this way in order to provoke a significant blow-up when translating programs by means of successive applications of distributivity. The above head parameters, '`3_2_0`', enforce heads (being disjunctions) composed of 3 conjunctions, containing themselves 2 literals, along with zero literal disjuncts. The body parameters, '`2_3_1`', lead to conjunctions with 2 disjunctions, containing 3 literals, and 1 literal conjunct, viz. '`not not p11`' in the rule above.[5] Finally, the filename ends with an estimated blow-up factor for the language-preserving translation, viz. '`30.2`' in our example.

Given a nested logic program Π, its blow-up factor, $\theta(\Pi)$, is computed as follows:

$$\theta(\Pi) = \frac{[r \cdot (u_1 + u_3 + v_1 + v_3) \cdot v_2^{v_1} \cdot u_2^{u_1}] + [h \cdot (u_1 + u_3) \cdot u_2^{u_1}] + [b \cdot (v_1 + v_3) \cdot v_2^{v_1}]}{[(r + h) \cdot (u_1 \cdot u_2 + u_3)] + [(r + b) \cdot (v_1 \cdot v_2 + v_3)]},$$

[5] The number of literals in the secondary disjuncts and conjuncts must always be greater or equal 2 in order to be distinct from the literals on the first level.

```
p;not_t:-not q,q.
p;not_t:-not q,not s.
not_t:-not t.
:-t,not_t.
```

Figure 1: Resulting code for Π under the "language-preserving" translation $\zeta \circ \lambda$ (`example.dlp`).

```
p:-l3,l5.
l3:-not l2.
:-l3,l2.
q;l1:-l2.
l2:-q.
l2:-l1.
l1:-not t.
:-l1,t.
q;l4:-l5.
l5:-q.
l5:-l4.
l4:-not s.
:-l4,s.
```

Figure 2: Resulting code for Π under the first structural translation σ (`example.str`).

```
p;not_t:-not q,l1.
q;not_s:-l1.
l1:-q.
l1:-not s.
not_t:-not t.
:-t,not_t.
not_s:-not s.
:-s,not_s.
```

Figure 3: Resulting code for Π under the second structural translation τ (`example.htl`).

```
l1:-l10.
l1:-p.
p:-l1.
l10:-l6,l9.
l6:-l10.
l9:-l10.
l6:-not l5.
:-l6,l5.
l2;l4:-l5.
l5:-l2.
l5:-l4.
l4:-not l3.
:-l4,l3.
l3:-t.
t:-l3.
l2;l8:-l9.
l9:-l2.
l9:-l8.
l2:-q.
q:-l2.
l8:-not l7.
:-l8,l7.
l7:-s.
s:-l7.
```

```
l1:-l8.
l1:-p.
p:-l1.
l8:-l4,l7.
l4:-l8.
l7:-l8.
l4:-l2,l3.
l2:-l4.
l3:-l4.
l2:-not q.
not_q:-l2.
l3;not_t.
:-l3,not t.
l5;l6:-l7.
l7:-l5.
l7:-l6.
l5:-q.
q:-l5.
l6:-not s.
not_s:-l6.
not_q:-not q.
:-q,not_q.
not_t:-not t.
:-t,not_t.
not_s:-not s.
:-s,not_s.
```

Figure 4: Non-optimized code for Π under the two structural translations σ and τ (`example.str`/`example.htl`), respectively.

where r stands for the number of rules; h, b gives the number of heads and bodies in the program; u_1, u_2, u_3 are the head parameters; and v_1, v_2, v_3 are the body parameters. The denominator of $\theta(\Pi)$ gives the size of Π in terms of its number of literals, whilst the numerator captures the size of the disjunctive program resulting from the language-preserving translation.

Cardinality constraints series: The second generator is based on the idea that (single headed) logic programs with cardinality literals can be transformed into nested logic programs [9]. A cardinality literal has the form $m\{L_1, \ldots, L_n\}n$ and means informally that the resulting answer set must contain at least m and at most n literals out of $\{L_1, \ldots, L_n\}$; see [35] for a detailed semantics. Such literals can then be used in the same way as ordinary literals in the head as well as the body of rules.

One may argue that this test series is less significant since the major decision problems for this class of programs are still on the first level of the polynomial hierarchy, whereas those of nested logic programs are on the second one. However, cardinality constraints constitute nowadays one of the major modeling technique in answer set programming, so that it is interesting to see how the different translations behave on this type of examples. Although our test sets remain somewhat artificial, they reflect the topology of practicably relevant programs using cardinality constraints.

As above, we start with generating a parameterised program with cardinality literals. Similarly, our test cases are parameterised by 11 options, reflected in the respective file name. For instance, a file name `test_35.20_5_0.1_3_2.1_2_1_3.135.2.wc` [6] contains rules over 35 variables; it includes 20 rules, 5 facts, and 0 constraints. The following three numbers, viz. `1_3_2`, parameterise the cardinality literals in the rule heads. They give the lower bound, number of literals, and the upper bound, respectively. This induces rule heads of form `1:{p9,p23,p11}:2` (where the two colons '`:`' are inserted for easier parsing by Prolog). The next quadruple, `1_2_1_3`, determines rule bodies: the lower bound, the number of literals, the upper bound, and the number of cardinality constraints in the body. Our exemplary file contains then rules of form

```
1:{p9,p23,p11}:2 :-
        1:{p1,not p20}:1, 1:{p17,p8}:1, 1:{not p29,not p9}:1.
```

This file is then translated into a nested program according to [9]. The blow-up factor, viz. `135.2`, estimates (as above) the factor of the language-preserving translation applied to the obtained nested logic program.

These generators, along with the fully detailed test series, can be downloaded at URL

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

In all, we generated 4661 nested logic programs, all of which where translated by each of the three translations and the respective image was subsequently passed to DLV. All tests were run under Linux (Mandrake 8.1) on bi-processors based on AMD Athlon MP 1200 MHz with 1GByte main memory. For simplicity, we fixed the initial size of the programs: While in the first series all nested programs contain 20 rules and 5 facts, the same holds in the second series for the programs containing cardinality constraints.[7] For simplicity, both series are further divided into subseries according to the number of variables involved; in fact,

[6] The extension `wc` stands for weight constraints, although our test series deal with cardinality constraints only (so far).

[7] The nested logic programs obtained in the second series are slightly larger, varying from 30 to 55 rules, depending on the respective constraints.

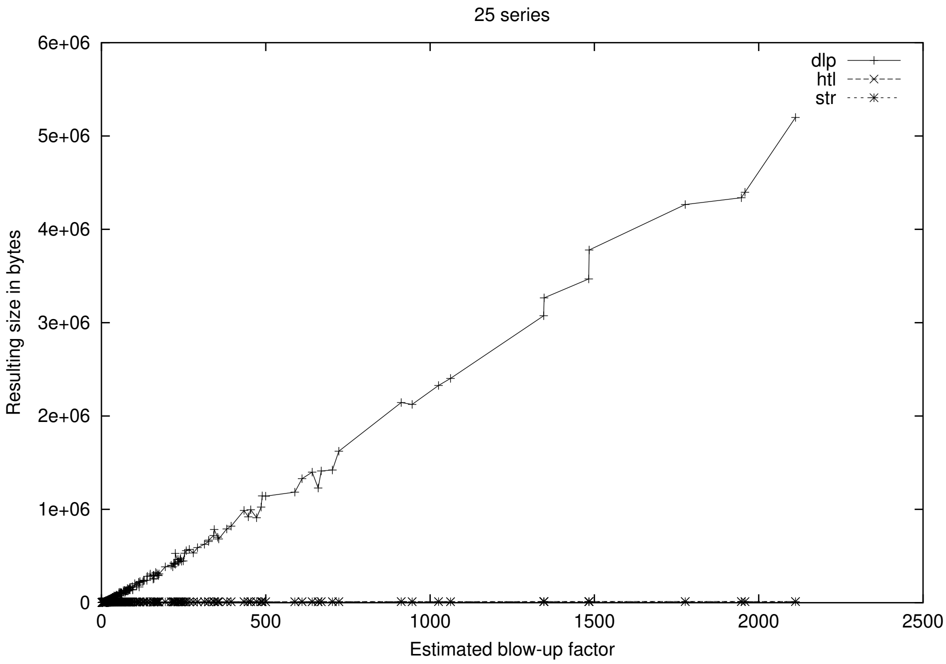

Figure 5: Normal form: Resulting size in bytes.

we ran subseries with 10, 15, 20,...,45, 50 variables. Since the results of all these subseries are reported at the aforecited URL, we concentrate here on one representative, given by the 25 variables subseries. This subseries comprises 260 normal form programs and 426 cardinality constraints programs. The obtained results are given in Figures 5–10 and Figures 11–16, respectively.[8] In all figures, the x-axis represents the estimated blow-up factor; this allows us to impose a certain order on the respective features, expressed by the y-axis. In particular, this factor allows us to identify thresholds indicating when which translation outperforms another (see below).

While Figure 5 and 11, respectively, give the size of the translated files in bytes, Figure 6 and 12 do the same in terms of rules.

In all cases, we observe, as expected, a significant increase in the size of the image under the language-preserving translation. This is confirmed by Figure 7 and 13, respectively, where the ratios between the size of the output and input file (in bytes) are shown. Also, looking at Table 2, we see that the majority of test cases has a smaller image under the structural transformations than under the language-preserving one. However, this is less significant for the cardinality constraints series than for the normal form series.

