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

Manh Thang Nguyen, Danny De Schreye,
Jürgen Giesl, Peter Schneider-Kamp

Report CW 558, July 2009

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

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

Manh Thang Nguyen[*], *Danny De Schreye*[†], *Jürgen Giesl*[‡], *Peter Schneider-Kamp*[§]

Report CW 558, July 2009

Department of Computer Science, K.U.Leuven

Abstract

This paper reports on work that was done in a project called "Termination analysis: crossing paradigm borders". The aim of the project is to study the feasibility of porting termination analysis techniques developed for one programming paradigm to another paradigm. In this paper, we report on part of the results of this project, namely, the study of porting termination analysis techniques based on polynomial interpretation - very well known in the context of term rewrite systems (TRS) - to obtain new (non-transformational) termination analysis techniques for definite logic programs (LP). We show how this leads to an approach that can be seen as a direct generalization of the traditional techniques in termination analysis of LP, 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 propose a constraint based approach for automatically generating poly- nomial interpretations that satisfy the termination conditions. Based on this approach, we implement a new tool, called Polytool, for automatic termination analysis of logic programs.

Keywords : Termination analysis, acceptability, polynomial interpretations.

---

[*]Deceased on 3 May 2009

[†]Department of Computer Science, K.U.Leuven
[‡]LuFG, Informatik, RWTH Aachen
[§]LuFG, Informatik, RWTH Aachen

Polytool: Polynomial Interpretations as a Basis for Termination Analysis of Logic Programs

Manh Thang Nguyen[1] Danny De Schreye[2], Jürgen Giesl[3], and Peter Schneider-Kamp[3]

[1] Deceased on 3 May 2009
[2] Department of Computer Science, K. U. Leuven
Celestijnenlaan 200A, B-3001, Heverlee, Belgium
Danny.DeSchreye@cs.kuleuven.ac.be
[3] LuFG Informatik 2, RWTH Aachen
Ahornstr. 55, D-52074 Aachen, Germany
{giesl, psk}@informatik.rwth-aachen.de

Abstract. This paper reports on work that was done in a project called "Termination analysis: crossing paradigm borders". The aim of the project is to study the feasability of porting termination analysis techniques developed for one programming paradigm to another paradigm.
In this paper, we report on part of the results of this project, namely, the study of porting termination analysis techniques based on polynomial interpretation - very well known in the context of term rewrite systems (TRS) - to obtain new (non-transformational) termination analysis techniques for definite logic programs (LP). We show how this leads to an approach that can be seen as a direct generalisation of the traditional techniques in termination analysis of LP, where linear norms and level mappings are used. Our extension generalises these to arbitrary polynomials. We extend a number of standard concepts and results on termination analysis to the context of polynomial interpretations. We propose a constraint based approach for automatically generating polynomial interpretations that satisfy the termination conditions. Based on this approach, we implement a new tool, called Polytool, for automatic termination analysis of logic programs.

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

For LP, 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 in LP 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 [?] infers one specific linear norm and linear level mapping. The analysers TermiLog [?], TerminWeb [?], and cTI [?] use a concrete linear norm (TermiLog, cTI) or a combination of several linear norms (TerminWeb) to obtain an approximation of the program and then infer a linear level mapping for termination analysis of the latter. 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, introduced in [?], is inspired by a similar term rewriting example from [?].

Example 1 (der).

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

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

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

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

We are interested in proving termination of this program w.r.t. the set of queries $S = \{\, d(t_1, t_2) \mid t_1$ is a ground term and t_2 is an arbitrary term$\}$. It turns out that all the existing, non-transformational termination analysers for LP, mentioned above, fail to prove termination for 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 TRS was first introduced by Lankford in [?]. It is currently one of the best known and most widely used techniques in TRS termination analysis.

We develop the approach within an LP context. We redefine and extend several known concepts and results from LP termination analysis to polynomial interpretations. We show how polynomial interpretations can be seen as a direct generalisation of currently used techniques in LP termination based on linear norms and linear level-mappings. Classical approaches in LP termination use interpretations that map to natural numbers (using linear polynomial functions). In contrast, we will use interpretations that map to polynomials (using arbitrary polynomial functions). The concepts that link the two approaches are those of the "abstract norm" and "abstract level mapping" [?]. We show that under this approach, examples as Example 1 can be solved.

We also developed an automated tool (Polytool) for termination analysis based on this approach [?]. We embedded this within the constraint-based ap-

proach developed in [?] and combined it with the non-linear Diophantine constraint solver developed by Fuhs et al. [?] (implemented in the AProVE system [?]) to provide a completely automated system.

The paper is organised as follows. In the next section, we present some preliminaries. In Section 3, we introduce the notion of polynomial interpretations in logic programming. We show how this approach can be used to prove termination with some examples. 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 LP 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 LP [?,?]. In the following, P denotes a definite logic program. We use Var_P, Fun_P, and $Pred_P$ to denote the set 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 $p \simeq q$. Let $Term_P$ and $Atom_P$ denote, respectively, the sets of all terms and atoms that can be constructed from P. Given two expressions E and F (terms, atoms, n-tuples of terms, or n-tuples of atoms), we denote by $mgu(E, F)$ their most general unifier.

In this paper, we focus our attention on definite logic programs and SLD-derivations where the left-to-right selection rule is used. Such derivations are referred to as 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 [?]). In the following, we usually speak of "termination" instead of "LD-termination" or "left-termination".

2.2 Norms and Level Mappings

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

Definition 2 (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 [?]. One of the most commonly used norms is the *list-length norm* which maps lists to their lengths and any other term to 0. Another frequently used norm is *term-size* 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 3 (linear norm and level mapping [?]). *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 $n \geq 0$.*

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 $n \geq 0$.*

2.3 Conditions for Termination w.r.t. General Orders

A *quasi-order* on a set S is a reflexive and transitive binary relation $\succsim$ defined on elements of S. We define the *associated equivalence relation* $\approx$ as $s \approx t$ if and only if $s \succsim t$ and $t \succsim s$. 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* $(\succsim, \succ)$ consists of a quasi-order $\succsim$ and a well-founded order $\succ$ that are *compatible* (i.e., $t_1 \succsim t_2 \succ t_3$ implies $t_1 \succ t_3$). We also need the following notion of a call set.

Definition 4 (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, the set S of queries of interest is infinite. 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 $Call(P, S)$ will need to be computed, using a mode or a type inference technique (e.g., [?,?,?]).

In order to obtain a termination criterion that is suitable for automation, one usually estimates the effect of the body atoms $B_1, \ldots, B_{i-1}$ by suitable *interargument relations*. This notion can be defined for reduction pairs.

Definition 5 (interargument relation - adapted from [?]). *Let P be a program, p be a predicate in P, and $(\succsim, \succ)$ be a reduction pair on $Term_P$. An* interargument relation *for p in P w.r.t. $(\succsim, \succ)$ is a relation R_p with the same arity as p, $R_p = \{p(t_1, \ldots, t_n) \mid for\ i = 1, \ldots, n : t_i \in Term_P \wedge \varphi_p(t_1, \ldots, t_n)\}$, where:*

- *$\varphi_p(t_1, \ldots, t_n)$ is a boolean expression (in terms of disjunction, conjunction and negation) of inequalities $s \succsim s'$ 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.* $(\succsim, \succ)$ *if and only if for every* $p(t_1, \dots, t_n) \in Atom_P$*:* $P \models p(t_1, \dots, t_n)$ *implies* $p(t_1, \dots, t_n) \in R_p$*.*

Finally, we need the notion of *rigidity*, in order to avoid backpropagation of bindings through unification in LD-derivations. We reformulate rigidity for reduction pairs.

