TLP **13** (4–5): Online Supplement, July 2013.
URL: http://dx.doi.org/10.1017/S147106841300046X

Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments

SERGIO GRECO, CRISTIAN MOLINARO
and IRINA TRUBITSYNA
DIMES, Università della Calabria
E-mail: {*greco,cmolinaro,trubitsyna*}*@dimes.unical.it*

submitted 10 April 2013; revised 23 June 2013; accepted 5 July 2013

Abstract

Recent years have witnessed an increasing interest in enhancing answer set solvers by allowing function symbols. Since the introduction of function symbols makes common inference tasks undecidable, research has focused on identifying classes of programs allowing only a restricted use of function symbols while ensuring decidability of common inference tasks. *Finitely-ground programs*, introduced in Calimeri *et al.* (2008), are guaranteed to admit a finite number of stable models with each of them of finite size. Stable models of such programs can be computed and thus common inference tasks become decidable. Unfortunately, checking whether a program is finitely-ground is semi-decidable. This has led to several decidable criteria, called *termination criteria*, providing sufficient conditions for a program to be finitely-ground. This paper presents a new technique that, used in conjunction with current termination criteria, allows us to detect more programs as finitely-ground. Specifically, the proposed technique takes a logic program $\mathscr{P}$ and transforms it into an adorned program $\mathscr{P}^\mu$ with the aim of applying termination criteria to $\mathscr{P}^\mu$ rather than $\mathscr{P}$. The transformation is sound in that if the adorned program satisfies a certain termination criterion, then the original program is finitely-ground. Importantly, applying termination criteria to adorned programs rather than the original ones strictly enlarges the class of programs recognized as finitely-ground.

KEYWORDS: logic programming with function symbols, bottom-up evaluation, program evaluation termination, stable models

1 Introduction

Recent developments of answer set solvers have seen significant progresses towards providing support for function symbols. As common inference tasks become undecidable in the presence of function symbols, research has focused on identifying classes of programs, allowing a restricted use of function symbols, for which stable models can be computed.

Finitely-ground programs, defined in Calimeri *et al.* (2008), are guaranteed to admit a finite number of stable models, each of finite size. Stable models of such programs can be computed and thus common inference tasks become decidable.

As the problem of deciding whether a program is finitely-ground is semi-decidable, decidable subclasses have been proposed (we discuss them in the following section). However, they are not able to identify as terminating even simple programs whose bottom-up evaluation always terminates. Below is an example.

Example 1

Consider the following program $\mathscr{P}_1$

$$\begin{array}{l} \mathtt{p(X,X)} \leftarrow \mathtt{base(X)}. \\ \mathtt{q(X,Y)} \leftarrow \mathtt{p(X,Y)}. \\ \mathtt{p(f(X),g(X))} \leftarrow \mathtt{q(X,X)}. \end{array}$$

where base is a base predicate symbol. The bottom-up evaluation of $\mathscr{P}_1$ terminates whatever set of facts for base is added to the program. Nevertheless, none of the termination criteria introduced so far is able to recognize this program as terminating.

For instance, the *argument-restricted* criterion (Lierler and Lifschitz 2009) consists of checking if there exists a function ϕ that assigns a natural number to each of p[1], p[2], q[1], q[2], so that the following two conditions are both satisfied.

1. As in the second rule X and Y are propagated from p[1] to q[1] and from p[2] to q[2], respectively, then ϕ must be s.t. $\phi(\mathtt{q[1]}) \geqslant \phi(\mathtt{p[1]})$ and $\phi(\mathtt{q[2]}) \geqslant \phi(\mathtt{p[2]})$.
2. Because in the third rule X is propagated from q[1] and q[2] to p[1] and p[2] by adding a function symbol, then ϕ must be s.t. *(i)* $\phi(\mathtt{p[1]}) > \phi(\mathtt{q[1]})$ or $\phi(\mathtt{p[1]}) > \phi(\mathtt{q[2]})$, and *(ii)* $\phi(\mathtt{p[2]}) > \phi(\mathtt{q[1]})$ or $\phi(\mathtt{p[2]}) > \phi(\mathtt{q[2]})$.

Clearly, the conditions above cannot be satisfied and thus $\mathscr{P}_1$ is not argument-restricted.

The Γ-*acyclicity* criterion (Greco *et al.* 2012b) builds a labelled directed graph which keeps track of how values are propagated from rule bodies to rule heads. In this case, the vertices of the graph are p[1], p[2], q[1], q[2]. The set of edges contains, among others, the edge (p[1], q[1]) labeled with ϵ because X is propagated from p[1] to q[1] in the second rule, and the edge (q[1], p[1]) labeled with f because X is propagated from q[1] to p[1] with the addition of function symbol f in the third rule. Because of this cycle expressing a possible non-terminating generation of terms, $\mathscr{P}_1$ is not Γ-acyclic.

This paper presents a new technique that, used in conjunction with current termination criteria, allows us to detect more programs as finitely-ground. The proposed technique takes a logic program $\mathscr{P}$ and transforms it into an adorned program $\mathscr{P}^\mu$ with the aim of applying termination criteria to $\mathscr{P}^\mu$ rather than $\mathscr{P}$. The transformation is sound in that if $\mathscr{P}^\mu$ satisfies a certain termination criterion, then $\mathscr{P}$ is finitely-ground.

Example 2

Consider again program $\mathscr{P}_1$ of Example 1. The technique proposed in this paper transforms $\mathscr{P}_1$ into the following adorned program

$$\begin{array}{l}
\mathtt{p}^{\epsilon\epsilon}(\mathtt{X},\mathtt{X}) \leftarrow \mathtt{base}^{\epsilon}(\mathtt{X}).\\
\mathtt{q}^{\epsilon\epsilon}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{\epsilon\epsilon}(\mathtt{X},\mathtt{Y}).\\
\mathtt{p}^{\mathtt{f_1 g_1}}(\mathtt{f}(\mathtt{X}),\mathtt{g}(\mathtt{X})) \leftarrow \mathtt{q}^{\epsilon\epsilon}(\mathtt{X},\mathtt{X}).\\
\mathtt{q}^{\mathtt{f_1 g_1}}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{\mathtt{f_1 g_1}}(\mathtt{X},\mathtt{Y}).
\end{array}$$

The adorned program above is "equivalent" to $\mathscr{P}_1$ in that the minimal model of $\mathscr{P}_1$ can be obtained from the minimal model of the transformed program by dropping adornments. Each adorned rule is obtained from a rule in the original program by adding adornments which keep track of the structure of the terms that can be propagated during the bottom-up evaluation. As adorning predicate symbols possibly breaks "cyclic" dependencies among arguments and/or rules, this often allows us to recognize more programs as finitely-ground than if termination criteria are applied to the original programs. For instance, as opposed to the original program $\mathscr{P}_1$, the transformed program above is not recursive and thus is easily recognized as terminating by all current termination criteria. This allows us to say that $\mathscr{P}_1$ is terminating because of the aforementioned equivalence.

