TLP **11** (1): 33–63, 2011. © Cambridge University Press 2010
doi:10.1017/S1471068410000025 First published online 25 February 2010

Polytool: Polynomial interpretations as a basis for termination analysis of logic programs

MANH THANG NGUYEN* and DANNY DE SCHREYE
Department of Computer Science, K. U. Leuven
Celestijnenlaan 200A, B-3001, Heverlee, Belgium
(*e-mail:* Danny.DeSchreye@cs.kuleuven.be)

JÜRGEN GIESL
LuFG Informatik II, RWTH Aachen,
Ahornstr. 55, 52074 Aachen, Germany
(*e-mail:* giesl@informatik.rwth-aachen.de)

PETER SCHNEIDER-KAMP
Department of Mathematics and Computer Science, University of Southern Denmark,
Campusvej 55, DK-5230 Odense M, Denmark
(*e-mail:* petersk@imada.sdu.dk)

submitted 18 February 2008; revised 30 November 2009; accepted 11 December 2009

Abstract

Our goal is to study the feasibility of porting termination analysis techniques developed for one programming paradigm to another paradigm. In this paper, we show how to adapt termination analysis techniques based on polynomial interpretations—very well known in the context of term rewrite systems—to obtain new (nontransformational) termination analysis techniques for definite logic programs (LPs). This leads to an approach that can be seen as a direct generalization of the traditional techniques in termination analysis of LPs, where linear norms and level mappings are used. Our extension generalizes these to arbitrary polynomials. We extend a number of standard concepts and results on termination analysis to the context of polynomial interpretations. We also propose a constraint-based approach for automatically generating polynomial interpretations that satisfy the termination conditions. Based on this approach, we implemented a new tool, called Polytool, for automatic termination analysis of LPs.

KEYWORDS: termination analysis, acceptability, polynomial interpretations

1 Introduction

Termination analysis plays an important role in the study of program correctness. A termination proof is mostly based on a mapping from computational states to some well-founded ordered set. Termination is guaranteed if the mapped values of

* Deceased on June 3, 2009

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

the encountered states during a computation, under this mapping, decrease w.r.t. the order.

For logic programs (LPs), termination analysis is done by mapping terms and atoms to a well-founded set of natural numbers by means of norms and level mappings. Proving termination is based on the search for a suitable norm and level mapping such that the resulting predicate calls decrease under the mapping.

Until now, most termination techniques for LPs are based on the use of linear norms and linear level mappings, which measure the size of each term or atom as a linear combination of the sizes of its subterms. For example, the Hasta-La-Vista system (Serebrenik and De Schreye 2003) infers one specific linear norm and linear level mapping. In the context of numerical computations, it includes a refinement on this, based on a case analysis. The tool cTI (Mesnard and Bagnara 2005) uses a concrete linear norm. The analyzers TermiLog (Lindenstrauss and Sagiv 1997; Lindenstrauss 2000) and TerminWeb (Codish and Taboch 1999; Taboch *et al.* 2002) use a combination of several linear norms to obtain an approximation of the program and then infer linear level mappings for termination analysis of the approximated program. However, the restriction to linear norms and level mappings limits the power of termination analysis considerably. To illustrate this point, consider the following example, *der*, that formulates rules for computing the repeated derivative of a function in some variable *u*. This example from Dershowitz *et al.* (1997) and De Schreye and Serebrenik (2002) is inspired by a similar term rewriting example from Dershowitz (1995).

Example 1 (*der*)

$$d(der(u), 1), \tag{1}$$

$$d(der(X + Y), DX + DY) \text{ :- } d(der(X), DX), d(der(Y), DY), \tag{2}$$

$$d(der(X * Y), X * DY + Y * DX) \text{ :- } d(der(X), DX), d(der(Y), DY), \tag{3}$$

$$d(der(der(X)), DDX) \text{ :- } d(der(X), DX), d(der(DX), DDX). \tag{4}$$

We are interested in proving termination of this program with respect to the set of queries $S = \{ d(t_1, t_2) \mid t_1$ is a ground term and t_2 is an arbitrary term$\}$. So the set of queries is specified by a *mode* that considers the first argument of d as an input argument and the second as an output argument.

As shown in Dershowitz *et al.* (1997) and Nguyen and De Schreye (2005), the termination proof is impossible when using a linear norm and a linear level mapping. Indeed, it turns out that all existing nontransformational termination analyzers for LPs mentioned above fail to prove termination of this example. □

In this paper, we propose a general framework for termination proofs of LPs based on polynomial interpretations. Using polynomial interpretations as a basis for ordering terms in TRSs was first introduced by Lankford in 1979. It is currently one of the best known and most widely used techniques in term rewrite systems (TRS) termination analysis.

We develop the approach within the LP context. Classical approaches in LP termination use interpretations that map to natural numbers (using linear polynomial

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

functions). In contrast, we will use interpretations that map to polynomials (using arbitrary polynomial functions). To adapt the classical LP approaches to polynomial interpretations, we use the concepts of "abstract norm" and "abstract level mapping" (Verschaetse and De Schreye 1991). We show that with our new approach, one can also prove termination of programs like Example 1.

We also developed an automated tool (Polytool) for termination analysis based on our approach (Nguyen and De Schreye 2007). We embedded this within the constraint-based approach developed in Decorte *et al.* (1999) and combined it with the nonlinear Diophantine constraint solver developed by Fuhs *et al.* (2007) (implemented in the AProVE system Giesl *et al.* 2006a) to provide a completely automated system.

The paper is organized as follows. In the next section, we present some preliminaries. In Section 3, we introduce the notion of polynomial interpretations in logic programming and show how this approach can be used to prove termination. In Section 4, we discuss the automation of the approach. In Section 5, we provide and discuss the results of our experimental evaluation. We end with a conclusion in Section 6.

2 Preliminaries

After introducing the basic terminology of LPs in Section 2.1, we recapitulate the concepts of *norms* and *level mappings* in Section 2.2 and explain their use for termination proofs in Section 2.3.

2.1 Notations and terminology

We assume familiarity with LP concepts and with the main results of logic programming (Lloyd 1987; Apt 1990). In the following, P denotes a definite logic program. We use Var_P, Fun_P, and $Pred_P$ to denote the sets of variables, function, and predicate symbols of P. Given an atom A, $rel(A)$ denotes the predicate occurring in A. Let p, q be predicates occurring in the program P. We say that p *refers to* q if there is a clause in P such that p is in its head and q is in its body. We say that p *depends on* q if (p, q) is in the transitive closure of the relation "refers to". If p depends on q and vice versa, p and q are called *mutually recursive*, denoted by $p \simeq q$. A clause in P with a predicate p in its head and a predicate q in its body, such that p and q are mutually recursive, is called a *(mutually) recursive clause*. Within such a recursive clause, the body-atoms with predicate symbol q are called *(mutually) recursive atoms*. Let $Term_P$ and $Atom_P$ denote, respectively, the sets of all terms and atoms that can be constructed from P.

In this paper, we focus our attention on definite logic programs and linear resolution for definite horn clauses using a selection function (SLD) derivations where the left-to-right selection rule is used. Such derivations are referred to as linear resolution for definite horn clauses (LD) derivations; the corresponding derivation tree is called *LD-tree*. We say that a query Q *LD-terminates* for a program P, if the LD-tree for (P, Q) is finite (left-termination; Lloyd 1987). In the following, we usually speak of "termination" instead of "LD-termination" or "left-termination".

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

2.2 Norms and level mappings

The concepts of *norm* and *level mapping* are central in termination analysis of logic programs.

Definition 1 (*norm, level mapping*)
A *norm* is a mapping $\|.\| : Term_P \to \mathbb{N}$. A *level-mapping* is a mapping $|.| : Atom_P \to \mathbb{N}$.

Several examples of norms can be found in the literature (Bossi et al. 1991). One of the most commonly used norms is the *list-length norm* $\|.\|_\ell$ which maps lists to their lengths and any other term to 0. Another frequently used norm is the *term-size* norm $\|.\|_\tau$ which counts the number of function symbols in a term. Both of them belong to a class of norms called linear norms which is defined as follows.

Definition 2 (*linear norm and level mapping; Serebrenik 2003*)
A norm $\|.\|$ is a *linear* norm if it is recursively defined by means of the following schema:

- $\|X\| = 0$ for any variable X,
- $\|f(t_1,\ldots,t_n)\| = f_0 + \sum_{i=1}^{n} f_i\|t_i\|$ where $f_i \in \mathbb{N}$ and $n \geqslant 0$.

Similarly, a level mapping $|.|$ is a *linear* level mapping if it is defined by means of the following schema:

- $|p(t_1,\ldots,t_n)| = p_0 + \sum_{i=1}^{n} p_i\|t_i\|$ where $p_i \in \mathbb{N}$ and $n \geqslant 0$.

2.3 Conditions for termination w.r.t. general orders

A *quasi-order* on a set S is a reflexive and transitive binary relation $\gtrsim$ defined on elements of S. We define the *associated equivalence relation* $\approx$ as $s \approx t$ if and only if $s \gtrsim t$ and $t \gtrsim s$. A *well-founded order* on S is a transitive relation $\succ$ where there is no infinite sequence $s_0 \succ s_1 \succ \ldots$ with $s_i \in S$. A *reduction pair* $(\gtrsim, \succ)$ consists of a quasi-order $\gtrsim$ and a well-founded order $\succ$ that are *compatible* (i.e., $t_1 \gtrsim t_2 \succ t_3$ implies $t_1 \succ t_3$). We also need the following notion of a call set.

Definition 3 (*call set*)
Let P be a program and S be a set of atomic queries. The *call set*, $Call(P,S)$, is the set of all atoms A, such that a variant of A is the selected atom in some derivation for (P,Q), for some $Q \in S$.

Most often, one regards infinite sets S of queries. For instance, this is the case in Example 1. As in Example 1, S is then specified in terms of modes or types. As a consequence, in an automated approach, a safe over-approximation of $Call(P,S)$ needs to be computed, using a mode or a type inference technique (e.g., Janssens and Bruynooghe 1992; Heaton *et al.* 2000; Bruynooghe *et al.* 2005; Gallagher *et al.* 2005).

In order to obtain a termination criterion that is suitable for automation, one usually estimates the effect of the atoms in the bodies of clauses by suitable *interargument relations*. This notion can be defined for arbitrary reduction pairs.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Definition 4 (*interargument relation De Schreye and Serebrenik 2002*)
Let P be a program, p be a predicate in P, and $(\gtrsim, \succ)$ be a reduction pair on $Term_P$. An *interargument relation* for p in P w.r.t. $(\gtrsim, \succ)$ is a relation R_p with the same arity as p: $R_p = \{p(t_1, \ldots, t_n) \mid t_i \in Term_P$ for all $1 \leqslant i \leqslant n$, and $\varphi_p(t_1, \ldots, t_n)\}$, where:

- $\varphi_p(t_1, \ldots, t_n)$ is a boolean expression (in terms of disjunction, conjunction, and negation) of inequalities $s \gtrsim s'$ or $s \succ s'$, in which
- s, s' are constructed from $t_1, \ldots, t_n$ by applying function symbols from Fun_P.

R_p is a *valid* interargument relation for p in P w.r.t. $(\gtrsim, \succ)$ if and only if for every $p(t_1, \ldots, t_n) \in Atom_P$: $P \models p(t_1, \ldots, t_n)$ implies $p(t_1, \ldots, t_n) \in R_p$.