Definition 6 (rigidity - adapted from [?]). *A term or atom* $A \in Term_P \cup Atom_P$ *is called* rigid *w.r.t. a reduction pair* $(\succsim, \succ)$ *if* $A \approx A\sigma$ *holds for any substitution* σ*. A set of terms (or atoms)* S *is called* rigid *w.r.t.* $(\succsim, \succ)$ *if all its elements are rigid w.r.t.* $(\succsim, \succ)$*.*

Example 7 (rigidity). The list $[X|t]$ (X is a variable, t is a ground term) is rigid w.r.t. the reduction pair $(\succsim, \succ)$ corresponding to the list-length norm $\|.\|_\ell$, i.e., $t_1 \succsim t_2$ if and only if $\|t_1\|_\ell \geq \|t_2\|_\ell$, $t_1 \succ t_2$ if and only if $\|t_1\|_\ell > \|t_2\|_\ell$. 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. $(\succsim, \succ)$.

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

We can now recall the definition of *rigid order-acceptability* w.r.t. a set of atoms.

Definition 8 (rigid order-acceptability - adapted from [?]). *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* $(\succsim, \succ)$ *on* $Atom_P$ *where* $Call(P, S)$ *is rigid w.r.t.* $(\succsim, \succ)$ *and where for each predicate* p *in* P*, there is a valid interargument relation* R_p *in* P *w.r.t.* $(\succsim, \succ)$ *such that*

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

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

Rigid order-acceptability is a sufficient condition for termination.

Theorem 9 (termination criterion with rigid order-acceptability [?]). *If* P *is rigid order-acceptable w.r.t.* S*, then* P *terminates for any query in* S*.*

We refer to [**?**], Theorems 3.32 and 3.54, for the proof of Theorem .

Rigid order-acceptability is sufficient for termination, but is not necessary for it (see [**?**]). With Definition 8 and Theorem 2.3, proving termination of a program requires verifying the rigidity of the call sets and verifying the validity of interargument relations for predicates and the decrease conditions for the (mutually) recursive clauses.

Note that, in general, the question of whether rigidity of the call set and validity of interargument relations are decidable is hard to answer. The decidability depends on the particular setting of the analysis: Is the order given or is an order infered? If it is infered, what kind of orders are considered? Is the set S given or is a set of terminating queries infered? What kind of abstraction is used to finitely represent the set S and to infer the call set (modes, types)? What expressivity is allowed for expressing the interargument relations? Most of the research works on LP termination analysis of the past 25 years provide partial answers to the question.

In the remainder of this paper we provide some answers to the question in the setting of a given set S, an infered order based on polynomial interpretations, abstractions of S based on types, with type inference for the call set and interarguments based on inequations of 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 LP based on a general orders on terms and atoms. In this section, we specialise it to orders based on polynomial interpretations.

We first introduce polynomial interpretations in Section 3.1. Then in Section 3.2 we re-formulate the termination conditions for LP from Section 2.3 for polynomial interpretations.

3.1 Polynomial Interpretations

In this paper, we only consider polynomials with natural numbers as coefficients. If $X_1, \ldots, X_n$ are all the variables occuring in a polynomial p, we will often denote p as $p(X_1, \ldots, X_n)$. There is an associated polynomial function, $F_p : \mathbb{N}^n \to \mathbb{N}$, $P_p = \lambda X_1, \ldots, X_n \ . \ p(X_1, \ldots, X_n)$. For natural numbers $x_1, \ldots, x_n$, $p(x_1, \ldots, x_n)$ denotes the function value $F_p(x_1, \ldots, x_n)$.

Given $p(X_1, \ldots, X_n)$ and m extra variables, $Y_1, \ldots, Y_m$, we also have an associated polynomial function $F_{p,m} : \mathbb{N}^{n+m} \to \mathbb{N}$, $F_{p,m} = \lambda X_1, \ldots, X_n, Y_1, \ldots, Y_m$. $p(X_1, \ldots, X_n)$. For such an associated function on an extended domain, we will write $p(x_1, \ldots, x_n, y_1, \ldots, y_m)$ to denote $F_{p,m}(x_1, \ldots, x_n, y_1, \ldots, y_m) = p(x_1, \ldots, x_n)$.

Definition 10 (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 $\succsim_{\mathbb{N}}$ is defined as $p \succsim_{\mathbb{N}} q$ if and only if $p(x_1, \ldots, x_n) \geq q(x_1, \ldots, x_n)$ 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 $(\succsim_{\mathbb{N}}, \succ_{\mathbb{N}})$ is a reduction pair. That is: $\succ_{\mathbb{N}}$ is well-founded, $\succsim_{\mathbb{N}}$ is reflexive and transitive, and $\succsim_{\mathbb{N}}$ and $\succ_{\mathbb{N}}$ are compatible. With Σ we denote the set of all polynomials with natural coefficients.

Given a polynomial, in addition to the polynomial function, we can also associate to it a function on polynomials.

Definition 11 (polynomial to polynomial mapping). *Let $p(X_1, \ldots, X_n)$ be a polynomial, the associated* polynomial to polynomial mapping*, $Gp : \Sigma^n \to \Sigma$, is the function $Gp = \lambda p_1, \ldots, p_n \ . \ p(X_1 \to P_1, \ldots, X_n \to p_n)$, where $X \to p$ denotes replacement of X by p.*

A *polynomial interpretation* associates to each functor and each predicate symbol of the program a polynomial to polynomial mapping associated to a polynomial.

Definition 12 (polynomial interpretation). *A* polynomial interpretation *I for a logic program P associates to each functor or predicate symbol f of arity n in $Fun_P \cup Pred_P$ a polynomial to polynomial mapping Gp_f of arity n.*

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

Definition 13 (polynomial level mapping). *The* level mapping *associated with a polynomial interpretation I, is a mapping $|.|_I : Atom_P \cup Term_P \to \Sigma$, which is defined recursively as:*

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

Every polynomial interpretation induces corresponding orders.

Definition 14 (reduction pair corresponding to polynomial interpretation). *Let I be a polynomial interpretation. We define the relations $\succsim_I$ and $\succ_I$ on $Term_P \cup Atom_P$ as follows:*

- $s \succsim_I t$ *if and only if* $|s|_I \succsim_{\mathbb{N}} |t|_I$ *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.

3.2 Termination of Logic Programs by Polynomial Interpretations

We now re-state Definition 8 and Theorem 2.3 for the special case of polynomial interpretations.

Instead of interargument relations for arbitrary orders, we now use interargument relations *w.r.t. polynomial interpretations.*

Definition 15 (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 iff R_p is an interargument relation for p in P w.r.t. $(\succsim_I, \succ_I)$.*

Instead of rigidity w.r.t. general orders as in Definition 6, we define *rigidity w.r.t. polynomial interpretations.*

Definition 16 (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 *iff A is rigid w.r.t. $(\succsim_I, \succ_I)$, i.e., iff $A \approx_I A\sigma$ 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 17 (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 $\{t \to X\}$ of a term t for X such that $A\{t \to X\} \not\approx_I A$.*

Example 18 (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 19 (alternative characterisation of rigidity). *Let I be a polynomial interpretation and A be a term or atom. Then A is rigid w.r.t. I iff A has no relevant variables w.r.t. I.*

Proof. Obvious from Definitions 6 and 17. □

Using the notions of interargument relations and rigidity w.r.t. a polynomial interpretation, as a refinement of Theorem 2.3 we obtain:

Corollary 20 (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 $rel(B_i) \simeq rel(A)$,*
- *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 2.3. □

Corollary 20 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 21 (a procedure for automatic termination analysis) *The termination proof procedure derived from Corollary 20 contains the following four 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 intermediate 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.*

Regarding step 2, following the standard approach in LP for verifying whether a relation R holds for all elements of the Herbrand model (see e.g. [**?**]), we need to verify that $T_P(R) \subset (R)$, where T_P is the immediate consequence operator. 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 relation for the head should also hold.