Example 3

The following program $\mathscr{P}_3$ checks if a given list can be partitioned into two identical sublists:

$$\begin{array}{l}
r_0 : \mathtt{part}([2,2,7,7],[\,],[\,]).\\
r_1 : \mathtt{part}(\mathtt{L_1},[\mathtt{X}|\mathtt{L_2}],\mathtt{L_3}) \leftarrow \mathtt{part}([\mathtt{X}|\mathtt{L_1}],\mathtt{L_2},\mathtt{L_3}), \neg\mathtt{part}(\mathtt{L_1},\mathtt{L_2},[\mathtt{X}|\mathtt{L_3}]).\\
r_2 : \mathtt{part}(\mathtt{L_1},\mathtt{L_2},[\mathtt{X}|\mathtt{L_3}]) \leftarrow \mathtt{part}([\mathtt{X}|\mathtt{L_1}],\mathtt{L_2},\mathtt{L_3}), \neg\mathtt{part}(\mathtt{L_1},[\mathtt{X}|\mathtt{L_2}],\mathtt{L_3}).\\
r_3 : \mathtt{sol} \leftarrow \mathtt{part}([\,],\mathtt{L},\mathtt{L}).\\
r_4 : \mathtt{p} \leftarrow \neg\mathtt{sol}, \neg\mathtt{p}.
\end{array}$$

The last rule enforces `sol` to be true in every stable model. This program has a standard structure that can be used to express several well-known NP problems, such as binary partition, subset sum, and others. For instance, by replacing rule r_3 with

$$\mathtt{sol} \leftarrow \mathtt{sum}([\,],1,\mathtt{X}), \mathtt{sum}([\,],2,\mathtt{X}).$$

where predicate symbol `sum` is defined as follows

$$\begin{array}{l}
\mathtt{sum}(\mathtt{L_1},1,0) \leftarrow \mathtt{part}([\,],\mathtt{L_1},\mathtt{L_2}).\\
\mathtt{sum}(\mathtt{L_2},2,0) \leftarrow \mathtt{part}([\,],\mathtt{L_1},\mathtt{L_2}).\\
\mathtt{sum}(\mathtt{L},\mathtt{I},\mathtt{X}+\mathtt{C}) \leftarrow \mathtt{sum}([\mathtt{X}|\mathtt{L}],\mathtt{I},\mathtt{C}).
\end{array}$$

we express *binary partition*, a classical NP-complete problem. As another example, we can express the *subset sum* problem by replacing r_3 with the rule $\mathtt{sol} \leftarrow \mathtt{sum}([\,],1,0)$.

The evaluation of the programs discussed above always terminates, but current termination criteria are not able to realize it. However, if termination criteria are applied to adorned programs, then they are able to detect termination of the original programs.

Related Work. A significant body of work has been done on termination of logic programs under top-down evaluation (De Schreye and Decorte 1994; Marchiori

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

1996; Ohlebusch 2001; Bonatti 2004; Codish *et al.* 2005; Serebrenik and De Schreye 2005; Bruynooghe *et al.* 2007; Nguyen *et al.* 2007; Baselice *et al.* 2009; Schneider-Kamp *et al.* 2009a; Schneider-Kamp *et al.* 2009b; Nishida and Vidal 2010; Schneider-Kamp *et al.* 2010; Voets and De Schreye 2011). Our work is also akin to work done in the area of term rewriting (Zantema 1994; Zantema 1995; Ferreira and Zantema 1996; Arts and Giesl 2000; Endrullis *et al.* 2008; Sternagel and Middeldorp 2008). In this paper, we consider logic programs with function symbols *under the stable model semantics* (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991), and thus, as already noticed and discussed in Calimeri *et al.* (2010), Alviano *et al.* (2010), all the excellent works above cannot straightforwardly be applied to our setting. As for the context considered in this paper, recent years have witnessed an increasing interest in the problem of identifying logic programs with function symbols for which a finite set of finite stable models exists and can be computed. The class of *finitely-ground programs*, guaranteeing the aforementioned desirable property, has been proposed in Calimeri *et al.* (2008). Since membership in the class is not decidable, recent research has concentrated on the identification of sufficient conditions, that we call *termination criteria*, for a program to be finitely-ground. Efforts in this direction are *ω-restricted programs* (Syrjänen 2001), *λ-restricted programs* (Gebser *et al.* 2007), and *finite domain programs* (Calimeri *et al.* 2008). More general classes are *argument-restricted programs* (Lierler and Lifschitz 2009), *safe* and *Γ-acyclic programs* (Greco *et al.* 2012b). This paper presents a technique that can be used in conjunction with the aforementioned termination criteria to recognize more programs as finitely-ground.

Our work is also related to research done in the database community on termination of the chase procedure (Fagin *et al.* 2005; Deutsch *et al.* 2008; Marnette 2009; Meier *et al.* 2009; Greco and Spezzano 2010; Greco *et al.* 2011; Krötzsch and Rudolph 2011; Grau *et al.* 2012). A survey on this topic can be found in Greco *et al.* (2012a). The fundamental difference is that the setting considered in this paper is much more general than the chase setting. In fact, while our approach can be applied to the chase setting by considering logic programs obtained via skolemization of the existential rules used with the chase, the vice versa is not true. The logic rules obtained via skolemization of existential rules are of a very restricted form: function symbols appear only in rule heads, each function symbol occurs at most once, there is no nesting of function symbols. In contrast, we consider logic programs allowing an arbitrary use of function symbols: they can appear in both the head and the body of rules, may be nested, and the same function symbol can appear multiple times. While in the chase setting determining the adornment symbol of a variable occurrence in the body of a rule can be straightforwardly done by looking at predicate symbol adornments (because there are no function symbols in the body), this problem is more complex in our setting because of the presence of possibly nested functions that can occur multiple times. Thus, a more complex (recursive) analysis is performed (cf. Definition 1) which uses *adornment definitions* (introduced in this paper and not used in the chase techniques) to memorize the "history" of complex term adornments. Furthermore, while determining head adornments is trivial in the chase setting, it gets more involved in our case because of nested complex terms (cf. Definition 3).

Organization. The paper is organized as follows. First, preliminaries on logic programs with function symbols are reported. Then, we present our transformation technique. Finally, we show different properties of our approach and conclude.

2 Logic programs with function symbols

Syntax. We assume to have infinite sets of *constants*, *variables*, *predicate symbols*, and *function symbols*. Each predicate and function symbol is associated with an *arity*, which is a non-negative integer for predicate symbols and a positive integer for function symbols.

A *term* is either a constant, a variable, or an expression of the form $f(t_1,\ldots,t_m)$, where f is a function symbol of arity m and the t_i's are terms (in the first two cases we say the term is *simple* while in the last case we say it is *complex*).

An *atom* is of the form $p(t_1,\ldots,t_n)$, where p is a predicate symbol of arity n and the t_i's are terms—we also use the notation $p(\bar{t})$ to refer to an atom, where $\bar{t}$ is understood to be a sequence of n terms. A *literal* is either an atom A (*positive* literal) or its negation $\neg A$ (*negative* literal). A (*disjunctive*) *rule* r is of the form $A_1 \vee \cdots \vee A_m \leftarrow L_1, \cdots, L_k$ where $m > 0$, $k \geqslant 0$, the A_i's are atoms, the L_j's are literals. The disjunction $A_1 \vee \cdots \vee A_m$ is called the *head* of r and is denoted by $head(r)$; the conjunction $L_1, \cdots, L_k$ is called the *body* of r and is denoted by $body(r)$. With a slight abuse of notation we use $head(r)$ (resp. $body(r)$) to also denote the *set* of atoms (resp. literals) appearing in the head (resp. body) of r. If $m = 1$, then r is *normal*; if all the L_j's are positive literals, r is *positive*.

A *program* is a finite set of rules. A program is *normal* (resp. *positive*) if every rule in it is normal (resp. positive). A term (resp. atom, literal, rule, program) is *ground* if no variables occur in it. A ground normal rule with empty body is also called a *fact*. We assume that programs are *range restricted*, i.e., for each rule, variables appearing in the head or in negative body literals also appear in some positive body literal. The *definition* of a predicate symbol p appearing in a program $\mathscr{P}$ consists of all rules in $\mathscr{P}$ having p in the head. Predicate symbols are partitioned into two classes: *base* predicate symbols, whose definition can contain only facts, and *derived* predicate symbols, whose definition can contain any rule. A *base* (resp. *derived*) atom is an atom whose predicate symbol is base (resp. derived). Facts defining base predicate symbols are called *database facts*.