Example 2 (*interargument relation*)
Let P be the standard *append* program that computes list concatenation. Then there are a number of valid interargument relations. Consider the reduction pair $(\gtrsim, \succ)$ corresponding to the list-length norm $\|.\|_\ell$, i.e., $t_1 \gtrsim t_2$ if and only if $\|t_1\|_\ell \geqslant \|t_2\|_\ell$ and $t_1 \succ t_2$ if and only if $\|t_1\|_\ell > \|t_2\|_\ell$. For instance, valid interargument relations for *append* w.r.t. $(\gtrsim, \succ)$ are $R_{append} = \{append(t_1, t_2, t_3) \mid t_1, t_2, t_3 \in Term_P \wedge \varphi_{append}(t_1, t_2, t_3)\}$, where $\varphi_{append}(t_1, t_2, t_3)$ could be

- $t_3 \gtrsim t_2 \wedge t_3 \gtrsim t_1$,
- $t_3 \gtrsim t_2$,
- $[t_1, t_2 | t_3] \succ [t_2 | t_3]$, or
- *true*.

Of course, usually only the first two interargument relations are useful for termination analysis. □

Finally, we need the notion of *rigidity*, in order to deal with bindings that are due to unification in LD-derivations. These bindings would have to be back-propagated to the variables in the initial goal. We reformulate rigidity for arbitrary reduction pairs.

Definition 5 (*rigidity – adapted from De Schreye and Serebrenik 2002*)
A term or atom $A \in Term_P \cup Atom_P$ is called *rigid* w.r.t. a reduction pair $(\gtrsim, \succ)$ if $A \approx A\sigma$ holds for any substitution σ. A set of terms (or atoms) S is called *rigid* w.r.t. $(\gtrsim, \succ)$ if all its elements are rigid w.r.t. $(\gtrsim, \succ)$.

Example 3 (*rigidity*)
The list $[X|t]$ (X is a variable, t is a ground term) is rigid w.r.t. the reduction pair $(\gtrsim, \succ)$ corresponding to the list-length norm. For any substitution σ, we have $\|[X|t]\sigma\|_\ell = 1 + \|t\|_\ell = \|[X|t]\|_\ell$. Therefore, $[X|t]\sigma \approx [X|t]$ w.r.t. $(\gtrsim, \succ)$.

However, the list $[X|t]$ is not rigid w.r.t. the reduction pair $(\gtrsim', \succ')$ corresponding to the term-size norm $\|.\|_\tau$, i.e., $t_1 \gtrsim' t_2$ if and only if $\|t_1\|_\tau \geqslant \|t_2\|_\tau$ and $t_1 \succ' t_2$ if and only if $\|t_1\|_\tau > \|t_2\|_\tau$. □

The following definition introduces the desired termination criterion, i.e., it recalls the definition of *rigid order-acceptability* w.r.t. a set of atoms.

Definition 6 (*rigid order-acceptability De Schreye and Serebrenik 2002*)
Let S be a set of atomic queries. A program P is *rigid order-acceptable* w.r.t. S if there exists a reduction pair $(\gtrsim, \succ)$ on $Atom_P$ where $Call(P, S)$ is rigid w.r.t. $(\gtrsim, \succ)$

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

and where for each predicate p in P, there is a valid interargument relation R_p in P w.r.t. $(\gtrsim, \succ)$ such that

- for any clause $A :\!- B_1, B_2, \ldots, B_n$ in P,
- for any atom $B_i \in \{B_1 \ldots, B_n\}$ such that $rel(B_i) \backsimeq rel(A)$,
- for any substitution θ such that the atoms $B_1\theta, \ldots, B_{i-1}\theta$ are elements of their associated interargument relations $R_{rel(B_1)}, \ldots, R_{rel(B_{i-1})}$:

$$A\theta \succ B_i\theta.$$

Theorem 1 states that *rigid order-acceptability* is a sufficient condition for termination. We refer to Serebrenik (2003), Theorems 3.32, and 3.54 for the proof of Theorem 1.

Theorem 1 (*termination criterion by rigid order-acceptability*)
If P is rigid order-acceptable w.r.t. S, then P terminates for any query in S.

Rigid order-acceptability is sufficient for termination, but is not necessary for it (see De Schreye and Serebrenik 2002). With Definition 6 and Theorem 1, proving termination of a program requires verifying the rigidity of the call set, verifying the validity of interargument relations for predicates, and verifying the decrease conditions for the (mutually) recursive clauses.

We will not discuss here the decidability or undecidability results related to various problems concerning: (i) the rigidity of the call set and (ii) the validity of interargument relations. The interested reader may refer to the relevant literature.

In the remainder of this paper we provide some answers to the question in the setting of a given set S, an inferred order based on polynomial interpretations, abstractions of S based on types, type inference to approximate the call set, and interargument relations based on inequalities between polynomials.

3 Polynomial interpretation of a logic program

The approach presented in the previous section can be considered a theoretical framework for termination analysis of LPs based on general orders on terms and atoms. In this section, we specialize it to orders based on polynomial interpretations.

We first introduce polynomial interpretations in Section 3.1. Then in Section 3.2 we reformulate the termination conditions for LPs from Section 2.3 for polynomial interpretations.

3.1 Polynomial interpretations

In this paper, we only consider polynomials with natural numbers as coefficients (so-called natural coefficients). Because natural numbers will occur many times in this paper, we will simply refer to them as "numbers".

We say that a variable X occurs in a polynomial p if the polynomial contains a monomial with a coefficient different from 0 and X occurs in this monomial. If $X_1, \ldots, X_n$ are all the variables occurring in a polynomial p, we often denote

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

p as $p(X_1,\ldots,X_n)$. For every polynomial p, there is an associated polynomial function $F_p = \lambda X_1,\ldots,X_n.\ p(X_1,\ldots,X_n)$. For numbers or polynomials $x_1,\ldots,x_n$, we often write "$p(x_1,\ldots,x_n)$" instead of "$F_p(x_1,\ldots,x_n)$." Given $p(X_1,\ldots,X_n)$ and $m \geqslant 1$ we also have an associated polynomial function $F_{p,m} = \lambda X_1,\ldots,X_n, Y_1,\ldots,Y_m.\ p(X_1,\ldots,X_n)$. For such an associated function on an extended domain, we often write "$p(x_1,\ldots,x_n,y_1,\ldots,y_m)$" to denote "$F_{p,m}(x_1,\ldots,x_n,y_1,\ldots,y_m)$."

Definition 7 (*orders on polynomials*)
Let p and q be two polynomials. Let $X_1,\ldots,X_n$ be all variables occurring in p or q. The quasi-order $\gtrsim_{\mathbb{N}}$ is defined as $p \gtrsim_{\mathbb{N}} q$ if and only if $p(x_1,\ldots,x_n) \geqslant q(x_1,\ldots,x_n)$ for all $x_1,\ldots,x_n \in \mathbb{N}$. The strict order $\succ_{\mathbb{N}}$ is defined as $p \succ_{\mathbb{N}} q$ if and only if $p(x_1,\ldots,x_n) > q(x_1,\ldots,x_n)$ for all $x_1,\ldots,x_n \in \mathbb{N}$.

Observe that $(\gtrsim_{\mathbb{N}}, \succ_{\mathbb{N}})$ is a reduction pair. In other words, $\succ_{\mathbb{N}}$ is well founded and transitive, $\gtrsim_{\mathbb{N}}$ is reflexive and transitive, and $\gtrsim_{\mathbb{N}}$ and $\succ_{\mathbb{N}}$ are compatible. Let Σ denote the set of all polynomials with natural coefficients. Note that all these polynomials p are *weakly monotonic*, i.e., $x_i \geqslant y_i$ for all $1 \leqslant i \leqslant n$ implies $p(x_1,\ldots,x_n) \geqslant p(y_1,\ldots,y_n)$.

A *polynomial interpretation* maps each function and each predicate symbol of the program to a polynomial.

Definition 8 (*polynomial interpretation*)
A *polynomial interpretation* I for a logic program P maps each symbol f of arity n in $Fun_P \cup Pred_P$ to a polynomial $p_f(X_1,\ldots,X_n)$.

Every polynomial interpretation induces a norm and a level mapping. Although it is standard in logic programming to distinguish between norms and level mappings, to simplify the formalization, here we will only introduce a level mapping and define it on both terms and atoms.

Definition 9 (*polynomial level mapping*)
The *level mapping* associated with a polynomial interpretation I, is a mapping $|.|_I : Term_P \cup Atom_P \to \Sigma$, which is defined recursively as:

- $|X|_I = X$ if X is a variable,
- $|f(t_1,\ldots,t_n)|_I = p_f(|t_1|_I,\ldots,|t_n|_I)$, where $p_f = I(f)$.

Every polynomial interpretation induces corresponding orders.

Definition 10 (*reduction pair corresponding to polynomial interpretation*)
Let I be a polynomial interpretation. We define the relations $\gtrsim_I$ and $\succ_I$ on $Term_P \cup Atom_P$ as follows:

- $s \gtrsim_I t$ if and only if $|s|_I \gtrsim_{\mathbb{N}} |t|_I$ for any $s,t \in Term_P \cup Atom_P$,
- $s \succ_I t$ if and only if $|s|_I \succ_{\mathbb{N}} |t|_I$ for any $s,t \in Term_P \cup Atom_P$.

Again, observe that the orders induced by a polynomial interpretation form a reduction pair.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Example 4 (*polynomial interpretation for "der"*)
Let I be a polynomial interpretation with

$$\begin{array}{rcccccl} I(+) & = & I(*) & = & p_+(X_1,X_2) & = & p_*(X_1,X_2) = X_1+X_2+2, \\ I(u) & = & I(1) & = & p_u & = & p_1 = 1, \\ & & I(der) & = & p_{der}(X) & & = X^2+2X+2, \\ & & I(d) & = & p_d(X_1,X_2) & & = X_1. \end{array}$$

Then $d(der(X+Y),DX+DY) \succ_I d(der(X),DX)$, since $|d(der(X+Y),DX+DY)|_I = (X+Y+2)^2+2(X+Y+2)+2 \succ_{\mathbb{N}} |d(der(X),DX)|_I = X^2+2X+2$.

3.2 Termination of logic programs by polynomial interpretations

We now restate Definition 6 and Theorem 1 for the special case of polynomial interpretations. So instead of interargument relations for arbitrary orders as in Definition 4, we now use interargument relations w.r.t. *polynomial interpretations.*

Definition 11 (*interargument relation w.r.t. a polynomial interpretation*)
Let P be a program, p be a predicate in P, and I be a polynomial interpretation. R_p is an *interargument relation* for p in P w.r.t. I if and only if R_p is an interargument relation for p in P w.r.t. $(\gtrsim_I, \succ_I)$.

Instead of rigidity w.r.t. general orders as in Definition 5, we define *rigidity w.r.t. polynomial interpretations.*

Definition 12 (*rigidity w.r.t. a polynomial interpretation*)
A term or atom $A \in Term_P \cup Atom_P$ is called *rigid* w.r.t. *a polynomial interpretation* I if and only if A is rigid w.r.t. $(\gtrsim_I, \succ_I)$, i.e., if and only if $A \approx_I A\sigma$ holds for any substitution σ. A set of terms (or atoms) S is called *rigid* w.r.t. I if all its elements are rigid w.r.t. I.

For polynomial interpretations, rigidity can also be characterized in an alternative way using *relevant variables.*

Definition 13 (*relevant variables*)
Let I be a polynomial interpretation and A be a term or atom. A variable X in A is called *relevant* w.r.t. I if there exists a substitution $\{X \to t\}$ of a term t for X, such that $A\{X \to t\} \not\approx_I A$.

Example 5 (*relevant variables*)
Let $A = [X|Y]$ and I be the interpretation corresponding to the list-length norm $\|.\|_\ell$, i.e., $|[H|T]|_I = 1 + |T|_I$. Then the only relevant variable of A is Y. □

Proposition 1 (*alternative characterization of rigidity*)
Let I be a polynomial interpretation and A be a term or atom. Then A is rigid w.r.t. I if and only if A has no relevant variables w.r.t. I.

