On Acyclic and Head-Cycle Free Nested Logic Programs*

Thomas Linke[1], Hans Tompits[2], and Stefan Woltran[2]

[1] Institut für Informatik, Universität Potsdam,
Postfach 90 03 27, D–14439 Potsdam, Germany
linke@cs.uni-potsdam.de
[2] Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Vienna, Austria
[tompits,stefan]@kr.tuwien.ac.at

Abstract. We define the class of head-cycle free nested logic programs, and its proper subclass of acyclic nested programs, generalising similar classes originally defined for disjunctive logic programs. We then extend several results known for acyclic and head-cycle free disjunctive programs under the stable-model semantics to the nested case. Most notably, we provide a propositional semantics for the program classes under consideration. This generalises different extensions of Fages' theorem, including a recent result by Erdem and Lifschitz for tight logic programs. We further show that, based on a shifting method, head-cycle free nested programs can be rewritten into normal programs in polynomial time and space, extending a similar technique for head-cycle free disjunctive programs. All this shows that head-cycle free nested programs constitute a subclass of nested programs possessing a lower computational complexity than arbitrary nested programs, providing the polynomial hierarchy does not collapse.

1 Introduction

This paper deals with generalisations and refinements of several reducibility results for *nested logic programs* (NLPs) under the stable-model semantics. This class of programs is characterised by the condition that arbitrarily nested formulas, formed from atoms using negation as failure, conjunction, and disjunction, serve as bodies and heads of rules, extending the well-known classes of *normal logic programs* (nLPs), *disjunctive logic programs* (DLPs), and *generalised disjunctive logic programs* (GDLPs). Nested logic programs under the stable-model semantics (or rather under the answer-set semantics, by allowing also strong negation) were introduced by Lifschitz, Tang, and Turner [18], and currently receive increasing interest in the literature, both from a logical as well as from a computational point of view.

In complexity theory, a frontier is identified having DLPs, GDLPs and NLPs on the one side, and nLPs and so-called *nested normal programs* (NnLPs), for which only

* This work was partially supported by the Austrian Science Fund (FWF) under projects Z29-N04 and P15068-INF, by the German Science Foundation (DFG) under grants FOR 375/1 and SCHA 550/6, TP C, as well as by the European Commission under projects FET-2001-37004 WASP and IST-2001-33570 INFOMIX.

positive literals are allowed as heads of rules (cf. Table 1 below), on the other side. For the former program classes, the main reasoning tasks lie at the second level of the polynomial hierarchy [9, 26], while for the latter classes, the main reasoning tasks have NP complexity [24, 2].[1] There are various translatability results between the different syntactic subclasses of programs. Among them, there are translations between nested programs and GDLPs [18], and between DLPs and nLPs [8], both requiring exponential space in the worst case. Additionally, there exist linear-time constructible translations between NLPs and DLPs [25], and between GDLPs and DLPs [14, 15]. Note that, unless the polynomial hierarchy collapses, the above mentioned complexity gap does not allow for polynomial translations between, e.g., nested logic programs and normal logic programs. However, one can seek for subclasses of NLPs where such a translation is possible.

In this paper, we identify non-trivial subclasses of nested programs for which we establish two forms of reductions:

1. reductions to classical propositional logic; and
2. reductions to normal logic programs.

More specifically, we introduce the classes of *head-cycle free* (*HCF*) *nested programs* and its proper subclass of *acyclic nested programs*. Both program classes are defined as generalisations of similar kinds of programs originally introduced as syntactic subclasses of disjunctive logic programs a decade ago by Ben-Eliyahu and Dechter [1]. Moreover, the reductions we provide here are, on the one hand, *extensions* of previous results, established for more restricted kinds of programs, and, on the other hand, with respect to head-cycle free and acyclic nested programs, *optimisations* of general translatability results developed in [26, 25]. We detail the main aspects of our results in the following.

Concerning the reduction to classical propositional logic, we construct mappings, $\mathcal{T}[\cdot]$ and $\mathcal{T}^*[\cdot]$, assigning to each program a propositional theory such that

1. given an acyclic nested program Π, the stable models of Π are given by the models of the classical theory $\mathcal{T}[\Pi]$; and
2. given a head-cycle free nested program Π, the stable models of Π are given by sets of form $I \cap V$, where I is a model of the classical theory $\mathcal{T}^*[\Pi]$ and V is the set of atoms occurring in Π.

In both cases, the size of the assigned classical theory is *polynomial* in the size of the input program. Moreover, the translation $\mathcal{T}^*[\cdot]$ is defined using newly introduced auxiliary variables, whereas for $\mathcal{T}[\cdot]$, no new variables are required.

These results are generalisations of similar characterisations given by Ben-Eliyahu and Dechter [1] for acyclic and head-cycle free DLPs. Moreover, our results generalise results relating the stable-model semantics to *Clark's completion* [5]. Recall that Clark's completion was one of the first semantics proposed for programs containing default negation, in which a normal logic program Π is associated with a propositional theory, $\mathsf{COMP}[\Pi]$, called the *completion of* Π. Although every stable model of Π is also a

[1] The NP-completeness for NnLPs can be derived from a translation from NnLPs to nLPs due to You, Yuan, and Zhang [29].

model of the completion of Π, the converse does not always hold. In fact, Fages [12] showed that the converse holds providing Π satisfies certain syntactic restrictions. Our results generalise Fages' characterisation in the sense that, if Π is a normal program, then both $\mathcal{T}[\Pi]$ and $\mathcal{T}^*[\Pi]$ coincide with $\mathsf{COMP}[\Pi]$.

Fages' theorem was generalised in several directions: Erdem and Lifschitz [10, 11] extended it for NnLPs, providing the given programs are *tight*. We extend the notion of tightness to *arbitrary* nested programs and refine our results by showing that

> if a nested program Π is HCF and tight on an interpretation I, then I is a stable model of Π iff I is a model of $\mathcal{T}[\Pi]$.

