Towards decidable classes of logic programs with function symbols

Marco Calautti, Sergio Greco, Cristian Molinaro, Irina Trubitsyna

DIMES, Università della Calabria
87036 Rende (CS), Italy
{calautti,greco,cmolinaro,trubitsyna}@dimes.unical.it

Abstract

Function symbols are widely acknowledged as an important feature in logic programming, but unfortunately, common inference tasks become undecidable in their presence. To cope with this issue, recent research has focused on identifying decidable classes of programs allowing only a restricted use of function symbols while ensuring decidability of common inference tasks. In this paper, we give an overview of current *termination criteria*. We also present a technique which can be used in conjunction with current termination criteria to enlarge the class of programs recognized as terminating.

1998 ACM Subject Classification F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages

Keywords and phrases Logic programming with function symbols, bottom-up evaluation, stable model semantics

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction and Preliminaries

In recent years, there has been a great deal of interest in enhancing answer set solvers by supporting function symbols. Function symbols often make modeling easier and the resulting encodings more readable and concise, but unfortunately, common inference tasks become undecidable in their presence. The class of *finitely ground programs*, proposed in [1], guarantees decidability of common inference tasks. In particular, finitely ground programs are terminating, i.e. the bottom-up evaluation of these programs gives a finite number of finite stable models. Since membership in the class is semi-decidable, research has focused on identifying sufficient conditions for a program to be finitely ground, leading to different criteria, called *termination criteria*. Efforts in this direction are *ω-restricted programs* [9], *λ-restricted programs* [2], *finite domain programs* [1], *argument-restricted programs* [8], *safe programs* [7], *Γ-acyclic programs* [7], and *bounded programs* [5]. In this paper, we give an overview of recent research on this topic. Specifically, we present some recently proposed decidable termination criteria, able to recognize the termination of disjunctive logic programs, and an orthogonal technique that can be used in conjunction with them to enlarge the class of programs recognized as finitely-ground [6].

We assume the reader is familiar with logic programs with function symbols under the *stable model semantics* [3] (see [5] for a brief overview). Given a program $\mathcal{P}$ we denote by $arg(\mathcal{P})$ the set of all arguments of $\mathcal{P}$, i.e., expressions of the form $p[i]$ where p is a predicate symbol of arity n appearing in $\mathcal{P}$ and $1 \leq i \leq n$. We use $body(r)$ and $head(r)$ to denote the body and the head of a rule r in $\mathcal{P}$; $body^+(r)$ denotes the conjunction of all positive literals in $body(r)$. For any rule r, $ground(r)$ denotes the set of rules obtained by replacing

variables with ground terms which can be constructed using constants and function symbols occurring in $\mathcal{P}$. An argument $q[i] \in arg(\mathcal{P})$ is said to be *limited* if it takes values from a finite domain, that is, if for every (stable) model M of $\mathcal{P}$ the projection of Q over the i-th arguments is a finite set, where Q is the set of q-atoms in M. We consider programs where rules are range restricted, that is all variables occurring in a rule r also occur in $body^+(r)$ and distinguish *base predicate symbols*, defined only by facts (i.e., ground rules with empty body) from *derived predicate symbols*, defined by arbitrary rules. For ease of presentation, we sometimes consider only positive programs as the techniques described can be easily extended to programs with negative body literals and disjunction in head.

2 Basic Termination Criteria

In this section, we describe the most general "basic" termination criterion proposed in the literature, namely *argument-restricted programs* [8]. We shall not discuss other well-known basic termination criteria, such as *ω-restricted programs* [9], *λ-restricted programs* [2] and *finite domain programs* [1], as they have been generalized by argument-restricted programs. We named the aforementioned termination criteria "basic" as their definition does not rely on other termination criteria.

Termination criteria are used to determine sets of arguments which are limited. In the following we shall use the following notations. Given a program $\mathcal{P}$ and a criterion W, $W(\mathcal{P})$ denotes the set of arguments which are recognized as limited by criterion W, whereas $\mathcal{W}$ denotes the class of programs which are recognized as terminating by W, that is the class of programs such that $arg(\mathcal{P}) = W(\mathcal{P})$.