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 polynomial associated with each function and predicate symbol 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 [**?**], we do not choose a particular polynomial interpretation and particular interargument relations. Instead, we introduce a general symbolic form for the polynomial associated with each function and predicate symbol 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 to polynomial mapping Gp_q, with $q(X, Y) = X^2 + 2XY$ to a predicate symbol q of arity 2, we would for example assign the Gp_q of $q(X, Y) = q_{00} + q_{10}X + q_{01}Y + q_{11}XY + q_1X^2 + q_2Y^2$, where the q_i and q_{ij} are unknown symbolic coefficients ranging over $\mathbb{N}$. The strategy of the analysis is to:

- introduce symbolic versions of the polynomials associated with each function and predicate symbol,
- express all conditions resulting from Corollary 20 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 a concrete valid interargument relation such that all conditions of Corollary 20 are satisfied. Therefore, each solution gives a termination proof.

To assign symbolic functions of polynomials to the functor and predicate symbols, we have to restrict ourselves to fixed types of polynomials, since there does not exist a symbolic polynomial, with finitely many monomials and coefficients, that allows to represent all possible concrete polynomials. Specifically, we

will associate linear functions of polynomials to predicate symbols and linear or simple-mixed functions of polynomials to functor. These classes of polynomials are defined as follows:

- *The linear class:* each monomial of a polynomial in this class contains at most one variable of at most degree 1:
  $p(X_1, \ldots, X_n) = p_0 + \sum_{k=1}^{n} p_k * X_k$
- *The simple-mixed class:* each monomial of a polynomial in this class consists of either a single variable of at most degree 2 or several variables of at most 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 [**?**,**?**].

In Section 4.1, we first reformulate the conditions of our termination criterion in Corollary (20) using the above symbolic forms of polynomials. Then in Section 4.2, we transform these symbolic conditions into constraints on the symbolic coefficients of the polynomials. Afterwards, in Section 4.3 we show how these resulting *Diophantine* constraints can be solved automatically.

4.1 Reformulating the Termination Conditions Symbolically

In this subsection, we reformulate all termination conditions in Corollary (20), i.e., in Procedure 21. 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 21, step 1)

There are several ways to approximate $Call(P, S)$ [**?**,**?**,**?**,**?**]. In this paper, we apply the approximation technique of [**?**]. More precisely, we first specify the set of queries as a set of rigid type graphs. Each rigid type graph represents a certain set of atoms. Then the technique in [**?**] 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 [**?**].

In the following, we recall 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 [**?**], which are based on linear norms and level-mappings, to the case of general polynomial interpretations.

Definition 22 (rigid type graph [?]). *A rigid type graph T is a 5-tuple, $(Nodes, ForArcs, BackArcs, Label, ArgPos)$, where*

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

This and the following concepts are illustrated in Example 26.

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 iff 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 23 (critical path [?]). *Let* $T = $ (*Nodes*, *ForArcs*, *BackArcs*, *Label*, *ArgPos*) *be a rigid type graph. A critical path in* T *is a path of arcs from ForArcs which goes from the root node to a node labeled* **MAX**.

The following proposition is extended from [?], where in [?] each function and predicate symbol is associated with a linear norm or level mapping. It provides a method to generate constraints for rigidity.

Proposition 24 (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 functor or predicate symbol* f/k *we have* $I(f) = Gp_f$ *with* $p_f(X_1, \ldots, X_k) = \sum_{0 \leq j_1, \ldots, j_k \leq M_f} f_{j_1 \ldots j_k} X_1^{j_1} \ldots X_k^{j_k}$, *the associated polynomial to polynomial mapping. The set* S *is rigid w.r.t.* I *iff on every critical path of* T *there exists an arc* (n, m) *with Label*$(n) = f/k$ *and ArgPos*$(n, m) = i$ *such that* $\sum_{j_i > 0} f_{j_1 \ldots j_k} = 0$.

Proof. Since we only regard polynomials with non-negative 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 19, this is equivalent to rigidity w.r.t. I. □

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 25 (symbolic condition for checking rigidity). *Let T represent 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 $d \in \{1, \ldots, e\}$, $Label(n^d) = f^d/k^d$ is a function or predicate symbol and $ArgPos(n^d, m^d) = i^d$. If for any such CP we have*

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

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

Example 26 (symbolic polynomial interpretation and rigidity constraints for the "der"-program). For the "*der*"-program (Example 1), we define the symbolic form for the polynomial interpretation I as follows. For each functor, constant and predicate symbol, we provide the symbolic polynomial on which the associated polynomial to polynomial mapping is based $I(+) : p_1X_1^2 + p_2X_2^2 + p_{11}X_1X_2 + p_{10}X_1 + p_{01}X_2 + p_{00}$,
$I(*) : m_1X_1^2 + m_2X_2^2 + m_{11}X_1X_2 + m_{10}X_1 + m_{01}X_2 + m_{00}$,
$I(der) : der_2X^2 + der_1X + der_0$,
$I(u) : c_u$,
$I(1) : c_1$,
$I(d) : d_0 + d_1X_1 + d_2X_2$.

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

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, $+/2$, $*/2$ and $der/1$, and t_2 is an arbitrary term$\}$. S_1 is represented in the type graph in Fig. 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 $Q \in S \setminus S_1$ (i.e. $Q \in S$ and $Q \notin S_1$), the program trivially terminates by finite failure. More generally, as rigid-type graphs can represent richer (or at least equally rich) sets of queries than modes, one can use a simple and safe transformation from termination analysis w.r.t. mode-based queries to termination analysis w.r.t. type-based queries: each ground argument of a mode-based query pattern is replaced by the rigid-type graph associated with this ground argument.

In our example, type inference ([?]) computes the call set $Call(P, S_1) = S_1$. From the graph, the following rigidity condition is generated:

$$d_2 = 0$$

□

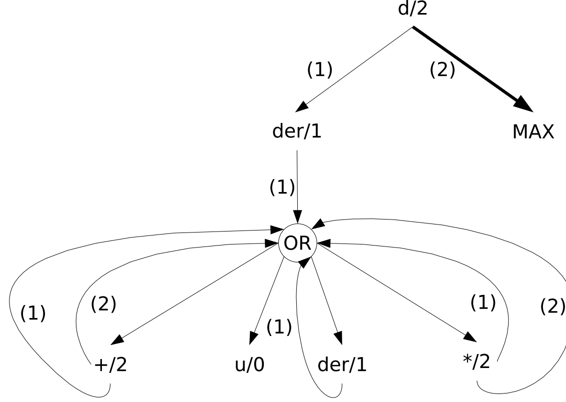

Fig. 1. Rigid type graph for S

4.1.2 Valid Interargument Relations (Procedure 21, step 2)

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

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

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

There are a number of works on inferring valid interargument relations of predicates. In [?], 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, since we will not use tham in our approach, but informally, inputs are the arguments of the predicate 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:

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

where i/n and o/n are polynomials with non-negative integer coefficients.

The form of interargument relations in [?] can be considered a special case of the form (7) above, where $i(|t_1|_I, \ldots, |t_n|_I)$ is constructed from the input arguments and $o(|t_1|_I, \ldots, |t_n|_I)$ is constructed from the outputs.

Since the approach in [?] only considers relations between the input and output arguments of the predicates, it has some limitations. In some cases, the desired relation is not between the inputs and outputs, but among the inputs and the output themselves. In particular, if all arguments of a predicate are inputs (or outputs), then the approach in [?] fails to infer any useful relation among them. The following example that calculates the natural division of the first and second arguments of the predicate $div/3$ and returns the result in its third argument, shows this point.

Example 27 (div).

$$
\begin{array}{l}
div(X, s(Y), 0) :- \; less(X, s(Y)). \\
div(X, s(Y), s(Z)) :- \; sub(X, s(Y), R), div(R, s(Y), Z). \qquad (8) \\
sub(X, 0, X). \\
sub(s(X), s(Y), Z) :- \; sub(X, Y, Z). \\
less(0, s(Y)). \\
less(s(X), s(Y)) :- \; less(X, Y).
\end{array}
$$

We regard 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 valid interargument relation for $sub/3$ such that the size of its first argument is greater than that of its third argument. However, if we apply the approach in [?], where a symbolic form of valid interargument relations based on modes is used, inferring such an interargument relation for $sub/3$ 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 or equal than the size of the third argument. Then, we could not conclude that for any successful answer substitution for the call $sub(X, s(Y), R)$ in Clause (8), the size of the first argument X is strictly greater than the size of the third argument R.