Other generalisations of Fages' theorem drop the syntactic proviso but add instead additional so-called *loop formulas* guaranteeing equivalence between the stable models of the given program and the classical models of the resultant theory. This idea was pursued by Lin and Zhao [19] for normal programs and subsequently extended by Lee and Lifschitz [17] for disjunctive programs with nested formulas in rule bodies. In contrast to Clark's completion for normal programs, the size of the resultant theories in these approaches is in the worst case *exponential* in the size of the input programs. We further note that, for the sort of programs dealt with in [17], the notion of completion defined there coincides with our transformation $\mathcal{T}[\cdot]$.

The reductions to classical propositional logic allow us also to draw immediate complexity results for acyclic and HCF nested programs. As noted above, the main reasoning tasks associated with arbitrary nested programs lie at the second level of the polynomial hierarchy [26], whereas our current results imply that analogous tasks for acyclic and HCF nested programs have NP or $\mathrm{co\text{-}NP}$ complexity (depending on the specific reasoning task). Thus, providing the polynomial hierarchy does not collapse, acyclic and HCF programs are computationally simpler than arbitrary nested programs.

Let us now turn to our results concerning the reductions to normal logic programs. As was shown by Ben-Eliyahu and Dechter [1], HCF disjunctive programs can be transformed into equivalent normal programs by shifting head atoms into the body (cf. also [6, 13]). For instance, a rule of form $p \vee q \leftarrow r$ is replaced by this method by the two rules $p \leftarrow r, \neg q$ and $q \leftarrow r, \neg p$ (where "$\neg$" denotes the negation-as-failure operator). We generalise this method for HCF nested programs, obtaining a polynomial reduction from HCF nested programs (and thus, in particular, also from acyclic nested programs) into nLPs. Note that applying such a shifting technique for programs which are not HCF does in general not retain the stable models.

Previous to our work, Inoue and Sakama [14] already defined the notions of acyclicity and head-cycle freeness for *generalised disjunctive programs*, extending the respective notions introduced in [1]. They showed that GDLPs satisfying either of these extended notions can likewise be transformed to nLPs by shifting head atoms to the bodies of rules and thus have the same worst-case complexity as normal programs. However, their notions of acyclicity and head-cycle freeness are more restrictive than ours, with respect to GDLPs, and hence our results hold for a larger class of programs.

We finally note that, at first glance, one may attempt to construct a polynomial translation from acyclic or head-cycle free NLPs into normal programs or into classical propositional logic by first applying the polynomial translation from NLPs to DLPs due to Pearce *et al.* [25], and afterwards by transforming the resultant programs into

either normal programs or into classical logic using the results of Ben-Eliyahu and Dechter [1]. However, since the translation of [25] does neither preserve acyclicity nor head-cycle freeness, such an approach does not work in general.

The paper is organised as follows. The next section supplies some background on logic programs and the stable-model semantics. In Section 3, we introduce acyclic and head-cycle free nested programs and show some invariance theorems. Section 4 discusses the reductions to classical propositional logic, and Section 5 deals with the generalised shifting technique for reducing HCF programs into nLPs. Section 6 is devoted to tight nested programs, and Section 7 concludes the paper.

2 Preliminaries

We deal with propositional languages and use the logical symbols $\top$, $\bot$, $\neg$, $\vee$, $\wedge$, $\rightarrow$, and $\leftrightarrow$ to construct formulas over propositional variables (or atoms) in the usual way. A formula using only $\wedge$, $\vee$, or $\neg$ as its sentential connectives is called an *expression*. *Literals* are formulas of form v or $\neg v$, where v is some variable, or one of $\top, \bot$. We refer to a literal of form v (where v is as before) as a *positive literal* and to a literal of form $\neg v$ as a *negative literal*. Disjunctions of form $\bigvee_{i\in I}\phi_i$ are assumed to stand for the logical constant $\bot$ whenever $I = \emptyset$, and, likewise, conjunctions of form $\bigwedge_{i\in I}\phi_i$ with $I = \emptyset$ stand for $\top$. The set of all atoms occurring in a formula ϕ is denoted by $Atm(\phi)$.

By an interpretation, I, we understand a set of variables. Informally, a variable v is true under I iff $v \in I$. Interpretations induce truth values (in the sense of classical logic) of arbitrary formulas in the usual way. The set of models of a formula ϕ is denoted by $Mod(\phi)$. Two formulas, ϕ and ψ, are (*logically*) *equivalent*, iff $Mod(\phi) = Mod(\psi)$. For a set V and a family of sets S, by $S|_V$ we denote the family $\{I \cap V \mid I \in S\}$.

The fundamental objects of our investigation are *nested logic programs* (NLPs), introduced by Lifschitz, Tang, and Turner [18]. NLPs are characterised by the condition that the bodies and heads of rules are given by arbitrary expressions as defined above. For reasons of simplicity, we deal here only with languages containing one kind of negation, corresponding to default negation. Therefore, $\neg$ refers to default negation, whenever used in logic programs.

In more formal terms, a *rule*, r, is a 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. If $B(r) = \top$, then r is a *fact*, and if $H(r) = \bot$, then r is a *constraint*. A nested logic program, or simply a *program*, is a finite set of rules.

We associate to every program Π the propositional formula

$$\mathsf{c}(\Pi) = \bigwedge_{r\in\Pi} \big(B(r) \rightarrow H(r)\big).$$

Furthermore, $Atm(\Pi)$ denotes the set of all atoms occurring in program Π.

Let Π be a program and I an interpretation. Then, the *reduct, Π^I, of Π with respect to I* is obtained from Π by replacing every occurrence of an expression $\neg\psi$ in Π which

Table 1. Classes of programs.

Class	Heads	Bodies
NLP	expression	expression
GDLP	disjunction of literals	conjunction of literals
DLP	disjunction of atoms	conjunction of literals
NnLP	positive literal	expression
nLP	positive literal	conjunction of literals

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 $\mathsf{c}(\Pi^I)$. The collection of all answer sets of Π is denoted by $AS(\Pi)$. Two logic programs, Π_1 and Π_2, are *equivalent* iff $AS(\Pi_1) = AS(\Pi_2)$.

By restricting the syntactic form of bodies and heads of rules, different classes of programs are identified. Besides NLPs, for our purposes, the following classes are of interest: *generalised disjunctive logic programs* (GDLPs), *disjunctive logic programs* (DLPs), *nested normal logic programs* (NnLPs), and *normal logic programs* (nLPs). Table 1 summarises the defining attributes of these classes.