Semantics. Consider a program $\mathscr{P}$. The *Herbrand universe* $H_{\mathscr{P}}$ of $\mathscr{P}$ is the possibly infinite set of ground terms which can be built using constants and function symbols appearing in $\mathscr{P}$. The *Herbrand base* $B_{\mathscr{P}}$ of $\mathscr{P}$ is the set of ground atoms which can be built using predicate symbols appearing in $\mathscr{P}$ and ground terms of $H_{\mathscr{P}}$. A rule r' is a *ground instance* of a rule r in $\mathscr{P}$ if r' can be obtained from r by substituting every variable in r with some ground term in $H_{\mathscr{P}}$; $ground(\mathscr{P})$ denotes the set of all ground instances of the rules in $\mathscr{P}$. An *interpretation* of $\mathscr{P}$ is any subset I of $B_{\mathscr{P}}$. The truth value of a ground atom A w.r.t. I, denoted $value_I(A)$, is *true* if $A \in I$, *false* otherwise. The truth value of $\neg A$ w.r.t. I, denoted $value_I(\neg A)$, is *true* if $A \notin I$, *false* otherwise. A ground rule r is *satisfied* by I if there is a ground literal L in $body(r)$ s.t. $value_I(L) = false$ or there is a ground atom A in $head(r)$ s.t.

$value_I(A) = true$. Thus, if the body of r is empty, r is satisfied by I if there is an atom A in $head(r)$ s.t. $value_I(A) = true$. An interpretation M of $\mathcal{P}$ is a *model* of $\mathcal{P}$ if M satisfies every ground rule in $ground(\mathcal{P})$. A model M of $\mathcal{P}$ is minimal if no proper subset of M is a model of $\mathcal{P}$. The set of minimal models of $\mathcal{P}$ is denoted by $\mathcal{MM}(\mathcal{P})$.

Given an interpretation I of $\mathcal{P}$, let $\mathcal{P}^I$ denote the ground positive program derived from $ground(\mathcal{P})$ by *(i)* removing every rule containing a negative literal $\neg A$ in the body with $A \in I$, and *(ii)* removing all negative literals from the remaining rules. An interpretation I is a *stable model* of $\mathcal{P}$ if and only if $I \in \mathcal{MM}(\mathcal{P}^I)$. The set of stable models of $\mathcal{P}$ is denoted by $\mathcal{SM}(\mathcal{P})$. Stable models are minimal models. Furthermore, $\mathcal{SM}(\mathcal{P}) = \mathcal{MM}(\mathcal{P})$ if $\mathcal{P}$ is a positive program. Positive normal programs have a unique minimal model.

3 Program adornment

In this section, we present a technique for checking termination of the bottom-up evaluation of logic programs with function symbols. For ease of presentation, we initially restrict ourselves to positive normal programs; the extension to arbitrary programs with disjunction (in the head) and negation (in the body) will be discussed at the end of the section. Checking finiteness of the minimal model of a positive normal program is equivalent to checking termination of the program bottom-up evaluation. For the sake of simplicity and without loss of generality, we assume that database facts do not contain complex terms (hence, we can assume that base atoms in rule bodies do not contain complex terms). For instance, the set of facts $\{\mathtt{base(a)}, \mathtt{base(f(b))}\}$ can be replaced with the set of facts $\{\mathtt{base(a)}, \mathtt{base(b)}\}$ and the rules $\{\mathtt{base_d(a)} \leftarrow \mathtt{base(a)}, \mathtt{base_d(f(b))} \leftarrow \mathtt{base(b)}\}$, where $\mathtt{base_d}$ is a derived predicate symbol. Additionally, atoms appearing in rule bodies and having $\mathtt{base}$ as predicate symbol are replaced with the same atoms where $\mathtt{base_d}$ replaces $\mathtt{base}$. Finally, since database facts are not relevant for the proposed technique, they are not shown in our examples. In fact, as discussed in the following section, our technique allows us to conclude that a program terminates for any set of database facts.

We start by introducing notations and terminology used hereafter. Given a program $\mathcal{P}$, we define the *adornment alphabet* $\Lambda = \{\epsilon\} \cup \{f_i \mid f$ is a function symbol in $\mathcal{P}$ and $i \in \mathbb{N}\}$; elements of Λ are called *adornment symbols*. An *adornment* α for a predicate symbol p of arity n is a string of length n over the alphabet Λ; the expression p^α is an *adorned predicate symbol* and $p^\alpha(t_1, \ldots, t_n)$ is an *adorned atom*, where the t_i's are terms. An *adorned conjunction* is a conjunction of adorned atoms. An *adorned rule* is a rule containing only adorned atoms. Given an adornment symbol f_i in $\Lambda - \{\epsilon\}$, an *adornment definition* for f_i is an expression of the form $f_i = f(\alpha_1, \ldots, \alpha_m)$, where m is the arity of function symbol f and the α_i's are adornment symbols. As an example, if our technique derives an adorned predicate symbol $\mathtt{p}^{\mathtt{f_1 g_1}}$ with adornment definitions $\mathtt{f_1} = \mathtt{f}(\epsilon)$ and $\mathtt{g_1} = \mathtt{g(f_1)}$, this means that the bottom-up evaluation of the considered program might yield atoms of the form

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

$\mathtt{p(f(c_1), g(f(c_2)))}$ with $\mathtt{c_1}$ and $\mathtt{c_2}$ being constants.[1] Intuitively, adornment definitions are used to keep track of what kind of complex terms can be propagated.

Roughly speaking, our transformation technique works as follows. It maintains a set of adorned predicate symbols, a set of adornment definitions, and a set of adorned rules. Whenever we find a rule whose body can be adorned in a "coherent" way (we will make clear what this means in Definition 2), we derive an adorned predicate symbol from the rule head (using the body adornments), and generate an adorned rule. In this step, new adornment definitions might be generated as well. New adorned predicate symbols are used to generate further adorned rules. Below is an example to illustrate the basic idea.

Example 4

Consider the following program $\mathscr{P}_4$ where `base` is a base predicate symbol.

$$\begin{aligned} r_0 :&\quad \mathtt{p(X, f(X)) \leftarrow base(X).} \\ r_1 :&\quad \mathtt{p(X, f(X)) \leftarrow p(Y, X), base(Y).} \\ r_2 :&\quad \mathtt{p(X, Y) \leftarrow p(f(X), f(Y)).} \end{aligned}$$

First, base predicate symbols are adorned with strings of ϵ's; thus, we get the adorned predicate symbol $\mathtt{base}^{\epsilon}$. This is used to adorn the body of r_0 so as to get

$$\rho_0 : \quad \mathtt{p}^{\epsilon \mathtt{f_1}}\mathtt{(X, f(X))} \leftarrow \mathtt{base}^{\epsilon}\mathtt{(X).}$$

from which we derive the new adorned predicate symbol $\mathtt{p}^{\epsilon \mathtt{f_1}}$, and the adornment definition $\mathtt{f_1} = \mathtt{f}(\epsilon)$. Next, $\mathtt{p}^{\epsilon \mathtt{f_1}}$ and $\mathtt{base}^{\epsilon}$ are used to adorn the body of r_1 so as to get

$$\rho_1 : \quad \mathtt{p}^{\mathtt{f_1 f_2}}\mathtt{(X, f(X))} \leftarrow \mathtt{p}^{\epsilon \mathtt{f_1}}\mathtt{(Y, X)}, \mathtt{base}^{\epsilon}\mathtt{(Y)}$$

from which we derive the new adorned predicate symbol $\mathtt{p}^{\mathtt{f_1 f_2}}$, and the adornment definition $\mathtt{f_2} = \mathtt{f(f_1)}$. Intuitively, the body of ρ_1 is coherently adorned because `Y` is always associated with the same adornment symbol ϵ. Using the new adorned predicate symbol $\mathtt{p}^{\mathtt{f_1 f_2}}$, we can adorn rule r_2 and get

$$\rho_2 : \quad \mathtt{p}^{\epsilon \mathtt{f_1}}\mathtt{(X, Y)} \leftarrow \mathtt{p}^{\mathtt{f_1 f_2}}\mathtt{(f(X), f(Y)).}$$

At this point, we are not able to generate new adorned rules (using the adorned predicate symbols generated so far) with coherently adorned bodies and the transformation terminates. In fact, $\mathtt{p}^{\mathtt{f_1 f_2}}\mathtt{(Y, X)}, \mathtt{base}^{\epsilon}\mathtt{(Y)}$ is not coherently adorned because the same variable `Y` is associated with both $\mathtt{f_1}$ and ϵ; moreover, $\mathtt{p}^{\epsilon \mathtt{f_1}}\mathtt{(f(X), f(Y))}$ is not coherently adorned because `f(X)` does not comply with the (simple) term structure described by ϵ.