In contrast, if we use Form (7), then it is possible to infer the following valid interargument relation for $sub/3$:

$$R_{sub/3} = \{sub(t_1, t_2, t_3) \mid |t_1|_I \succsim_{\mathbb{N}} |t_2|_I + |t_3|_I\}$$

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 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 $|s(Y)|_I \succsim_{\mathbb{N}} 1$. Our implementation in the system Polytool is indeed able to infer this interargument relation, because, through constraint solving, it is able to find any instance of (7), given that the functions i/n and o/n satisfy some restrictions stated below. It is therefore also able to prove termination of "*div*" If we used the form of interargument relations in [?] instead, Polytool would not be able to solve this problem. $\Box$.

In our experiments with several collections of programs, there are also several other examples that cannot be proved terminating by the technique in [?] due to the restriction of the form of interargument relations based on modes.

Similar to the symbolic form of polynomial interpretations, we also need to restrict the symbolic form of polynomial interargument relations. To this end, we use the symbolic forms of linear or simple-mixed polynomials for the left- and right-hand side polynomials i/n and o/n.

Example 28 ("der"-program continued: symbolic interargument relation). We continue Example 26. For the "*der*"-program, we use linear polynomials for the symbolic form of the polynomial interargument relation of the predicate $d/2$:

$$R_{d/2} = \{d(t_1, t_2) \mid i_0 + i_1|t_1|_I + i_2|t_2|_I \succsim_{\mathbb{N}} o_0 + o_1|t_1|_I + o_2|t_2|_I\},$$

where $i(X, Y) = i_0 + i_1X + i_2Y$ corresponds to the left-hand side polynomial of the interargument relation and $o(X, Y) = o_0 + o_1X + o_2Y$ corresponds to the right-hand side polynomial of the relation. □

For the inference of valid interargument relations, we use the technique proposed in [**?**], cf. Procedure 21, step 2. More precisely, for every predicate p, we use two symbolic polynomials i_p and o_p in order to define the interargument relation.

$$R_{p/n} = \{p(t_1, \ldots, t_n) \mid i_p(|t_1|_I, \ldots, |t_n|_I) \succsim_{\mathbb{N}} o_p(|t_1|_I, \ldots, |t_n|_I)\} \qquad (9)$$

For any sequence of terms $t_1, \ldots, t_n$, let $\mathbf{R}_p(t_1, \ldots, t_n)$ abbreviate the inequality $i_p(|t_1|_I, \ldots, |t_n|_I) \geq o_p(|t_1|_I, \ldots, |t_n|_I)$.

The goal is to impose constraints on all the polynomials i_p and o_p which ensure that the corresponding interargument relations are valid. To this end, we generate for every clause of the program:

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

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 29 ("der"-program continued: constraints for the symbolic interargument relation). There are four clauses (1) - (4) from which valid interargument conditions are inferred. We only present the conditions resulting from the last clause (4):

$$d(der(der(X)), DDX) :- d(der(X), DX), d(der(DX), DDX)$$

Here, we obtain the constraint

$$\begin{aligned} &\forall X, DX, DDX \in \mathbb{N} : \\ &\mathbf{R}_d(der(X), DX) \wedge \mathbf{R}_d(der(DX), DDX) \Rightarrow \mathbf{R}_d(der(der(X)), DDX), \quad (10) \end{aligned}$$

□

4.1.3 Decrease Conditions (Procedure 21, 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

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

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

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

Obviously, the formula is in Form (6).

Example 30 ("der"-program continued: constraints for the decrease conditions). There are 3 recursive clauses where decrease conditions can be inferred. We present the decrease condition for the recursive body atom $d(der(DX), DDX)$ of the last clause:

$$\begin{aligned}
&\forall X, DX, DDX \in \mathbb{N}: \\
&i_0 + i_1(der_2X^2 + der_1X + der_0) + i_2DX \geq \\
&o_0 + o_1(der_2X^2 + der_1X + der_0) + o_2DX \\
&\Rightarrow \\
&d_0 + d_1(der_2(der_2X^2 + der_1X + der_0)^2 + \\
&\qquad der_1(der_2X^2 + der_1X + der_0) + der_0) + d_2DDX \geq \\
&d_0 + d_1(der_2DX^2 + der_1DX + der_0) + d_2DDX + 1.
\end{aligned} \tag{11}$$

□

4.2 From Symbolic Conditions to Constraints on Coefficients

Our goal is to find a polynomial interpretation such that all symbolic conditions generated in the previous section are satisfied. To this end, we transform all these conditions into a set of Diophantine constraints. In this transformation, we first eliminate implications, cf. Section 4.2.1. Afterwards, in Section 4.2.2, the universally quantified variables (e.g., $X, DX, DDX, \ldots$) are removed and the former symbolic 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 analysed in Section 4.1.1, all generated rigidity constraints have Form (5). Hence, these are already Diophantine constraints which only contain symbolic coefficients, but no universally quantified variables.

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

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

where $n \geq 0$ 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 symbolic 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

$$\forall \overline{X} \in \mathbb{N} : \quad p \geq 0. \tag{12}$$

However, here p is a polynomial with integer (i.e., possibly negative) coefficients. The transformation is *sound*: if the new constraints of Form (12) are satisfied by some substitution of the symbolic coefficients to natural numbers, then this mapping 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 $p_i \geq q_i$. We first transform them into constraints with at most one premise. Obviously, $p_1 \geq q_1 \wedge \ldots \wedge p_n \geq q_n$ implies $p_1 + \ldots + p_n \geq q_1 + \ldots q_n$. Thus, instead of (6), it would be sufficient to demand

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

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

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

A similar method was also used for termination analysis of logic programs in [**?**] and for termination of term rewriting in [**?**, Section 7.2] to transform disjunctions of polynomial inequalities into one single inequality.

For example, the constraint

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

can now be transformed into

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

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/n in order to combine the n inequalities in the premise. The reason is that if prem/n 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

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

would be transformed into the invalid constraint

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

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

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

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., $\mathit{prem}(X_1, X_2) = X_1 + X_2^2$).

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

$$\mathit{prem}(p_1, \ldots, p_n) \geq \mathit{prem}(q_1, \ldots, q_n) \;\Rightarrow\; p_{n+1} \geq q_{n+1}, \tag{13}$$

it is sufficient to demand

$$p_{n+1} - q_{n+1} \geq \mathit{prem}(p_1, \ldots, p_n) - \mathit{prem}(q_1, \ldots, q_n). \tag{14}$$

This observation was already used in the work of [**?**] and also in termination techniques for term rewriting to handle such conditional polynomial inequalities [**?**,**?**].

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

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

With the transformation of (13) to (14) above, it would be transformed into the unconditional constraint

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

which however is invalid. We have encountered several examples of this type in our experiments, which motivates a further extension. In this example and similar ones, it would be better to apply a suitable polynomial $\mathit{conc}/1$ to the polynomials X and 1 in the former conclusion. Then we would obtain

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

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

So to summarize, in the first phase of our transformation, any constraint of the form

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

is transformed into the unconditional constraint

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

Here, prem/n and $\mathit{conc}/1$ are two arbitrary new polynomials. The only requirement that we have to impose is that $\mathit{conc}/1$ must not be a constant. Of course, the above unconditional constraint can easily be transformed further into an inequality with 0 on its right-hand side. The following proposition proves the soundness of this transformation.

Proposition 31 (Soundness of Removing Implications). *Let prem/n and $\mathit{conc}/1$ be two polynomials with natural coefficients, where $\mathit{conc}/1$ 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*

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

is valid, then

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

is also valid.

Proof. For any tuple of natural 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 $p(X, Y)$ is the polynomial $X_1^2 + 2X_1X_2$, then $p(2, 1) = 8$.