Following Lloyd and Topor [23] (cf. also [11]), we define the completion of an NnLP Π as the propositional formula

$$\mathsf{COMP}[\Pi] = \bigwedge_{p \in A} \Big(p \leftrightarrow \bigvee_{r \in \Pi, H(r) = p} B(r) \Big),$$

where $A = Atm(\Pi) \cup \{\bot\}$.

Finally, we recall some graph-theoretical notations. A (*directed*) *graph*, G, is a pair (V, E) such that V is a finite set of *nodes* and $E \subseteq V \times V$ is a set of *edges*. A *path from* v *to* v' in G is a sequence $P_{v,v'} = (v_1, \ldots, v_n)$ of nodes such that $v = v_1$, $v' = v_n$, and $(v_i, v_{i+1}) \in E$, for each $1 \leq i < n$. A graph $G = (V, E)$ is *acyclic* iff, for each node $v \in V$, there is no path from v to itself. A *strongly connected component* (or *component*, for short) of a graph G is a maximal set S of nodes such that, for any two nodes p and q in S, there is a path from p to q in G. Strongly connected components can be identified in linear time [28]. The *size* of a component is the length (i.e., number of edges) of the longest acyclic path in it.

3 Acyclic and Head-Cycle Free Nested Programs

We start our formal elaboration by recalling the notion of a dependency graph for nested logic programs. Based on this, we define acyclic and head-cycle free nested programs, and show that these notions are invariant with respect to rewritings into DLPs.

We commence with the following auxiliary notions.

Definition 1. *Let p be an atom and φ an expression. Then, the* polarity *of a specific occurrence of p in φ is* positive *iff it is not in the scope of a negation, and* negative

otherwise. The set of atoms having a positive occurrence in φ is denoted by $Atm^+(\varphi)$. For a program Π, we define $Atm^+(\Pi) = \bigcup_{r\in\Pi}(Atm^+(H(r)) \cup Atm^+(B(r)))$.

With this notation at hand, we are able to define the concept of dependency graphs for NLPs, following Lee and Lifschitz [17].

Definition 2. *The* (positive) dependency graph *of a program Π is given by $G_\Pi = (Atm(\Pi), E_\Pi)$, where $E_\Pi \subseteq Atm(\Pi) \times Atm(\Pi)$ is defined by the condition that $(p, q) \in E_\Pi$ iff there is some $r \in \Pi$ such that $p \in Atm^+(B(r))$ and $q \in Atm^+(H(r))$.*

Our first category of programs is then introduced as follows:

Definition 3. *A nested logic program Π is* acyclic *iff its dependency graph G_Π is acyclic.*

It is a straightforward matter to check that this definition generalises acyclic DLPs as introduced by Ben-Eliyahu and Dechter [1], i.e., a DLP Π is acyclic in the sense of Definition 3 iff it is acyclic in the sense of [1].

Example 1. Consider the following two programs:

$$\Pi_1 = \{p \vee q \leftarrow; p \leftarrow q; q \leftarrow p\}; \qquad \Pi_2 = \{p \vee q \leftarrow; p \leftarrow \neg\neg q; q \leftarrow \neg\neg p\}.$$

Programs Π_1 and Π_2 have dependency graphs $G_{\Pi_1} = (\{p, q\}, \{(p, q), (q, p)\})$ and $G_{\Pi_2} = (\{p, q\}, \emptyset)$, respectively. Thus, Π_1 is not acyclic, whereas Π_2 is. One may verify that both programs have the same stable models, viz. $AS(\Pi_1) = AS(\Pi_2) = \{\{p, q\}\}$.

Next, we generalise the notion of a head-cycle free DLP to the class of NLPs. To this end, we need the following definition.

Definition 4. *Two distinct atoms, p and q, are* joint-positive *in an expression ϕ iff there exists a subformula $\phi_1 \vee \phi_2$ of ϕ with $p \in Atm^+(\phi_1)$ and $q \in Atm^+(\phi_2)$, or vice versa. Moreover, p and q are called* head-sharing *in a program Π iff p and q are joint-positive in $H(r)$, for some $r \in \Pi$.*

From this, the class of head-cycle free NLPs is characterised in the following way:

Definition 5. *A nested program Π is* head-cycle free (*HCF*) *iff its dependency graph G_Π does not contain a directed cycle going through two head-sharing atoms in Π.*

Again, it can be shown that a DLP Π is HCF in the above sense iff it is HCF in the sense of [1]. Thus, the class of HCF NLPs is a proper generalisation of the class of HCF DLPs. Furthermore, it is easy to see that every acyclic NLP is HCF.

Example 2. Consider the programs Π_1 and Π_2 from Example 1. Observe that p and q are head-sharing in both Π_1 and Π_2. Hence, Π_1 is not HCF, since there is a cycle in G_{Π_1} involving p and q. On the other hand, Π_2 is HCF, and, as we already know from the above, acyclic.

Table 2. Replacements in logic programs. It is assumed that ϕ, ψ, and φ are expressions, A is a set of atoms, p is an atom, L and L_p are new atoms, and $\circ \in \{\wedge, \vee\}$.