Proof
Obvious from Definitions 12 and 13. □

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Using the notions of interargument relations and rigidity w.r.t. a polynomial interpretation, we obtain the following specialization of Theorem 1:

Corollary 1 (*termination criterion with polynomial rigid order-acceptability*)
Let S be a set of atomic queries and P be a program. Let I be a polynomial interpretation, where $Call(P,S)$ is rigid w.r.t. I and where for each predicate p in P, there is a valid interargument relation R_p in P w.r.t. I such that

- for any clause $A :\!- B_1, B_2, \ldots, B_n$ in P,
- for any atom $B_i \in \{B_1 \ldots, B_n\}$ such that $rel(B_i) \trianglelefteq rel(A)$,
- for any substitution θ such that the atoms $B_1\theta, \ldots, B_{i-1}\theta$ are elements of their associated interargument relations $R_{rel(B_1)}, \ldots, R_{rel(B_{i-1})}$:

$$A\theta \succ_I B_i\theta.$$

Then P terminates for any query in S.

Proof
The corollary immediately follows from Theorem 1. □

Corollary 1 can be applied to verify termination of a logic program w.r.t. a set of queries. More precisely, we have to check that all conditions in the following termination proof procedure are satisfied by some polynomial interpretation I. In Section 4 we will discuss how to find such an interpretation automatically.

Procedure 1 (*a procedure for automatic termination analysis*)
The termination proof procedure derived from Corollary 1 contains the following three steps:

Step 1: The call set $Call(P,S)$ must be rigid w.r.t. I. In other words, no query A in the call set may have a relevant variable w.r.t. I.

Step 2: For a clause that has body-atoms between the head and a (mutually) recursive body-atom, valid interargument relations of those atoms w.r.t. I need to be inferred.

Step 3: For every clause, the polynomial level mapping of the head w.r.t. I should be larger than that of any (mutually) recursive body-atom, given that interargument relations for intermediate body-atoms hold.

For Step 2, we can follow the standard approach for LPs to verify that a relation R holds for all elements of the Herbrand model (see, e.g., Lloyd 1987). To this end, one has to verify $T_P(R) \subseteq R$, where T_P is the immediate consequence operator corresponding to the program P. Thus, we verify the validity of interargument relations by first checking whether they are correct for the facts in the program. Then for every clause, if the interargument relations hold for all body-atoms, the interargument relation for the head should also hold.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Example 6 (*applying Corollary 1 to the "der"-program*)
Consider again the "*der*"-program from Example 1 and the set of queries $S = \{d(t_1, t_2) \mid t_1 \text{ is a ground term and } t_2 \text{ is an arbitrary term}\}$. Note that here, $Call(P, S) = S$. Let I be the polynomial interpretation from Example 4. Then no $A \in Call(P, S)$ has a relevant variable w.r.t. I. This means that $Call(P, S)$ is rigid w.r.t. I.

Let $R_d = \{d(t_1, t_2) \mid t_1, t_2 \in Term_P, t_1 \succ_I t_2\}$ be an interargument relation for the predicate d. Checking the validity of R_d is equivalent to verifying the correctness of the following conditions for any substitution θ:

$$der(u)\theta \succ_I (1)\theta,$$

$$der(X)\theta \succ_I DX\,\theta \text{ and } der(Y)\theta \succ_I DY\,\theta \text{ implies } der(X + Y)\theta \succ_I (DX + DY)\theta,$$

$$der(X)\theta \succ_I DX\,\theta \text{ and } der(Y)\theta \succ_I DY\,\theta \text{ implies } der(X * Y)\theta \succ_I (X * DY + Y * DX)\theta,$$

$$der(X)\theta \succ_I DX\,\theta \text{ and } der(DX)\theta \succ_I DDX\,\theta \text{ implies } der(der(X))\theta \succ_I DDX\,\theta.$$

To prove termination, we also need the following decrease conditions for any substitution θ:

$$d(der(X + Y), DX + DY)\theta \succ_I d(der(X), DX)\theta,$$

$$d(der(X), DX)\theta \text{ satisfies } R_d \text{ implies } d(der(X + Y), DX + DY)\theta \succ_I d(der(Y), DY)\theta,$$

$$d(der(X * Y), X * DY + Y * DX)\theta \succ_I d(der(X), DX)\theta,$$

$$d(der(X), DX)\theta \text{ satisfies } R_d \text{ implies } d(der(X * Y), X * DY + Y * DX)\theta \succ_I d(der(Y), DY)\theta,$$

$$d(der(der(X)), DDX)\theta \succ_I d(der(X), DX)\theta,$$

$$d(der(X), DX)\theta \text{ satisfies } R_d \text{ implies } d(der(der(X)), DDX)\theta \succ_D d(der(DX), DDX)\theta.$$

The conditions above are equivalent to the following inequalities on the variables X, Y, DX, DY, DDX. For the conditions on the valid interargument relation, we obtain

$$5 > 1$$

$$\forall X, Y, DX, DY \in \mathbb{N}\ X^2 + 2X + 2 > DX \wedge Y^2 + 2Y + 2 > DY \Rightarrow (X + Y + 2)^2 + 2(X + Y + 2) + 2 > DX + DY + 2,$$

$$\forall X, Y, DX, DY \in \mathbb{N}\ X^2 + 2X + 2 > DX \wedge Y^2 + 2Y + 2 > DY \Rightarrow (X + Y + 2)^2 + 2(X + Y + 2) + 2 > x + DY + Y + DX + 3,$$

$$\forall X, DX, DDX \in \mathbb{N}\ X^2 + 2X + 2 > DX \wedge DX^2 + 2DX + 2 > DDX \Rightarrow (X^2 + 2X + 2)^2 + 2(X^2 + 2X + 2) + 2 > DDX.$$

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

And for the decrease conditions we obtain

$$\begin{array}{rcl}
\forall X, Y \in \mathbb{N}\ (X+Y+2)^2+2(X+Y+2)+2 & > & X^2+2X+2,\\
\forall X, Y, DX \in \mathbb{N}\ X^2+2X+2 > DX \Rightarrow (X+Y+2)^2+2(X+Y+2)+2 & > & Y^2+2Y+2,\\
\forall X, Y \in \mathbb{N}\ (X+Y+2)^2+2(X+Y+2)+2 & > & X^2+2X+2,\\
\forall X, Y, DX \in \mathbb{N}\ X^2+2X+2 > DX \Rightarrow (X+Y+2)^2+2(X+Y+2)+2 & > & Y^2+2Y+2,\\
\forall X \in \mathbb{N}\ (X^2+2X+2)^2+2(X^2+2X+2)+2 & > & X^2+2X+2,\\
\forall X, DX \in \mathbb{N}\ X^2+2X+2 > DX \Rightarrow (X^2+2X+2)^2+2(X^2+2X+2)+2 & > & DX^2+2DX+2.
\end{array}$$

The above inequalities are easily verified for all instantiations of the variables by numbers. Hence, the program terminates w.r.t. the set of queries S. □

4 Automating the termination proof

A key question is how to automate the search for a polynomial interpretation and for interargument relations. In other words, to prove termination of a logic program, one has to synthesize the coefficients of the polynomials associated with the function and predicate symbols as well as the formulas $\varphi_p(t_1, \ldots, t_n)$ defining the interargument relations. In the philosophy of the constraint-based approach in Decorte *et al.* (1999), we do not choose a particular polynomial interpretation and particular interargument relations. Instead, we introduce a general symbolic form for the polynomials associated with the function and predicate symbols and for the interargument relations. As an example, assume that polynomials of degree 2 are selected for the interpretation. Then instead of assigning the polynomial $p_q(X_1, X_2) = X_1^2 + 2X_1X_2$ to a predicate symbol q of arity 2, we would, for example, assign the *symbolic* polynomial $p_q(X_1, X_2) = q_{00} + q_{10}X_1 + q_{01}X_2 + q_{11}X_1X_2 + q_1X_1^2 + q_2X_2^2$, where the q_i and q_{ij} are unknown coefficients ranging over $\mathbb{N}$. So our approach for termination analysis works as follows:

- introduce symbolic versions of the polynomials associated with function and predicate symbols,
- express all conditions resulting from Corollary 1 as constraints on the coefficients (e.g., $q_{00}, q_{10}, q_{01}, \ldots$),
- solve the resulting system of constraints to obtain values for the coefficients.

Each solution for this constraint system gives rise to a concrete polynomial interpretation and to concrete valid interargument relations such that all conditions of Corollary 1 are satisfied. Therefore, each solution gives a termination proof.

In order to assign symbolic polynomials to the function and predicate symbols, we make the decision of assigning linear polynomials to predicate symbols and linear or simple-mixed polynomials to function symbols. These classes of polynomials are defined as follows:

(1) *The linear class:* Each monomial of a polynomial in this class contains at most one variable of at most degree 1:
$p(X_1, \ldots, X_n) = p_0 + \sum_{k=1}^{n} p_k X_k$.

(2) *The simple-mixed class:* Each monomial of a polynomial in this class contains either a single variable of at most degree 2 or several variables of at most

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

degree 1:
$p(X_1, \ldots, X_n) = \sum_{j_k \in \{0,1\}} p_{j_1 \ldots j_n} X_1^{j_1} \ldots X_n^{j_n} + \sum_{k=1}^{n} p_k X_k^2$.

The above classes of polynomials have proved to be particularly useful for automated termination proofs of TRSs. For more details on these classes of polynomials we refer to Steinbach (1992) and Contjean *et al.* (2005). In our work, these choices resulted from extensive experiments with different kinds of polynomials, where our goal was to optimize both the efficiency and the power of the termination analyzer.

In Section 4.1, we first reformulate the conditions of our termination criterion in Corollary 1, using the above symbolic forms of polynomials. Then in Section 4.2, we transform these symbolic conditions into constraints on the unknown coefficients of the symbolic polynomials. Afterward, in Section 4.3 we show how these resulting Diophantine constraints can be solved automatically. Finally, we conclude with a comparison of our contributions with related work from term rewriting in Section 4.4.

4.1 Reformulating the termination conditions

In this subsection, we reformulate all termination conditions of Corollary 1, i.e., of Procedure 1. These include the rigidity property (Step 1), the valid interargument relations (Step 2), and the decrease conditions (Step 3). The reformulation results in symbolic constraints, based on the symbolic forms of the polynomial interpretations.

4.1.1 Rigidity conditions (procedure 1, step 1)

There are several ways to approximate $Call(P, S)$ (e.g., Janssens and Bruynooghe 1992; Heaton *et al.* 2000; Bruynooghe *et al.* 2005; Gallagher *et al.* 2005). In this paper, we apply the approximation technique of Janssens and Buynooghe (1992) and Gallagher *et al.* (2005) . More precisely, we first specify the set of queries as a set of rigid type graphs. Then the technique in Janssens and Brunooghe (1992) and Gallagher *et al.*(2005) is used to compute a new, finite set of rigid type graphs which approximate $Call(P, S)$. Each of these new rigid type graphs represents a so-called call pattern. For further details, we refer to Janssens and Brunooghe (1992) and Gallagher *et al.* (2005).

In the following, we recapitulate the notion of rigid type graphs and show how rigidity conditions are derived from the set of call patterns. First, we recall and extend some basic definitions from Jansses and Bruynooghe (1992),, which are based on linear norms and level-mappings, to the case of general polynomial interpretations. Example 7 will illustrate these definitions.