Clearly, the size of the image affects also the compilation time.[9] We thus get a similar behavior in Figure 8 and 14, respectively, as regards compilation time, as we obtained for the size of the image (cf. also Table 2). Things become more interesting when comparing the respective run times (for computing *all* answer sets of a given program); these are given in Figure 9 and 15, respectively. For the first time, we obtain a kind of threshold beyond which the structural transformations always outperform the language-preserving one. While this is the case for an estimated blow-up factor of 500 for the normal-form test cases, the one

[8]These figures contain only those test cases on which all three transformations remained under the time limit.

[9]In fact, our implementation contains a subprocedure having quadratic time complexity, which amplifies the effect of the (intermediate) program size on the compilation time.

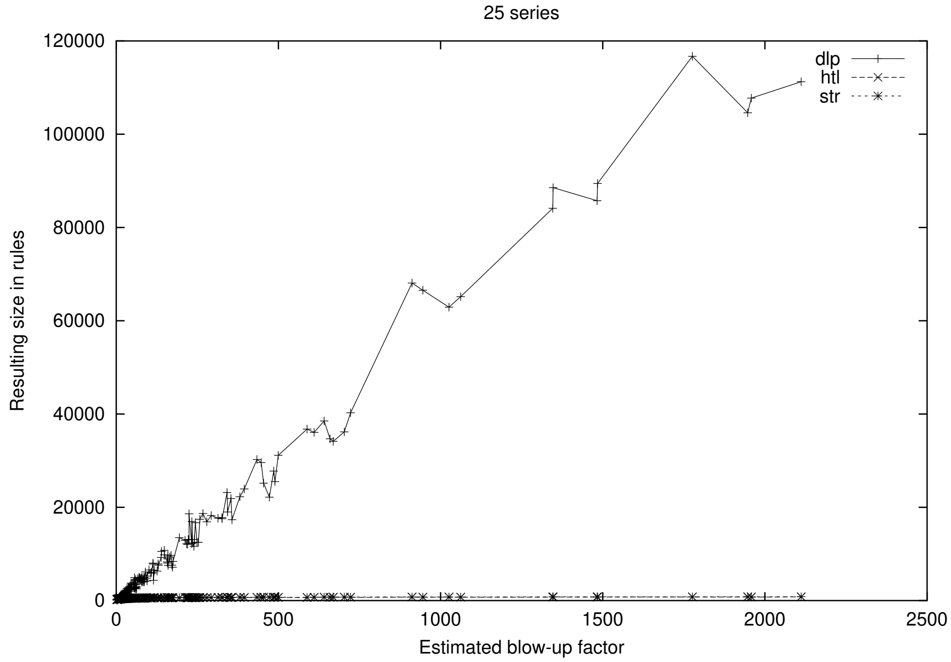

Figure 6: Normal form: Resulting size in rules.

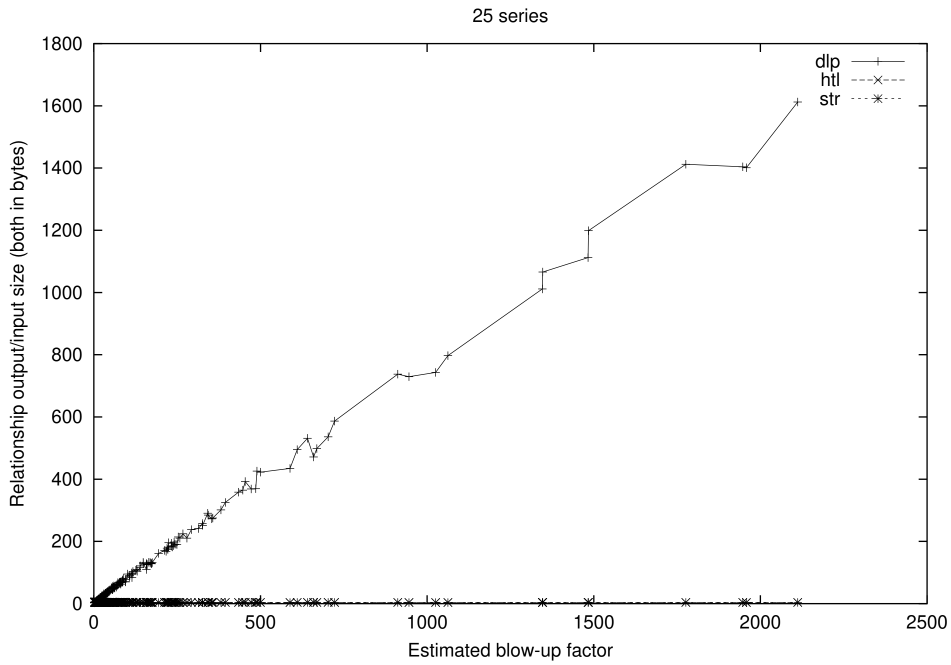

Figure 7: Normal form: Relationship output/input size (both in bytes).

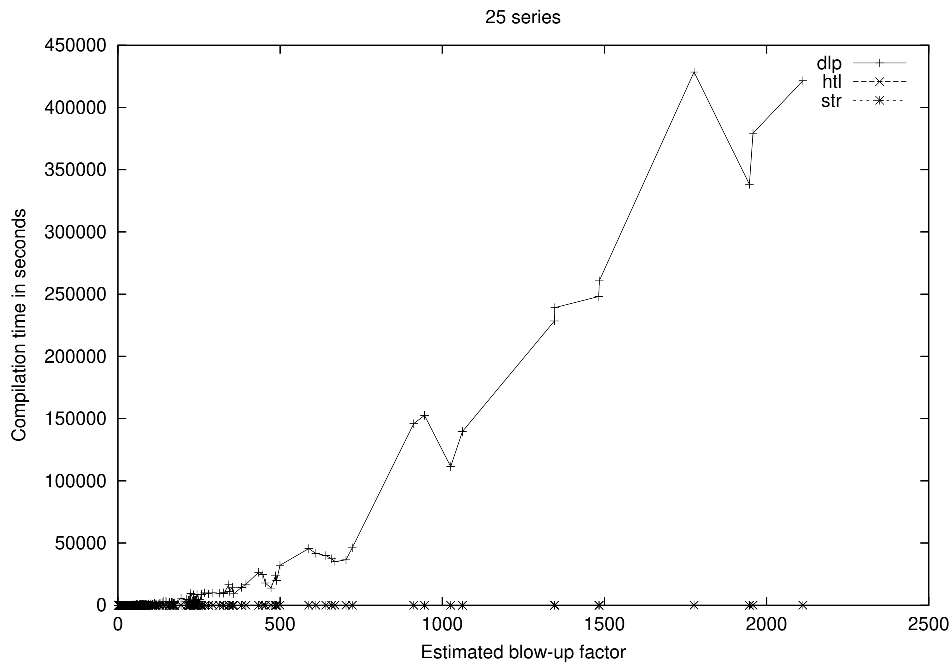

Figure 8: Normal form: Compilation time.

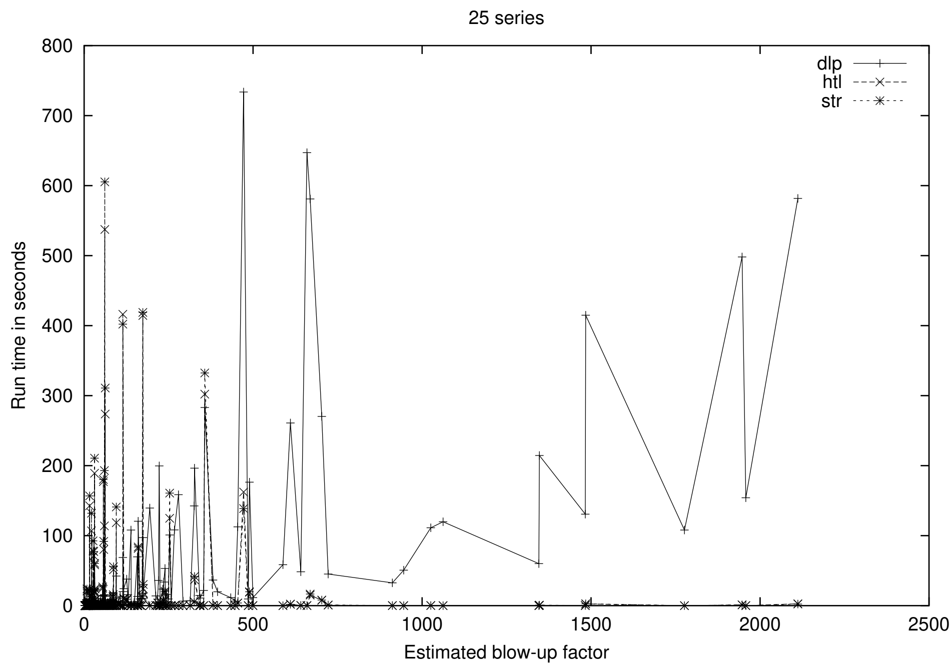

Figure 9: Normal form: Run time.