Name	Occurrence α	Replaced by β
(L1)	$\phi \circ \psi$	$\psi \circ \phi$
(L2)	$(\phi \circ \psi) \circ \varphi$	$\phi \circ (\psi \circ \varphi)\ \circ \in \{\wedge, \vee\}$
(L3)	$\phi \circ \phi$	ϕ
(L4)	$(\phi \wedge \psi) \vee \varphi$	$(\phi \vee \varphi) \wedge (\psi \vee \varphi)$
(L5)	$(\phi \vee \psi) \wedge \varphi$	$(\phi \wedge \varphi) \vee (\psi \wedge \varphi)$
(L6)	$\neg(\phi \vee \psi)$	$(\neg\phi \wedge \neg\psi)$
(L7)	$\neg(\phi \wedge \psi)$	$(\neg\phi \vee \neg\psi)$
(L8)	$\neg\neg\neg\phi$	$\neg\phi$
(L9)	$\phi \leftarrow \psi \vee \varphi$	$\phi \leftarrow \psi$; $\phi \leftarrow \varphi$
(L10)	$\phi \wedge \psi \leftarrow \varphi$	$\phi \leftarrow \varphi$; $\psi \leftarrow \varphi$
(L11)	$\phi \leftarrow \psi \wedge \neg\neg\varphi$	$\phi \vee \neg\varphi \leftarrow \psi$
(L12)	$\neg\neg\phi \vee \psi \leftarrow \varphi$	$\psi \leftarrow \varphi \wedge \neg\phi$
(J)	$\neg p \vee \psi \leftarrow \phi$	$L_p \vee \psi \leftarrow \phi$; $\bot \leftarrow p \wedge L_p$; $L_p \leftarrow \neg p$
(S)[1]	$\phi \vee \psi \leftarrow \varphi$	$\phi \leftarrow \varphi \wedge \neg\psi$; $\psi \leftarrow \varphi \wedge \neg\phi$
(T*)[2]	$\phi \leftarrow \psi$	$\{\phi[(A \setminus \{p\})/\top] \leftarrow \psi[p/\bot];\ \phi[p/\top] \leftarrow \psi[(A \setminus \{p\})/\bot] \mid p \in A\}$
(D)	$\phi \vee \psi \leftarrow \varphi$	$\phi \vee L \leftarrow \varphi$; $\psi \leftarrow L$
(Y1)	$p \leftarrow \psi \wedge (\phi \vee \varphi)$	$p \leftarrow \psi \wedge L$; $L \leftarrow \phi \vee \varphi$
(Y2)	$p \leftarrow \psi \wedge \neg\neg q$;	$p \leftarrow \psi \wedge \neg L$; $L \leftarrow \neg q$
(C)	$\phi \wedge \psi \leftarrow \varphi$	$L \leftarrow \varphi$; $\phi \leftarrow L$; $\psi \leftarrow L$

[1] applicable only if $Atm^+(\phi) \cap Atm^+(\psi) = \emptyset$.
[2] applicable only for non-empty $A \subseteq Atm^+(\phi) \cap Atm^+(\psi)$.

In what follows, we review the translations introduced by Lifschitz, Tang, and Turner [18] and Janhunen [15], which, jointly applied, allow for translating any nested program into a DLP via the substitutions (L1)–(L12) and (J) from Table 2. Observe that the DLPs obtained in this way may be exponential in the size of the respective input programs. Our goal is to show that our notions of acyclicity and head-cycle freeness for NLPs are invariant with respect to the applications of these substitutions.

Any substitution σ from Table 2 is applied as follows: We say that a program Π' is *obtained from* Π *via* σ by replacing[2] an occurrence of an expression, or a single rule α, by β, which itself is an expression, a rule, or a set of rules. Moreover, $\theta[p/x]$ denotes the replacement of *all positive* occurrences of an atom p in θ by expression x. Accordingly, for a set of atoms S, $\theta[S/x]$ denotes the replacement of all positive occurrences of $p \in S$ in θ by x. Thus, $\theta[S/x] = \theta$ whenever $S \cap Atm^+(\theta)$ is empty. We sometimes use a substitution σ in the reverse way, i.e., replacing β by α. This is made explicit by writing $\sigma^{\leftarrow}$. We note that in this section we require only a part of the translations given in Table 2; the remaining ones are needed later on.

We start with the translation from NLPs to GDLPs due to Lifschitz, Tang, and Turner [18]. This translation is based on substitutions (L1)–(L12) from Table 2.

[2] For (L11), (L12), (J), (Y1), and (Y2), we allow the component ψ in α to be vacuous; in this case, for β, ψ is set to $\top$ in (L11) and to $\bot$ in (L12).

Proposition 1 ([18]). *Let Π be a nested program.*

Then, for every program Π' obtained from Π via any substitution from (*L1*)–(*L12*)*, it holds that $AS(\Pi) = AS(\Pi')$. Moreover, there is a program Π'' obtained from Π via a sequence of substitutions from* (*L1*)–(*L12*) *such that Π'' is a GDLP and $AS(\Pi) = AS(\Pi'')$.*

Next, we close the gap between GDLPs and DLPs. To this end, we require the substitution rule (J) of Table 2, which is a generalised stepwise variant of a labeling technique discussed in [15], introducing a globally new atom L_p, for any atom p. Observe that acyclicity and head-cycle freeness are invariant with respect to applications of substitutions (L1)–(L12) and (J). More precisely, (L1)–(L12) preserve both the dependency graph and the pairs of head-sharing atoms. An application of Substitution (J), however, changes the dependency graph $G_\Pi = (Atm(\Pi), E_\Pi)$, for a given program Π, to $G'_\Pi = (Atm(\Pi) \cup \{L_p\}, E_\Pi \cup \{(q, L_p) \mid q \in Atm^+(\phi)\})$ and yields additional pairs q and L_p of head-sharing atoms, for any $q \in Atm^+(\psi)$ (cf. Table 2), but no additional cycles are introduced in G'_Π. This gives us the desired result.

Theorem 1. *Let Π be a nested program, and let Π' be obtained from Π by applying any sequence of substitutions from* (*L1*)–(*L12*) *and* (*J*)*.*

Then, the following properties hold:

1. *$AS(\Pi) = AS(\Pi')|_{Atm(\Pi)}$;*
2. *Π is acyclic iff Π' is acyclic; and*
3. *Π is HCF iff Π' is HCF.*

This theorem states that the properties of being acyclic and of being HCF are invariant with respect to any sequence of substitutions from (L1)–(L12) and (J). The next theorem demonstrates that substitutions (L1)–(L12) and (J) are suffi cient to transform a given NLP into a corresponding DLP.

Theorem 2. *For any nested program Π, there is a program Π' obtained from Π via a sequence of substitutions from* (*L1*)–(*L12*) *and* (*J*) *obeying the conditions from Theorem 1 and such that Π' is a DLP.*

Example 3. For Π_2 from Example 1, we derive the following DLP by applying substitutions (L11) and (J) to both of the last two rules of Π_2:

$$\Pi' = \{p \vee q \leftarrow;\ p \vee L_q \leftarrow;\ q \vee L_p \leftarrow\} \cup \{\bot \leftarrow v \wedge L_v;\ v \leftarrow \neg L_v \mid v \in \{p, q\}\}.$$

The dependency graph of this program is $(\{p, q, L_p, L_q\}, \emptyset)$, and thus it is still HCF and acyclic. As well, the only stable model of Π' is $\{p, q\}$.