Argument-restricted programs [8]. The argument-restricted criterion tests the possibility to find for each argument a *finite* upper bound of the depth of terms that may occur in that argument during the program evaluation. This test is based on the notion of *argument ranking function* defined below. For any atom A of the form $p(t_1, ..., t_n)$, A^0 denotes the predicate symbol p, and A^i denotes term t_i, for $1 \leq i \leq n$.

▶ **Definition 1.** An *argument ranking* for a program $\mathcal{P}$ is a partial function ϕ from $arg(\mathcal{P})$ to non-negative integers such that, for every rule r of $\mathcal{P}$, every atom A occurring in the head of r, and every variable X occurring in an argument term A^i, if $\phi(A^0[i])$ is defined, then $body^+(r)$ contains an atom B such that X occurs in an argument term B^j, $\phi(B^0[j])$ is defined, and the following condition is satisfied

$$\phi(A^0[i]) - \phi(B^0[j]) \geq d(X, A^i) - d(X, B^j)$$

where $d(X,t) = 0$ if $t = X$; if $t = f(v_1, \ldots, v_k)$, then $d(X,t) = 1 + \max_{v_l \text{ contains } X} d(X, v_l)$.
The set of *restricted arguments* of $\mathcal{P}$ is $AR(\mathcal{P}) = \{p[i] \mid p[i] \in arg(\mathcal{P}) \wedge \exists \phi \text{ s.t. } \phi(p[i]) \text{ is defined}\}$.
A program $\mathcal{P}$ is said to be *argument restricted* iff $AR(\mathcal{P}) = arg(\mathcal{P})$. □

▶ **Example 2.** Consider the following logic program P_2:

```
p(f(X)) ← q(X).
q(X) ← p(f(X)).
```

The program is recognized to be argument-restricted. In particular, the argument-restricted function ϕ can be defined as follows: $\phi(\mathtt{p[1]}) = 1$ and $\phi(\mathtt{q[1]}) = 0$. □

3 Iterated Termination Criteria

In this section we present recently proposed criteria which, starting from a set of limited arguments defined through the application of a basic criterion, computes a possibly larger set of limited arguments.

Safe programs [7]. The first technique is obtained by introducing a fixpoint function, called *safe function*, which, iteratively, extends a given set of limited arguments. Its definition is based on the notion of activation graph.

The *activation graph* of a program $\mathcal{P}$, denoted $\Omega(\mathcal{P})$, is a directed graph whose nodes are the rules of $\mathcal{P}$, and there is an edge (r_i, r_j) in the graph iff r_i activates r_j, i.e. there exist two ground rules $r'_i \in ground(r_i)$, $r'_j \in ground(r_j)$ and a set of ground atoms D such that *(i)* $D \not\models r'_i$, *(ii)* $D \models r'_j$, and *(iii)* $D \cup head(r'_i) \not\models r'_j$. This intuitively means that if D does not satisfy r'_i, D satisfies r'_j, and $head(r'_i)$ is added to D to satisfy r'_i, this causes r'_j not to be satisfied anymore (and then to be "activated").

▶ **Definition 3.** Given a program $\mathcal{P}$ and a basic termination criterion W, the set of *W-safe arguments* $S\text{-}W(\mathcal{P})$ is computed by first setting $S\text{-}W(\mathcal{P}) = W(\mathcal{P})$ and next iteratively adding each argument $q[k]$ such that for all rules $r \in \mathcal{P}$ where q appears in the head (i) r does not depend on a cycle of $\Omega(\mathcal{P})$, or (ii) for every head atom $q(t_1, ..., t_n)$, every variable X appearing in t_k appears also in some safe argument in $body^+(r)$. A program $\mathcal{P}$ is said to be *W-safe* if $S\text{-}W(\mathcal{P}) = arg(\mathcal{P})$. □

The criterion obtained by combining basic criterion W with the safe function is denoted by $S\text{-}W$.