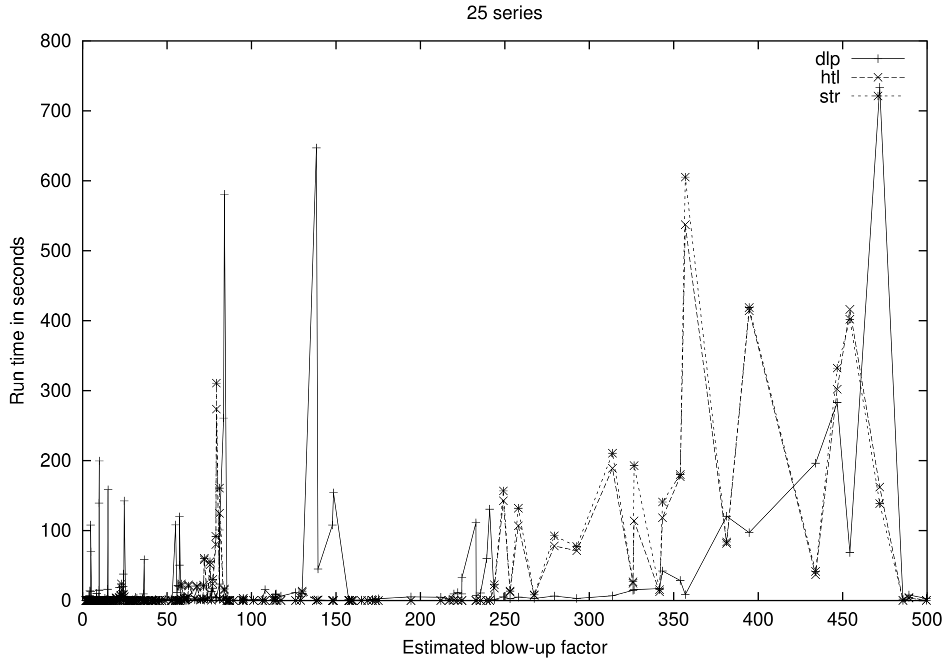

Figure 10: Normal form: Run time, focused.

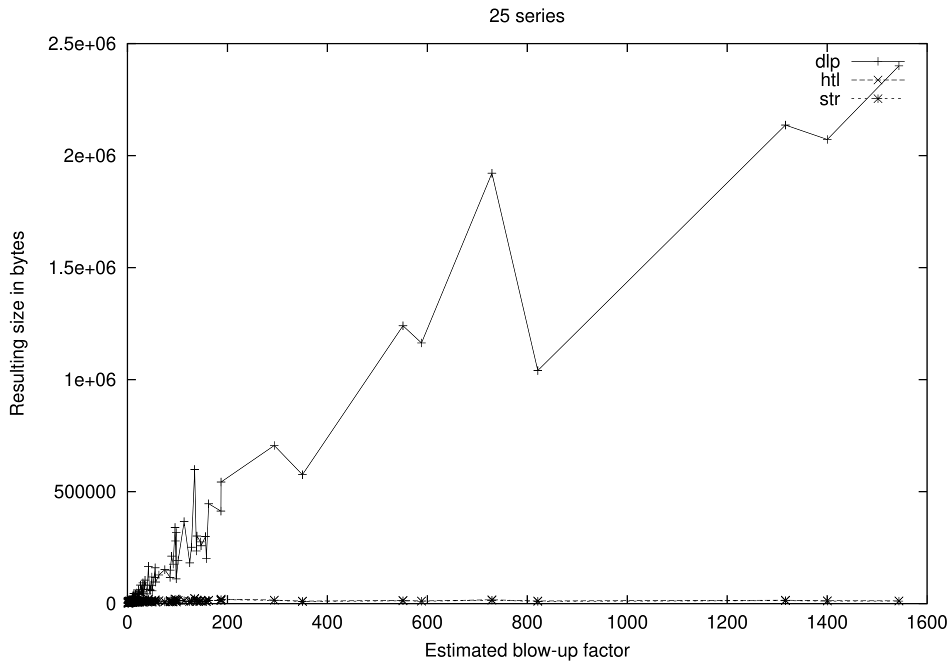

Figure 11: Cardinality constraints: Resulting size in bytes.

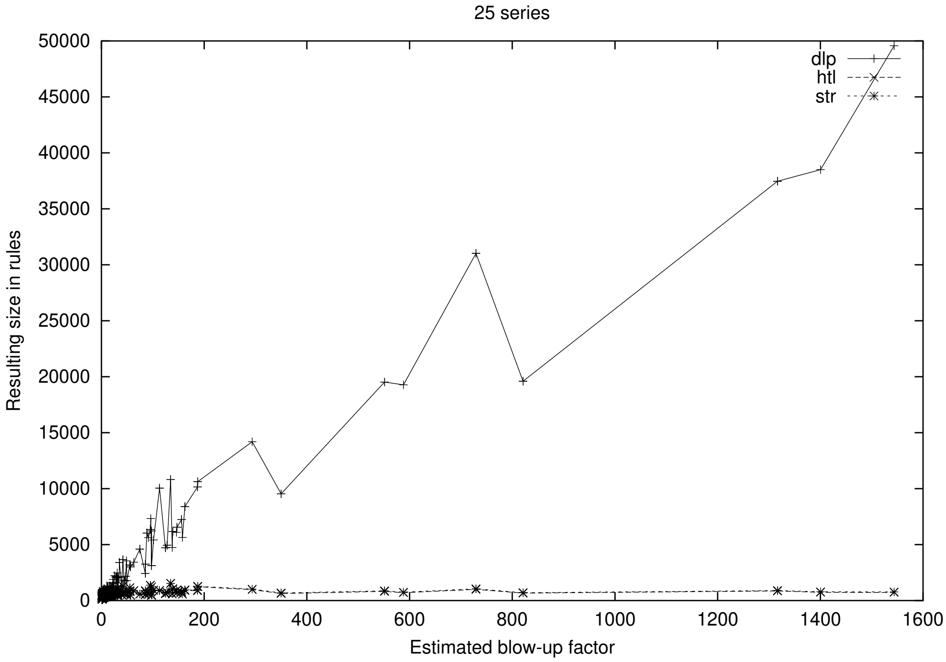

Figure 12: Cardinality constraints: Resulting size in rules.

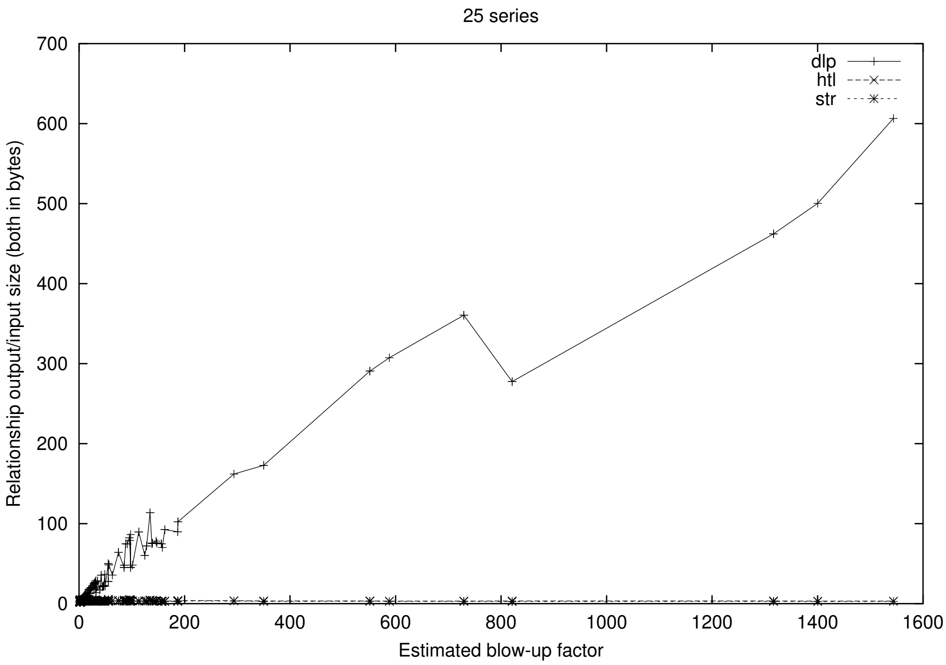

Figure 13: Cardinality constraints: Relationship output/input size (both in bytes).

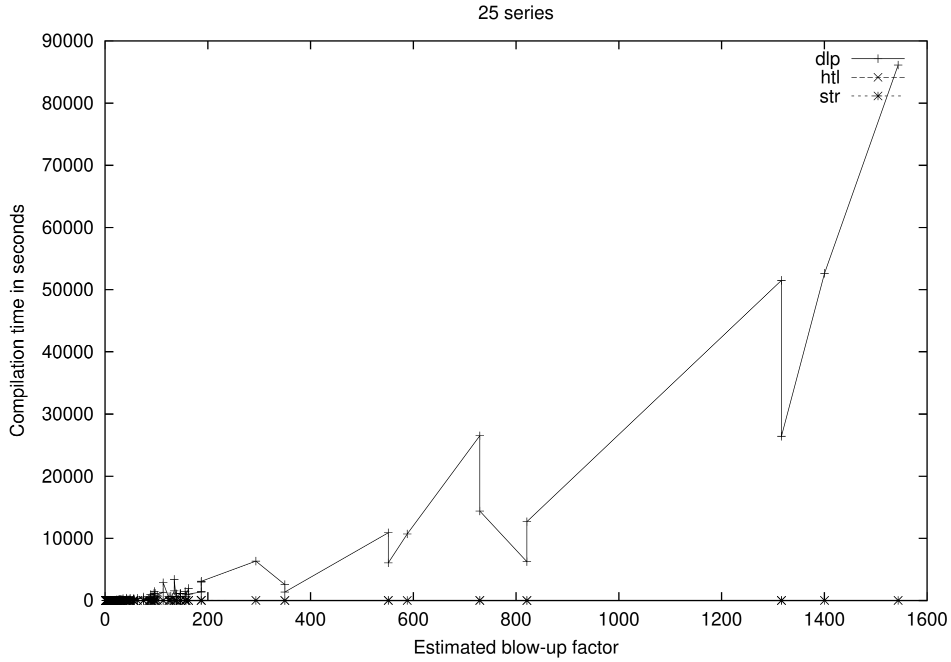

Figure 14: Cardinality constraints: Compilation time.