4 Reductions to Classical Propositional Logic

We now proceed with assigning a propositional semantics to acyclic and head-cycle free nested programs, in the sense that a program Π is transformed into a propositional formula ϕ such that the answer sets of Π are given by the models of ϕ. Observing that

these encodings yield propositional formulas whose sizes are *polynomial* in the sizes of the input programs, we also draw some immediate complexity results.

We have the following building blocks. Let Π be a nested program. For any $p \in Atm^+(\Pi)$ occurring in a strongly connected component of size $l > 1$ in G_Π, we introduce globally new variables $p_1, \ldots, p_k$, where $k = \lceil \log_2(l-1) \rceil$. For two atoms p, q occurring in the same component of size $l > 1$ of the dependency graph, we define

$$prec_\Pi[q,p] = \bigwedge_{i=1}^{k}(q_i \rightarrow \bigvee_{j=i}^{k} p_j) \wedge \neg \bigwedge_{i=1}^{k}(p_i \rightarrow q_i).$$

Informally, $prec_\Pi[\cdot,\cdot]$ assigns a strict partial order to the atoms in Π, based on a binary encoding technique.

Now we are ready to define our two main transformations, $\mathcal{T}[\cdot]$ and $\mathcal{T}^*[\cdot]$, from nested logic programs into formulas of propositional logic.

Definition 6. *Let Π be a nested program, and let Π_p be the program resulting from Π by taking those rules $r \in \Pi$ where $p \in Atm^+(H(r))$ and replacing each positive occurrence of p in a head by $\bot$. Furthermore, let Π_p^* be the program resulting from Π_p by replacing each positive occurrence of an atom $q \neq p$ in a body by the formula $q \wedge prec_\Pi[q,p]$, providing q is in the same component as p in G_Π.*

Then, define

$$\mathcal{T}[\Pi] = \mathsf{c}(\Pi) \wedge \bigwedge_{p \in Atm(\Pi)} \big(p \rightarrow \neg \mathsf{c}(\Pi_p)\big) \quad \textit{and}$$

$$\mathcal{T}^*[\Pi] = \mathsf{c}(\Pi) \wedge \bigwedge_{p \in Atm(\Pi)} \big(p \rightarrow \neg \mathsf{c}(\Pi_p^*)\big).$$

Note that both the size of $\mathcal{T}[\Pi]$ as well as the size of $\mathcal{T}^*[\Pi]$ is polynomial in the size of Π. Furthermore, if Π is an NnLP, then $\mathcal{T}[\Pi]$ is equivalent to the completion $\mathsf{COMP}[\Pi]$, and if Π is an acyclic NLP, then $\mathcal{T}^*[\Pi] = \mathcal{T}[\Pi]$. Moreover, it can be shown that, for any DLP Π, the theories $\mathcal{T}[\Pi]$ and $\mathcal{T}^*[\Pi]$ are equivalent to the encodings given by Ben-Eliyahu and Dechter [1] for acyclic and HCF DLPs, respectively. The main characterisations of [1] can thus be paraphrased as follows:

Proposition 2 ([1]). *For any DLP Π,*

1. *if Π is acyclic, then $I \in AS(\Pi)$ iff $I \in Mod(\mathcal{T}[\Pi])$, for all $I \subseteq Atm(\Pi)$; and*
2. *if Π is HCF, then $AS(\Pi) = Mod(\mathcal{T}^*[\Pi])|_{Atm(\Pi)}$.*

The restriction in the second result is used to "hide" the newly introduced variables in formulas $prec_\Pi[\cdot,\cdot]$ in $\mathcal{T}^*[\Pi]$.

With the next results, we generalise Proposition 2 to HCF and acyclic nested programs. To begin with, we have the following theorem.

Theorem 3. *Let Π be a HCF nested logic program.*
Then, $AS(\Pi) = Mod(\mathcal{T}^*[\Pi])|_{Atm(\Pi)}$.

This theorem is proved by showing that models of $\mathcal{T}^*[\cdot]$ are invariant (modulo the introduction of new atoms) under substitution rules (L1)–(L12) and (J).

As an immediate consequence, we obtain the following corollary for acyclic nested programs.

Corollary 1. *Let Π be an acyclic nested logic program, and let $I \subseteq Atm(\Pi)$. Then, $I \in AS(\Pi)$ iff $I \in Mod(\mathcal{T}[\Pi])$.*

Let us briefly mention that our encodings easily extend to typical reasoning tasks associated to logic programs. Following [1], we define the following inference operators. Let Π be a logic program and S a finite set of atoms.

1. *Brave consequence*: $\Pi \vdash_b S$ iff S is contained in some answer set of Π.
2. *Skeptical consequence*: $\Pi \vdash_s S$ iff S is contained in all answer sets of Π.
3. *Disjunctive entailment*: $\Pi \vdash_d S$ iff, for each answer set I of Π, there is some $p \in S$ such that $p \in I$.

We then obtain the following straightforward encodings:

Theorem 4. *Let S be a finite set of atoms.*

1. *For any acyclic NLP Π, we have that*
   (*a*) *$\Pi \vdash_b S$ iff $\mathcal{T}[\Pi] \wedge \bigwedge_{p \in S} p$ is satisfiable;*
   (*b*) *$\Pi \vdash_s S$ iff $\mathcal{T}[\Pi] \rightarrow \bigwedge_{p \in S} p$ is valid; and*
   (*c*) *$\Pi \vdash_d S$ iff $\mathcal{T}[\Pi] \rightarrow \bigvee_{p \in S} p$ is valid.*
2. *For any HCF NLP Π, we have that*
   (*a*) *$\Pi \vdash_b S$ iff $\mathcal{T}^*[\Pi] \wedge \bigwedge_{p \in S} p$ is satisfiable;*
   (*b*) *$\Pi \vdash_s S$ iff $\mathcal{T}^*[\Pi] \rightarrow \bigwedge_{p \in S} p$ is valid; and*
   (*c*) *$\Pi \vdash_d S$ iff $\mathcal{T}^*[\Pi] \rightarrow \bigvee_{p \in S} p$ is valid.*

Observing that the above encodings are clearly constructible in polynomial time, we derive the following immediate complexity results:

Theorem 5. *Checking whether $\Pi \vdash_b S$ holds, for a given acyclic or HCF NLP Π and a given finite set S of atoms, is* NP*-complete. Furthermore, checking whether $\Pi \vdash_s S$ or whether $\Pi \vdash_d S$ holds, given Π and S as before, is* co-NP*-complete.*

Note that the upper complexity bounds follow from the complexity of classical propositional logic, and the lower complexity bounds are inherited from the complexity of normal logic programs.