Definition 14 (*rigid type graph Janssens and Bruynooghe 1992*)
A rigid type graph T is a 5-tuple $(Nodes, ForArcs, BackArcs, Label, ArgPos)$, where

(1) $Nodes$ is a finite nonempty set of nodes.
(2) $ForArcs \subseteq Nodes \times Nodes$ such that $(Nodes, ForArcs)$ is a tree.
(3) $BackArcs \subseteq Nodes \times Nodes$ such that for every arc $(m, n) \in BackArcs$, node n is an ancestor of node m in the tree $(Nodes, ForArcs)$.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

(4) *Label* is a function $Nodes \rightarrow Fun_P \cup Pred_P \cup \{\mathbf{MAX}, \mathbf{OR}\}$.
(5) If a node n is labeled with $f \in Fun_P \cup Pred_P$ and f has arity k, then the node n has exactly k outgoing arcs (counting both *ForArcs* and *BackArcs*). These arcs are labeled with the numbers $1, \ldots, k$. For every such arc (n, m), $ArgPos(n, m)$ returns the corresponding label from $\{1, \ldots, k\}$.

The intuition behind rigid type graphs is related to the tree representation of terms and atoms in LP. A rigid type graph generalizes the tree representation of an atom by allowing:

- nodes labeled by **MAX**, denoting any term,
- nodes labeled by **OR**, denoting the union of all denotations of the subgraphs rooted at this node,
- backarcs, denoting repeated traversals of a subgraph.

For each rigid type graph representing a set of atoms S, each node **MAX** in the graph corresponds to a possible occurrence of a variable in the atoms of S. The set S is rigid w.r.t. the polynomial interpretation I if and only if all these variables are not relevant w.r.t. I. In the following, we formulate this rigidity condition syntactically based on the rigid type graph.

Definition 15 (*critical path Decorte et al. 1999*)
Let $T = (Nodes, ForArcs, BackArcs, Label, ArgPos)$ be a rigid type graph. A critical path in T is a path of arcs from the tree *ForArcs* which goes from the root node of the tree to a node labeled **MAX**.

The following proposition is extended from Decorte *et al.* (1993), where each function or predicate symbol is associated with a linear norm or level mapping. It provides a method to generate constraints for rigidity.

Proposition 2 (*checking rigidity by critical paths*)
Let P be a program and $T = (Nodes, ForArcs, BackArcs, Label, ArgPos)$ be a rigid type graph representing a set of atoms S. Let I be a polynomial interpretation, where for any function or predicate symbol f of arity k we have $I(f) = p_f(X_1, \ldots, X_k) = \sum_{0 \leqslant j_1, \ldots, j_k \leqslant M_f} f_{j_1 \ldots j_k} X_1^{j_1} \ldots X_k^{j_k}$. The set S is rigid w.r.t. I if and if and only if on every critical path of T there exists an arc (n, m) with $Label(n) = f$, $arity(f) = k$, and $ArgPos(n, m) = i$ such that $\sum_{j_i > 0} f_{j_1 \ldots j_k} = 0$, where k is the arity of f.

Proof
Since we only regard polynomials with nonnegative coefficients $f_{j_1 \ldots j_k}$, the condition $\sum_{j_i > 0} f_{j_1 \ldots j_k} = 0$ is equivalent to the requirement that $f_{j_1 \ldots j_k} = 0$, whenever $j_i > 0$. This in turn is equivalent to the condition that X_i is not involved in $p_f(X_1, \ldots, X_k)$. Hence, the condition in the above proposition is equivalent to the requirement that for any **MAX** node, there is at least one function or predicate symbol f on the critical path to this **MAX** node, for which the argument position corresponding to the path is not involved in p_f. So equivalently, the atoms in the set S have no relevant variables w.r.t. I. According to Proposition 1, this is equivalent to rigidity w.r.t. I. □

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

The following corollary shows how to express the above rigidity check as a constraint on the coefficients of the polynomial interpretation. To this end, we express the existence condition of an appropriate arc (n, m) by a suitable multiplication.

Corollary 2 (*symbolic condition for checking rigidity*)
Let T be a rigid type graph representing a set of atoms S and let CP be a critical path of T. Let $(n^1, m^1), \ldots, (n^e, m^e)$ be all arcs in CP such that for all $d \in \{1, \ldots, e\}$, $Label(n^d) = f^d$ is a function or predicate symbol of some arity k^d and $ArgPos(n^d, m^d) = i^d$. If for any such CP we have

$$\prod_{d=1}^{e} \left(\sum_{j_{(i^d)} > 0} f^d_{j_1 \ldots j_{(k^d)}} \right) = 0, \tag{5}$$

then S is rigid w.r.t. I.

Example 7 (symbolic polynomial interpretation and rigidity constraints for the "der"-program)

For Example 1, we define a symbolic polynomial interpretation I as follows:

$$\begin{aligned} I(+) &= p_1 X_1^2 + p_2 X_2^2 + p_{11} X_1 X_2 + p_{10} X_1 + p_{01} X_2 + p_{00}, \\ I(*) &= m_1 X_1^2 + m_2 X_2^2 + m_{11} X_1 X_2 + m_{10} X_1 + m_{01} X_2 + m_{00}, \\ I(der) &= der_2 X^2 + der_1 X + der_0, \\ I(u) &= c_u, \\ I(1) &= c_1, \\ I(d) &= d_0 + d_1 X_1 + d_2 X_2. \end{aligned}$$

We will reformulate the termination conditions for this example in symbolic form. However for reasons of space, we will not give all polynomial constraints. Instead, in order to illustrate the main ideas, in each subsection we only present one constraint for the corresponding type of conditions.

Instead of checking termination of the "*der*"-program w.r.t. the set of queries $S = \{d(t_1, t_2) \mid t_1$ is a ground term, t_2 is an arbitrary term$\}$ as in Example 1, we now regard the set of queries $S_1 = \{d(t_1, t_2) \mid t_1$ is of the form $der(t_1')$, where t_1' is a ground term constructed from the function symbols u, $+$, $*$, der, and t_2 is an arbitrary term$\}$. S_1 is represented by the type graph in Figure 1.

Obviously, termination of the program w.r.t. S_1 also implies termination w.r.t. S. This can be proved easily by showing that for any query $Q \in S \setminus S_1$, the program trivially terminates by finite failure.

In our example, type inference (Janssens and Bruynooghe 1992) computes the call set $Call(P, S_1) = S_1$, i.e., the graph in Figure 1 also represents $Call(P, S_1)$. Its only critical path consists of just the arc from the root to the node labeled **MAX**. Hence from the graph, the following rigidity condition is generated according to Corollary 2:

$$d_2 = 0.$$

□

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

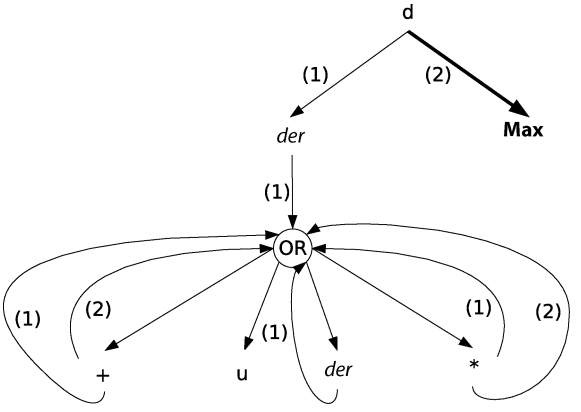


Fig. 1. Rigid type graph for Example 7.

4.1.2 Valid interargument relations (procedure 1, Step 2)

Next we consider the other symbolic constraints, derived for valid interargument relations and decrease conditions. We will show that they all take the form:

$$\forall \overline{X} \in \mathbb{N}: \quad p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \quad \Rightarrow \quad p_{n+1} \geqslant q_{n+1}, \tag{6}$$

where $n \geqslant 0$ and p_i, q_i are polynomials with natural coefficients. Here, $\overline{X}$ is the tuple of all variables occurring in $p_1, \ldots, p_{n+1}, q_1, \ldots, q_{n+1}$.

There are a number of works on inferring valid interargument relations of predicates. In Decorte *et al.* (1999), interargument relations are formulated as inequalities between a linear combination of the "*inputs*" and a linear combination of the "*outputs*". We will not define input and output arguments formally in this paper, since we do not use them in our approach, but informally, inputs are the arguments of a predicate symbol which are only called with ground terms and outputs are the remaining arguments.

We propose a new form of interargument relation, namely *polynomial* interargument relations, which are of the following form:

$$R_p = \{p(t_1, \ldots, t_n) \mid i_p(|t_1|_I, \ldots, |t_n|_I) \gtrsim_{\mathbb{N}} o_p(|t_1|_I, \ldots, |t_n|_I)\}, \tag{7}$$

where i_p and o_p are polynomials with natural coefficients.

The form of interargument relations in Decorte *et al.* (1999) can be considered a special case of the form (7) above, where $i_p(|t_1|_I, \ldots, |t_n|_I)$ is constructed from the input arguments only and $o_p(|t_1|_I, \ldots, |t_n|_I)$ is constructed only from the outputs.

Since the approach in Decorte *et al.* (1999) only considers relations between the input and output arguments of the predicates, it has some limitations. In some cases, the desired relation does not compare inputs with outputs, but the relation holds among the inputs only or among the outputs only. In particular, if all arguments of a predicate are inputs (or outputs), then the approach in Decorte *et al.* (1999) fails to infer any useful relation among them. The following example shows this point. It computes the natural division of the first and second arguments of the predicate *div* and returns the result in its third argument.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Example 8 (*div*)

$$
\begin{aligned}
&div(X, s(Y), 0) \;:\!-\; less(X, s(Y)),\\
&div(X, s(Y), s(Z)) \;:\!-\; sub(X, s(Y), R), div(R, s(Y), Z),\\
&sub(X, 0, X),\\
&sub(s(X), s(Y), Z) \;:\!-\; sub(X, Y, Z),\\
&less(0, s(Y)),\\
&less(s(X), s(Y)) \;:\!-\; less(X, Y).
\end{aligned}
\tag{8}
$$

We consider the set of queries $S = \{\, div(t_1, t_2, t_3) \mid t_1$ and t_2 are ground terms, and t_3 is an arbitrary term$\}$. This program terminates for all these queries. If we look at Clause (8), the decrease in size between the head and the recursive body-atom can be established if we can infer a suitable valid interargument relation for *sub*. This relation should imply that within Clause (8), the first argument of *sub* is greater than its third argument. However, if we apply the approach in Decorte *et al.* (1999), inferring such an interargument relation for *sub* is impossible. Since the first two *sub*-arguments are used as input and the last one is output, the approach can only infer interargument relations where a linear combination of the sizes of the first and second arguments is greater than or equal to the size of the third argument. Then, we cannot conclude that for every successful answer substitution for the call $sub(X, s(Y), R)$ in Clause (8), the first *sub*-argument X is strictly greater than the third *sub*-argument R.

In contrast, if we use Form (7), then it is possible to infer the following valid interargument relation for *sub*:

$$R_{sub} = \{sub(t_1, t_2, t_3) \mid |t_1|_I \gtrsim_{\mathbb{N}} |t_2|_I + |t_3|_I\}.$$

Note that in the right-hand side $|t_2|_I + |t_3|_I$ of the above inequality, we have both an input argument t_2 and an output argument t_3. This valid polynomial interargument relation guarantees that for any successful answer substitution for the call $sub(X, s(Y), R)$ in Clause (8), we have $|X|_I \succ_{\mathbb{N}} |R|_I$ if $|s(Y)|_I \gtrsim_{\mathbb{N}} 1$. Our implementation in the system Polytool is indeed able to infer this interargument relation using the constraint solving technique explained below. Therefore, Polytool can prove termination of "*div*". If we used the form of interargument relations in Decorte *et al.* (1999) instead, Polytool would not be able to solve this problem. □.