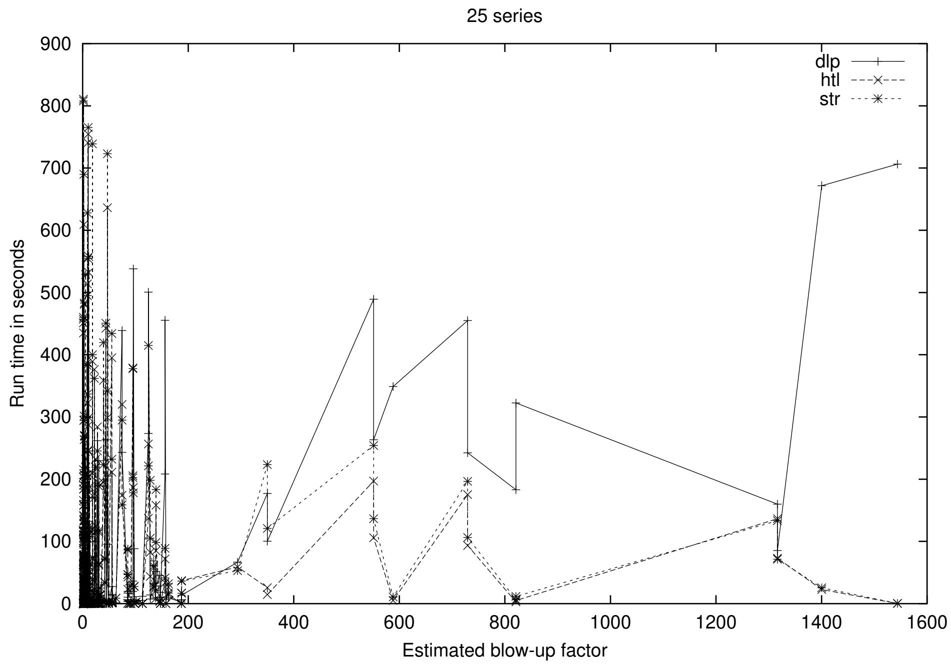

Figure 15: Cardinality constraints: Run time.

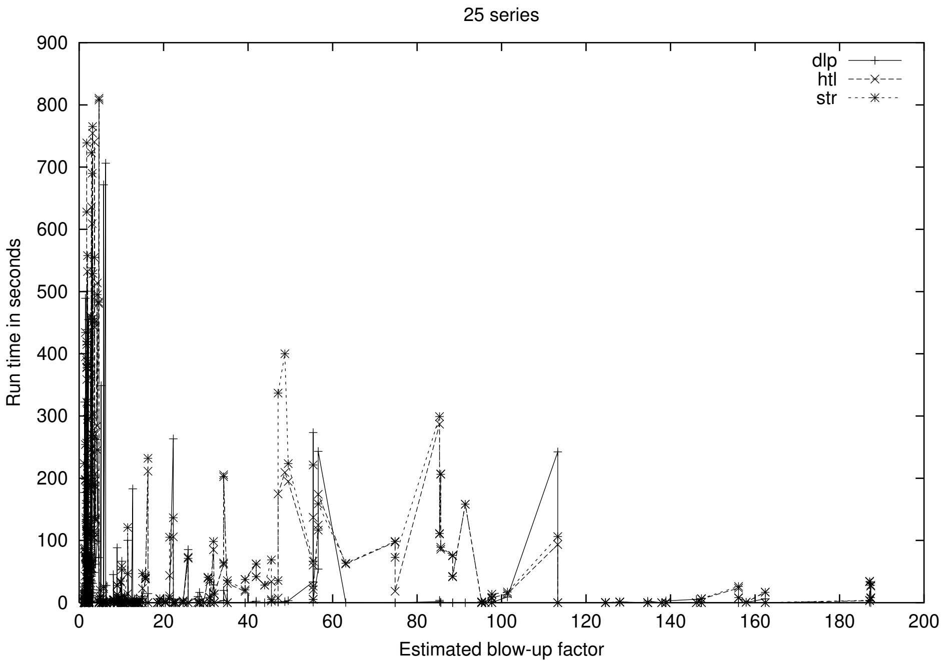

Figure 16: Cardinality constraints: Run time, focused.

for the cardinality-constraint test cases is at 200. The fact that the results of all translations appear to be scattered below this threshold is confirmed by the detailed results under the threshold given in Figure 10 and 16, respectively. Interestingly, both figures reveal that the language-preserving translation may even be outperformed on rather small blow-up factors, as witnessed by the range of 0-50 on the x-axis in Figure 10. Although these thresholds are sometimes less clear cut in the other test series, their existence seems to be a recurring feature.

7 Conclusion

We have developed several translations of logic programs with nested expressions into disjunctive logic programs. We have proven that our translations are 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 major translations as a front-end for the system DLV [7, 8]. The corresponding

Table 2: Quantitative comparison.

	Normal form				Cardinality constraints			
Feature	dlp<htl	htl<dlp	dlp<str	str<dlp	dlp<htl	htl<dlp	dlp<str	str<dlp
Size in Bytes	15	245	13	247	146	279	122	303
Size in Rules	22	238	25	235	208	217	209	216
Compilation time	10	250	13	247	182	243	196	227
Runtime	63	190	63	192	282	115	307	104

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, because our goal is to *reconstruct the answer sets* of the original programs from the translated ones. Furthermore, we note that our translations amount to *semi-representability* results in the sense of [25], insofar as we embed a full-fledged formalism into one of its (syntactically) restricted fragments by means of language extension.

With the particular labeling technique employed here, our translation avoids the risk of an exponential blow-up in the worst-case, faced by a previous approach of Lifschitz *et al.* [23] 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 translations, we have also implemented the distributive translation into disjunctive logic programs in order to conduct an initial experimental study. Although we have concentrated above on a certain representative subseries, we have seen that there is an unconditional benefit beyond a certain threshold.

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.

A Proofs

First, we state some simple technical result which we need later on.

Proposition 8 *Let C_ϕ be a formula over some alphabet $\mathcal{A}$ containing one or more designated occurrences of a formula ϕ, and let C_p be the result of replacing the designated occurrences of ϕ in C_ϕ by a new atom $p \notin \mathcal{A}$.*

Then, for any HT-interpretation $\mathcal{I} = \langle I_H, I_T \rangle$ over $\mathcal{A}$ and any $w \in \{H, T\}$, it holds that

$$\nu_{\mathcal{I}}(w, C_\phi) = \nu_{\mathcal{J}}(w, C_p),$$

where $\mathcal{J} = \langle J_H, J_T \rangle$ is given by $J_{w'} = I_{w'} \cup \{p\}$ if $\nu_{\mathcal{I}}(w', \phi) = 1$, and $J_{w'} = I_{w'}$ otherwise, for each $w' \in \{H, T\}$.

Proof. The proof proceeds by straightforward induction on the logical complexity of ϕ. □

A.1 Proof of Lemma 1

Let $\rho : NLP_{\mathcal{A}_1} \to NLP_{\mathcal{A}_2}$ be some function with $\mathcal{A}_1 \subseteq \mathcal{A}_2$, and assume that $\alpha_\rho : INT_{\mathcal{A}_1} \times NLP_{\mathcal{A}_1} \to INT_{\mathcal{A}_2}$ is a ρ-associated HT-homomorphism. We must show that ρ is faithful, i.e., that for each $\Pi \in NLP_{\mathcal{A}_1}$, there is a one-to-one correspondence between the answer sets of Π and the answer sets of $\rho(\Pi)$ such that

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

holds. In order to show this, we construct, for each $\Pi \in NLP_{\mathcal{A}_1}$, a function $\beta_{\rho,\Pi} : EM_{\mathcal{A}_1}(\Pi) \to EM_{\mathcal{A}_2}(\rho(\Pi))$ from the set of equilibrium models of Π over $\mathcal{A}_1$ into the set of equilibrium models of $\rho(\Pi)$ over $\mathcal{A}_2$ satisfying the following properties:

(a) $\beta_{\rho,\Pi}$ is bijective; and

(b) $\beta_{\rho,\Pi}^{-1} : EM_{\mathcal{A}_2}(\rho(\Pi)) \to EM_{\mathcal{A}_1}(\Pi)$ is given by $\beta_{\rho,\Pi}^{-1}(\langle J, J\rangle) = \langle J \cap \mathcal{A}_1, J \cap \mathcal{A}_1\rangle$, for each $\langle J, J\rangle \in EM_{\mathcal{A}_2}(\rho(\Pi))$.

From this, by invoking Proposition 1, faithfulness of ρ follows.

We proceed with the construction of function $\beta_{\rho,\Pi}$. To begin with, we show that under the conditions of the lemma, the following two properties hold, for any program $\Pi \in NLP_{\mathcal{A}_1}$:

(P_1) if $\mathcal{I}$ is an equilibrium model of Π, then $\alpha_\rho(\mathcal{I}, \Pi)$ is an equilibrium model of $\rho(\Pi)$, for every total HT-interpretation $\mathcal{I}$ over $\mathcal{A}_1$; and

(P_2) if $\mathcal{J}$ is an equilibrium model of $\rho(\Pi)$, then there is some $\mathcal{I} \in EM_{\mathcal{A}_1}(\Pi)$ such that $\mathcal{J} = \alpha_\rho(\mathcal{I}, \Pi)$, for every total HT-interpretation $\mathcal{J}$ over $\mathcal{A}_2$.