To determine termination of the bottom-up evaluation of $\mathscr{P}_4$, we can apply current termination criteria to $\mathscr{P}_4^{\mu} = \{\rho_0, \rho_1, \rho_2\}$ rather than $\mathscr{P}_4$. In fact, our technique ensures that if $\mathscr{P}_4^{\mu}$ is recognized as terminating, so is $\mathscr{P}_4$. Notice that both $\mathscr{P}_4$ and $\mathscr{P}_4^{\mu}$ are recursive, but while some termination criteria (e.g., the argument-restricted and Γ-acyclicity criteria) detect $\mathscr{P}_4^{\mu}$ as terminating, none of the current termination criteria is able to realize that $\mathscr{P}_4$ terminates.

[1] Here predicate symbol `p` is assumed of arity 2, and function symbols `f` and `g` are assumed of arity 1.

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

In the following, we formally present our technique. First, we define how to determine the adornment symbols associated with the variables in an adorned conjunction, and how to check if the conjunction is coherently adorned. Then, we define how to determine the adornment of a rule head when its body is coherently adorned. Finally, we present the complete technique.

Checking adornment coherency. The aim of adornment coherency is to check if the adorned conjunction in the body of an adorned rule satisfies two conditions that are necessary for the rule to "trigger". First, for each adorned atom $p^{\alpha_1 \cdots \alpha_n}(t_1, \ldots, t_n)$ in the conjunction, we check if t_i complies with the term structure corresponding to α_i. As an example, in the adorned atom $\mathtt{p}^{\mathtt{f}_1}(\mathtt{g}(\mathtt{X}))$ with adornment definition $\mathtt{f}_1 = \mathtt{f}(\epsilon)$, we have that $\mathtt{g}(\mathtt{X})$ does not comply with the term structure $\mathtt{f}(\mathtt{c})$ corresponding to $\mathtt{f}_1$, where $\mathtt{c}$ is an arbitrary constant. Second, we determine the adornment symbol associated with each variable occurrence in the conjunction and check if, for every variable, all its occurrences are associated with adornment symbols describing compatible term structures. As an example, if $\mathtt{p}^{\mathtt{f}_1\mathtt{g}_1}(\mathtt{X},\mathtt{X})$ is an atom in the conjunction with adornment definitions $\mathtt{f}_1 = \mathtt{f}(\epsilon)$ and $\mathtt{g}_1 = \mathtt{g}(\epsilon)$, then two different term structures are associated with two occurrences of the same variable and the conjunction is not coherently adorned.

Function *TermAdn* below determines the adornment symbols associated with the variables in a term t_i in an adorned atom $p^{\alpha_1 \cdots \alpha_n}(t_1, \ldots, t_n)$ on the basis of α_i and a set of adornment definitions S. Function *BodyAdn* simply collects the adornment symbols for all variables in an adorned conjunction (using *TermAdn*) and is used to check if the conjunction is coherently adorned.

Definition 1
Let $body^\sigma$ be an adorned conjunction and S a set of adornment definitions. We define

$$BodyAdn(body^\sigma, S) = \bigcup_{\substack{p^{\alpha_1 \cdots \alpha_n}(t_1,\ldots,t_n) \,\in\, body^\sigma \wedge \\ 1 \leqslant i \leqslant n}} TermAdn(t_i, \alpha_i, S);$$

where *TermAdn* is recursively defined as follows:

1. $TermAdn(t_i, \epsilon, S) = \emptyset$, if t_i is a constant;
2. $TermAdn(t_i, \alpha_i, S) = \{t_i/\alpha_i\}$, if t_i is a variable;
3. $TermAdn(f(u_1, \ldots, u_m), f_i, S) = \bigcup_{j=1}^{m} TermAdn(u_j, \alpha_j, S)$, if $f_i = f(\alpha_1, \ldots, \alpha_m)$ is in S;
4. $TermAdn(t_i, \alpha_i, S) = \{fail\}$, otherwise.

Notice that there is a non-deterministic choice to be made in item 3 above when there are multiple adornment definitions for the same f_i in S. Depending on the choice, $BodyAdn(body^\sigma, S)$ can return different sets; we define $SBodyAdn(body^\sigma, S)$ as the set of all possible outcomes; notice that if $body^\sigma$ is the empty conjunction, $SBodyAdn(body^\sigma, S)$ contains only the empty set.

Definition 2
Consider an adorned conjunction $body^\sigma$ and a set of adornment definitions S, and let $W \in SBodyAdn(body^\sigma, S)$. We say that $body^\sigma$ is *coherently adorned* w.r.t. W iff

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

$fail \notin W$ and for every two distinct X/α and X/β in W it is the case that $\alpha = f_i$ and $\beta = f_j$, where f is a function symbol and $i, j \in \mathbb{N}$.

Given a set $W \in SBodyAdn(body^\sigma, S)$, we define $\mathscr{S}(W)$ as the set of all subsets $\mathscr{T}_\mathscr{S}$ of W containing exactly one expression of the form X/α for every variable X in $body^\sigma$.

Example 5
Consider the set of adornment definitions $S = \{\mathtt{f}_2 = \mathtt{f}(\mathtt{f}_1), \mathtt{f}_1 = \mathtt{f}(\epsilon), \mathtt{g}_1 = \mathtt{g}(\epsilon)\}$. For the adorned conjunction $\mathtt{p}^{\mathtt{f}_2\mathtt{g}_1}(\mathtt{f}(\mathtt{f}(\mathtt{X})), \mathtt{g}(\mathtt{X}))$, we have that $BodyAdn(\mathtt{p}^{\mathtt{f}_2\mathtt{g}_1}(\mathtt{f}(\mathtt{f}(\mathtt{X})), \mathtt{g}(\mathtt{X})), S)$ can return only the set $W = TermAdn(\mathtt{f}(\mathtt{f}(\mathtt{X})), \mathtt{f}_2, S) \cup TermAdn(\mathtt{g}(\mathtt{X}), \mathtt{g}_1, S) = TermAdn(\mathtt{f}(\mathtt{X}), \mathtt{f}_1, S) \cup TermAdn(\mathtt{X}, \epsilon, S) = TermAdn(\mathtt{X}, \epsilon, S) \cup \{\mathtt{X}/\epsilon\} = \{\mathtt{X}/\epsilon\} \cup \{\mathtt{X}/\epsilon\} = \{\mathtt{X}/\epsilon\}$ and $\mathtt{p}^{\mathtt{f}_2\mathtt{g}_1}(\mathtt{f}(\mathtt{f}(\mathtt{X})), \mathtt{g}(\mathtt{X}))$ is coherently adorned w.r.t. W. Considering $\mathtt{q}^{\mathtt{f}_2}(\mathtt{f}(\mathtt{g}(\mathtt{X})))$, we have that $BodyAdn(\mathtt{q}^{\mathtt{f}_2}(\mathtt{f}(\mathtt{g}(\mathtt{X}))), S)$ can return only $W = TermAdn(\mathtt{f}(\mathtt{g}(\mathtt{X})), \mathtt{f}_2, S) = TermAdn(\mathtt{g}(\mathtt{X}), \mathtt{f}_1, S) = \{fail\}$ and $\mathtt{q}^{\mathtt{f}_2}(\mathtt{f}(\mathtt{g}(\mathtt{X})))$ is not coherently adorned w.r.t. W. Considering $\mathtt{p}^{\mathtt{f}_2\mathtt{g}_1}(\mathtt{f}(\mathtt{X}), \mathtt{g}(\mathtt{X}))$, we have that $BodyAdn(\mathtt{p}^{\mathtt{f}_2\mathtt{g}_1}(\mathtt{f}(\mathtt{X}), \mathtt{g}(\mathtt{X})), S)$ returns only $W = TermAdn(\mathtt{f}(\mathtt{X}), \mathtt{f}_2, S) \cup TermAdn(\mathtt{g}(\mathtt{X}), \mathtt{g}_1, S) = TermAdn(\mathtt{X}, \mathtt{f}_1, S) \cup TermAdn(\mathtt{X}, \epsilon, S) = \{\mathtt{X}/\mathtt{f}_1\} \cup \{\mathtt{X}/\epsilon\} = \{\mathtt{X}/\mathtt{f}_1, \mathtt{X}/\epsilon\}$ and $\mathtt{p}^{\mathtt{f}_2\mathtt{g}_1}(\mathtt{f}(\mathtt{X}), \mathtt{g}(\mathtt{X}))$ is not coherently adorned w.r.t. W.