Similar to the symbolic form of polynomial interpretations, we also use a symbolic form of polynomial interargument relations. To this end, we take symbolic polynomials i_p and o_p. For the inference of valid interargument relations, we then apply the technique proposed in Decorte *et al.* (1999, cf. Procedure 1, Step 2). For any sequence of terms $t_1, \ldots, t_n$, let $\mathbf{R}_p(t_1, \ldots, t_n)$ abbreviate the inequality $i_p(|t_1|_I, \ldots, |t_n|_I) \geqslant o_p(|t_1|_I, \ldots, |t_n|_I)$. The goal is to impose constraints on the polynomials i_p and o_p, which ensure that the corresponding interargument relation $R_p = \{p(t_1, \ldots, t_n) \mid \forall \overline{X} \in \mathbb{N}\, \mathbf{R}_p(t_1, \ldots, t_n)\}$ is valid. To this end, we generate for every

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

clause of the program:

$$p(\bar{t}) \coloneq p_1(\overline{t_1}), \ldots, p_n(\overline{t_n}),$$

the constraint

$$\forall \overline{X} \in \mathbb{N}: \quad \mathbf{R}_{p_1}(\overline{t_1}) \wedge \ldots \wedge \mathbf{R}_{p_n}(\overline{t_n}) \Rightarrow \mathbf{R}_p(\bar{t}).$$

It is clear that this formula has Form (6).

Example 9 (*symbolic interargument relation for the "der"-program*)

We continue Example 7 and use linear polynomials for i_{der} and o_{der}, i.e., $i_{der}(X, Y) = i_0 + i_1 X + i_2 Y$ and $o_{der} = o_0 + o_1 X + o_2 Y$. Hence, the symbolic form of the polynomial interargument relation for the predicate d is

$$R_d = \{d(t_1, t_2) \mid i_0 + i_1|t_1|_I + i_2|t_2|_I \gtrsim_{\mathbb{N}} o_0 + o_1|t_1|_I + o_2|t_2|_I\}.$$

There are four clauses (1)–(4) from which constraints for valid interargument relations are inferred. We only present the constraint resulting from the last clause (4):

$$d(der(der(X)), DDX) \coloneq d(der(X), DX), d(der(DX), DDX)$$

Here, we obtain the constraint

$$\begin{aligned} &\forall X, DX, DDX \in \mathbb{N}: \\ &\mathbf{R}_d(der(X), DX) \wedge \mathbf{R}_d(der(DX), DDX) \Rightarrow \mathbf{R}_d(der(der(X)), DDX). \end{aligned} \tag{9}$$

□

4.1.3 Decrease conditions (procedure 1, step 3)

Finally, one has to require the decrease condition between the head and any (mutually) recursive body-atom in any (mutually) recursive clause. So for any clause

$$p(\bar{t}) \coloneq p_1(\overline{t_1}), \ldots, p_n(\overline{t_n})$$

of the program where $p \backsimeq p_i$ (i.e., where p and p_i are mutually recursive), we require

$$\forall \overline{X} \in \mathbb{N} \quad \mathbf{R}_{p_1}(\overline{t_1}) \wedge \ldots \wedge \mathbf{R}_{p_{i-1}}(\overline{t_{i-1}}) \Rightarrow |p(\bar{t})|_I \geqslant |p_i(\overline{t_i})|_I + 1.$$

Obviously, the formula is in Form (6).

Example 10 (*constraints for the decrease conditions of "der"*)

There are three recursive clauses (2)–(4) where decrease conditions can be inferred. We present the decrease condition for the recursive body-atom $d(der(DX), DDX)$ of

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

the last clause (4):

$$\begin{aligned}
&\forall X, DX, DDX \in \mathbb{N}: \\
&i_0 + i_1(der_2 X^2 + der_1 X + der_0) + i_2 DX \geqslant \\
&o_0 + o_1(der_2 X^2 + der_1 X + der_0) + o_2 DX \\
&\Rightarrow \\
&d_0 + d_1(der_2(der_2 X^2 + der_1 X + der_0)^2 + \\
&\qquad der_1(der_2 X^2 + der_1 X + der_0) + der_0) + d_2 DDX \geqslant \\
&d_0 + d_1(der_2 DX^2 + der_1 DX + der_0) + d_2 DDX + 1.
\end{aligned} \tag{10}$$

□

4.2 *From symbolic conditions to constraints on coefficients*

Our goal is to find a polynomial interpretation such that all constraints generated in the previous section are satisfied. To this end, we transform all these constraints into Diophantine constraints. In this transformation, we first eliminate implications, cf. Section 4.2.1. Afterward, in Section 4.2.2, the universally quantified variables (e.g., $X, DX, DDX, \ldots$) are removed and the former unknown coefficients (e.g., $der_0, der_1, der_2, \ldots$) become the new variables. If the resulting Diophantine constraints can be solved, then the program under consideration is terminating.

As we analyzed in Section 4.1.1, all generated rigidity constraints have the Form (5). Hence, these are already Diophantine constraints which only contain unknown coefficients, but no universally quantified variables.

The other constraints, generated for the valid interargument relations and the decrease conditions, have the following form:

$$\forall \overline{X} \in \mathbb{N}: \quad p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \quad \Rightarrow \quad p_{n+1} \geqslant q_{n+1}, \tag{6}$$

where $n \geqslant 0$ and p_i, q_i are polynomials with natural coefficients.

In the following, we introduce a two-phase method to transform all constraints of Form (6) into Diophantine constraints on the unknown coefficients.

4.2.1 *First phase: removing implications*

The constraints of Form (6) are implications. In the first phase, such constraints are transformed into inequalities without premises, i.e., into constraints of the form

$$\forall \overline{X} \in \mathbb{N}: \quad p \geqslant 0. \tag{11}$$

However, here p is a polynomial with integer (i.e., possibly negative) coefficients. The transformation is *sound*: if the new constraints of Form (11) are satisfied by some substitution which instantiates the unknown coefficients with numbers, then this substitution also satisfies the original constraints of Form (6).

The idea for the transformation is the following. Constraints of the form (6) may have an arbitrary number n of premises $p_i \geqslant q_i$. We first transform them into

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

constraints with at most one premise. Obviously, $p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n$ implies $p_1 + \ldots + p_n \geqslant q_1 + \ldots q_n$. Thus, instead of (6), it would be sufficient to demand

$$\forall \overline{X} \in \mathbb{N} : \quad p_1 + \ldots + p_n \geqslant q_1 + \ldots q_n \quad \Rightarrow \quad p_{n+1} \geqslant q_{n+1}.$$

So in order to combine the n polynomials in the premise, we can use the polynomial $prem(X_1, \ldots, X_n) = X_1 + \ldots + X_n$. Then instead of (6), we may require

$$\forall \overline{X} \in \mathbb{N} : \quad prem(p_1, \ldots, p_n) \geqslant prem(q_1, \ldots, q_n) \quad \Rightarrow \quad p_{n+1} \geqslant q_{n+1}.$$

A similar method was also used for termination analysis of logic programs in Decorte *et al.* (1999) and for termination of term rewriting in Section 7.2 in Giesl *et al.* (2006) to transform disjunctions of polynomial inequalities into one single inequality.

For example, the constraint

$$\forall X_1, X_2, X_3 \in \mathbb{N} : \quad X_1 \geqslant X_2 \wedge X_2 \geqslant X_3 \quad \Rightarrow \quad X_1 \geqslant X_3$$

can now be transformed into

$$\forall X_1, X_2, X_3 \in \mathbb{N} : \quad X_1 + X_2 \geqslant X_2 + X_3 \quad \Rightarrow \quad X_1 \geqslant X_3$$

Since the latter constraint is valid, the former one is valid as well.

However, in order to make the approach more powerful, one could also use other polynomials *prem* in order to combine the n inequalities in the premise. The reason is that if *prem* is restricted to be the addition, then many valid constraints of the form (6) would be transformed into invalid ones. For example, the valid constraint

$$\forall X_1, X_2, X_3 \in \mathbb{N} : \quad X_1 \geqslant X_2^2 \wedge X_2 \geqslant X_3^2 \quad \Rightarrow \quad X_1 \geqslant X_3^4$$

would be transformed into the invalid constraint

$$\forall X_1, X_2, X_3 \in \mathbb{N} : \quad X_1 + X_2 \geqslant X_2^2 + X_3^2 \quad \Rightarrow \quad X_1 \geqslant X_3^4.$$

For instance, the constraint does not hold for $X_1 = 4$, $X_2 = 0$, and $X_3 = 2$.

To make the transformation more general and more powerful, we therefore permit the use of *arbitrary* polynomials *prem* with natural coefficients. In the above example, now the resulting constraint

$$\forall X_1, X_2, X_3 \in \mathbb{N} : \quad prem(X_1, X_2) \geqslant prem(X_2^2, X_3^2) \quad \Rightarrow \quad X_1 \geqslant X_3^4$$

would indeed be valid for a suitable choice of *prem*. For instance, one could choose *prem* to be the addition of the first argument with the square of the second argument (i.e., $prem(X_1, X_2) = X_1 + X_2^2$).

By the introduction of the new polynomial *prem*, every constraint of the form (6) can now be transformed into an implication with at most one premise. It remains to transform such implications further into unconditional inequalities. Obviously, instead of

$$prem(p_1, \ldots, p_n) \geqslant prem(q_1, \ldots, q_n) \Rightarrow p_{n+1} \geqslant q_{n+1}, \tag{12}$$

it is sufficient to demand

$$p_{n+1} - q_{n+1} \geqslant prem(p_1, \ldots, p_n) - prem(q_1, \ldots, q_n). \tag{13}$$

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

This observation was already used in the work of Decorte *et al.* (1999) and also in termination techniques for term rewriting to handle such conditional polynomial inequalities (Brauburger and Giesl 1998; Giesl *et al.* 2007).

However, the approach can still be improved. Recall that we used an arbitrary polynomial *prem* to combine the polynomials in the former premises. In a similar way, one could also apply an arbitrary polynomial *conc* to the polynomials p_{n+1} and q_{n+1} in the former conclusion. To see why this can be necessary, consider the valid constraint

$$\forall X \in \mathbb{N}: \quad 2X \geqslant 2 \quad \Rightarrow \quad X \geqslant 1.$$

With the transformation of (12) into (13) above, it would be transformed into the unconditional constraint

$$\forall X \in \mathbb{N}: \quad X - 1 \geqslant 2X - 2,$$

which is invalid. We have encountered several examples of this kind in our experiments, which motivates this further extension. In such examples, it would be better to apply a suitable polynomial *conc* to the polynomials X and 1 in the former conclusion. Then we would obtain

$$\forall X \in \mathbb{N}: \quad conc(X) - conc(1) \geqslant 2X - 2$$

instead. By choosing $conc(X) = 2X$, now the resulting constraint is valid.

So to summarize, in the first phase of our transformation, any constraint of the form (6) is transformed into the unconditional constraint

$$\forall \overline{X} \in \mathbb{N}: conc(p_{n+1}) - conc(q_{n+1}) \geqslant prem(p_1, \ldots, p_n) - prem(q_1, \ldots, q_n). \qquad (14)$$

Here, *prem* and *conc* are two arbitrary new polynomials. The only requirement that we have to impose is that *conc* must not be a constant. Indeed, if *conc* would be a constant, then (14) no longer implies that (12) holds for all instantiations of the variables in the polynomials $p_1, \ldots, p_{n+1}, q_1, \ldots, q_{n+1}$. Note that we do not need a similar requirement on *prem*. If a constant *prem* would satisfy (14), then (6) trivially holds. The following proposition proves the soundness of this transformation.