We first show Condition (P_1). Consider some $\Pi \in NLP_{\mathcal{A}_1}$, and let $\mathcal{I} = \langle I, I\rangle$ be an equilibrium model of Π, i.e., $\mathcal{I}$ is an HT-model of Π, and there is no $K \subset I$ such that $\langle K, I\rangle$ is also an HT-model of Π. Given that $\mathcal{I}$ is a total HT-model of Π, Conditions (H_1) and (H_2) of Definition 3 imply that $\alpha_\rho(\mathcal{I}, \Pi)$ is a total HT-model of $\rho(\Pi)$. Consider $\langle J, J\rangle = \alpha_\rho(\mathcal{I}, \Pi)$. Since $\alpha_\rho(\mathcal{I}, \Pi)$ is a ρ-associated HT-embedding, it holds that $J \cap \mathcal{A}_1 = I$ and $J \setminus \mathcal{A}_1 \subseteq var(\rho(\Pi))$. Hence, $J \subseteq \mathcal{A}_1 \cup var(\rho(\Pi))$. Suppose that there is some $L \subset J$ such that $\langle L, J\rangle$ is an HT-model of $\rho(\Pi)$. Then, according to Condition (H_4) of Definition 3, there is some $\langle K', I'\rangle \in INT_{\mathcal{A}_1}$ such that $\alpha_\rho(\langle K', I'\rangle) = \langle L, J\rangle$. But since α_ρ is a ρ-associated HT-embedding, we have $I' = J \cap \mathcal{A}_1 = I$. Hence, given that $L \subset J$, by applying Condition (H_3) of Definition 3, we get that $K' \subset I$. Furthermore, by Condition (H_1) of Definition 3, $\langle K', I\rangle$ is an HT-model of Π. This contradicts our hypothesis that $\mathcal{I} = \langle I, I\rangle$ is an equilibrium model of Π. It follows that there is no $L \subset J$ such that $\langle L, J\rangle$ is an HT-model of $\rho(\Pi)$. Consequently, $\alpha_\rho(\mathcal{I}, \Pi)$ is an equilibrium model of $\rho(\Pi)$. This proves that $\mathcal{I}$ is an equilibrium model of Π only if $\alpha_\rho(\mathcal{I}, \Pi)$ is an equilibrium model of $\rho(\Pi)$.

As for Property (P_2), let $\mathcal{J} = \langle J, J\rangle$ be an equilibrium model of $\rho(\Pi)$. Then, $J \subseteq var(\rho(\Pi))$, and Condition (H_4) implies that there is some $\mathcal{I} = \langle I_H, I_T\rangle \in INT_{\mathcal{A}_1}$ such that $\mathcal{J} = \alpha_\rho(\mathcal{I}, \Pi)$. We claim that $\mathcal{I}$ is an equilibrium model of Π. Indeed, since $\mathcal{J}$ is a total HT-model of $\rho(\Pi)$, by Conditions (H_1) and (H_2) it follows that $\mathcal{I}$ is a total HT-model of Π. So, in particular, we have that $I_H = I_T = I$. Additionally, Condition (H_3) guarantees that there is no $K \subset I$ such that $\langle K, I\rangle$ is an HT-model of Π. Consequently, $\mathcal{I}$ is an equilibrium model of Π, which proves the claim.

Now, in view of Properties (P_1) and (P_2), we can define the mapping $\beta_\rho^\Pi : EM_{\mathcal{A}_1}(\Pi) \to EM_{\mathcal{A}_2}(\rho(\Pi))$ by setting

$$\beta_{\rho,\Pi}(\mathcal{I}) = \alpha_\rho(\mathcal{I}, \Pi),$$

for any equilibrium model $\mathcal{I}$ of ρ. It remains to show that Conditions (a) and (b) hold.

Concerning Condition (a), we must show that $\beta_{\rho,\Pi}$ is one-to-one and onto. First of all, in view of Property (P_2), $\beta_{\rho,\Pi}$ is clearly onto. As well, since α_ρ satisfies the condition that $\alpha_\rho(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{J}, \Pi)$ only if $\mathcal{I} = \mathcal{J}$, $\beta_{\rho,\Pi}$ is one-to-one.

Finally, as for Condition (b), consider the function $\xi_{\rho,\Pi} : EM_{\mathcal{A}_2}(\rho(\Pi)) \to EM_{\mathcal{A}_1}(\Pi)$, given by

$$\xi_{\rho,\Pi}(\langle J, J\rangle) = \langle J \cap \mathcal{A}_1, J \cap \mathcal{A}_1\rangle,$$

for each $\langle J, J\rangle \in EM_{\mathcal{A}_2}(\rho(\Pi))$. Since $\beta_{\rho,\Pi}$ is onto and $\alpha_\rho(\mathcal{I}, \Pi)$ satisfies the condition that $J \cap \mathcal{A}_1 = I$, for each $\langle I, I\rangle \in EM_{\mathcal{A}_1}(\Pi)$ with $\langle J, J\rangle = \alpha_\rho(\mathcal{I}, \Pi)$, it is easy to check that, for each $\langle I, I\rangle \in EM_{\mathcal{A}_1}(\Pi)$ and each $\langle J, J\rangle \in EM_{\mathcal{A}_2}(\rho(\Pi))$, we have that

$$\langle J, J\rangle = \beta_{\rho,\Pi}(\langle I, I\rangle) \quad \text{iff} \quad \langle I, I\rangle = \xi_{\rho,\Pi}(\langle J, J\rangle).$$

Hence, $\xi_{\rho,\Pi} = \beta^{-1}_{\rho,\Pi}$.

A.2 Proof of Theorem 1

Let $\rho : NLP_{\mathcal{A}_1} \to NLP_{\mathcal{A}_2}$ with $\mathcal{A}_1 \subseteq \mathcal{A}_2$ be a modular translation and $\alpha_\rho : INT_{\mathcal{A}_1} \times NLP_{\mathcal{A}_1} \to INT_{\mathcal{A}_2}$ a corresponding ρ-associated HT-homomorphism. For any pair $P = \langle \Pi_1, \Pi_2\rangle$, where $\Pi_1, \Pi_2 \in NLP_{\mathcal{A}_1}$, we define a function $\rho^P : NLP_{\mathcal{A}_1} \to NLP_{\mathcal{A}_2}$ as follows:

$$\rho^P(\Pi) = \begin{cases} \rho(\Pi_1) \cup \Pi_2 & \text{if } \Pi = \Pi_1 \cup \Pi_2; \\ \rho(\Pi) & \text{otherwise.} \end{cases}$$

We show that each such ρ^P is faithful. From this, strong faithfulness of ρ is an immediate consequence. Indeed, fix some program $\Pi_1 \in NLP_{\mathcal{A}_1}$, and consider any $\Pi_2 \in NLP_{\mathcal{A}_1}$. Then, for $P_0 = \langle \Pi_1, \Pi_2\rangle$, we have the following two properties, given that ρ^{P_0} is faithful:

1. There is a one-to-one correspondence between the answer sets of $\Pi_1 \cup \Pi_2$ and the answer sets of $\rho^{P_0}(\Pi_1 \cup \Pi_2)$; and
2. $AS_{\mathcal{A}_1}(\Pi_1 \cup \Pi_2) = \{J \cap \mathcal{A}_1 \mid J \in AS_{\mathcal{A}_2}(\rho^{P_0}(\Pi_1 \cup \Pi_2))\}$.

Since $\rho^{P_0}(\Pi_1 \cup \Pi_2) = \rho(\Pi_1) \cup \Pi_2$, the above two conditions express strong faithfulness of ρ.

We continue with the proof that ρ^P is faithful, for every $P = \langle \Pi_1, \Pi_2\rangle$. We first show that under the conditions of the theorem, the following property holds:

($*$) For any $\Pi \in NLP_{\mathcal{A}_1}$, any $\Pi' \subseteq \Pi$, and any $\mathcal{I} \in INT_{\mathcal{A}_1}$, if $\mathcal{I}$ is an HT-model of Π, then

$$\alpha_\rho(\mathcal{I}, \Pi') = \langle J_H \cap (var(\rho(\Pi')) \cup \mathcal{A}_1), J_H \cap (var(\rho(\Pi')) \cup \mathcal{A}_1)\rangle,$$

where $\langle J_H, J_T\rangle = \alpha_\rho(\mathcal{I}, \Pi)$.

Let Π, Π', and $\langle J_H, J_T\rangle$ be as above, and assume that $\mathcal{I} = \langle I_H, I_T\rangle$ is an HT-model of Π. Since α_ρ is an HT-homomorphism, it follows that $\langle J_H, J_T\rangle$ is an HT-model of $\rho(\Pi)$. By the modularity of ρ, we have that $\rho(\Pi) = \rho(\Pi') \cup \rho(\Pi \setminus \Pi')$. Hence, $\langle J_H, J_T\rangle$ is an HT-model of $\rho(\Pi')$. In particular

$$\langle J'_H, J'_T\rangle = \langle J_H \cap (var(\rho(\Pi')) \cup \mathcal{A}_1), J_H \cap (var(\rho(\Pi')) \cup \mathcal{A}_1)\rangle$$

is an HT-model of $\rho(\Pi')$ as well, since variables not occurring in $var(\rho(\Pi')) \cup \mathcal{A}_1$ have no influence on the truth-value evaluation process. Applying Property (H_4) of Definition 3 for α_ρ, we thus get that there is some $\langle I'_H, I'_T\rangle$ such that $\langle J'_H, J'_T\rangle = \alpha_\rho(\langle I'_H, I'_T\rangle, \Pi')$. It remains to show that $\langle I_H, I_T\rangle = \langle I'_H, I'_T\rangle$. This can be seen as follows. Let $w \in \{H, T\}$. By the absorption law of set theory, we have that

$$J'_w \cap \mathcal{A}_1 = \big(J_w \cap (var(\rho(\Pi')) \cup \mathcal{A}_1)\big) \cap \mathcal{A}_1 = J_w \cap \mathcal{A}_1.$$

But since α_ρ is an HT-embedding, it holds that $J'_w \cap \mathcal{A}_1 = I'_w$and $J_w \cap \mathcal{A}_1 = I_w$. Hence, $I_w = I'_w$. This concludes the proof of Property $(*)$.