5 A Generalised Shifting Approach

The result that HCF nested programs have NP or co-NP complexity motivates to seek a polynomial translation from HCF programs to NnLPs and furthermore to nLPs. We do this by introducing a generalised variant of the well-known shifting technique [1, 6]. Recall that shifting for DLPs is defined as follows: Let $r \in \Pi$ be a disjunctive rule in a

HCF DLP Π. Then, following [1], Π is equivalent to the program resulting from Π by replacing r by the following set of rules:[3]

$$\{p \leftarrow B(r) \wedge \neg(Atm(H(r)) \setminus \{p\}) \mid p \in Atm(H(r))\}. \quad (1)$$

For generalising this shifting technique to nested programs, we introduce the substitution rule (S), depicted in Table 2, which allows the replacement of $\phi \vee \psi \leftarrow \varphi$ by the two rules $\phi \leftarrow \varphi \wedge \neg\psi$ and $\psi \leftarrow \varphi \wedge \neg\phi$, where ϕ and ψ are arbitrary expressions, providing $Atm^+(\phi) \cap Atm^+(\psi) = \emptyset$. Observe that (S) preserves head-cycle freeness.

In view of its proviso, (S) is not always applicable, even if a given program is HCF. But this problem is already apparent in the case of disjunctive programs. Indeed, in (1), we used the set $Atm(H(r))$ rather than the disjunction $H(r)$ explicitly, otherwise we would have run into problems: For instance, the DLP $\{p \vee p \leftarrow\}$ is clearly not equivalent to $\{p \leftarrow \neg p\}$.

We can establish the following property:

Lemma 1. *Let Π be a HCF nested program, and let Π' be obtained from Π via* (S)*. Then, $AS(\Pi) = AS(\Pi')$.*

This lemma follows from the property that models of $\mathcal{T}^*[\cdot]$ are preserved under application of (S), together with Theorem 3.

Theorem 6. *Let Π be a nested program.*

Then, there is a program $\mathcal{S}_{exp}[\Pi]$ obtained from Π via a finite sequence of substitutions from (*L1*)–(*L4*)*,* (*L6*)–(*L8*)*,* (*L10*)*,* (*L11*)$^{\leftarrow}$*,* (*L12*)*, and* (*S*)*, such that* (*i*) *$\mathcal{S}_{exp}[\Pi]$ is an NnLP, and* (*ii*) *if Π is HCF, then $AS(\Pi) = AS(\mathcal{S}_{exp}[\Pi])$.*

The "strategy" to obtain $\mathcal{S}_{exp}[\Pi]$ from Π is as follows: First, we translate Π into a program where all heads are disjunctions of atoms. Then, via (L1), (L2), and (L3), we can easily eliminate repeated occurrences of an atom p in a head. Finally, we then apply (S) to replace each (proper) disjunctive rule into a set of nested normal rules.

Observe that the subscript "exp" in $\mathcal{S}_{exp}[\cdot]$ indicates that the size of $\mathcal{S}_{exp}[\Pi]$ may be exponential in the size of Π in the worst case. The reason is the use of substitution rule (L4). We can circumvent the application of (L4), and thus the exponential blow-up, if we could use (S) more directly. To this end, we introduce the two substitution rules (D) and (T*), as given in Table 2. Observe that (T*) is a generalisation of an optimisation rule called (TAUT) due to Brass and Dix [3]. In fact, we want to apply (D) instead of (S), but (D) may introduce new head cycles according to its definition. In particular, this situation occurs whenever an atom occurs positively in both the body and the head of the considered rule. Hence, the strategy is then as follows: If (S) is not applicable, we first use (T*) to eliminate all atoms which occur positively in both the body and the head of the considered rule. After applying (D), we are clearly allowed to apply (S) to the resulting rules of form $\phi \vee L \leftarrow \varphi$, since L is a new atom not occurring in ϕ. In order to apply (S) after (D) and (T*), it is required that acyclicity and head-cycle freeness are invariant under application of (D) and (T*), which is indeed the case. Given that both substitutions can be shown to be answer-set preserving for HCF programs as well, we obtain the following theorem.

[3] For a finite set S of atoms, $\neg S$ denotes $\bigwedge_{s \in S} \neg s$.

Theorem 7. *Let Π be a nested program.*

Then, there is a program $\mathcal{S}_{poly}[\Pi]$ obtained from Π via a polynomial sequence of substitutions from (*L1*)–(*L3*), (*L6*)–(*L8*), (*L10*), (*L11*)$^{\leftarrow}$, (*L12*), (*S*), (*T**), *and* (*D*) *such that* (*i*) *$\mathcal{S}_{poly}[\Pi]$ is an NnLP, and* (*ii*) *if Π is HCF, then $AS(\Pi) = AS(\mathcal{S}_{poly}[\Pi])|_{Atm(\Pi)}$.*

Note that $\mathcal{S}_{poly}[\Pi]$ is polynomial in the size of Π, since the distributivity rule (L4) is not included. Indeed, new atoms are only introduced by (D).

So far, we showed how to translate HCF nested programs into NnLPs in polynomial time. In order to obtain a reduction to nLPs, we consider two additional rules, (Y1) and (Y2), depicted in Table 2. The following result holds:

Proposition 3 ([29]). *Let Π be an NnLP, and let Π' be obtained from Π via* (*Y1*) *or* (*Y2*).

Then, $AS(\Pi) = AS(\Pi')|_{Atm(\Pi)}$.