Proposition 3 (*Soundness of Removing Implications*)
Let *prem* and *conc* be two polynomials with natural coefficients, where *conc* is not a constant. Moreover, let $p_1, \ldots, p_{n+1}, q_1, \ldots, q_{n+1}$ be arbitrary polynomials with natural coefficients. If

$$\forall \overline{X} \in \mathbb{N}: \quad conc(p_{n+1}) - conc(q_{n+1}) - prem(p_1, \ldots, p_n) + prem(q_1, \ldots, q_n) \geqslant 0$$

is valid, then

$$\forall \overline{X} \in \mathbb{N}: \quad p_1 \geqslant q_1 \wedge \ldots \wedge p_n \geqslant q_n \quad \Rightarrow \quad p_{n+1} \geqslant q_{n+1}$$

is also valid.

Proof
For any tuple of numbers $\overline{x}$, let $p_i(\overline{x})$ and $q_i(\overline{x})$ denote the numbers that result from p_i and q_i by instantiating the variables $\overline{X}$ by the numbers $\overline{x}$. So if $p(X_1, X_2)$ is the polynomial $X_1^2 + 2X_1X_2$, then $p(2, 1) = 8$.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Suppose that there is a tuple of numbers $\overline{x}$ with $p_i(\overline{x}) \geqslant q_i(\overline{x})$ for all $i \in \{1,\ldots,n\}$. We have to show that then $p_{n+1}(\overline{x}) \geqslant q_{n+1}(\overline{x})$ holds as well.

Since *prem* has only natural coefficients, it is weakly monotonic. Thus, $p_i(\overline{x}) \geqslant q_i(\overline{x})$ for all $i \in \{1,\ldots,n\}$ implies $prem(p_1(\overline{x}),\ldots,p_n(\overline{x})) \geqslant prem(q_1(\overline{x}),\ldots,q_n(\overline{x}))$ and thus, $prem(p_1(\overline{x}),\ldots,p_n(\overline{x})) - prem(q_1(\overline{x}),\ldots,q_n(\overline{x})) \geqslant 0$. The prerequisites of the proposition ensure

$$conc(p_{n+1}) - conc(q_{n+1}) \;\geqslant\; prem(p_1,\ldots,p_n) - prem(q_1,\ldots,q_n)$$

for all instantiations of the variables. Hence, we also obtain $conc(p_{n+1}(\overline{x})) - conc(q_{n+1}(\overline{x})) \geqslant 0$ or, equivalently,

$$conc(p_{n+1}(\overline{x})) \geqslant conc(q_{n+1}(\overline{x})). \tag{15}$$

Now suppose that $p_{n+1}(\overline{x}) \not\geqslant q_{n+1}(\overline{x})$. Since $p_{n+1}(\overline{x})$ and $q_{n+1}(\overline{x})$ are *numbers* (not polynomials with variables), we would then have $p_{n+1}(\overline{x}) < q_{n+1}(\overline{x})$. Since *conc* only has nonnegative coefficients and since it is not a constant, it is strictly monotonic. Thus, $p_{n+1}(\overline{x}) < q_{n+1}(\overline{x})$ would imply

$$conc(p_{n+1}(\overline{x})) < conc(q_{n+1}(\overline{x}))$$

in contradiction to (15). Hence, we have $p_{n+1}(\overline{x}) \geqslant q_{n+1}(\overline{x})$, as desired. □

For the symbolic form of *prem* and *conc*, we again choose linear or simple-mixed polynomials. From our experiments, this choice provided good results on the benchmark programs, while remaining reasonably efficient. By applying Proposition 3, we can now transform all constraints for the termination proof into unconditional constraints of the form (11). If there exists a substitution of the unknown coefficients by numbers that makes the resulting unconditional constraints valid, then the same substitution also satisfies the original conditional constraints.

Example 11 (*applying Proposition 3 to the "der"-program*)
We choose the decrease condition (10) in Example 10 as an example showing how to transform an implication into an unconditional constraint.

Since the constraint (10) has only one premise, here the polynomial *prem* has arity 1. We choose a simple mixed form for *prem* and a linear form for *conc*:

$$prem(X) = prem_0 + prem_1 X + prem_2 X^2 \quad conc(X) = conc_0 + conc_1 X.$$

Since *conc* must not be a constant, one also has to impose the constraint

$$conc_1 > 0.$$

Now we can transform (10) into an unconditional constraint. Here, we use the following abbreviations:

$$\begin{aligned}
p_1 &= i_0 + i_1(der_2 X^2 + der_1 X + der_0) + i_2 DX,\\
q_1 &= o_0 + o_1(der_2 X^2 + der_1 X + der_0) + o_2 DX,\\
p_2 &= d_0 + d_1(der_2(der_2 X^2 + der_1 X + der_0)^2\\
&\quad + der_1(der_2 X^2 + der_1 X + der_0) + der_0) + d_2 DDX,\\
q_2 &= d_0 + d_1(der_2 DX^2 + der_1 DX + der_0) + d_2 DDX + 1.
\end{aligned}$$

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Then (10) is the constraint

$$\forall X, DX, DDX \in \mathbb{N}: \quad p_1 \geqslant q_1 \quad \Rightarrow p_2 \geqslant q_2$$

and its transformation yields

$$\begin{aligned} \forall X, DX, DDX \in \mathbb{N}: \quad & conc_0 + conc_1\, p_2 - conc_0 - conc_1\, q_2 \\ & -prem_0 - prem_1\, p_1 - prem_2\, p_1^2 \\ & +prem_0 + prem_1\, q_1 + prem_2\, q_1^2 \qquad \geqslant 0. \end{aligned}$$

By applying standard simplifications, the constraint can be rewritten to the following form:

$$\begin{aligned} \forall X, DX \in \mathbb{N}: \quad & M_1 X^4 + M_2 X^3 + M_3 X^2 + M_4 X + \\ & M_5 DX^2 + M_6 DX + M_7 X^2 DX + M_8 X DX + M_9 \geqslant 0, \end{aligned} \tag{16}$$

where $M_1, \ldots, M_9$ are polynomials over the unknown coefficients $prem_j$, i_j, o_j, der_j, and d_j with $j \in \{0, 1, 2\}$ and $conc_j$ with $j \in \{0, 1\}$. For example, we have

$$M_1 =_{def} conc_1\, d_1\, der_2^3 + prem_2\, o_1^2\, der_2^2 - prem_2\, i_1^2\, der_2^2.$$

□

4.2.2 Second phase: removing universally quantified variables

In this phase, we transform any constraint of the form

$$\forall \overline{X} \in \mathbb{N}: \quad p \geqslant 0 \tag{11}$$

into a set of Diophantine constraints on the unknown coefficients. The transformation is again *sound*: if there is a solution for the resulting set of Diophantine constraints, then this solution also satisfies the original constraint (11).

We use a straightforward transformation proposed by Hong and Jakuš (1998), which is also used in all related tools for termination of term rewriting. One only requires that all coefficients of the polynomial p are nonnegative integers. Obviously, the criterion is only sufficient, because, for instance, $p(X) = (X-1)^2 \geqslant 0$, but $X^2 - 2X + 1$ does not have nonnegative coefficients only.

Example 12 (*removing universally quantified variables for the "der"-program*)
We continue the transformation of Example 11. Here, we obtained the constraint (16). We derive the following set of Diophantine constraints which contains the unknown coefficients $conc_j$, $prem_j$, i_j, o_j, der_j, and d_j as variables: $M_1 \geqslant 0, M_2 \geqslant 0, \ldots, M_9 \geqslant 0$. □

4.3 Solving Diophantine constraints

The previous sections showed that one can formulate all termination conditions in symbolic form and that one can transform them automatically into a set of Diophantine constraints. The problem then becomes solving a system of nonlinear Diophantine constraints with the unknown coefficients as variables. If the

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Diophantine constraints are solvable, then the logic program under consideration is terminating. Solving such problems has been studied intensively, especially in the context of constraint logic programming. Moreover, there are approaches from termination of term rewriting in order to solve such restricted Diophantine constraints automatically (e.g., Contejean *et al.* 2005; Fuhs *et al.* 2007; Borralleras *et al.* 2009). In Fuhs *et al.* (2007), Diophantine constraints are encoded as a SAT-problem, and then a SAT solver is used to solve the resulting SAT-problem. As shown in Fuhs *et al.* (2007), this approach is significantly more efficient than solving Diophantine constraints by dedicated solvers like Contejean *et al.* (2005) or by standard implementations of constraint logic programming like in SICStus Prolog.

Example 13 (*solving Diophantine constraints for the"der"-program*)
We start with the symbolic polynomial interpretation from Example 7 (e.g., with $I(der) = der_2X^2 + der_1X + der_0$) and obtain the solution $der_2 = 1$ and $der_0 = der_1 = 2$, which corresponds to $X^2 + 2X + 2$. Similarly, we start with the symbolic form of the polynomial interargument relation as in Example 9:

$$R_d = \{d(t_1, t_2) \mid i_0 + i_1|t_1|_I + i_2|t_2|_I \gtrsim_{\mathbb{N}} o_0 + o_1|t_1|_I + o_2|t_2|_I\}.$$

Then we get the solution $i_1 = 1$, $i_0 = i_2 = 0$, $o_2 = 1$, $o_0 = o_1 = 0$. This corresponds to the interargument relation $R_d = \{d(t_1, t_2) \mid |t_1|_I \gtrsim_{\mathbb{N}} |t_2|_I\}$. So we obtain the concrete simple-mixed polynomial interpretation from Example 4 and the concrete interargument relation from Example 6. □

4.4 Relation to approaches from term rewriting

Finally, we briefly discuss the connection between our approach for automated LP termination proofs from Section 4.1–4.3 and related approaches used for termination analysis of TRSs.

Section 4.1 describes how to obtain constraints for a symbolic polynomial order which guarantee that the requirements of our termination criterion are fulfilled. This is similar to related approaches used in term rewriting. Here, one also chooses a symbolic polynomial interpretation and constructs corresponding inequalities. If one applies polynomial interpretations directly for termination analysis of TRSs, then these inequalities ensure that every rewrite rule is strictly decreasing. If one uses more sophisticated termination techniques like the dependency pair method (Arts and Giesl 2000; Hirokawa and Middeldorp 2005; Giesl *et al.* 2006), then one builds inequalities which ensure that dependency pairs are weakly or strictly decreasing and that rules are weakly decreasing. The decrease conditions of dependency pairs correspond to our decrease conditions in Section 4.1.3 and the requirement that rules are weakly decreasing roughly corresponds to our symbolic constraints for valid interargument relations in Section 4.1.2. Still, there are subtle differences. For example, in LPs, a predicate symbol may have several output arguments which is the reason for the different polynomials i_p and o_p in our polynomial interargument relations. Moreover, while term rewriting uses matching for evaluation, in logic

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Table 1. *The results for 296 benchmarks of the TPDB*

	TALP	cTI-1.1	TerminWeb	Polytool	AProVE
YES	163 [2.54]	167 [0.06]	177 [0.54]	214 [4.28]	232 [6.34]
FAILURE	112 [1.45]	129 [0.05]	118 [0.6]	62[10.48]	57 [19.08]
TIMEOUT	21	0	1	20	7

programming one uses unification. This is the reason for our additional rigidity conditions in Section 4.1.1.

The approach in Section 4.2 shows how to find suitable values for the symbolic coefficients. This is the same problem as in the corresponding techniques for term rewriting. However, the usual techniques in term rewriting can only handle unconditional inequalities. Therefore, we have developed a new method in Section 4.2.1 to remove conditions. This is a new contribution of the present paper. In fact, after having developed this contribution for the current paper, due to its success in the tool Polytool, two of the authors of the current paper later even adapted this method to term rewriting (see Fuhs *et al.* 2008, Footnote 14).