Fix now some pair $P = \langle \Pi_1, \Pi_2\rangle$, with $\Pi_1, \Pi_2 \in NLP_{\mathcal{A}_1}$. In view of Lemma 1, for showing that ρ^P is faithful, it suffices to show that there exists some ρ^P-associated HT-homomorphism. To this end, consider the function $\alpha_{\rho^P} : INT_{\mathcal{A}_1} \times NLP_{\mathcal{A}_1} \to INT_{\mathcal{A}_2}$, given by

$$\alpha_{\rho^P}(\mathcal{I}, \Pi) = \begin{cases} \alpha_\rho(\mathcal{I}, \Pi_1) & \text{if } \Pi = \Pi_1 \cup \Pi_2; \\ \alpha_\rho(\mathcal{I}, \Pi) & \text{otherwise.} \end{cases}$$

We claim that α_{ρ^P} is the desired ρ^P-associated HT-homomorphism. We first show that α_{ρ^P} is a ρ^P-associated HT-embedding.

Let $\mathcal{I} = \langle I_H, I_T\rangle$ be an HT-interpretation over $\mathcal{A}_1$, Π a program over $\mathcal{A}_1$, and $w \in \{H, T\}$. We must show that Conditions (E_1) and (E_2) of Definition 2 are satisfied. If $\Pi \neq \Pi_1 \cup \Pi_2$, then $\alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi)$, and (E_1) and (E_2) hold trivially, since α_ρ is a ρ-associated HT-embedding and $\rho^P(\Pi) = \rho(\Pi)$. Now assume that $\Pi = \Pi_1 \cup \Pi_2$. In this case, we have that $\alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi_1)$ and $\rho^P(\Pi) = \rho(\Pi_1) \cup \Pi_2$. Again, (E_1) is a direct consequence of the fact that α_ρ is a ρ-associated HT-embedding. Now let $\langle J_H, J_T\rangle = \alpha_{\rho^P}(\mathcal{I}, \Pi)$. We obtain that $J_w \setminus \mathcal{A}_1 \subseteq var(\rho(\Pi_1))$, for $w \in \{H, T\}$, since α_ρ is a ρ-associated HT-embedding and $\alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi_1)$. But given that $\rho^P(\Pi) = \rho(\Pi_1) \cup \Pi_2$, and thus $var(\rho(\Pi_1)) \subseteq var(\rho^P(\Pi))$, it follows that $J_w \setminus \mathcal{A}_1 \subseteq var(\rho^P(\Pi))$, for $w \in \{H, T\}$. This shows that Condition (E_2) is again satisfied.

It remains to show that α_{ρ^P} satisfies Conditions (H_1)–(H_4) of Definition 3, for any $\mathcal{I}, \mathcal{I}' \in INT_{\mathcal{A}_1}$ and any $\Pi \in NLP_{\mathcal{A}_1}$.

So, consider some $\mathcal{I}, \mathcal{I}'$, and Π as required. If $\Pi \neq \Pi_1 \cup \Pi_2$, then (H_1)–(H_4) clearly hold, given that $\alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi)$ and $\rho^P(\Pi) = \rho(\Pi)$, and since α_ρ is a ρ-associated HT-homomorphism. Assume now that $\Pi = \Pi_1 \cup \Pi_2$. Then, $\alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi_1)$ and $\rho^P(\Pi) = \rho(\Pi_1) \cup \Pi_2$.

Suppose that $\mathcal{I}$ is an HT-model of Π. Then, $\mathcal{I}$ is *a fortiori* an HT-model of Π_1, and since α_ρ is a ρ-associated HT-homomorphism, we obtain that $\alpha_\rho(\mathcal{I}, \Pi_1)$ is an HT-model of $\rho(\Pi_1)$. Furthermore, since $\mathcal{I}$ is also an HT-model of Π_2 and α_ρ preserves truth values over $\mathcal{A}_1$, it follows that $\alpha_\rho(\mathcal{I}, \Pi_1)$ is an HT-model of Π_2. Consequently, $\alpha_\rho(\mathcal{I}, \Pi_1)$ is an HT-model of $\rho^P(\Pi) = \rho(\Pi_1) \cup \Pi_2$. This proves that $\mathcal{I}$ is a model of Π only if $\alpha_{\rho^P}(\mathcal{I}, \Pi)$ is an HT-model of $\rho^P(\Pi)$. Conversely, let $\alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi_1)$ be an HT-model of $\rho^P(\Pi) = \rho(\Pi_1) \cup \Pi_2$. Then $\alpha_\rho(\mathcal{I}, \Pi_1)$ is an HT-model of $\rho(\Pi_1)$, and therefore $\mathcal{I}$ is an HT-model of Π_1, since α_ρ is an HT-homomorphism. Moreover, $\mathcal{I}$ is also an HT-model of Π_2. Thus, $\mathcal{I}$ is an HT-model of $\Pi = \Pi_1 \cup \Pi_2$. This proves Condition (H_1).

Concerning Condition (H_2), we have that $\mathcal{I}$ is a total HT-interpretation iff $\alpha_\rho(\mathcal{I}, \Pi_1)$ is a total HT-interpretation, since α_ρ is a ρ-associated HT-homomorphism. Since $\alpha_\rho(\mathcal{I}, \Pi_1) = \alpha_{\rho^P}(\mathcal{I}, \Pi)$, we obtain that $\mathcal{I}$ is a total HT-interpretation iff $\alpha_{\rho^P}(\mathcal{I}, \Pi)$, and thus (H_2) holds.

Now consider Condition (H_3). Let $\mathcal{I} = \langle I_H, I_T\rangle$ and $\mathcal{I}' = \langle I'_H, I'_T\rangle$ be HT-models of $\rho(\Pi)$. We must show that $I_H \subset I'_H$ and $I_T = I'_T$ iff $J_H \subset J'_H$ and $J_T = J'_T$, for

$$\langle J_H, J_T\rangle = \alpha_{\rho^P}(\mathcal{I}, \Pi) = \alpha_\rho(\mathcal{I}, \Pi_1) \quad \text{and} \quad \langle J'_H, J'_T\rangle = \alpha_{\rho^P}(\mathcal{I}', \Pi) = \alpha_\rho(\mathcal{I}', \Pi_1).$$

First of all, if $J_H \subset J'_H$ and $J_T = J'_T$, we immediately get $I_H \subset I'_H$ and $I_T = I'_T$, since α_ρ is a ρ-associated HT-homomorphism. For the converse, assume $I_H \subset I'_H$ and $I_T = I'_T$, and let $\langle K_H, K_T\rangle = \alpha_\rho(\mathcal{I}, \Pi)$ and $\langle K'_H, K'_T\rangle = \alpha_\rho(\mathcal{I}', \Pi)$. Given that α_ρ is a ρ-associated HT-homomorphism, it follows that $K_H \subset K'_H$ and $K_T = K'_T$. By Property $(*)$, we have that

$$J_w = K_w \cap (\mathcal{A}_1 \cup var(\rho(\Pi_1))) \quad \text{and} \quad J'_w = K'_w \cap (\mathcal{A}_1 \cup var(\rho(\Pi_1))), \tag{21}$$

for $w \in \{H, T\}$. Hence, $J_H \subseteq J'_H$ and $J_T = J'_T$. We claim that $J_H \subset J'_H$ must hold. Suppose otherwise, i.e., assume that $J_H = J'_H$. Then, given that α_ρ is a ρ-associated HT-embedding, we obtain that $I_H = I'_H$. This, however, contradicts our assumption that $I_H \subset I'_H$. It follows that $I_H \subset I'_H$ and $I_T = I'_T$ only if $J_H \subset J'_H$ and $J_T = J'_T$.

Finally, consider Condition (H_4). We must show that an HT-interpretation $\mathcal{J}$ over $\mathcal{A}_1 \cup var(\rho^P(\Pi))$ is an HT-model of $\rho^P(\Pi)$ only if $\mathcal{J} \in \alpha_{\rho^P}(INT_{\mathcal{A}_1}, \Pi)$. So, consider an HT-model $\mathcal{J}$ of $\rho^P(\Pi) = \rho(\Pi_1) \cup \Pi_2$. Then, $\mathcal{J}$ is an HT-model of $\rho(\Pi_1)$, and we obtain that $\mathcal{J} \in \alpha_\rho(INT_{\mathcal{A}_1}, \Pi_1)$, since α_ρ is an HT-homomorphism. But $\alpha_{\rho^P}(INT_{\mathcal{A}_1}, \Pi) = \alpha_\rho(INT_{\mathcal{A}_1}, \Pi_1)$, by definition of α_{ρ^P}. Hence, $\mathcal{J} \in \alpha_{\rho^P}(INT_{\mathcal{A}_1}, \Pi)$.