Head adornment. When the conjunction in the body of a rule can be coherently adorned, adornments are propagated from the body to the head. The adornment of the head predicate symbol is determined on the basis of the structure of the terms in the head, and the adornment symbols associated with the variables in the body. As an example, consider the rule $\mathtt{p}(\mathtt{X}, \mathtt{f}(\mathtt{X}, \mathtt{g}(\mathtt{X}))) \leftarrow \mathtt{b}(\mathtt{X})$ and the adorned body conjunction $\mathtt{b}^\epsilon(\mathtt{X})$. The adornment symbol associated with variable $\mathtt{X}$ is ϵ, which intuitively means that the bottom-up evaluation of the program might yield atoms of the form $\mathtt{b}(\mathtt{c})$, with $\mathtt{c}$ being a constant. Thus, the rule above might yield atoms of form $\mathtt{p}(\mathtt{c}, \mathtt{f}(\mathtt{c}, \mathtt{g}(\mathtt{c})))$. To keep track of this, the head predicate symbol is adorned as $\mathtt{p}^{\epsilon\mathtt{f}_1}$, and the adornment definitions $\mathtt{f}_1 = \mathtt{f}(\epsilon, \mathtt{g}_1)$ and $\mathtt{g}_1 = \mathtt{g}(\epsilon)$ are derived. We start by introducing a special (asymmetric) "union operator", denoted by $\sqcup$, which takes as input a set of adornment definitions S and a set containing a single adornment definition $f_h = f(\alpha_1, \ldots, \alpha_m)$, and gives as output a set S' of adornment definitions where $S \subseteq S'$. Operator $\sqcup$ is defined as follows:
$S \sqcup \{f_h = f(\alpha_1, \ldots, \alpha_m)\} = S$, if there exists $f_k = f(\alpha_1, \ldots, \alpha_m)$ in S;
$S \sqcup \{f_h = f(\alpha_1, \ldots, \alpha_m)\} = S \cup \{f_h = f(\alpha_1, \ldots, \alpha_m)\}$, if there is no $f_k = f(\alpha_1, \ldots, \alpha_m)$ in S.

We are now ready to define how rule heads are adorned.

Definition 3
Consider a positive normal rule $p(t_1, \ldots, t_n) \leftarrow body$, a set of adornment definitions S_0, and an adorned conjunction $body^\sigma$ obtained by adding adornments to all atoms in $body$. Let W be an element of $SBodyAdn(body^\sigma, S_0)$ s.t. $body^\sigma$ is coherently adorned w.r.t. W, and $\mathscr{T}_\mathscr{S} \in \mathscr{S}(W)$. The adornment of the head atom $p(t_1, \ldots, t_n)$ w.r.t. $\mathscr{T}_\mathscr{S}$ and S_0 is

$$SetHeadAdn(p(t_1, \ldots, t_n), \mathscr{T}_\mathscr{S}, S_0) = \langle p^{\alpha_1 \ldots \alpha_n}(t_1, \ldots, t_n), S_n \rangle$$

where $\langle \alpha_1, S_1 \rangle = Adn(t_1, \mathscr{T}_\mathscr{S}, S_0)$, $\langle \alpha_2, S_2 \rangle = Adn(t_2, \mathscr{T}_\mathscr{S}, S_1)$, $\ldots$, $\langle \alpha_n, S_n \rangle = Adn(t_n, \mathscr{T}_\mathscr{S}, S_{n-1})$ and function Adn is defined as follows:

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

- $Adn(t, \mathcal{T}_{\mathcal{S}}, S) = \langle \epsilon, S \rangle$, if t is a constant;
- $Adn(t, \mathcal{T}_{\mathcal{S}}, S) = \langle \alpha_i, S \rangle$, if t is a variable X and X/α_i is in T;[2]
- $Adn(f(u_1, \ldots, u_m), \mathcal{T}_{\mathcal{S}}, S) = \langle f_j, S' \rangle$ where
  - — $\langle \beta_1, S_1 \rangle = Adn(u_1, \mathcal{T}_{\mathcal{S}}, S)$;
  - — $\langle \beta_2, S_2 \rangle = Adn(u_2, \mathcal{T}_{\mathcal{S}}, S_1)$;

    $\vdots$
  - — $\langle \beta_m, S_m \rangle = Adn(u_m, \mathcal{T}_{\mathcal{S}}, S_{m-1})$;
  - — $S' = S_m \sqcup \{f_i = f(\beta_1, \ldots, \beta_m)\}$, with $i = \max\{k \mid f_k = f(\gamma_1, \ldots, \gamma_m) \in S_m\} + 1$;
  - — j is s.t. $f_j = f(\beta_1, \ldots, \beta_m)$ is in S'.

Example 6

Consider the rule $\mathtt{p(f(a,nil),f(X,f(Y,nil)))} \leftarrow \mathtt{base(X,Y)}$ and the adorned body $\mathtt{base}^{\epsilon\epsilon}\mathtt{(X,Y)}$. Then, $SBodyAdn(\mathtt{base}^{\epsilon\epsilon}\mathtt{(X,Y)}, \emptyset)$ has one element $W = \{\mathtt{X}/\epsilon, \mathtt{Y}/\epsilon\}$ and $\mathcal{T}_{\mathcal{S}} = \{\mathtt{X}/\epsilon, \mathtt{Y}/\epsilon\}$ is the only element in $\mathcal{S}(W)$. Then, $SetHeadAdn(\mathtt{p(f(a,nil),f(X,f(Y,nil)))}, \mathcal{T}_{\mathcal{S}}, \emptyset)$ gives $\langle \mathtt{p}^{\mathtt{f_1 f_2}}\mathtt{(f(a,nil),f(X,f(Y,nil)))}, S_2 \rangle$, where $S_2 = \{\mathtt{f_1} = \mathtt{f}(\epsilon, \epsilon), \mathtt{f_2} = \mathtt{f}(\epsilon, \mathtt{f_1})\}$. In fact, by Definition 3,

- $Adn(\mathtt{f(a,nil)}, T, \emptyset)$ gives $\langle \mathtt{f_1}, S_1 \rangle$, where $S_1 = \{\mathtt{f_1} = \mathtt{f}(\epsilon, \epsilon)\}$, since
  - $Adn(\mathtt{a}, T, \emptyset)$ gives $\langle \epsilon, \emptyset \rangle$, and
  - $Adn(\mathtt{nil}, T, \emptyset)$ gives $\langle \epsilon, \emptyset \rangle$.
- Then, $Adn(\mathtt{f(X,f(Y,nil))}, T, S_1)$ gives $\langle \mathtt{f_2}, S_2 \rangle$ as
  - $Adn(\mathtt{X}, T, S_1)$ gives $\langle \epsilon, S_1 \rangle$, and
  - $Adn(\mathtt{f(Y,nil)}, T, S_1)$ gives $\langle \mathtt{f_1}, S_1 \rangle$ as
    - $Adn(\mathtt{Y}, T, S_1)$ gives $\langle \epsilon, S_1 \rangle$, and
    - $Adn(\mathtt{nil}, T, S_1)$ gives $\langle \epsilon, S_1 \rangle$.