▶ **Example 4.** The following simple logic program $\mathcal{P}_4$ is not recognized as terminating by any basic termination criteria introduced so far.

$$\mathtt{p(X,X)} \leftarrow \mathtt{base(X)}.$$
$$\mathtt{p(f(X),g(X))} \leftarrow \mathtt{p(X,X)}.$$

However, this program is terminating and $\mathcal{P}_4 \in \mathcal{S}\text{-}\mathcal{W}$, for every basic criterion W, since the activation graph of $\mathcal{P}_4$ does not contain any cycle. □

Bounded Programs [5]. The definition of bounded programs relies on the notion of *labelled argument graph*. This graph, denoted $\mathcal{G}_L(\mathcal{P})$, is derived from the argument graph by labelling edges as follows: for each pair of nodes $p[i], q[j] \in arg(\mathcal{P})$ and for every rule $r \in \mathcal{P}$ such that *(i)* an atom $p(t_1, ..., t_n)$ appears in $head(r)$, *(ii)* an atom $q(u_1, ..., u_m)$ appears in $body^+(r)$, *(iii)* terms t_i and u_j have a common variable X, there is an edge $(q[j], p[i], \langle \alpha, r, h, k \rangle)$, where h and k are natural numbers denoting the positions of $p(t_1, ..., t_n)$ in $head(r)$ and $q(u_1, ..., u_m)$ in $body^+(r)$, respectively[1], whereas $\alpha = \epsilon$ if $t_i = u_j$, $\alpha = f$ if $u_j = X$ and $t_i = f(..., X, ...)$, $\alpha = \bar{f}$ if $u_j = f(..., X, ...)$ and $t_i = X$. For the sake of simplicity, without loss of generality, we assume that if a variable X appears in two terms occurring in the head and body of a rule respectively, then only one of the two terms is a complex term and that the nesting level of complex terms is at most one.

[1] We assume that literals in the head (resp. body) are ordered with the first one being associated with 1, the second one with 2, etc.

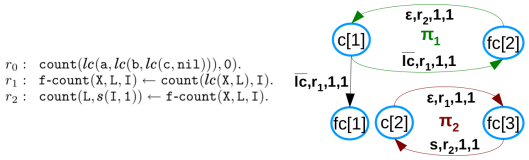

Figure 1 Rewriting of $\mathcal{P}6$ and corresponding labelled argument graph.

Given a path $\rho = (a_1, b_1, \langle \alpha_1, r_1, h_1, k_1 \rangle), \ldots, (a_m, b_m, \langle \alpha_m, r_m, h_m, k_m \rangle)$, we define $\lambda_1(\rho) = \alpha_1 \ldots \alpha_m$, $\lambda_2(\rho) = r_1, ..., r_m$, and $\lambda_3(\rho) = \langle r_1, h_1, k_1 \rangle \ldots \langle r_m, h_m, k_m \rangle$. Given a cycle π consisting of n labelled edges $e_1, ..., e_n$, we can derive n different cyclic paths starting from each of the e_i's—we use $\tau(\pi)$ to denote the set of such cyclic paths.

Given two cycles π_1 and π_2, we write $\pi_1 \approx \pi_2$ iff $\exists \rho_1 \in \tau(\pi_1)$ and $\exists \rho_2 \in \tau(\pi_2)$ such that $\lambda_3(\rho_1) = \lambda_3(\rho_2)$. Given a program $\mathcal{P}$, we say that a cycle π in $\mathcal{G}_L(\mathcal{P})$ is *active* iff $\exists \rho \in \tau(\pi)$ such that $\lambda_2(\rho) = r_1, ..., r_m$ and $(r_1, r_2), ..., (r_{m-1}, r_m), (r_m, r_1)$ is a cyclic path in the activation graph $\Omega(\mathcal{P})$.

Given a program $\mathcal{P}$ and a path ρ in $\mathcal{G}_L(\mathcal{P})$, we denote with $\hat{\lambda}_1(\rho)$ the string obtained from $\lambda_1(\rho)$ by iteratively eliminating pairs of the form $\gamma\bar{\gamma}$ from the string until the resulting string cannot be further reduced.

Given a program $\mathcal{P}$, a cycle π in $\mathcal{G}_L(\mathcal{P})$ can be classified as follows. We say that π is i) *balanced* if $\exists \rho \in \tau(\pi)$ s.t. $\hat{\lambda}_1(\rho)$ is empty, ii) *growing* if $\exists \rho \in \tau(\pi)$ s.t. $\hat{\lambda}_1(\rho)$ does not contain a symbol of the form $\bar{\gamma}$, iii) *failing* otherwise.