The techniques of the short sections 4.2.2 and 4.3 are identical to the corresponding approaches used in term rewriting. We only included them here in order to have a self-contained presentation of our approach and to finish its illustration with the "*der*"-example.

5 Experimental evaluation

In this section we discuss the experimental evaluation of our approach. We implemented our technique in a system called Polytool (Nguyen and De Schreye 2007) written in SICStus Prolog[1]. Essentially, the Polytool system consists of four modules: The first module is the type inference engine, where we use the inference system of Gallenger *et al.* (2005). The second module generates all termination conditions using symbolic polynomials as in Section 4.1. The third module transforms the resulting polynomial constraints into Diophantine constraints, as in Section 4.2. The final module is a Diophantine constraint solver, cf. Section 4.3. We selected the SAT-based Diophantine solver (Fuhs *et al.* 2007) of the AProVE tool (Giesl *et al.* 2006).

We tested the performance of Polytool on a collection of 296 examples. The collection (Table 1) consists of all benchmarks for logic programming from the *Termination Problem Data Base* (TPDB)[2], where all examples that contain arithmetic or built-in predicates were removed.

Polytool applies the following strategy: first, we search for a linear polynomial interpretation. If we cannot find such an interpretation satisfying the termination conditions, then we search for a simple-mixed polynomial interpretation. More

[1] For the source code, we refer to `http://www.cs.kuleuven.be/~manh/polytool`.
[2] `http://www.termination-portal.org/wiki/Termination_Competition`

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

precisely, we still interpret predicate symbols by linear polynomials, but we map function symbols to simple-mixed polynomials. We use similar symbolic polynomials for *conc* and *prem* from Section 4.2.1: if the polynomial interpretation is linear, then both *conc* and *prem* are linear. Otherwise, we use a linear form for *conc* and a simple-mixed form for *prem*. The domain for all unknown coefficients in the generated Diophantine constraints is fixed to the set $\{0, 1, 2\}$. The experiments were performed on an AMD 64 bit, 2GB RAM running Linux.

We performed an experimental comparison with other leading systems for automated termination analysis of logic programs, namely: Polytool-WST07, cTI-1.1 (Mesnard and Bagnara 2005), TerminWeb (Codish and Taboch 1999; Taboch *et al.* 2002), TALP (Ohlebusch *et al.* 2000) and AProVE (Giesl *et al.* 2006a). For TALP, the option of nonlinear polynomial interpretations was chosen. For cTI-1.1, we selected the "default" option. For AProVE and TerminWeb, the fully automatic modes were chosen. We did not include the tool Hasta-La-Vista (Serebrenik and De Schreye 2003) in the evaluation because it is a predecessor of Polytool. We used a time limit of 60 seconds for testing each benchmark on each termination tool. This time limit is also used in the annual termination competition.

In Table 1, we give the numbers of benchmarks which are proved terminating ("YES"), the number of benchmarks which could not be proved terminating but where processing ended within the time limit ("FAILURE"), and the number of benchmarks where the tool did not stop before the timeout ("TIMEOUT"). The number in square brackets is the average runtime (in seconds) that a particular tool uses to prove termination of benchmarks (or fails to prove termination of them within the time limit). The detailed experiments (including also the source code of the benchmarks and the termination proofs produced by the tools) can be found at `http://www.cs.kuleuven.be/~dtai/projects/polytool`. Note that the two examples *der* and *div* presented in this paper do not occur in the TPDB. For completeness we just mention that Polytool and AProVE succeed on *der*, whereas cTI-1.1 and TerminWeb fail, and TALP reaches the timeout. For *div*, all systems except TALP succeed. In the next subsections we discuss the results of the experiments. For a more detailed discussion, we refer to (Nguyen 2009).

5.1 *Comparison between* polytool *and* cTI-1.1

Similar to Polytool, cTI-1.1 deploys a global constraint-based approach to termination analysis. However, different from Polytool, in cTI-1.1 termination inference of the analyzed program relies on its two main abstract approximations: a program in CLP($\mathbb{N}$), where all terms of the program are mapped to expressions in $\mathbb{N}$ according to a fixed symbolic norm (e.g., the symbolic[3] term-size norm by default), and a program in CLP($\mathbb{B}$), where $\mathbb{B}$ denotes the booleans, which is obtained from the program in CLP($\mathbb{N}$) by mapping any number to 1, any variable to itself, and addition to logical

[3] The difference between the "term-size norm" and the "*symbolic* term-size norm" is that the "term-size norm" maps all variables to 0, whereas the "symbolic term-size norm" maps any variable to itself (as in polynomial interpretations).

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

conjunction. The purpose of these abstractions is to capture the decrease conditions (the program in CLP($\mathbb{N}$)) and the boundedness information (the program in CLP($\mathbb{B}$)) of the program.

As shown in Table 1, Polytool outperforms cTI-1.1. The only benchmark where cTI-1.1 can prove termination and Polytool fails is the example incomplete2.pl in the directory SGST06 of the TPDB. However, if we reset the range for the values of the unknown coefficients in the generated Diophantine constraints to $\{0,\ldots,8\}$, then Polytool can prove termination for the example as well.

There are several reasons for the less powerful performance of cTI-1.1 in comparison with Polytool. First of all, cTI-1.1 uses a fixed symbolic norm to map the analyzed program to a program in CLP($\mathbb{N}$), for which all termination conditions are formulated. However, in some cases, the selected symbolic norm is not suitable to capture the decrease in the analyzed program. Then as a result, cTI-1.1 cannot prove termination. The TPDB contains a number of such benchmarks, e.g., flat.pl, normal.pl in the talp directory and countstack.pl, factor.pl, flatten.pl in the SGST06 directory.

Secondly, when we use the term-size or list-length norm for the abstract approximation in cTI-1.1, all constant symbols are mapped to the same number in $\mathbb{N}$. As a result, cTI-1.1 fails for examples where the difference among constant symbols plays a role for the termination behavior. In Polytool, different constant symbols can be mapped to different numbers in $\mathbb{N}$. Therefore, termination of examples such as simple.pl in the talp directory, pl2.3.1.pl in the plumer directory, at.pl in the SGST06 directory, etc. can be proved, whereas cTI-1.1 fails.

Thirdly, since termination analysis of cTI-1.1 is based on linear symbolic norms, it cannot prove termination of programs such as Example 1 or the example hbal_tree.pl in the TPDB. In contrast, Polytool can prove termination of these examples using simple-mixed polynomial interpretations.

Finally, there are examples like applast.pl, bappend.pl, blist.pl, btappend.pl, btapplast.pl, confdel.pl and btree.pl in the SGST06 directory, whose termination cannot be proved by cTI-1.1, since cTI-1.1 only uses groundness instead of type analysis. The termination proof of these examples also fails with TALP for the same reason. In contrast, Polytool and AProVE succeed for them and TerminWeb succeeds for some of them (i.e., applast.pl, bappend.pl, blist.pl, confdel.pl). The success of Polytool and TerminWeb is due to the use of types instead of modes and AProVE succeeds because of so-called *argument filterings* which remove argument positions of function and predicate symbols that are irrelevant for termination. But TerminWeb still fails on some of these examples, since it uses a fixed norm for part of its analysis.

A strong point of cTI-1.1 is that it is very fast (it is by far the fastest tool in the experiments). The reason is that cTI-1.1 fixes the norm in advance. Therefore it requires much less unknown coefficients to formulate termination conditions. Another strong point of cTI-1.1 is its ability of performing termination inference (i.e., it can try to detect all terminating modes for a program), which is impossible for Polytool at this moment. Finally, recent extensions of cTI-1.1 include nontermination proofs, which are not supported by the other systems in our experiments.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

5.2 *Comparison between* polytool *and* terminWeb

Similar to cTI-1.1, TerminWeb also uses fixed symbolic norms, e.g., the term-size norm, the list-length norm, or (as in our experiments) a combination of type-based norms (Bruynooghe *et al.* 2007) to approximate the analyzed program. Therefore, it has similar problems as cTI-1.1. In fact, termination of examples such as flat.pl, normal.pl, countstack.pl, factor.pl, flatten.pl discussed in Section 5.1 cannot be proved by TerminWeb either.

Different from Polytool and cTI-1.1, TerminWeb applies a local approach to termination analysis, where different norms and level mappings are used for different loops in the program (Codish and Taboch 1999). Hence, TerminWeb can prove termination of a class of programs where lexicographic orders are required (e.g., the benchmarks ackermann.pl and vangelder.pl in the TPDB). In fact, these programs could already be proven terminating by TermiLog (Lindenstrauss and Sagiv 1997; Lindenstrauss 2000), the first generally available automatic termination analyzer for LPs. TermiLog succeeds on these programs due to the query-mapping pairs approach (Lindenstrauss *et al.* 2004), which has some similarity to the dependency pair approach (Arts and Giesl 2000; Hirokawa and Middeldorp 2005; Giesl *et al.* 2006b). For termination of such programs, the global technique based on polynomial interpretations deployed in Polytool is insufficient. We are working on an extension using dependency graphs that is able to deal with such programs as well (Nguyen *et al.* 2008; Schneider-Kamp *et al.*, forthcoming).

Similar to cTI-1.1, TerminWeb is much faster than Polytool. This is again due to the fact that TerminWeb uses a fixed symbolic norm to approximate the analyzed program.

5.3 *Comparison between* polytool, AProVE, *and* TALP

A point of similarity between Polytool, TALP, and AProVE is that all these systems use polynomial interpretations as the basis for the termination analysis. However in TALP and AProVE, polynomial interpretations are applied indirectly: given a logic program and a set of queries, these tools first transform them into a TRS whose termination is sufficient for the termination of the original logic program. Then, termination analysis is applied to the resulting TRS. Due to this transformational approach, several other termination techniques developed for TRSs become applicable for the analysis of LPs as well. In particular, AProVE uses many different methods for proving termination.

A limitation of the transformational approach in TALP is that it can only handle well-moded logic programs. There are many nonwell-moded examples in the TPDB that can be solved by most other tools but not by TALP.

AProVE instead applies a quite strong transformational approach, which can also deal with nonwell-moded logic programs (Schneider-Kamp *et al.* 2009). Together with the powerful back-end TRS termination prover, this makes AProVE a very strong LP termination system. In fact, in both our experiments and in the termination competitions, AProVE was always in the first place. In particular, it can

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

prove termination of most examples whenever some other tool can. Nevertheless, there exists one example in the TPDB (i.e., incomplete.pl) where AProVE fails to prove its termination but Polytool succeeds. In general, the most important observation when comparing Polytool and AProVE is that although Polytool only uses polynomial interpretations and AProVE uses a large collection of different termination techniques, Polytool is already almost as powerful as AProVE.

Similar to TerminWeb, cTI-1.1, and TALP, AProVE uses mode analysis and does not provide the expressivity of types. However, it can express classes like bounded lists, since it uses *argument filterings*. Nevertheless, in some cases, the effect of *argument filterings* is not "deep" enough to represent redundant argument positions adequately, cf. (Nguyen 2009). Finally, as shown in Table 1, AProVE is the slowest tool in the experiments. One reason is that the transformation may generate quite complex TRSs that require more time for termination analysis. Another reason is that AProVE contains much more different termination techniques than the other tools and it tries to apply them all after each other.

6 Conclusions

Since a few years, the LP and the TRS termination analysis communities jointly organize the "*International Workshop on Termination*" (WST). As a part of this workshop, the *International Competition of Termination Tools* is organized annually, allowing different termination tools from different categories, including term rewriting and logic programming, to compete. These workshops have raised a considerable interest in gaining a better understanding of each other's approaches. It soon became clear that there has to be a close relationship between one of the most popular techniques for TRSs, polynomial interpretations, and one of the key techniques for LPs, acceptability with linear norms and level mappings. However, partly because of the distinction between orders over the numbers (LPs) versus orders over polynomials (TRSs), the actual relation between the approaches was unclear.