Putting the previous results together, the following property can be shown:

Theorem 8. *Let Π be a nested program.*

Then, there is a program $\mathcal{S}[\Pi]$ obtained from Π via a polynomial sequence of substitutions from (*L1*)–(*L3*), (*L6*)–(*L9*), (*L10*), (*L11*)$^{\leftarrow}$, (*L12*), (*S*), (*T**), (*D*), (*Y1*), *and* (*Y2*), *such that* (*i*) *$\mathcal{S}[\Pi]$ is normal, and* (*ii*) *if Π is HCF, then $AS(\Pi) = AS(\mathcal{S}[\Pi])|_{Atm(\Pi)}$.*

Example 4. Observe that program Π_2 from Example 1 can be translated into the nLP

$$\mathcal{S}[\Pi] = \{p \leftarrow \neg q;\ q \leftarrow \neg p;\ p \leftarrow \neg L_1;\ L_1 \leftarrow \neg q; q \leftarrow \neg L_2;\ L_2 \leftarrow \neg p\}.$$

6 Tight Nested Logic Programs

It is well known that every stable model of an NnLP Π is a model of $\mathsf{COMP}[\Pi]$ (cf., e.g., [11]). However, the converse holds only providing certain syntactic restrictions are enforced. Such conditions were first given by Fages [12] for nLPs, and subsequently extended by Erdem and Lifschitz [11] for NnLPs. In the latter work, the notion of *tight nested normal logic programs* is introduced. In this section, we extend tightness to general nested logic programs and show that HCF NLPs which satisfy tightness can be reduced to theories of classical propositional logic by means of translation $\mathcal{T}[\cdot]$. That is, the resultant theories are equivalent to $\mathsf{COMP}[\Pi]$ in case of an NnLP Π.

Following [11], we define the *positive conjunctive components* of an expression ϕ, denoted $cc(\phi)$, as follows: First, every expression ϕ can be written in the form $\phi_1 \wedge \cdots \wedge \phi_n$ ($n \geq 1$), where each ϕ_i is not a conjunction. The formulas $\phi_1, \ldots, \phi_n$ are called the *conjunctive components* of ϕ. Then, $cc(\phi)$ is the conjunction of all those conjunctive components of ϕ such that at least one atom occurs positively in it. Note that, e.g., $cc(\neg p) = \top$, where p is some atom.

Definition 7. *A nested program Π is* tight on an interpretation *I iff there exists a function λ from $Atm(\Pi)$ to ordinals such that, for each rule $r \in \Pi$, if $I \in Mod(H(r) \wedge B(r))$, then $\lambda(p) < \lambda(q)$, for each $p \in Atm(cc(B(r)))$ and each $q \in Atm^+(H(r))$.*

Obviously, this definition generalises the one in [11]. Using our translation $\mathcal{T}[\cdot]$, we can reformulate the main theorem in [11] as follows:

Proposition 4 ([11]). *Let Π be an NnLP, and let $I \subseteq Atm(\Pi)$ be an interpretation such that Π is tight on I.*
Then, $I \in AS(\Pi)$ iff $I \in Mod(\mathcal{T}[\Pi])$.

We generalise this proposition by showing that $\mathcal{T}[\Pi]$ is applicable to tight HCF nested programs as well. To this end, we make partly use of the results discussed in the previous section showing how nested programs can be reduced to NnLPs. Note that, whenever such a translation simultaneously retains tightness and models of $\mathcal{T}[\cdot]$, we directly get the desired generalisation, according to Proposition 4.

Lemma 2. *Let Π be a nested program, let I be an interpretation, and let Π' be obtained from Π via any substitution from* (*L1*)–(*L8*), (*L12*), (*L11*)$^{\leftarrow}$, *or* (*S*).
Then, Π' is tight on I whenever Π is tight on I.

Lemma 3. *Let Π be a nested program, and let Π' be obtained from Π via any substitution from* (*L1*)–(*L12*), (*L11*)$^{\leftarrow}$, *or* (*S*).
Then, $Mod(\mathcal{T}[\Pi]) = Mod(\mathcal{T}[\Pi'])$.

Observe that not all substitution rules from Table 2 used in Theorem 6 to obtain NnLPs are included in Lemma 2. In fact, there is some problem with (L10). Consider the program $\Pi = \{a \leftarrow b; b \wedge c \leftarrow a\}$, which is tight on interpretation $I = \{a, b\}$, since only for the first rule $r = a \leftarrow b$ the condition $I \in Mod(H(r) \wedge B(r))$ from Definition 7 holds. Applying (L10), we obtain $\Pi' = \{a \leftarrow b; b \leftarrow a; c \leftarrow a\}$ which is not tight on $\{a, b\}$ anymore, because now, both $I \in Mod(H(r) \wedge B(r))$ and $I \in Mod(H(r') \wedge B(r'))$ holds, for $r = a \leftarrow b$ and $r' = b \leftarrow a$. We therefore replace (L10) by the new rule (C) from Table 2, which can be shown to retain tightness, models of $\mathcal{T}[\cdot]$ (modulo newly introduced atoms), and head-cycle freeness.

By these invariance results, we get the main result of this section.

Theorem 9. *Let Π be a HCF nested program, and let $I \subseteq Atm(\Pi)$ be an interpretation such that Π is tight on I.*
Then, $I \in AS(\Pi)$ iff $I \in Mod(\mathcal{T}[\Pi])$.

7 Conclusion

In this paper, we introduced the classes of acyclic and head-cycle free nested programs as generalisations of similar classes originally introduced for disjunctive logic programs. We furthermore extended several results related to Clark's completion to our classes of programs, by introducing the polynomial reductions $\mathcal{T}[\cdot]$ and $\mathcal{T}^*[\cdot]$ to classical propositional logic. Moreover, we extended the notion of tightness to nested programs, and we constructed a polynomial translation of HCF nested programs into normal programs by applying a generalised shifting technique. We also derived immediate complexity results, showing that acyclic and HCF nested programs have a lower complexity than arbitrary NLPs, providing the polynomial hierarchy does not collapse.

Transformations $\mathcal{T}[\cdot]$ and $\mathcal{T}^*[\cdot]$ can also be viewed as optimisations of a translation studied in [26], in which (arbitrary) nested programs are efficiently mapped to *quantified Boolean formulas* such that the stable models of the former are given by the models of the latter. Hence, the present results show that, in case of acyclic and HCF programs, a reduction to classical formulas suffices instead of a reduction to the more expressive class of quantified Boolean formulas.