Transformation function. Before presenting the complete transformation technique, we introduce some further notations and terminology. An *adornment substitution* θ is a set of pairs the form f_i/f_j with $i > j$ that does not contain two pairs f_i/f_j and f_j/f_k—i.e., a symbol f_i cannot be replaced by a symbol g_h and a symbol f_j used to replace a symbol f_i cannot be substituted in θ by a symbol f_k—where f_i, f_j, f_k, g_h are in $\Lambda - \{\epsilon\}$. For instance, $\{f_2/f_1, g_3/g_1\}$ is an adornment substitution, but $\{f_1/g_1\}$ and $\{f_3/f_2, f_2/f_1\}$ are not. The result of applying θ to an adorned rule r, denoted $r\theta$, is the adorned rule obtained from r by substituting each f_i appearing in r with f_j, where f_i/f_j belongs to θ. The result of applying θ to a set of adorned rules $\mathcal{P}^\mu$ (resp. adorned predicate symbols AP, adornment definitions S), denoted $\mathcal{P}^\mu\theta$ (resp. $AP\theta$, $S\theta$), is analogously defined.

The set of the *adorned versions* of an atom $p(\bar{t})$ w.r.t. a set of adorned predicate symbols AP is $\mathcal{A}(p(\bar{t}), AP) = \{p^\alpha(\bar{t}) \mid p^\alpha \in AP\}$. The set of the *adorned versions* of a conjunction of atoms $body = A_1, \ldots, A_k$ w.r.t. AP is $\mathcal{A}(body, AP) = \{AA_1, \ldots, AA_k \mid AA_i \in \mathcal{A}(A_i, AP) \text{ for } 1 \leqslant i \leqslant k\}$. If *body* is the empty conjunction, then $\mathcal{A}(body, AP)$ contains only the empty conjunction.

[2] Notice that X always appears in $body^\sigma$ as we consider range restricted programs.

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

Algorithm 1 Adorn

Input: Positive normal program $\mathscr{P}$.
Output: Adorned positive normal program $\mathscr{P}^{\mu}$.
1: $S = \emptyset$; $\mathscr{P}^{\mu} = \emptyset$;
2: $AP = \{p^{\alpha_1 \ldots \alpha_n} \mid p$ is a base predicate symbol appearing in $\mathscr{P}$ of arity n and every $\alpha_i = \epsilon\}$;
3: **repeat**
4: $\quad AP' = AP$;
5: $\quad$ **for each** rule $p(\bar{t}) \leftarrow body$ in $\mathscr{P}$ **do**
6: $\quad\quad$ **for each** $body^{\sigma}$ in $\mathscr{A}(body, AP)$ **do**
7: $\quad\quad\quad$ **for each** W in $SBodyAdn(body^{\sigma}, S)$ **do**
8: $\quad\quad\quad\quad$ **if** $body^{\sigma}$ is coherently adorned w.r.t. W **then**
9: $\quad\quad\quad\quad\quad$ **for each** $\mathscr{T}_{\mathscr{S}}$ in $\mathscr{S}(W)$ **do**
10: $\quad\quad\quad\quad\quad\quad \langle p^{\alpha}(\bar{t}), S' \rangle = SetHeadAdn(p(\bar{t}), \mathscr{T}_{\mathscr{S}}, S)$;
11: $\quad\quad\quad\quad\quad\quad AP = AP \cup \{p^{\alpha}\}$; $S = S'$;
12: $\quad\quad\quad\quad\quad\quad ar = p^{\alpha}(\bar{t}) \leftarrow body^{\sigma}$;
13: $\quad\quad\quad\quad\quad\quad \mathscr{P}^{\mu} = \mathscr{P}^{\mu} \cup \{ar\}$;
14: $\quad\quad\quad\quad\quad\quad$ **if** $\exists r \in \mathscr{P}^{\mu} \wedge \exists substitution\ \theta \neq \emptyset\ s.t.\ ar\theta = r$ **then**
15: $\quad\quad\quad\quad\quad\quad\quad \mathscr{P}^{\mu} = \mathscr{P}^{\mu}\theta$; $AP = AP\theta$; $S = S\theta$;
16: **until** $AP' = AP$
17: **return** $\mathscr{P}^{\mu}$;

Function *Adorn* performs the transformation of a positive normal program. It maintains a set of adornment definitions S, a set of adorned rules $\mathscr{P}^{\mu}$ (eventually, this will be the output), and a set AP of adorned predicate symbols. Initially, S and $\mathscr{P}^{\mu}$ are empty (line 1), and AP contains all base predicate symbols in $\mathscr{P}$ adorned with strings of ϵ's (line 2). Then, for each coherently adorned body $body^{\sigma}$ of a rule $p(\bar{t}) \leftarrow body$ in the original program, we determine the adorned head $p^{\alpha}(\bar{t})$ and the set of adornment definitions S' using function *SetHeadAdn* (line 10). The set AP is extended with p^{α}, S' is assigned to S (line 1), and a new adorned rule ar of the form $p^{\alpha}(\bar{t}) \leftarrow body^{\sigma}$ is added to $\mathscr{P}^{\mu}$ (line 1). If there exists an adornment substitution θ that applied to ar gives a rule r in $\mathscr{P}^{\mu}$, then θ is applied to $\mathscr{P}^{\mu}$, AP, and S (line 15). This ensures termination of *Adorn*.

Example 7
Consider the following program $\mathscr{P}_7$ computing the reverse of list $[\mathtt{a}, \mathtt{b}, \mathtt{c}]$:

$$r_0 : \quad \mathtt{reverse(f(a, f(b, f(c, nil))), nil).}$$
$$r_1 : \quad \mathtt{reverse(L_1, f(X, L_2))} \leftarrow \mathtt{reverse(f(X, L_1), L_2).}$$

Here function symbol $\mathtt{f}$ denotes the list constructor operator "|"and constant $\mathtt{nil}$ denotes the empty list "[ ]". The bottom-up evaluation of $\mathscr{P}_7$ terminates and the reverse $\mathtt{L}$ of $[\mathtt{a}, \mathtt{b}, \mathtt{c}]$ can be retrieved from the atom $\mathtt{reverse([\,], L)}$ in the minimal model of $\mathscr{P}_7$.