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

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

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

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

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

Now suppose that $p_{n+1}(\overline{x}) \not\geq q_{n+1}(\overline{x})$. 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 non-negative 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

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

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

An good choice for the symbolic form of $prem/n$ and $conc/1$ is again to use linear, or simple-mixed polynomials. By applying Proposition 31, we can now transform all constraints for the termination proof to unconditional constraints of the form (12). If there exists a mapping from the symbolic coefficients to natural numbers that makes the resulting constraints valid, then the same mapping also satisfies the original constraints.

Example 32 (applying Proposition 31 to the "der"-program). We continue with the termination proof of the "*der*"-program. We choose the decrease condition (11) in Example 30 as an example showing how to transform an implication to an unconditional constraint.

Since the constraint (11) 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*:

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

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

$$conc_1 > 0.$$

Now we can transform (11) into an unconditional constraint. Here, we use the following abbreviations:

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

Then (11) is the constraint

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

and its transformation yields

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

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

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

where $M_1, \ldots, M_9$ are expressions without the universally quantified variables X and DX. In other words, they are polynomials over the symbolic coefficients $conc_j$, $prem_j$, i_j, o_j, der_j, and d_j with $j \in \{0, 1, 2\}$. For example, we have

$$M_1 \;=\; conc_1 * d_1 * der_2^3 + prem_2 * o_1^2 * der_2^2 - prem_2 * i_1^2 * der_2^2.$$

□

4.2.2 Second Phase: Removing Universally Quantified Variables

In this phase, we transform any constraint of the form

$$\forall \overline{X} \in \mathbb{N}: \quad p \geq 0 \tag{12}$$

to a set of Diophantine constraints on symbolic coefficients. The transformation is again *sound*: if there is a solution for the resulting set of Diophantine constraints (i.e., a suitable mapping from symbolic coefficients to natural numbers), then this mapping also satisfied the original constraint (12).

We use a straightforward transformation proposed by [**?**], which is also used in all related tools for termination of term rewriting. One only requires that all coefficients of the polynomial p are non-negative. Obviously, the criterion is only sufficient (i.e., the "only if" direction does not hold). For example $p(X) = (X-1)^2 \geq 0$, but $X^2 - 2X + 1$ does not have non-negative coefficients only.

Example 33 (removing universally quantified variables for the "der"-program). We continue the transformation of Example 32. Here, we obtained the constraint

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

We derive the following set of Diophantine constraints which contains symbolic coefficients $conc_j$, $prem_j$, i_j, o_j, der_j, and d_j as variables: $M_1 \geq 0, M_2 \geq 0, \ldots, M_9 \geq 0$. □

4.3 Solving Diophantine Constraints

The previous sections showed that one can formulate all termination conditions in symbolic form and one can transform them automatically to a set of Diophantine constraints. The problem then becomes solving a system of non-linear Diophantine constraints with the symbolic coefficients as variables. If the 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., [**?**,**?**]. In [**?**], Diophantine constraints are encoded as a SAT-problem, and then a back-end SAT solver is used to solve the resulting SAT-problem. As shown in [**?**], this approach is significantly more efficient than solving Diophantine constraints by dedicated solvers like [**?**] or by standard implementations of constraint logic programming like in SICStus Prolog.

Example 34 (solving Diophantine constraints for the "der"-program). We start with the symbolic polynomial interpretation from Example 26 (e.g., with $I(der) : der_2 X^2 + der_1 X + der_0$) and obtain the solution $der_2 = 2$ and $der_1 = 1$ $der_0 = 2$

which corresponds to $2X^2 + X + 2$. Similarly, we start with the symbolic form of the polynomial interargument relation as in Example 28:

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

Then we obtain the solution $i_1 = 2$, $i_0 = i_2 = 0$, $o_0 = 1$, $o_2 = 2$, and $o_1 = 0$, which corresponds to the interargument relation $R_d = \{d(t_1, t_2) \mid 2|t_1|_I \succsim_{\mathbb{N}} 1 + 2|t_2|_I\}$. In the transformation of the implication constraints, we used the coefficients of the polynomials $prem(X) = prem_0 + prem_1 X + prem_2 X^2$ and $conc(X) = conc_0 + conc_1 X$. We obtain the solution $prem_2 = 1$, $prem_1 = 1$, $prem_0 = 0$, $conc_1 = 1$, and $conc_0 = 0$, i.e., $prem(X) = X^2 + X$ and $conc(X) = X$. We have used simple-mixed non-linear polynomials both in the polynomial level mapping and for polynomials like *prem* from the transformation. □

5 Experimental Evaluation

In this section we discuss an experimental evaluation of the presented approach. The technique was implemented in a system called Polytool.[4] The work reported on in this paper (i.e., the development of the underlying theory, the design of the automation, the implementation and extensive experiments), has been performed over a period of several years, from 2004 to 2007. As a result, the system has evolved considerably during this period. Essentially, the Polytool system consists of four modules:

Until recently, the first module used in Polytool was the type inference engine of Janssens and Bruynooghe [?]. Recently, we have replaced this module by the type inference system of Gallagher, Henriksen, and Banda [?], based on well-typings [?]. This system allows to infer the same information as the one of [?], but it is more efficient, and more importantly, it is implemented in SICStus Prolog, which is generally available.

The second module is the core of the system. It generates the set of all termination conditions using symbolic polynomials as in Section 4.1. This part was also implemented in SICStus Prolog.

The third module of Polytool transforms the resulting polynomial constraints into Diophantine constraints, based on the approach discussed in Section 4.2. It is again implemented in SICStus Prolog.

The final module is a back-end Diophantine constraint solver, cf. Section 4.3. We selected the SAT-based Diophantine solver [?] of the AProVE tool [?]

We tested the performance of Polytool on 296 examples. The collection (Table 1) includes all benchmarks for logic programming from the *Termination Problem Data Base* 2007 (TPDB 4.0) [?], where all examples that contain arithmetic or built-in predicates were removed The Termination Problem Data Base contains examples from several sources, and is used in the annual *International Competition of Termination Tools* [?].

[4] For the source code, we refer to `http://www.cs.kuleuven.be/~manh/polytool`.

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 precisely, then we still interpret predicate symbols by linear polynomials, but we map function symbols to simple-mixed polynomials. We use similar symbolic polynomials for *conc/1* and *prem/n* from Section 4.2.1: if the polynomial interpretation is linear, then both *conc/1* and *prem/n* are linear. Otherwise, we use the linear form for *conc/1* and the simple-mixed form for *prem/n*. The domain for all symbolic 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 [?], TerminWeb [?], TALP [?], (which uses CiME 2.02 [?] as a back-end), and AProVE [?]. Polytool-WST07 is the version of Polytool that participated in the *International Competition of Termination Tools* 2007. Similarly, the version of AProVE used in our experiments is also the variant that participated in this competition. The only difference between Polytool-WST07 and the version of Polytool in this paper is that Polytool-WST07 applies the groundness analysis in [?] and linear polynomial interpretations instead of type analysis and simple-mixed polynomial interpretations. For TALP, the option of non-linear 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 [?] 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 corresponds to the rules which have been applied for evaluating tools in the annual termination competition in recent years (2004-2006) [?].

In Table 1, we give the number 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/~manh/polytool/`.