The translation of acyclic and HCF nested programs to nLPs optimises a polynomial translation presented in [25] from arbitrary nested programs into disjunctive logic programs, in the sense that our current method (i) introduces in general fewer additional variables, and (ii) translates a subclass of NLPs into a (presumably) less complex subclass of DLPs, viz. normal programs.

Furthermore, our translation extends and optimises also a recent result due to Eiter *et al.* [8] which discusses a general method to eliminate disjunctions from a given DLP under different notions of equivalence. To wit, under ordinary equivalence (i.e., preservance of stable models), the aforementioned method allows to transform a given DLP into an nLP by applying the usual shifting technique [1] and by adding suitable rules in order to retain equivalence between the programs. However, in general, the size of the resultant programs is exponential in the size of the input programs. Hence, for HCF programs, we obtain not only a generalisation of this general result to the nested case but also a *polynomial* method to achieve a transformation to nLPs.

Following the remarks in [29], our polynomial transformations from HCF nested programs into normal programs can be used to utilise extant answers-set solvers, like DLV [7], Smodels [27], or ASSAT [19], for computing answer sets of HCF nested programs. Furthermore, the present results indicate how to compute answer sets of HCF NLPs directly by generalising graph based methods as described in [4, 20, 16]. More precisely, we may define $Atm^{-}(\varphi)$ as the set of atoms having negative occurrences in φ, which enables us to express positive as well as negative dependencies between atoms in expressions. Therefore, graph coloring techniques as described in [16, 21], and used as basis of the noMoRe system [22], may be generalised to HCF NLPs. Hence, our approach offers different ways for answer-set computation of nested programs.

Although our current results are established for programs containing only one kind of negation, viz. default negation, they can be extended to programs allowing strong negation as well. Furthermore, another issue is the lifting of the notions of acyclic and head-cycle free nested programs to the first-order case, which can be done along the lines of [14].

References

1. R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.
2. N. Bidoit and C. Froidevaux. Negation by Default and Unstratifiable Logic Programs. *Theoretical Computer Science*, 78:85–112, 1991.
3. S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999.
4. G. Brignoli, S. Costantini, O. D'Antona, and A. Provetti. Characterizing and Computing Stable Models of Logic Programs: The Non-stratified Case. In *Proc. of the 2nd International Conference on Information Technology* (*CIT-99*), pp. 197–201, 1999.

5. K. L. Clark. Negation as Failure. In *Logic and Databases*, pp. 293–322. Plenum, 1978.
6. J. Dix, G. Gottlob, and V. Marek. Reducing Disjunctive to Non-Disjunctive Semantics by Shift-Operations. *Fundamenta Informaticae*, XXVIII(1/2):87–100, 1996.
7. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. In *Logic-Based Artifi cial Intelligence*, pp. 79–103. Kluwer, 2000.
8. T. Eiter, M. Fink, H. Tompits, and S. Woltran. On Eliminating Disjunctions in Stable Logic Programming. In *Proc. KR-04*, 2004. To appear.
9. T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artifi cial Intelligence*, 15(3–4):289–323, 1995.
10. E. Erdem and V. Lifschitz. Fages' Theorem for Programs with Nested Expressions. In *Proc. ICLP-01*, LNCS 2237, pp. 242–254, Springer, 2001.
11. E. Erdem and V. Lifschitz. Tight Logic Programs. *Theory and Practice of Logic Programming*, 3(4–5):499–518, 2003.
12. F. Fages. Consistency of Clark's Completion and Existence of Stable Models. *Methods of Logic in Computer Science*, 1:51–60, 1994.
13. M. Gelfond, V. Lifschitz, H. Przymusinska, and M. Truszczyński. Disjunctive Defaults. In *Proc. KR-91*, pp. 230–237. Morgan Kaufmann, 1991.
14. K. Inoue and C. Sakama. Negation as Failure in the Head. *Journal of Logic Programming*, 35(1):39–78, 1998.
15. T. Janhunen. On the Effect of Default Negation on the Expressiveness of Disjunctive Rules. In *Proc. LPNMR-01*, LNCS 2173, pp. 93–106, Springer, 2001.
16. K. Konczak, T. Linke, and T. Schaub. Graphs and Colorings for Answer Set Programming: Abridged Report. In *Proc. LPNMR-04*, LNCS 2923, pp. 127–140. Springer, 2004.
17. J. Lee and V. Lifschitz. Loop Formulas for Disjunctive Logic Programs. In *Proc. ICLP-03*, LNCS 2916, pp. 451–465, Springer, 2003.
18. V. Lifschitz, L. Tang, and H. Turner. Nested Expressions in Logic Programs. *Annals of Mathematics and Artifi cial Intelligence*, 25(3-4):369–389, 1999.
19. F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proc. AAAI-02*, pp. 112–117, 2002.
20. T. Linke. Graph Theoretical Characterization and Computation of Answer Sets. In *Proc. IJCAI-01*, pp. 641–645. Morgan Kaufmann, 2001.
21. T. Linke. Suitable Graphs for Answer Set Programming. In *Proc. ASP-03*, pp. 15–28. CEUR Workshop Proceedings, volume 78, 2003.
22. T. Linke, C. Anger, and K. Konczak. More on `noMoRe`. In *Proc. JELIA-02*, LNCS 2424, pp. 468–480, Springer, 2002.
23. J. Lloyd and R. Topor. Making Prolog More Expressive. *Journal of Logic Programming*, 3:225–240, 1984.
24. W. Marek and M. Truszczyński. Autoepistemic Logic. *Journal of the ACM*, 38:588–619, 1991.
25. D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A Polynomial Translation of Logic Programs with Nested Expressions into Disjunctive Logic Programs: Preliminary Report. In *Proc. ICLP-02*, LNCS 2401, pp. 405–420. Springer, 2002.
26. D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proc. EPIA-01*, LNCS 2285, pp. 306–320. Springer, 2001.
27. P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artifi cial Intelligence*, 138:181–234, 2002.
28. R. Tarjan. Depth-first Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.
29. J. You, L. Yuan, and M. Zhang. On the Equivalence Between Answer Sets and Models of Completion for Nested Logic Programs. In *Proc. IJCAI-03*, pp. 859–865, 2003.