Our technique works as follows. Initially, $S = \emptyset$, $\mathscr{P}^{\mu} = \emptyset$, and $AP = \emptyset$. Using AP, the algorithm can determine a coherently adorned body conjunction only for the first rule (whose body is empty), from which we get the adorned rule

$$\rho_0 : \quad \mathtt{reverse}^{\mathtt{f}_3\epsilon}\mathtt{(f(a, f(b, f(c, nil))), nil).}$$

which is added to $\mathscr{P}^{\mu}$. Furthermore, $\mathtt{reverse}^{\mathtt{f}_3\epsilon}$ is added to AP, and the adornment definitions $\mathtt{f}_3 = \mathtt{f}(\epsilon, \mathtt{f}_2)$, $\mathtt{f}_2 = \mathtt{f}(\epsilon, \mathtt{f}_1)$, $\mathtt{f}_1 = \mathtt{f}(\epsilon, \epsilon)$ are added to S. Now, the algorithm

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

can obtain a coherently adorned body conjunction for r_1, and we get

$$\rho_1: \quad \mathtt{reverse}^{\mathtt{f}_2\mathtt{f}_1}(\mathtt{L}_1, \mathtt{f}(\mathtt{X}, \mathtt{L}_2)) \leftarrow \mathtt{reverse}^{\mathtt{f}_3\epsilon}(\mathtt{f}(\mathtt{X}, \mathtt{L}_1), \mathtt{L}_2).$$

Rule ρ_1 is added to $\mathscr{P}^\mu$, $\mathtt{reverse}^{\mathtt{f}_2\mathtt{f}_1}$ is added to AP, whereas S remains the same. Similarly, in the next two steps we derive the following rules (which are added to $\mathscr{P}^\mu$)

$$\begin{array}{lll} \rho_2: & \mathtt{reverse}^{\mathtt{f}_1\mathtt{f}_2}(\mathtt{L}_1, \mathtt{f}(\mathtt{X}, \mathtt{L}_2)) & \leftarrow \mathtt{reverse}^{\mathtt{f}_2\mathtt{f}_1}(\mathtt{f}(\mathtt{X}, \mathtt{L}_1), \mathtt{L}_2). \\ \rho_3: & \mathtt{reverse}^{\epsilon\mathtt{f}_3}(\mathtt{L}_1, \mathtt{f}(\mathtt{X}, \mathtt{L}_2)) & \leftarrow \mathtt{reverse}^{\mathtt{f}_1\mathtt{f}_2}(\mathtt{f}(\mathtt{X}, \mathtt{L}_1), \mathtt{L}_2). \end{array}$$

Moreover, $\mathtt{reverse}^{\mathtt{f}_1\mathtt{f}_2}$ and $\mathtt{reverse}^{\epsilon\mathtt{f}_3}$ are added to AP, while S does not change. At this point, no new coherently adorned body can be derived and the transformation terminates. The transformed program $Adorn(\mathscr{P}_7) = \{\rho_0, \rho_1, \rho_2, \rho_3\}$ is recognized as terminating by all current termination criteria, while $\mathscr{P}_7$ was not by all of them.

The following example shows the role of adornment substitutions.

Example 8

Consider the program $\mathscr{P}_8$ below where base is a base predicate symbol.

$$\begin{array}{l} \mathtt{p}(\mathtt{X}) \leftarrow \mathtt{base}(\mathtt{X}). \\ \mathtt{p}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}(\mathtt{X}). \end{array}$$

The transformation algorithm adds the following adorned rules to $\mathscr{P}^\mu$

$$\begin{array}{ll} \rho_0: & \mathtt{p}^\epsilon(\mathtt{X}) \leftarrow \mathtt{base}^\epsilon(\mathtt{X}). \\ \rho_1: & \mathtt{p}^{\mathtt{f}_1}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}^\epsilon(\mathtt{X}). \\ \rho_2: & \mathtt{p}^{\mathtt{f}_2}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}^{\mathtt{f}_1}(\mathtt{X}). \\ \rho_3: & \mathtt{p}^{\mathtt{f}_3}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}^{\mathtt{f}_2}(\mathtt{X}). \end{array}$$

Furthermore, the adornment definitions $\mathtt{f}_1 = \mathtt{f}(\epsilon)$, $\mathtt{f}_2 = \mathtt{f}(\mathtt{f}_1)$, $\mathtt{f}_3 = \mathtt{f}(\mathtt{f}_2)$ are added to S, and the adorned predicate symbols $\mathtt{p}^\epsilon$, $\mathtt{p}^{\mathtt{f}_1}$, $\mathtt{p}^{\mathtt{f}_2}$, $\mathtt{p}^{\mathtt{f}_3}$ are added to AP. At this point, the following adorned rule is derived and added to $\mathscr{P}^\mu$:

$$\rho_4: \quad \mathtt{p}^{\mathtt{f}_4}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}^{\mathtt{f}_3}(\mathtt{X}).$$

The adornment definition $\mathtt{f}_4 = \mathtt{f}(\mathtt{f}_3)$ is added to S and $\mathtt{p}^{\mathtt{f}_4}$ is added to AP. However, since there is an adornment substitution $\theta = \{\mathtt{f}_4/\mathtt{f}_2, \mathtt{f}_3/\mathtt{f}_1\}$ such that $\rho_4\theta = \rho_2$, then θ is applied to $\mathscr{P}^\mu$, AP, and S. Thus, $\mathscr{P}^\mu$ becomes $\{\rho_0, \rho_1, \rho_2, \rho_3\theta\}$, where $\rho_3\theta$ is

$$\mathtt{p}^{\mathtt{f}_1}(\mathtt{f}(\mathtt{X})) \leftarrow \mathtt{p}^{\mathtt{f}_2}(\mathtt{X}).$$

$AP = \{\mathtt{p}^\epsilon, \mathtt{p}^{\mathtt{f}_1}, \mathtt{p}^{\mathtt{f}_2}\}$ and $S = \{\mathtt{f}_1 = \mathtt{f}(\epsilon), \mathtt{f}_2 = \mathtt{f}(\mathtt{f}_1), \mathtt{f}_1 = \mathtt{f}(\mathtt{f}_2)\}$. At this point, no new adorned rule can be generated and the algorithm terminates. Notice that both $\mathscr{P}_8$ and $Adorn(\mathscr{P}_8)$ are not recognized as terminating by current termination criteria. Indeed, for any set of database facts containing at least one fact base(c), the minimal model is not finite and the bottom-up evaluation of both programs never terminates. Nevertheless, function *Adorn* terminates.

Disjunctive programs with negation. The extension of technique to programs with disjunction in the head and negation in the body can be carried out by checking termination of a positive normal program derived from a general one as follows. For any program $\mathscr{P}$, we use $st(\mathscr{P})$ to denote the positive normal program obtained from $\mathscr{P}$ by replacing each rule $A_1 \vee \cdots \vee A_m \leftarrow body$ with m positive normal rules

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

of the form $A_i \leftarrow body^+$ $(1 \leqslant i \leqslant m)$ where $body^+$ is obtained from *body* by deleting all negative literals. As we show in the following section, this allows us to apply our technique to general programs.

4 Properties of transformed programs

In this section, we show different properties of the proposed transformation technique.

Theorem 1
Function *Adorn* terminates for every positive normal program $\mathscr{P}$. □

Let *Unadn* be a function taking as input a set of adorned atoms and giving as output the same set where adornments from predicate symbols are dropped. The following theorem says that we can obtain the minimal model of a positive normal program $\mathscr{P}$ from the minimal model of $Adorn(\mathscr{P})$ by dropping adornments.

Theorem 2
Given a positive normal program $\mathscr{P}$, let M be the minimal model of $\mathscr{P}$ and M' the minimal model of $Adorn(\mathscr{P})$. Then, $M = Unadn(M')$.

We restrict ourselves to argument-restricted and Γ-acyclic programs (denoted as $\mathscr{AR}$ and $\mathscr{AP}$, respectively) as they include ω-restricted, λ-restricted, finite domain, and safe programs; however, our approach is an orthogonal technique that can be used with any of the aforementioned termination criteria. The theorem below states that our technique is sound for positive normal programs, that is, if $Adorn(\mathscr{P})$ is in $\mathscr{AR}$ or $\mathscr{AP}$ (and thus is recognized as finitely-ground), then $\mathscr{P}$ is finitely-ground—indeed, we can state that $\mathscr{P} \cup D$ is finitely-ground for any finite set of database facts D (recall that we assume that database facts do not contain complex terms). An important consequence of $\mathscr{P} \cup D$ being finitely-ground is that the minimal model of $\mathscr{P} \cup D$ is finite and can be computed.

Theorem 3
Given a positive normal program $\mathscr{P}$, if $Adorn(\mathscr{P}) \in \mathscr{T}$, then $\mathscr{P} \cup D$ is finitely-ground for any finite set of database facts D, for $\mathscr{T} \in \{\mathscr{AR}, \mathscr{AP}\}$. □

The theorem below states soundness of our technique for arbitrary programs.