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 analysed 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

	TALP	cTI-1.1	TerminWeb	Polytool-WST07	Polytool	AProVE
YES	163 [2.54]	167 [0.06]	177 [0.54]	204 [3.05]	214 [4.28]	232 [6.34]
FAILURE	112 [1.45]	129 [0.05]	118 [0.6]	82 [2.96]	62[10.48]	57 [19.08]
TIMEOUT	21	0	1	10	20	7

Table 1. The results for 296 benchmarks of the TPDB 4.0

$\mathbb{N}$ according to a fixed symbolic norm (e.g., the symbolic[5] term-size norm), and a program in $\mathsf{CLP}(\mathbb{B})$, where $\mathbb{B}$ denotes the booleans, which is obtained from the program in $\mathsf{CLP}(\mathbb{N})$ by mapping any natural number to 1, any variable to itself, and addition to logical conjunction. The purpose of these abstractions is to capture the decrease conditions (the program in $\mathsf{CLP}(\mathbb{N})$) and the boundedness information (the program in $\mathsf{CLP}(\mathbb{B})$) 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 to do so is the example incomplete2.pl in the directory SGST06 of the TPDB 4.0:

$$\begin{array}{l} f(X) :- g(s(s(s(X)))). \\ f(s(X)) :- f(X). \\ g(s(s(s(s(X))))) :- f(X). \end{array}$$

However, if we reset the range for the values of symbolic coefficients in the generated Diophantine constraints to $\{0, \ldots, 8\}$, then Polytool can find the following polynomial interpretation for the termination proof of this program:

$$\begin{array}{l} \phi(f(X)) = X + 8 \\ \phi(g(X)) = X + 1 \\ \phi(s(X)) = X + 2 \end{array}$$

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 (term-size norm by default) to map the analysed program to a program in $\mathsf{CLP}(\mathbb{N})$, for which all termination conditions are formulated. However, in some cases, the selected symbolic norm is not suitable to capture the decrease information hidden in the analysed program. As a result, cTI-1.1 cannot prove termination of such examples. The TPDB 4.0 contains a number of such benchmarks, e.g. flat.pl, normal.pl in the talp directory, countstack.pl, factor.pl, flatten.pl in the SGST06 directory, etc. One of these examples is presented in detail below.

[5] 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).

Example 35 (normal). Consider the following program w.r.t. the set of queries $S = \{normal(t_1, t_2) \mid \text{where } t_1 \text{ is a ground term and } t_2 \text{ is an arbitrary term}\}$:

$$
\begin{aligned}
&normal(F, N) :- \ rewrite(F, F_1), normal(F_1, N).\\
&normal(F, F).\\
&rewrite(op(op(A, B), C), op(A, op(B, C))).\\
&rewrite(op(A, op(B, C)), op(A, D)) :- \ rewrite(op(B, C), D).
\end{aligned}
$$

Using the term-size norm, cTI-1.1 maps this program to the following abstract program in CLP($\mathbb{N}$):

$$
\begin{aligned}
&normal(F, N) :- \ rewrite(F, F_1), normal(F_1, N).\\
&normal(F, F).\\
&rewrite(2 + A + B + C, 2 + A + B + C).\\
&rewrite(2 + A + B + C, 1 + A + D) :- \ rewrite(1 + B + C, D).
\end{aligned}
$$

However, this mapping does not preserve termination of the original program since its abstract counterpart is not terminating w.r.t. the set of queries $S' = \{normal(t_1, t_2) \mid t_1 \text{ is number in } \mathbb{N} \text{ and } t_2 \text{ is an arbitrary term}\}$, which is the approximation of the query set S w.r.t. the term-size norm.

In contrast, Polytool found the following more complex polynomial interpretation:

$$
\begin{aligned}
\phi(normal(X_1, X_2)) &= X_1 + 1\\
\phi(rewrite(X_1, X_2)) &= 2X_1 + 1\\
\phi(op(X_1, X_2)) &= 2X_1 + X_2 + 1
\end{aligned}
$$

and the following valid interargument relation:

$$R_{rewrite/2} = \{rewrite(t_1, t_2) \mid \|t_1\|_I \succsim_{\mathbb{N}} 1 + \|t_2\|_I\}.$$

In this way, termination of this example could be proved.

For users who already have certain knowledge about termination analysis of logic programs, cTI-1.1 has an option that allows to enter a user-defined norm manually. This option provides a semi-automatic way to prove termination of examples which require more complicated norms. For example, if we give cTI-1.1 the norm found above by Polytool, cTI-1.1 can find a termination proof for this example. □

Secondly, when we use the term-size norm (or list-length norm) for the abstract approximation, all constant symbols are mapped to the same number in $\mathbb{N}$ (i.e., all constant symbols are mapped to 1 if we use term-size norm and to 0 if we use list-length norm). As a result, termination analysis based on this approach may not work for examples where the difference among constant symbols plays a role for the termination behavior.

Example 36 (transitive closure). Consider the following transitive closure example and the query pattern $S = \{tc(t_1, t_2) \mid t_1$ is a ground term and t_2 is an arbitrary term$\}$:

$$
\begin{aligned}
&p(a, b).\\
&p(b, c).\\
&tc(X, X).\\
&tc(X, Y) :- p(X, Z), tc(Z, Y). \qquad (17)
\end{aligned}
$$

If the term-size norm is used to abstract the program, then it is impossible to detect the decrease between the head $tc(X, Y)$ and the recursive body atom $tc(Z, Y)$ in Clause (17). Thus, termination of this program cannot be verified.

In Polytool, different constant symbols can be mapped to different numbers in $\mathbb{N}$. Therefore, termination of Example 36 and of other examples such as simple.pl in the talp directory, pl2.3.1.pl in the plumer directory, at.pl in the SGST06 directory, etc. in the TPDB 4.0 could be proved, whereas cTI-1.1 fails. □

Next, since termination analysis of cTI-1.1 is based on linear symbolic norms such as the term-size norm, it cannot prove termination of programs such as Example 1 or the example hbal_tree.pl in the TPDB 4.0. This is also the case for Polytool-WST07 since for this version, we had fixed the settings to linear polynomials only, because in the competition, a small runtime was also an important factor. In contrast, Polytool can prove termination of these examples using simple-mixed polynomial interpretations.

Finally, there are examples whose termination cannot be proved by cTI-1.1 due to the restriction to groundness analysis, e.g., applast.pl, bappend.pl, blist.pl, btappend.pl, btapplast.pl, confdel.pl and btree.pl in the SGST06 directory of the TPDB 4.0. The termination proof of these examples also fails with Polytool-WST07 and TALP, due to 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). Polytool and TerminWeb succeed due to the use of types instead of modes and AProVE succeeds due to its argument filtering. But TerminWeb still fails on some of these examples because it uses a fixed norm for part of its analysis.

Example 37 (blist). We consider termination of the example blist.pl that builds a bounded list w.r.t. the set of queries $\{goal(t) \mid t$ is a ground term$\}$. Here, the predicate $s2l/2$ is used to convert a number n (denoted in the form $s^n(0)$) into a list of length n.

$$
\begin{aligned}
&list([\,]).\\
&list([X|Xs]) :- list(Xs).\\
&s2l(s(X), [Y|Xs]) :- s2l(X, Xs).\\
&s2l(0, [\,]).\\
&goal(X) :- s2l(X, Xs), list(Xs).
\end{aligned}
$$

Because cTI-1.1 uses groundness analysis (which can only distinguish between ground terms and arbitrary terms), for any element of the success set of the predicate *s2l/2*, it can only infer that if the first argument is a ground term, then the second argument is an arbitrary term. As a result, when investigating the set of queries $\{goal(t) \mid t \text{ is a ground term}\}$, then the inferred call set includes $S = \{list(t) \mid t \text{ is an arbitrary term}\}$. cTI-1.1 then fails to prove termination of this example because the program does not terminate w.r.t. S. The main reason for this failure is that the call set inferred by a groundness analysis is too large. Actually, S only has to include the set of bounded (i.e., [ ]-terminated) lists as in (**??**), but groundness analysis fails to capture this.

Using type-based inference in Polytool instead, it is sufficient to reason on the set of queries $S_1 = \{goal(t) \mid t \text{ is a term constructed from the function symbols } s/1 \text{ and } 0\}$, as discussed in Section 4.1.1. Then, the following call set is generated: $S' = S_1 \cup S_2 \cup S_3$, where $S_2 = \{s2l(t_1, t_2) \mid t_1 \text{ is a term constructed from the function symbols } s/1 \text{ and } 0,\ t_2 \text{ is an arbitrary term}\}$, and $S_3 = \{list(t) \mid t \text{ is a bounded list}\}$. Proving termination of the program with the latter call set succeeds in Polytool. □

A strong point of cTI-1.1 is that it is a very fast system. Table 1 shows that it is the fastest one in the experiments. The reason is that cTI-1.1 fixes the norm in advance. Therefore it requires much less symbolic 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 non-termination proofs, which are not supported by the other systems in our experiments.