A.3 Proof of Lemma 2

We must show that (i) $\alpha_{\zeta'}$ is an ζ'-associated HT-embedding and (ii) that for each HT-interpretation $\mathcal{I} \in INT_{\mathcal{A}}$ and each $\Pi \in NLP_{\mathcal{A}}$, Conditions (H_1)–(H_4) of Definition 3 are satisfied. For the case that Π is not a generalised disjunctive logic program, i.e., whenever $\Pi \in NLP_{\mathcal{A}} \setminus GDLP_{\mathcal{A}}$, these conditions are trivially fulfilled, since we have $\zeta'(\Pi) = \Pi$ and $\alpha_{\zeta'}(\mathcal{I}, \Pi) = \mathcal{I}$.

So assume that $\Pi \in GDLP_{\mathcal{A}}$, and consider $\mathcal{I} \in INT_{\mathcal{A}}$. By definition of $\alpha_{\zeta'}$, it is quite obvious that $\alpha_{\zeta'}$ is a ζ'-associated HT-embedding, since $\kappa(I_T, \Pi)$ extends both the here-part as well as the there-part of $\mathcal{I}$ only by atoms of form $\overline{p}$ which are guaranteed to occur in $\zeta'(\Pi) = \zeta(\Pi)$.

We proceed with Condition (H_1). Here, we must show that $\mathcal{I}$ is an HT-model of Π iff $\alpha_{\zeta'}(\mathcal{I}, \Pi)$ is an HT-model of $\zeta'(\Pi)$. First, we note the following property, which can be shown in a straightforward way:

$(*)$ An HT-interpretation $\langle J_H, J_T\rangle$ over $\mathcal{A} \cup \bar{\mathcal{A}}$ is an HT-model of

$$\{\bot \leftarrow (p \wedge \overline{p}), \overline{p} \leftarrow \neg p \mid \neg p \text{ occurs in the head of some rule in } \Pi\} \tag{22}$$

iff for each $p \in \mathcal{A}$ occurring in (22) and each $w \in \{H, T\}$, it holds that $\overline{p} \in J_w$ iff $p \notin J_T$.

Hence, applying this property, we immediately obtain that $\alpha_{\zeta'}(\mathcal{I}, \Pi)$ is an HT-model of (22). It remains to show that $\mathcal{I}$ is an HT-model of Π iff $\alpha_{\zeta'}(\mathcal{I}, \Pi)$ is an HT-model of $\overline{\Pi}$. But this a direct consequence of Proposition 8, by observing that $\overline{\Pi}$ results from Π by replacing each occurrence of an expression $\neg p$ in the heads of the rules in Π by $\overline{p}$, and given that $\nu_{\mathcal{I}}(H, \neg p) = \nu_{\mathcal{I}}(T, \neg p)$, for any $p \in \mathcal{A}$.

Concerning (H_2), it follows directly from the definition of $\alpha_{\zeta'}$ that $\mathcal{I}$ is total iff $\alpha_{\zeta'}(\mathcal{I}, \Pi)$ is total.

As for (H_3), let $\mathcal{I} = \langle I_H, I_T\rangle$ and $\mathcal{I}' = \langle I'_H, I'_T\rangle$. Clearly, $\kappa(I_T, \Pi) = \kappa(I'_T, \Pi)$ whenever $I_T = I'_T$. Therefore, $I'_H \subset I_H$ and $I_T = I'_T$ are jointly satisfied precisely if we have that $(I'_H \cup \kappa(I'_T, \Pi)) \subset (I_H \cup \kappa(I_T, \Pi))$ and $(I'_T \cup \kappa(I'_T, \Pi)) = (I_T \cup \kappa(I_T, \Pi))$. Hence, Condition (H_3) holds.

Finally, consider Condition (H_4). Observe that each HT-interpretation $\langle J_H, J_T\rangle$ over $\mathcal{A} \cup var(\zeta'(\Pi))$ not contained in $\alpha_{\zeta'}(INT_{\mathcal{A}}, \Pi)$ does not satisfy the condition that, for each $w \in \{H, T\}$ and each $p \in \mathcal{A}$ occurring in (22), $\overline{p} \in J_w$ iff $p \in J_T$. Therefore, by Condition $(*)$, $\langle J_H, J_T\rangle$ is not an HT-model of (22). In particular, $\langle J_H, J_T\rangle$ is then also not an HT-model of $\zeta'(\Pi)$, as (22) is a subset of $\zeta'(\Pi)$. This proves Condition (H_4).

A.4 Proof of Lemma 4

We show by induction on $i \geq 0$ that α_i^v is a σ_i^v-associated HT-homomorphism.

INDUCTION BASE: For $i = 0$, the proposition holds trivially, since for each $\mathcal{I} \in \mathit{INT}_{\mathcal{A}}$ and each $\Pi \in \mathit{NLP}_{\mathcal{A}}$, we have $\sigma_i^v(\Pi) = \Pi$ and $\alpha_i^v(\mathcal{I}, \Pi) = \mathcal{I}$.

INDUCTION STEP: Let $i > 0$ and assume that for each $j < i$, α_j^v is a σ_j^v-associated HT-homomorphism.

We must show that (i) α_i^v is an σ_i^v-associated HT-embedding and (ii) that for each HT-interpretation $\mathcal{I} \in \mathit{INT}_{\mathcal{A}}$ and each $\Pi \in \mathit{NLP}_{\mathcal{A}}$, Conditions (H_1)–(H_4) of Definition 3 are satisfied.

Fix some $\Pi \in \mathit{NLP}_{\mathcal{A}}$, and let ϕ_i be the i-th element of $v(\Pi)$.

For the case that $i > |\Sigma(\Pi)|$, we have $\sigma_i^v(\Pi) = \sigma_{i-1}^v(\Pi)$ and $\alpha_i^v(\mathcal{I}, \Pi) = \alpha_{i-1}^v(\mathcal{I}, \Pi)$. Therefore, by induction hypothesis, α_i^v is a σ_i^v-associated HT-homomorphism.

So assume that $i \leq |\Sigma(\Pi)|$. By definition of α_i^v, it is quite obvious that α_i^v is a σ_i^v-associated HT-embedding, since $\alpha_i^v(\mathcal{I}, \Pi)$ results from $\alpha_{i-1}^v(\mathcal{I}, \Pi)$ by extending at most both the here-part as well as the there-part by label $\mathbf{L}_{\phi_i}$. Moreover, by definition of σ_i^v, $var(\sigma_i^v(\Pi)) = var(\sigma_{i-1}^v(\Pi)) \cup \{\mathbf{L}_{\phi_i}\}$ holds. Hence, since by induction hypothesis α_{i-1}^v is a σ_{i-1}^v-associated HT-embedding, we get that α_i^v is a σ_i^v-associated HT-embedding.

We proceed with Condition (H_1). Here, we must show that $\mathcal{I}$ is an HT-model of Π iff $\alpha_i^v(\mathcal{I}, \Pi)$ is an HT-model of $\sigma_i^v(\Pi)$. First, we note that the following property holds, for any expression ϕ over $\mathcal{A}$ and $\gamma(\cdot)$ as in Definition 4:

$(*)$ An HT-interpretation $\langle J_H, J_T \rangle$ over $\mathcal{A} \cup \mathcal{A}_{\mathbf{L}}$ is an HT-model of $\gamma(\phi)$ iff for each $w \in \{H, T\}$, it holds that $\mathbf{L}_\phi \in J_w$ iff ϕ^* is true under $\langle J_H, J_T \rangle$ in w, where ϕ^* results from ϕ as described in Definition 5.

Additionally, ϕ^* is true under $\langle J_H, J_T \rangle$ in $w \in \{H, T\}$ iff ϕ^* is true under $\langle J_H \setminus \{\mathbf{L}_\phi\}, J_T \setminus \{\mathbf{L}_\phi\} \rangle$ in w, since $\mathbf{L}_\phi$ does not occur in ϕ^* by definition.

Applying Property $(*)$, we immediately obtain that $\alpha_i^v(\mathcal{I}, \Pi)$ is an HT-model of $\gamma(\phi_i)$. It remains to show that $\mathcal{I}$ is an HT-model of Π iff $\alpha_i^v(\mathcal{I}, \Pi)$ is an HT-model of $(\sigma_{i-1}^v(\Pi))[\phi_i^*/\mathbf{L}_{\phi_i}]$. By induction hypothesis, $\mathcal{I}$ is an HT-model of Π iff $\alpha_{i-1}^v(\mathcal{I}, \Pi)$ is an HT-model of $\sigma_{i-1}^v(\Pi)$. But, by Proposition 8, we immediately get that the latter holds iff $\alpha_i^v(\mathcal{I}, \Pi)$ is an HT-model of $(\sigma_{i-1}^v(\Pi))[\phi_i^*/\mathbf{L}_{\phi_i}]$, which proves Condition (H_1).

Concerning (H_2), by induction hypothesis we already have that $\mathcal{I}$ is total iff $\alpha_{i-1}^v(\mathcal{I}, \Pi)$ is total. But we also have that $\alpha_i^v(\mathcal{I}, \Pi)$ is total iff $\alpha_{i-1}^v(\mathcal{I}, \Pi)$ is total, which follows directly from the definition of α_i^v and the fact that a formula is true unter a total HT-interpretation iff it is true in the world T.