▶ **Definition 5.** *Given a program $\mathcal{P}$ and a basic termination criterion W, the set of* W-bounded arguments *B-$W(\mathcal{P})$ is computed by first setting B-$W(\mathcal{P}) = W(\mathcal{P})$ and next iteratively adding each argument $q[k]$ such that for each basic cycle π in $\mathcal{G}_L(\mathcal{P})$ on which $q[k]$ depends, at least one of the following conditions holds:*

1. *π is not active or is not growing;*
2. *π contains an edge $(s[j], p[i], \langle f, r, l_1, l_2 \rangle)$ and, letting $p(t_1, ..., t_n)$ be the l_1-th atom in the head of r, for every variable X in t_i, there is an atom $b(u_1, ..., u_m)$ in body$^+(r)$ s.t. X appears in a term u_h and $b[h]$ is W-bounded;*
3. *there is a basic cycle π' in $\mathcal{G}_L(\mathcal{P})$ s.t. $\pi' \approx \pi$, π' is not balanced, and π' passes only through W-bounded arguments.*

A program $\mathcal{P}$ is said to be W-bounded *if B-$W(\mathcal{P}) = arg(\mathcal{P})$.* □

The criterion obtained by combining basic criterion W with the bounded function is denoted by B-W. The class of W-bounded programs is denoted by $\mathcal{BW}$. A relevant aspect that distinguishes this technique from other works is that this technique analyzes how groups of arguments are each other related—this is illustrated in the following example.

▶ **Example 6.** Consider the following logic program $\mathcal{P}_6$:

r_0 : count([a, b, c], 0).
r_1 : count(L, I + 1) ← count([X|L], I).

The bottom-up evaluation of $\mathcal{P}_6$ terminates yielding the set of atoms count([a, b, c], 0), count([b, c], 1), count([c], 2), and count([ ], 3). The query goal count([ ], L) can be used to retrieve the length L of list [a, b, c].[2] □

[2] Notice that $\mathcal{P}_6$ has been written so as to count the number of elements in a list when evaluated in a

To comply with the syntactic restrictions required by the bounded technique, Figure 1 shows a rewriting of $\mathcal{P}_6$ and the corresponding labelled argument graph. where lc and s denote the list constructor and the sum operators respectively. Basically, considering the argument-restricted technique as the basic criterion W, after having established that argument $\texttt{count}[1]$ is limited, that is $\texttt{count}[1] \in B\text{-}AR(\mathcal{P}_6)$, by analyzing the two cycles involving arguments $\texttt{count}[1]$ and $\texttt{count}[2]$, respectively and using Condition 3 of Definition 5 it is possible to detect that also argument $\texttt{count}[2]$ is limited, that is $\texttt{count}[2] \in B\text{-}AR(\mathcal{P}_6)$. Consequently, $\mathcal{P}_6$ is AR-bounded.

4 Rewriting technique

In this section we present a rewriting technique [6] that, used in conjunction with current termination criteria, allows us to detect more programs as finitely-ground. This technique takes a logic program $\mathcal{P}$ and transforms it into an adorned program $\mathcal{P}^\mu$ with the aim of applying termination criteria to $\mathcal{P}^\mu$ rather than $\mathcal{P}$. The transformation is sound in that if the adorned program satisfies a certain termination criterion, then the original program satisfies this criterion as well and, consequently, is finitely-ground. Importantly, as showed by the below example, applying termination criteria to adorned programs rather than the original ones strictly enlarges the class of programs recognized as finitely-ground. This technique is much more general than those used to deal with chase termination (see [4]).

▶ **Example 7.** Consider the following program $\mathcal{P}_7$, where $\texttt{base}$ is a base predicate symbol defined by facts not showed here.

$$r_0: \quad \texttt{p}(\texttt{X}, \texttt{f}(\texttt{X})) \leftarrow \texttt{base}(\texttt{X}).$$
$$r_1: \quad \texttt{p}(\texttt{X}, \texttt{f}(\texttt{X})) \leftarrow \texttt{p}(\texttt{Y}, \texttt{X}),\ \texttt{base}(\texttt{Y}).$$
$$r_2: \quad \texttt{p}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{p}(\texttt{f}(\texttt{X}), \texttt{f}(\texttt{Y})).$$