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 [**?**] to approximate the analysed 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, transitive_closure.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 ranking functions are used for different loops in the program [**?**]. 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 4.0). For termination of such programs, the current global technique based on polynomial interpretations deployed in Polytool is insufficient. We are currently working on an extension using dependency graphs that is able to deal with such programs as well [**?**].

Example 38 (ackermann). Consider the following logic program computing the Ackermann function. We want to prove termination of this program w.r.t. the

set of queries $S = \{ack(t_1, t_2, t_3) \mid t_1$ and t_2 are ground terms, t_3 is an arbitrary term$\}$.

$$ack(0, X, s(X)).$$
$$ack(s(X), 0, Z) :- ack(X, s(0), Z).$$
$$ack(s(X), s(Y), Z) :- ack(s(X), Y, Z'), ack(X, Z', Z).$$

Proving termination of this example based on the local approach involves two ranking functions: the first one measures the size of the first argument and the other measures the size of the second argument of the predicate $ack/3$. However, with the global approach based on polynomial interpretations, it is impossible to find a single ranking function for the termination proof. TerminWeb can prove termination of this example, whereas both cTI-1.1 and Polytool fail. □

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 analysed 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 a 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 to 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 [?]. As a matter of fact, there are many non well-moded examples in the TPDB 4.0 that can be solved by most other tools, both direct and indirect, but not by TALP, e.g., map1.pl, zebra.pl in the talp directory, som.pl, transpose.pl, preorder_dl.pl in the terminweb directory, append-bff.pl, delete-bbf.pl, delete-bfb.pl, delete-fbf.pl, delete-ffb.pl, delmin-bff.pl, delmin-ffb.pl, flatlength-bfb.pl, flatlength-bff.pl, in-fb.pl, insert-fbf.pl, insert-ffb.pl, less-bf.pl, map_color.pl, maximum-bff.pl, maximum-fbf.pl, prefix-bf.pl, select-ffb.pl, sum-fbf.pl in the toplas directory, and pplus.pl, pplus2.pl, preorder.pl, giesl97.pl in the SGST06 directory.

AProVE instead applies a quite strong transformational approach which can also deal with non well-moded logic programs [?]. Together with the strong 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 for Termination Tools 2004-2007, AProVE was always in the first place. In particular, it can prove termination of most examples whenever some other tool can. Nevertheless, there exists one example in the TPDB 4.0 (i.e., incomplete.pl)

where AProVE fails to prove its termination but Polytool succeeds. In general, the main 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. Hence, if one also adapts further more sophisticated techniques from term rewriting to the logic programming domain, then Polytool might indeed become more powerful than AProVE.

Example 39 (incomplete). Consider the following example incomplete.pl, with the query set $S = \{p(t) \mid t \text{ is a ground term}\}$:

$$
\begin{aligned}
&p(X) :\!-\; q(f(Y)), p(Y).\\
&p(g(X)) :\!-\; p(X).\\
&q(g(Y)).
\end{aligned}
$$

AProVE fails to prove termination of this program due to a "variable condition" imposed on the translated term rewrite system which is violated in this example. For more details, we refer to http://aprove.informatik.rwth-aachen.de/eval/LP/. Polytool, in contrast, can prove termination of this example by checking the failure of the call $q(f(Y))$ in the first clause, where Y is a free variable. □

Similar to the other termination tools in the experiments (i.e., 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 so-called *argument filterings* to remove argument positions of function and predicate symbols which are irrelevant for termination. Nevertheless, in some cases, the effect of argument filterings is not "deep" enough to represent redundant argument positions adequately. The following example shows this point.

Example 40 (btranspose). Consider the following program which constructs a matrix and transposes it to a new one where the rows of the former become the columns of the latter. As in Example 37, the predicate $s2l/2$ is used to convert a number into a list. Similarly, the predicate $s2m/3$ converts two numbers x and y into a matrix with x rows and y columns. We want to verify termination of the program w.r.t. to the set of queries $S_1 = \{goal(t_1, t_2, t_3) \mid t_1, t_2$ are ground

terms, and t_3 is an arbitrary term}.

$$
\begin{aligned}
&goal(X,Y,M_t) \text{ :- } s2m(X,Y,M), trans(M,M_t).\\
&s2m(0,Y,[\,]).\\
&s2m(s(X),Y,[R|Rs]) \text{ :- } s2l(Y,R), s2m(X,Y,Rs).\\
&s2l(s(Y),[C|Cs]) \text{ :- } s2l(Y,Cs). \qquad (18)\\
&s2l(0,[\,]).\\
&trans(M,M_t) \text{ :- } trans(M,[\,],M_t).\\
&trans([R|Rs],_,[C|Cs]) \text{ :- } r2c(R,[C|Cs],Cs_1,[\,],Ac), trans(Rs,Ac,Cs_1). \quad (19)\\
&trans([\,],X,X).\\
&r2c([X|Xs],[[X|Y_s]|Cs],[Y_s|Cs_1],A,B) \text{ :- } r2c(Xs,Cs,Cs_1,[[\,]|A],B). \qquad (20)\\
&r2c([\,],[\,],[\,],A,A).
\end{aligned}
$$

To prove termination of this example, one has to verify that:

- the first and second arguments of any call to $s2m/3$ are ground terms constructed from 0 and $s/1$,
- the first argument of any call to $s2l/2$ is a ground term constructed from 0 and $s/1$,
- the first argument of any call to $r2c/5$ is a bounded list as in (**??**),
- the first argument of any call to $trans/3$ is a bounded matrix, i.e., a term of the form

$$\bullet(\ell_1, \bullet(\ell_2, \ \dots \ \bullet(\ell_n, [\,]) \ \dots)), \qquad (21)$$

  where $n \geq 0$ and where the ℓ_i are bounded lists.

In the termination proof attempt, AProVE filters away the first argument of $\bullet/2$ due to the free variable C in the head of Clause (18). But if the first argument of $\bullet/2$ is discarded, one also loses the information that the elements ℓ_i in a bounded matrix like (21) are bounded lists. Hence, one can no longer detect that the first argument of $r2c$ in Clause (19) is a bounded list and thus, one fails to prove the decrease between the head and the body of Clause (20). The main problem here is that $\bullet/2$ is used in the program with two different types, i.e., to construct the "lists" and to construct the "matrices" (the "list" of lists). The argument filtering technique applied in AProVE fails to recognize this difference.

For Polytool, we use the type inference module and the set of queries $S'_1 = \{goal(t_1,t_2,t_3) \mid t_1, t_2$ are ground terms constructed from the function symbols 0 and $s/1$, and t_3 is an arbitrary term}, which is sufficient for proving termination of the example with the set of queries S_1. Then the call patterns $goal(s,s,f)$, $s2m(s,s,f)$, $s2l(s,f)$, $trans(m,f)$, $trans(m,m,f)$, and $r2c(\ell,f,f,m,f)\}$ are derived. Here, "s" means ground terms built from 0 and $s/1$, "ℓ" means bounded lists as in (**??**), and "m" means bounded matrices as in (21).

Proving termination of the program w.r.t. these call patterns is now possible for Polytool, and the following polynomial interpretation is derived:

$$\begin{aligned}
&\phi(0) = 1,\\
&\phi([\,]) = 1,\\
&\phi(s(X)) = X + 1,\\
&\phi([X_1|X_2]) = X_2 + 1,\\
&\phi(r2c(X_1, X_2, X_3, X_4, X_5)) = X_1 + 1,\\
&\phi(trans(X_1, X_2, X_3)) = X_1,\\
&\phi(s2m(X_1, X_2, X_3)) = X_1 + X_2,\\
&\phi(s2l(X_1, X_2)) = X_1 + 1.
\end{aligned}$$

The interpretation of the predicate symbol *goal* and the interargument relations for the predicates $r2c/5$ and $s2l/2$ can be chosen arbitrarily. □

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.

