

Polynomial Interpretations as a Basis for Termination Analysis of Logic Programs

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

Report CW412, Revised version, August 2007



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

Polynomial Interpretations as a Basis for Termination Analysis of Logic Programs

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

Report CW 412, Revised version, August 2007

Department of Computer Science, K.U.Leuven

Abstract

This paper introduces a new technique for termination analysis of definite logic programs (LPs) based on polynomial interpretations. The principle of this technique is to map each function and predicate symbol to a polynomial over some subset of natural numbers, like it has been done in proving termination of term rewriting systems. Such polynomial interpretations can be seen as a direct generalisation of the traditional techniques in termination analysis of LPs, where (semi-)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.

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

Manh Thang Nguyen¹, Danny De Schreye¹, Jürgen Giesl², and Peter Schneider-Kamp²

¹ Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001, Heverlee, Belgium
{ManhThang.Nguyen, Danny.DeSchreye}@cs.kuleuven.ac.be

² Department of Computer Science, RWTH Aachen
Ahornstr. 55, D-52056 Aachen, Germany
{giesl, psk}@informatik.rwth-aachen.de

Abstract. This paper introduces a new technique for termination analysis of definite logic programs based on polynomial interpretations. The principle of this technique is to map each function and predicate symbol to a polynomial over some domain of natural numbers, like it has been done in proving termination of term rewriting systems. Such polynomial interpretations can be seen as a direct generalisation of the traditional techniques in termination analysis of LPs, where (semi-)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, namely 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 ordering.

For Logic Programming (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 mapped value of the initial predicate call is bounded and of the running predicate calls decrease under the mapping. Automated termination proof, however, does not take into account all computational states. It focuses on verifying the decrease in size of (mutually) recursive predicate calls,

which correspond to the loops that the execution passes through.

Until now, most termination techniques in LP are based on the use of semi-linear norms and level mappings, which measure the size of each term or atom as a linear combination of its subterms. For example, the Hasta La Vista system [31] infers one specific semi-linear norm and the TerminWeb analyser [34] uses a combination of several semi-linear norms for termination analysis. A restriction of semi-linear norms is that a lot of examples require more powerful norms to verify their termination. 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 was first introduced in [13] (see also [10]).

Example 1 (der).

$$\begin{aligned} & d(\text{der}(u), 1). \\ & d(\text{der}(X + Y), DX + DY) : -d(\text{der}(X), DX), d(\text{der}(Y), DY). \\ & d(\text{der}(X * Y), X * DY + Y * DX) : -d(\text{der}(X), DX), d(\text{der}(Y), DY). \\ & d(\text{der}(\text{der}(X)), DDX) : -d(\text{der}(X), DX), d(\text{der}(DX), DDX). \end{aligned}$$

We are interested in proving termination of this program w.r.t. the query set $S = \{d(t_1, t_2) \mid t_1 \text{ is a ground term, and } t_2 \text{ is a free variable}\}$. We consider the first argument of $d/2$ as an input argument and the second as an output.

Doing this on the basis of a semi-linear norm and level mapping is impossible. The function symbol *der/1* expresses a non-linear relation between the input and output of the original derivative function. In particular, assume that there exists such a semi-linear norm $\|\cdot\|$ and level mapping $|\cdot|$ of general forms such that: $\|u\| = c$, $\|t_1 + t_2\| = f_0^+ + f_1^+ \|t_1\| + f_2^+ \|t_2\|$, $\|t_1 * t_2\| = f_0^* + f_1^* \|t_1\| + f_2^* \|t_2\|$, $\|\text{der}(t)\| = f_0^d + f_1^d \|t\|$, $|d(t_1, t_2)| = d_0 + d_1 \|t_1\| + d_2 \|t_2\|$ where t, t_1, t_2 are terms and $c, f_0^+, f_1^+, f_2^+, f_0^*, f_1^*, f_2^*, f_0^d, f_1^d, d_0, d_1, d_2, n_0$ and n_1 are non-negative integers. Applying the general constraint based method in [12] shows a contradiction: the system of inequalities that is set up from the acceptability condition is unsolvable. A complete proof can be found in [25]. Of course this only proves that one particular approach is unable to prove termination on the basis of semi-linear mappings. \square