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

$$\rho_0: \quad \texttt{p}^{\epsilon\texttt{f}_1}(\texttt{X}, \texttt{f}(\texttt{X})) \leftarrow \texttt{base}^\epsilon(\texttt{X}).$$

from which we derive the new adorned predicate symbol $\texttt{p}^{\epsilon\texttt{f}_1}$, and the adornment definition $\texttt{f}_1 = \texttt{f}(\epsilon)$. Next, $\texttt{p}^{\epsilon\texttt{f}_1}$ and $\texttt{base}^\epsilon$ are used to adorn the body of r_1 so as to get

$$\rho_1: \quad \texttt{p}^{\texttt{f}_1\texttt{f}_2}(\texttt{X}, \texttt{f}(\texttt{X})) \leftarrow \texttt{p}^{\epsilon\texttt{f}_1}(\texttt{Y}, \texttt{X}), \texttt{base}^\epsilon(\texttt{Y})$$

from which we derive the new adorned predicate symbol $\texttt{p}^{\texttt{f}_1\texttt{f}_2}$, and the adornment definition $\texttt{f}_2 = \texttt{f}(\texttt{f}_1)$. Intuitively, the body of ρ_1 is coherently adorned because $\texttt{Y}$ is always associated with the same adornment symbol ϵ. Using the new adorned predicate symbol $\texttt{p}^{\texttt{f}_1\texttt{f}_2}$, we can adorn rule r_2 and get

$$\rho_2: \quad \texttt{p}^{\epsilon\texttt{f}_1}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{p}^{\texttt{f}_1\texttt{f}_2}(\texttt{f}(\texttt{X}), \texttt{f}(\texttt{Y})).$$

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, $\texttt{p}^{\texttt{f}_1\texttt{f}_2}(\texttt{Y}, \texttt{X}), \texttt{base}^\epsilon(\texttt{Y})$ is not coherently adorned because the same variable $\texttt{Y}$ is associated

bottom-up fashion, and therefore differs from the classical formulation relying on a top-down evaluation strategy. However, programs relying on a top-down evaluation strategy can be rewritten into programs whose bottom-up evaluation gives the same result.

with both $\mathtt{f}_1$ and ϵ; moreover, $\mathtt{p}^{\epsilon\mathtt{f}_1}(\mathtt{f}(\mathtt{X}), \mathtt{f}(\mathtt{Y}))$ is not coherently adorned because $\mathtt{f}(\mathtt{X})$ does not comply with the (simple) term structure described by ϵ.

To determine termination of the bottom-up evaluation of $\mathcal{P}_7$, we can apply current termination criteria to $\mathcal{P}_7^{\mu} = \{\rho_0, \rho_1, \rho_2\}$ rather than $\mathcal{P}_7$. □

It is worth noting that the rewriting technique ensures that if $\mathcal{P}_7^{\mu}$ is recognized as terminating, so is $\mathcal{P}_7$. Notice also that both $\mathcal{P}_7$ and $\mathcal{P}_7^{\mu}$ are recursive, but while some termination criteria (e.g., the argument-restricted and Γ-acyclicity criteria) detect $\mathcal{P}_7^{\mu}$ as terminating, none of the current termination criteria is able to realize that $\mathcal{P}_7$ terminates.

References

1 Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable functions in ASP: Theory and implementation. In *ICLP*, pages 407–424, 2008.

2 Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In *LPNMR*, pages 266–271, 2007.

3 Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.

4 Sergio Greco, Cristian Molinaro, and Francesca Spezzano. *Incomplete Data and Data Dependencies in Relational Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

5 Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Bounded programs: A new decidable class of logic programs with function symbols. In *IJCAI*, 2013.

6 Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Logic programming with function symbols: Checking termination of bottom-up evaluation through program adornments. In *ICLP (to appear in TPLP journal)*, 2013.

7 Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. On the termination of logic programs with function symbols. In *ICLP (Technical Communications)*, pages 323–333, 2012.

8 Yuliya Lierler and Vladimir Lifschitz. One more decidable class of finitely ground programs. In *ICLP*, pages 489–493, 2009.

9 Tommi Syrjanen. Omega-restricted logic programs. In *LPNMR*, pages 267–279, 2001.