Theorem 4
Given a program $\mathscr{P}$, if $Adorn(st(\mathscr{P})) \in \mathscr{T}$, then $\mathscr{P} \cup D$ is finitely-ground for any finite set of database facts D, for $\mathscr{T} \in \{\mathscr{AR}, \mathscr{AP}\}$. □

We use *Adorn*-$\mathscr{T}$ to denote the class of programs $\mathscr{P}$ such that $Adorn(\mathscr{P})$ is in $\mathscr{T}$, where $\mathscr{T}$ is one of $\mathscr{AR}$ and $\mathscr{AP}$. The following theorem allows us to say that the class of programs recognized as finitely-ground by a criterion $\mathscr{T}$ is strictly enlarged using function *Adorn*.

Theorem 5
$\mathscr{T} \subsetneq Adorn\text{-}\mathscr{T}$ for $\mathscr{T} \in \{\mathscr{AR}, \mathscr{AP}\}$. □

https://doi.org/10.1017/S147106841300046X Published online by Cambridge University Press

5 Conclusions

Identifying classes of logic programs with function symbols whose stable models can be computed has attracted a great deal of interest in recent years, leading to the development of different termination criteria. In this paper, we have proposed a new technique which transforms a program into adorned one with the aim of applying current termination criteria to the adorned program rather than the original one. Our technique strictly enlarges the class of programs recognized as finitely-ground by current termination criteria.

A possible direction for future work is to improve the proposed technique so as to perform a more refined analysis directly over a disjunctive program with negation, rather than the positive normal program $st(\mathscr{P})$ derived from the original one. We conjecture that our technique can also be used in conjunction with recently introduced termination criteria (Greco *et al.* 2013; Calautti *et al.* 2013); we plan to investigate this aspect too.

References

Alviano, M., Faber, W. and Leone, N. 2010. Disjunctive asp with functions: Decidable queries and effective computation. *Theory and Practice of Logic Programming 10,* 4-6, 497–512.

Arts, T. and Giesl, J. 2000. Termination of term rewriting using dependency pairs. *Theoretical Computer Science 236,* 1-2, 133–178.

Baselice, S., Bonatti, P. A. and Criscuolo, G. 2009. On finitely recursive programs. *Theory and Practice of Logic Programming 9,* 2, 213–238.

Bonatti, P. A. 2004. Reasoning with infinite stable models. *Artificial Intelligence 156,* 1, 75–111.

Bruynooghe, M., Codish, M., Gallagher, J. P., Genaim, S. and Vanhoof, W. 2007. Termination analysis of logic programs through combination of type-based norms. *ACM Transactions on Programming Languages and Systems 29,* 2.

Calautti, M., Greco, S. and Trubitsyna, I. 2013. Detecting decidable classes of finitely ground logic programs with function symbols. In *International Symposium on Principles and Practice of Declarative Programming* (to appear).

Calimeri, F., Cozza, S., Ianni, G. and Leone, N. 2008. Computable functions in asp: Theory and implementation. In *International Conference on Logic Programming*, 407–424.

Calimeri, F., Cozza, S., Ianni, G. and Leone, N. 2010. Enhancing asp by functions: Decidable classes and implementation techniques. In *AAAI Conference on Artificial Intelligence.*

Codish, M., Lagoon, V. and Stuckey, P. J. 2005. Testing for termination with monotonicity constraints. In *International Conference on Logic Programming*, 326–340.

De Schreye, D. and Decorte, S. 1994. Termination of logic programs: The never-ending story. *Journal of Logic Programming 19/20*, 199–260.

Deutsch, A., Nash, A. and Remmel, J. B. 2008. The chase revisited. In *Symposium on Principles of Database Systems*, 149–158.

Endrullis, J., Waldmann, J. and Zantema, H. 2008. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning 40,* 2-3, 195–220.

Fagin, R., Kolaitis, P. G., Miller, R. J. and Popa, L. 2005. Data ex- change: Semantics and query answering. *TCS 336,* 1, 89–124.

Ferreira, M. C. F. and Zantema, H. 1996. Total termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing 7,* 2, 133–162.

Gebser, M., Schaub, T. and Thiele, S. 2007. Gringo : A new grounder for answer set programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 266–271.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *International Joint Conference and Symposium on Logic Programming*, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9,* 3/4, 365–386.

Grau, B. C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B. and Wang, Z. 2012. Acyclicity conditions and their application to query answering in description logics. In *KR*.

Greco, S., Molinaro, C. and Spezzano, F. 2012a. *Incomplete Data and Data Dependencies in Relational Databases*, Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

Greco, S., Molinaro, C. and Trubitsyna, I. 2013. Bounded programs: A new decidable class of logic programs with function symbols. In *International Joint Conference on Artificial Intelligence* (to appear).

Greco, S. and Spezzano, F. 2010. Chase termination: A constraints rewriting approach. *Proceeding of the Very Large Data Base Conference 3,* 1, 93–104.

Greco, S., Spezzano, F. and Trubitsyna, I. 2011. Stratification criteria and rewriting techniques for checking chase termination. *Proceeding of the Very Large Data Base Conference 4,* 11, 1158–1168.

Greco, S., Spezzano, F. and Trubitsyna, I. 2012b. On the termination of logic programs with function symbols. In *International Conference on Logic Programming (Technical Communications)*, 323–333.

Krötzsch, M. and Rudolph, S. 2011. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI*, 963–968.

Lierler, Y. and Lifschitz, V. 2009. One more decidable class of finitely ground programs. In *International Conference on Logic Programming*, 489–493.

Marchiori, M. 1996. Proving existential termination of normal logic programs. In *Algebraic Methodology and Software Technology*, 375–390.

Marnette, B. 2009. Generalized schema-mappings: from termination to tractability. In *Symposium on Principles of Database Systems*, 13–22.

Meier, M., Schmidt, M. and Lausen, G. 2009. On chase termination beyond stratification. *Proceeding of the Very Large Data Base Conference 2,* 1, 970–981.

Nguyen, M. T., Giesl, J., Schneider-Kamp, P. and De Schreye, D. 2007. Termination analysis of logic programs based on dependency graphs. In *International Symposium on Logic-based Program Synthesis and Transformation*, 8–22.

Nishida, N. and Vidal, G. 2010. Termination of narrowing via termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing 21,* 3, 177–225.

Ohlebusch, E. 2001. Termination of logic programs: Transformational methods revisited. *Applicable Algebra in Engineering, Communication and Computing 12,* 1/2, 73–116.

Schneider-Kamp, P., Giesl, J. and Nguyen, M. T. 2009a. The dependency triple framework for termination of logic programs. In *International Symposium on Logic-based Program Synthesis and Transformation*, 37–51.

Schneider-Kamp, P., Giesl, J., Serebrenik, A. and Thiemann, R. 2009b. Automated termination proofs for logic programs by term rewriting. *ACM Transactions on Computational Logic 11,* 1.

Schneider-Kamp, P., Giesl, J., Ströder, T., Serebrenik, A. and Thiemann, R. 2010. Automated termination analysis for logic programs with cut. *Theory and Practice of Logic Programming 10,* 4-6, 365–381.

Serebrenik, A. and De Schreye, D. 2005. On termination of meta-programs. *Theory and Practice of Logic Programming 5,* 3, 355–390.

Sternagel, C. and Middeldorp, A. 2008. Root-labeling. In *Rewriting Techniques and Applications*, 336–350.

Syrjänen, T. 2001. Omega-restricted logic programs. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, 267–279.

Voets, D. and De Schreye, D. 2011. Non-termination analysis of logic programs with integer arithmetics. *Theory and Practice of Logic Programming 11,* 4-5, 521–536.

Zantema, H. 1994. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation 17,* 1, 23–50.

Zantema, H. 1995. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae 24,* 1/2, 89–105.