One main conclusion of the research that led to this paper is that the distinction is a superficial one. So one outcome of our work is that, indeed, the polynomial interpretations used for TRSs are a direct generalization of the current practice for LPs.

On the more technical level, the contribution of this paper is twofold. Firstly, we provide a complete and revised theoretical framework for polynomial interpretations in LP termination analysis (cf. Section 3). A first variant of such a framework was introduced in a preliminary version of this paper (Nguyen and De Schreye 2005). Parts of this build on the results of De Schreye and Serebrenik (2002) on order-acceptability and the results of Decorte *et al.* (1999) on the constraint-based approach for termination analysis. Another part extends the results of Bossi *et al.* (1991) on the syntactic characterization of rigidity. The main revisions are in the concept of polynomial interpretations and the concept of rigidity. Secondly, we adapt the constraint-based approach in Decorte *et al.* (1999) to represent all termination

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

conditions symbolically, and introduce a new approach to find such polynomial interpretations automatically (cf. Section 4).

We also developed an automated tool (Polytool (Nguyen and De Schreye 2007)) for termination proofs of LPs based on polynomial interpretations. The main contribution of the implementation is the integration of a number of techniques including the termination framework in Section 3, the call pattern inference tools in Janssens and Bruynooghe (1992), Heaton *et al.* (2000), Bruynooghe *et al.* (2005), Gallagher *et al.* (2005), the constraint-based approach in Section 4, and the Diophantine constraint solver in Fuhs *et al.* (2007), to provide a completely automated termination analyzer. Polytool participated in the annual *International Competitions of Termination Tools* since 2007 and reached the second place, just after AProVE.

We have also conducted extensive experimental evaluation for Polytool and compared it empirically with other termination analyzers such as cTI-1.1, TerminWeb, TALP, and AProVE, cf. Section 5. The evaluation shows that Polytool is powerful enough to solve a large number of benchmarks. In particular, it can also verify termination of examples for which nonlinear norms are required.

The current paper and the corresponding tool provide a good basis to adapt further techniques from the area of TRS termination to the LP domain. In this way, the power of automated termination analysis can be increased substantially. Moreover, such adaptations will clarify the connections between the numerous termination techniques developed for TRSs and for LPs, respectively. First steps into this direction are (Nguyen *et al.* 2008; Schneider-Kamp *et al.*, forthcoming).

Acknowledgements

Manh Thang Nguyen was partly supported by *GOA Inductive Knowledge Bases* and partly by *FWO Termination Analysis: Crossing Paradigm Borders*. Jürgen Giesl and Peter Schneider-Kamp were supported by the *Deutsche Forschungsgemeinschaft (DFG) grant GI 274/5-2*. We thank John Gallagher for making his type inference engine available, Carsten Fuhs for his SAT-based Diophantine constraint solver within AProVE, Frédéric Mesnard and Roberto Bagnara for providing us the cTI system and the Parma Polyhedra Library, Michael Codish and Samir Genaim for their TerminWeb system. We thank the anonymous reviewers for their valuable comments.

References

Apt, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, Vol. B, Ed. Jan van Leeuwen, MIT Press, 493–574.

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

Borralleras, C., Lucas, S., Navarro-Marset, R., Rodríguez-Carbonell, E. and Rubio, A. 2009. Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In *Proc. CADE '09*. Ed. Renate Schmidt, LNAI, Springer-Verlag, vol. 5663, 294–305.

Bossi, A., Cocco, N. and Fabris, M. 1991 Proving termination of logic programs by exploiting term properties. In *Proc. TAPSOFT '91*. Eds. S. Abramsky and T. Maibaum, LNCS, Springer-Verlag, vol. 494, 153–180.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Brauburger, J. and Giesl, J. 1998 Termination analysis by inductive evaluation. In *Proc. CADE '98*. Eds. C. Kirchner and H. Kirchner, LNAI, Springer-Verlag, vol. 1421, 254–269.

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, 10-es.

Bruynooghe, M., Gallagher, J. P. and Van Humbeeck, W. 2005. Inference of well-typings for logic programs with application to termination analysis. In *Proc.* SAS '05. Eds. C. Hankin and I. Siveroni, LNCS, Springer-Verlag, vol. 3672, 35–51.

Codish, M. and Taboch, C. 1999. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming 41*, 1, 103–123.

Contejean, E., Marché, C., Tomás, A. P. and Urbain, X. 2005. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning 34*, 4, 325–363.

Decorte, S., De Schreye, D. and Fabris, M. 1993. Automatic inference of norms: A missing link in automatic termination analysis. In *Proc. ILPS '93*. Ed. D. Miller, MIT Press, 420–436.

Decorte, S., De Schreye, D. and Vandecasteele, H. 1999. Constraint-based automatic termination analysis of logic programs. *ACM Transactions on Programming Languages and Systems 21*, 6, 1137–1195.

Dershowitz, N. 1995. 33 examples of termination. In *Proc. French Spring School in Theoretical Computer Science*. Eds. H. Comon and J. Jouannaud, LNCS, Springer-Verlag, vol. 909, 16–26.

Dershowitz, N., Lindenstrauss, N. and Sagiv, Y. 1997. What norms are useful for logic programs? In *Proc. WST '97*. Transparencies available from URL: `http://www.cs.huji.ac.il/~naomil/`.

De Schreye, D. and Serebrenik, A. 2002. Acceptability with general orderings. In *Computational Logic: Logic Programming and Beyond*. Eds. A. Kakas and F. Sadri, LNCS, Springer-Verlag, vol. 2407, 187–210.

Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R. and Zankl, H. 2007. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT '07*. Eds. J. Marques-Silva, K. Sakallah, LNCS, Springer-Verlag, vol. 4501, 340–354.

Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R. and Zankl, H. 2008. Maximal termination. In *Proc. RTA '08*, 110–125.

Gallagher, J. P., Henriksen, K. S. and Banda, G. 2005. Techniques for scaling up analyses based on pre-interpretations. In *Proc. ICLP '05*. Eds. M. Gabbrielli and G. Gupta, LNCS, Springer-Verlag, vol. 3668, 280–296.

Giesl, J., Schneider-Kamp, P. and Thiemann, R. 2006a. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*. Eds. U. Furbach and N. Shankar, LNAI, Springer-Verlag, vol. 4130, 281–286.

Giesl, J., Thiemann, R., Schneider-Kamp, P. and Falke, S. 2006b. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning 37*, 3, 155–203.

Giesl, J., Thiemann, R., Swiderski, S. and Schneider-Kamp, P. 2007. Proving termination by bounded increase. In *Proc. CADE '07*. Ed. F. Pfenning, LNAI, Springer-Verlag, vol. 4603, 443–459.

Heaton, A., Abo-Zaed, M., Codish, M. and King, A. 2000. A simple polynomial groundness analysis for logic programs. *Journal of Logic Programming 45*, 1–3, 143–156.

Hirokawa, N. and Middeldorp, A. 2005. Automating the dependency pair method. *Information and Computation 199*, 1–2, 172–199.

Hong, H. and Jakuš, D. 1998. Testing positiveness of polynomials. *Journal of Automated Reasoning 21*, 1, 23–38.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press

Janssens, G. and Bruynooghe, M. 1992. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Journal of Logic Programming 13*, 2–3, 205–258.

Lankford, D. S. 1979. *On Proving Term Rewriting Systems are Noetherian.* Tech. Rep., Mathematics Department, Louisiana Tech University, Ruston, LA.

Lindenstrauss, N. 2000. TermiLog: Termination analyzer for logic programs. URL: http://www.cs.huji.ac.il/~naomil/termilog.php.

Lindenstrauss, N. and Sagiv, Y. 1997. Automatic termination analysis of logic programs. In *Proc. ICLP '97*. Ed. L. Naish, MIT Press, 63–77.

Lindenstrauss, N., Sagiv, Y. and Serebrenik, A. 2004. Proving termination for logic programs by the query-mapping pairs approach. In *Program Development in Computational Logic*. Eds. M. Bruynooghe and K. Lau, LNCS, Springer-Verlag, vol. 3049, 453–498.

Lloyd, J. W. 1987. *Foundations of Logic Programming*. Springer Verlag, Berlin.

Mesnard, F. and Bagnara, R. 2005. cTI: A constraint-based termination inference tool for ISO-Prolog. *Theory and Practice of Logic Programming 5*, 1–2, 243–257.

Nguyen, M. T. 2009 *Termination Analysis: Crossing Paradigm Borders.* Ph.D. thesis, Department of Computer Science, K. U. Leuven, Belgium.

Nguyen, M. T. and De Schreye, D. 2005. Polynomial interpretations as a basis for termination analysis of logic programs. In *Proc. ICLP '05*. Eds. M. Gabbrielli and G. Gupta, LNCS, Springer-Verlag, vol. 3668, 311–325. Extended version appeared as Technical report, Department of Computer Science, K. U. Leuven, Belgium, URL: http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW412.pdf.

Nguyen, M. T. and De Schreye, D. 2007. Polytool: Proving termination automatically based on polynomial interpretations. In *Proc. LOPSTR '06*. Ed. G. Puebla, LNCS, Springer-Verlag, vol. 4407, 210–218. Extended version appeared as Technical report, Department of Computer Science, K. U. Leuven, Belgium, URL: http://www.cs.kuleuven.be/~manh/polytool/publications/tech-reports/reportCW442.pdf.

Nguyen, M. T., Giesl, J., Schneider-Kamp, P. and De Schreye, D. 2008. Termination analysis of logic programs based on dependency graphs. In *Proc. LOPSTR '07*. Ed. A. King, LNCS, Springer-Verlag, vol. 4915, 8–22.

Ohlebusch, E., Claves, C. and Marché, C. 2000. TALP: A tool for the termination analysis of logic programs. In *Proc. RTA '00*. Ed. L. Bachmair, LNCS, Springer-Verlag, vol. 1833, 270–273.

Schneider-Kamp, P., Giesl, J. and Nguyen, M. T. The dependency triple framework for termination of logic programs. In *Proc. LOPSTR '09*. LNCS. Forthcoming.

Schneider-Kamp, P., Giesl, J., Serebrenik, A. and Thiemann, R. 2009. Automated termination proofs for logic programs by term rewriting. *ACM Transactions on Computational Logic 11,* 1. Preliminary version appeared in *Proc. LOPSTR '06*, Ed. G. Puebla, LNCS, Springer-Verlag, vol. 4407, pp. 117–193, 2007.

Serebrenik, A. 2003. *Termination Analysis of Logic Programs.* Ph.D. thesis, Department of Computer Science, K. U. Leuven, Belgium. URL: http://www.cs.kuleuven.ac.be/publicaties/doctoraten/cw/CW2003_02.abs.html.

Serebrenik, A. and De Schreye, D. 2003. Hasta-La-Vista: Termination analyser for logic programs. In *Proc. WLPE '03*. Eds. F. Mesnard and A. Serebrenik, 60–74.

Steinbach, J. 1992. Proving polynomials positive. In *Proc. FSTTCS '92*. Ed. R. Shyamasundar, LNCS, Springer-Verlag, vol. 652, 18–20.

Taboch, C., Genaim, S. and Codish, M. 2002. TerminWeb: Semantics-based termination analyser for logic programs. URL: http://www.cs.bgu.ac.il/~mcodish/TerminWeb.

Verschaetse, K. and De Schreye, D. 1991. Deriving termination proofs for logic programs, using abstract procedures. In *Proc. ICLP '91*. Ed. K. Furukawa, MIT Press, 301–315.

https://doi.org/10.1017/S1471068410000025 Published online by Cambridge University Press