In this paper, we propose a general framework for termination proof of LPs based on the use of polynomial interpretations. Using polynomial interpretations as a basis for ordering terms in TRSs was first introduced by Lankford in [22]. 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 (semi-)linear norms and linear level-mappings. As one would expect, the generalisation is a move from linear polynomial functions to arbitrary polynomials, while the concepts that link the two approaches are those of the ‘abstract norm’ and ‘abstract level mapping’ [35]. We show that under this approach, examples as the

above can be solved.

We also develop an automated tool (Polytool) for termination analysis based on this approach. We embedded this within the constraint-based approach developed in [12] and combined it with the nonlinear Diophantine constraint solver developed by Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann and Harald Zankl [14] 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 the results of an experimental evaluation and discuss these results. We end with a conclusion in Section 6.

2 Preliminaries

2.1 Notations and Terminology

We assume familiarity with logic programming concepts and with the main results of logic programming [3, 23]. In the following, P denotes a definite logic program. We use Var_P , Fun_P , $Const_P$ and $Pred_P$ to denote the set of variables, function, constant and predicate symbols of P . Note that any constant symbol in the program is considered a function symbol with arity of 0. 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 refer 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 only 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 as the LD-tree. We say that a query Q *LD-terminates* for a program P , if the LD-tree for $Q \cup P$ is finite (left-termination [23]).

2.2 Norms and Level Mappings

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

Several examples of norms can be found in literature [6]. 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 the tree representation of a term. Both of them belong to a class of norms called linear norms which is defined as follows.

Definition 2 (linear norm). [30] A norm $\|\cdot\|$ is a linear norm if it is recursively defined by means of the following schema:

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

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

A quasi-ordering on a set S is a reflexive and transitive binary relation \succeq defined on elements of S . We define the associated equivalence relation \simeq as $s \simeq t$ if and only if $s \succeq t$ and $t \succeq s$. If neither $s \succeq t$, nor $t \succeq s$ we write $\|_{\succeq}$. To each quasi-ordering \succeq on S , we can associate a strict ordering \succ on S as $s \succ t$ if and only if $s \succeq t$ and it is not the case that $t \succeq s$. A strict ordering \succ is called *well-founded* if there is no infinite sequence $s_0 \succ s_1 \succ \dots$ with $s_i \in S$. Let T be a set such that $S \subseteq T$. A quasi-ordering \supseteq defined on T is called a *proper extension* of \succeq if

- $t_1 \succ t_2$ implies $t_1 \supseteq t_2$ for all $t_1, t_2 \in S$.
- $t_1 \succ t_2$ implies $t_1 \triangleright t_2$ for all $t_1, t_2 \in S$, where \triangleright is the strict ordering associated with \supseteq .

We also need the following notion of a call set.

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

In practice, the query set S is specified as a call pattern. The set $Call(P, S)$ can be computed by using a type inference technique (e.g.[20]).

Definition 4 (order-acceptability w.r.t. a set). [10] Let S be a set of atomic queries and P be a program. P is order-acceptable w.r.t. S if there exists a well-founded ordering \succ such that

- for any $A \in Call(P, S)$,
- for any clause $A' \leftarrow B_1, \dots, B_n$, such that $mgu(A, A') = \theta$ exists,
- for any atom B_i , such that $rel(B_i) \simeq rel(A)$,
- for any computed answer substitution σ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

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

The following theorem establishes the link between order-acceptability w.r.t. a set and LD-termination of a program.

Theorem 1. [10] A program P LD-terminates under the left-to-right selection rule for any query in S if and only if P is order-acceptable w.r.t. S .