In addition to the examples from the TPDB, we also did experiments with all additional examples that occur in the paper. Note that this table contains the example btranspose, for which only Polytool succeeds to prove termination. For the example der, Polytool was the first tool to prove termination automatically.[6]

Examples	TALP	cTI-1.1	TerminWeb	Polytool-WST07	Polytool	AProVE
btranspose	fail[0.22]	fail[0.05]	fail[0.37]	fail [2.50]	yes [23.83]	fail[19.25]
der	timeout	fail[0.03]	fail[0.11]	fail[4.65]	yes [15.70]	yes[17.74]
div	timeout	yes[0.02]	yes[0.25]	yes [3.23]	yes[2.64]	yes[7.61]

Table 2. The results for the examples in this paper

6 Conclusions

Since a few years, the LP termination analysis community and the TRS termination analysis community 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

[6] Due to recent improvements in the automation of recursive path orders [**?**], this example can now also be proved by AProVE.

from different categories, including term rewriting and logic programming, to compete, as well as to raise new challenging examples for termination research communities. 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 natural 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 [?]. Parts of this build on the results in [?] on order-acceptability and the results in [?] on the constraint-based approach for termination analysis, another part extends the results of Bossi et al. [?] 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 [?] to represent all termination conditions symbolically, and introduce a new approach to find such polynomial interpretations automatically (cf. Section 4).

We have also developed an automated tool (Polytool [?]) for termination proofs of LPs based on polynomial interpretations. It has required an intensive work in coding. Especially, this concerns the construction of symbolic polynomial constraints from the acceptability conditions and the transformation from these polynomial constraints to Diophantine constraints. 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 [?,?,?,?], the constraint-based approach in Section 4, and the Diophantine constraint solver in [?], to provide a completely automated termination analyser. The previous version of this system using groundness analysis (Polytool-WST07) participated in the *International Competition of Termination Tools* 2007 and reached the second place, just after AProVE.

We have also done an intensive experimental evaluation for Polytool and compared it empirically with other termination analysers 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 non-linear 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 as well. 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. A first step into this direction is [**?**].

7 Acknowledgements

Manh Thang Nguyen is partly supported by *GOA Inductive Knowledge Bases* and partly by *FWO Termination Analysis: Crossing Paradigm Borders*. Jürgen Giesl and Peter Schneider-Kamp are 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.

References

1. K. R. Apt. Logic programming. In *Handbook of Theoretical Computer Science*, volume B, pages 493–574. MIT Press, 1990.
2. A. Bossi, N. Cocco, and M. Fabris. Proving termination of logic programs by exploiting term properties. In *Proc. TAPSOFT '91*, LNCS 494, pages 153–180, 1991.
3. J. Brauburger and J. Giesl. Termination analysis by inductive evaluation. In *Proc. CADE '98*, LNAI 1421, pages 254–269, 1998.
4. M. Bruynooghe, J. P. Gallagher, and W. Van Humbeeck. Inference of well-typings for logic programs with application to termination analysis. In *Proc. SAS '05*, LNCS 3672, pages 35–51, 2005.
5. M. Bruynooghe, M. Codish, J. P. Gallagher, S. Genaim, and W. Vanhoof. Termination analysis of logic programs through combination of type-based norms. *ACM Transactions on Programming Languages and Systems*, 29(2), 2007.
6. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
7. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME, 2000. `http://cime.lri.fr`.
8. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.
9. D. De Schreye and A. Serebrenik. Acceptability with general orderings. In *Computational Logic: Logic Programming and Beyond*, LNCS 2407, pages 187–210. 2002.
10. S. Decorte, D. De Schreye, and M. Fabris. Automatic inference of norms: a missing link in automatic termination analysis. In *Proc. ILPS '93*, pages 420–436. MIT Press, 1993.
11. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based automatic termination analysis of logic programs. *ACM Transactions on Programming Languages and Systems*, 21(6):1137–1195, 1999.
12. N. Dershowitz. 33 examples of termination. In *Proc. French Spring School in Theoretical Computer Science*, LNCS 909, pages 16–26, 1995.

13. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. SAT '07*, LNCS 4501, pages 340–354, 2007.
14. J. P. Gallagher, K. S. Henriksen, and G. Banda. Techniques for scaling up analyses based on pre-interpretations. In *Proc. ICLP '05*, LNCS 3668, pages 280–296, 2005.
15. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. IJCAR '06*, LNAI 4130, pages 281–286, 2006.
16. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
17. J. Giesl, R. Thiemann, S. Swiderski, and P. Schneider-Kamp. Proving termination by bounded increase. In *Proc. CADE '07*, LNAI 4603, pages 443–459, 2007.
18. A. Heaton, M. Abo-Zaed, M. Codish, and A. King. A simple polynomial groundness analysis for logic programs. *Journal of Logic Programming*, 45(1-3):143–156, 2000.
19. H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
20. G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *Journal of Logic Programming*, 13(2-3):205–258, 1992.
21. D. S. Lankford. On proving term rewriting systems are noetherian. Technical report, Mathematics Department, Louisiana Tech. University, Ruston, LA, 1979.
22. N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In *Proc. ICLP '97*, pages 63–77. MIT Press, 1997.
23. J. W. Lloyd. *Foundations of Logic Programming.* Springer Verlag, Berlin, 1987.
24. C. Marché and H. Zantema. The termination competition. In *Proc. RTA '07*, LNCS 4533, pages 303–313, 2007. See also the website `http://www.lri.fr/~marche/termination-competition`.
25. F. Mesnard and R. Bagnara. cTI: A constraint-based termination inference tool for ISO-Prolog. *Theory and Practice of Logic Programming*, 5(1-2):243–257, 2005.
26. M. T. Nguyen and D. De Schreye. Polynomial interpretations as a basis for termination analysis of logic programs. In *Proc. ICLP '05*, LNCS 3668, pages 311–325, 2005.
27. M. T. Nguyen and D. De Schreye. Polytool: Proving termination automatically based on polynomial interpretations. In *Proc. LOPSTR '06*, LNCS 4407, pages 210–218, 2007. Extended version appeared as Technical report, Department of Computer Science, K. U. Leuven, Belgium.
28. M. T. Nguyen, J. Giesl, P. Schneider-Kamp, and D. De Schreye. Termination analysis of logic programs based on dependency graphs. In *Proc. LOPSTR '07*, LNCS, 2008. To appear.
29. E. Ohlebusch, C. Claves, and C. Marché. TALP: A tool for the termination analysis of logic programs. In *Proc. RTA '00*, LNCS 1833, pages 270–273, 2000.
30. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proc. LOPSTR '06*, LNCS 4407, pages 177–193, 2007.
31. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and SAT solving. In *Proc. FroCoS '07*, LNAI 4720, pages 267–282, 2007.
32. A. Serebrenik. *Termination Analysis of Logic Programs.* PhD thesis, Department of Computer Science, K. U. Leuven, Belgium, 2003. http://www.cs.kuleuven.ac.be/publicaties/doctoraten/cw/CW2003$_0$2*.abs.html*.

33. A. Serebrenik and D. De Schreye. Hasta-La-Vista: Termination analyser for logic programs. In *Proc. WLPE '03*, pages 60–74, 2003.
34. J. Steinbach. Proving polynomials positive. In *Proc. FSTTCS '92*, LNCS 652, pages 18–20, 1992.
35. C. Taboch, S. Genaim, and M. Codish. TerminWeb: Semantics-based termination analyser for logic programs, 2002. `http://www.cs.bgu.ac.il/~mcodish/TerminWeb`.
36. K. Verschaetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In *Proc. ICLP '91*, pages 301–315. MIT Press, 1991.