As for (H_3), let $\langle I_H, I_T \rangle$ and $\langle I'_H, I'_T \rangle$ be HT-interpretations over $\mathcal{A}$, and consider

$$\mathcal{K} = \langle K_H, K_T \rangle = \alpha_{i-1}^v(\langle I_H, I_T \rangle, \Pi) \quad \text{and} \quad \mathcal{K}' = \langle K'_H, K'_T \rangle = \alpha_{i-1}^v(\langle I'_H, I'_T \rangle, \Pi),$$

and

$$\langle J_H, J_T \rangle = \alpha_i^v(\langle I_H, I_T \rangle, \Pi) \quad \text{and} \quad \langle J'_H, J'_T \rangle = \alpha_i^v(\langle I'_H, I'_T \rangle, \Pi).$$

By induction hypothesis, $I'_H \subset I_H$ and $I_T = I'_T$ are jointly satisfied precisely if $K'_H \subset K_H$ and $K_T = K'_T$. So, it remains to show that $K'_H \subset K_H$ and $K_T = K'_T$ iff $J'_H \subset J_H$ and $J_T = J'_T$. First, assume that $K'_H \subset K_H$ and $K_T = K'_T$ holds. Observe that ϕ_i^* is true under $\mathcal{K}$ in T iff ϕ_i^* is true under $\mathcal{K}'$ in T, since evaluating a formula under an interpretation $\mathcal{J}$ in T solely depends on the there part of $\mathcal{J}$. Hence, $J_T = J'_T$. For the here parts, observe that $\mathbf{L}_{\phi_i}$ is neither contained in K'_H nor in K_H by definition. Since,

$J'_H \setminus \{\mathbf{L}_\phi\} = K'_H$ as well as $J_H \setminus \{\mathbf{L}_\phi\} = K_H$, we have $J'_H \subset J_H$ whenever $K'_H \subset K_H$. Now assume $J'_H \subset J_H$ and $J_T = J'_T$. We immediately get $K'_T = K_T$ and $K'_H \subseteq K_H$. Assume that $K'_H = K_H$. Then, $\mathcal{K} = \mathcal{K}'$, and so $\langle I_H, I_T\rangle = \langle I'_H, I'_T\rangle$. Hence, $\alpha_i^v(\langle I_H, I_T\rangle, \Pi) = \alpha_i^v(\langle I'_H, I'_T\rangle, \Pi)$, which implies $J_H = J'_H$. This, however, violates our hypothesis that $J'_H \subset J_H$. Consequently, $K'_H \subset K_H$.

Finally, consider Condition (H_4). We show that each $\langle J_H, J_T\rangle$ over $\mathcal{A} \cup var(\sigma_i^v(\Pi))$ not contained in $\alpha_i^v(INT_\mathcal{A}, \Pi)$ is not an HT-model of $\sigma_i^v(\Pi)$. Fix such a $\langle J_H, J_T\rangle \notin \alpha_i^v(INT_\mathcal{A}, \Pi)$. We distinguish two cases.

First, assume that for each $w \in \{H, T\}$, it holds that $\mathbf{L}_{\phi_i} \in J_w$ iff ϕ_i^* is true under $\langle J_H, J_T\rangle$ in w. Then, $\mathcal{K} = \langle J_H \setminus \{\mathbf{L}_{\phi_i}\}, J_T \setminus \{\mathbf{L}_{\phi_i}\}\rangle \notin \alpha_{i-1}^v(INT_\mathcal{A}, \Pi)$. By induction hypothesis, $\mathcal{K}$ is not an HT-model of $\sigma_{i-1}^v(\Pi)$. But since $\langle J_H, J_T\rangle$ results from K by setting, for each $w \in \{H, T\}$, $\mathbf{L}_{\phi_i} \in J_w$ iff ϕ_i^* is true under $\langle J_H, J_T\rangle$ in w, $\langle J_H, J_T\rangle$ cannot be an HT-model of $(\sigma_{i-1}^v(\Pi))[\phi_i^*/\mathbf{L}_{\phi_i}]$, by Proposition 8. Since the latter program is contained in $\sigma_i^v(\Pi)$, $\langle J_H, J_T\rangle$ is then also not an HT-model of $\sigma_i^v(\Pi)$.

Now assume that there is some $w \in \{H, T\}$ such that $\mathbf{L}_{\phi_i} \in J_w$ is not equivalent to the condition that ϕ_i^* is true under $\langle J_H, J_T\rangle$ in w. By Condition $(*)$, we immediately get that $\langle J_H, J_T\rangle$ is not an HT-model of $\gamma(\phi_i)$. In particular, $\langle J_H, J_T\rangle$ is then also not an HT-model of $\sigma_i^v(\Pi)$, as $\gamma(\phi_i)$ is a subset of $\sigma_i^v(\Pi)$.

References

[1] M. Baaz, U. Egly, and A. Leitsch. Normal Form Transformations. In A. Robinson and A. Voronkov, editors, *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 T. Eiter, W. Faber, and M. Truszczynski, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNCS*, pages 186–199. Springer-Verlag, 2001.

[4] D. de Jongh and L. Hendriks. Characterization of Strongly Equivalent Logic Programs in Intermediate Logics. *Theory and Practice of Logic Programming (TPLP)*, 2002. To appear.

[5] U. Egly. On Different Structure-Preserving Translations to Normal Form. *Journal of Symbolic Computation*, 22(2):121–142, 1996.

[6] U. Egly. On Definitional Transformations to Normal Form for Intuitionistic Logic. *Fundamenta Informaticae*, 29(1,2):165–201, 1997.

[7] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Nonmonotonic Reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97)*, volume 1265 of *LNCS*, pages 363–374. Springer-Verlag, 1997.

[8] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons and Benchmarks. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 406–417. Morgan Kaufmann, 1998.

[9] P. Ferraris and V. Lifschitz. Weight Constraints as Nested Expressions. Unpublished draft; available at `http://www.cs.utexas.edu/users/vl/`, 2002.

[10] M. Gelfond, M. Balduccini, and J. Galloway. Diagnosing Physical Systems in A-Prolog. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNCS*, pages 213–225. Springer-Verlag, 2001.

[11] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming (ICLP-88)*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

[12] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–385, 1991.

[13] K. Gödel. Zum intuitionistischen Aussagenkalkül. *Anzeiger der Akademie der Wissenschaften in Wien*, pages 65–66, 1932.

[14] R. Hähnle. Short Conjunctive Normal Forms in Finitely Valued Logics. *Journal of Logic and Computation*, 4(6):905–927, 1994.

[15] K. Heljanko and I. Niemelä. Bounded LTL Model Checking with Stable Models. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNCS*, pages 200–212. Springer-Verlag, 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 J. Dix, L. F. del Cerro, and U. Furbach, editors, *Proceedings of the 6th European Workshop on Logics in Artificial Intelligence (JELIA-98)*, volume 1489 of *LNCS*, pages 216–232, 1998.

[18] T. Janhunen. On the Effect of Default Negation on the Expressiveness of Disjunctive Rules. In T. Eiter, W. Faber, and M. Truszczynski, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNCS*, pages 93–106. Springer-Verlag, 2001.

[19] T. Janhunen, I. Niemelä, P. Simons, and J.-H. You. Unfolding Partiality and Disjunctions in Stable Model Semantics. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning (KR-00)*, pages 411–424. Morgan Kaufmann, 2000.

[20] V. Kowalski. The Calculus of the Weak "Law of Excluded Middle". *Mathematics of the USSR*, 8:648–658, 1968.

[21] V. Lifschitz. Answer Set Planning. In D. De Schreye, editor, *Proceedings of the 16th International Conference on Logic Programming (ICLP-99)*, pages 23–37. The MIT Press, 1999.

[22] V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[23] V. Lifschitz, L. Tang, and H. Turner. Nested Expressions in Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

[24] F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In F. Giunchiglia, D. Fensel, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR-02)*, pages 170–176. Morgan Kaufmann, 2002.

[25] V. Marek and M. Truszczyński. *Nonmonotonic Logic: Context-Dependent Reasoning*. Artifical Intelligence. Springer-Verlag, 1993.

[26] 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-Verlag, 1994.

[27] I. Niemelä and P. Simons. Smodels: An Implementation of the Stable Model and Well-Founded Semantics for Normal Logic Programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-97)*, volume 1265 of *LNCS*, pages 420–429. Springer-Verlag, 1997.

[28] D. Pearce. A New Logical Characterisation of Stable Models and Answer Sets. In J. Dix, L. Pereira, and T. Przymusinksi, editors, *Non-Monotonic Extensions of Logic Programming*, volume 1216 of *LNCS*, pages 57–70. Springer-Verlag, 1997.

[29] D. Pearce. From Here to There: Stable Negation in Logic Programming. In D. Gabbay and H. Wansing, editors, *What is Negation?* Kluwer, 1999.

[30] D. Pearce, I. de Guzmán, and A. Valverde. A Tableau Calculus for Equilibrium Entailment. In R. Dyckhoff, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, Proceedings (TABLEAUX-00)*, volume 1847 of *LNCS*, pages 352–367. Springer-Verlag, 2000.

[31] D. Pearce, I. de Guzmán, and A. Valverde. Computing Equilibrium Models Using Signed Formulas. In J. W. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic 2000 (CL-00)*, volume 1861 of *LNCS*, pages 688–702. Springer-Verlag, 2000.

[32] D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In P. Brazdil and A. Jorge, editors, *Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving, 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] J. Siekmann and G. Wrightson, editors. *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag, 1983.

[35] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[36] G. Tseitin. On the Complexity of Proofs in Propositional Logics. *Seminars in Mathematics*, 8, 1970. Reprinted in [34].