Definition 5 (interargument relation). Let P be a program, p/n be a predicate in P and \succ be an ordering on $Term_P$. An interargument relation for p/n is a relation $R_{p/n} = \{(t_1, \dots, t_n) \mid t_i \in Term_P \wedge \varphi_p(t_1, \dots, t_n)\}$, where:

- $\varphi_p(t_1, \dots, t_n)$ is a formula of an arbitrary boolean combination of inequalities,
- each inequality in φ_p is either $s_i \succeq s_j$, $s_i \succ s_j$, $s_i \preceq s_j$ or $s_i \ll s_j$, where s_i, s_j are constructed from t_1, \dots, t_n by applying functors of P .

$R_{p/n}$ is a valid interargument relation for p/n w.r.t. the ordering \succ if and only if for every $p(t_1, \dots, t_n) \in \text{Atom}_P$: $P \models p(t_1, \dots, t_n)$ implies $p(t_1, \dots, t_n) \in R_{p/n}$.

The concept of rigidity is also generalized to general orderings.

Definition 6 (rigidity). [10] A term or atom $A \in \text{Term}_P \cup \text{Atom}_P$ is called rigid w.r.t. a quasi-ordering \succeq if $\forall \sigma \in \text{Subs}$, $A \preceq A\sigma$. In this case, \succeq is said to be rigid on A . A set of terms (or atoms) S is called rigid w.r.t. a quasi-ordering \succeq if all its elements are rigid w.r.t. \succeq .

Example 2. The list $[X|t]$ (X is a variable, t is a ground term) is rigid w.r.t. the quasi-ordering \succeq imposed by the list-length norm $\|\cdot\|_\ell$, i.e. $t_1 \succeq t_2$ 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 , $\|[X|t]\sigma\|_\ell = 1 + \|t\|_\ell = \|[X|t]\|_\ell$. Therefore, $[X|t]\sigma \preceq [X|t]$. However, this list is not rigid w.r.t. the quasi-ordering \supseteq imposed by the term-size norm $\|\cdot\|_\tau$, i.e. $t_1 \supseteq t_2$ if and only if $\|t_1\|_\tau \geq \|t_2\|_\tau$, $t_1 \triangleright t_2$ if and only if $\|t_1\|_\tau > \|t_2\|_\tau$. For instance, with $\sigma_1 = \{X/a_1\}$, a_1 is a constant, $\|[X|t]\sigma_1\|_\tau = 1 + \|a_1\|_\tau + \|t\|_\tau = 2 + \|t\|_\tau$, while with $\sigma_2 = \{X/[a_1, a_2]\}$ a_1, a_2 are constants, $\|[X|t]\sigma_2\|_\tau = 1 + \|[a_1, a_2]\|_\tau + \|t\|_\tau = 6 + \|t\|_\tau$. That implies $[X|t]\sigma_2 \triangleright [X|t]\sigma_1$. \square

The following notion of rigid order-acceptability w.r.t. a set of atoms no longer forces us to reason on $\text{Call}(P, S)$. Instead, we only need to consider the rigidity of the call set. Furthermore, the condition in this notion is fully at the clause level and the condition on computed answer substitution is replaced by one on valid interargument relations.

Definition 7 (rigid order-acceptability w.r.t. a set). [10] Let S be a set of atomic queries and P be a program. Let \succeq be a well-founded quasi-ordering on Term_P and for each predicate p/n in P , let $R_{p/n}$ be a valid interargument relation for p/n w.r.t. \succeq . P is rigid order-acceptable w.r.t. S if there exists a proper extension \supseteq of \succeq on $\text{Term}_P \cup \text{Atom}_P$, which is rigid on $\text{Call}(P, S)$ such that

- for any clause $H \leftarrow B_1, B_2, \dots, B_n$,
- for any atom B_i in its body such that $\text{rel}(B_i) \supseteq \text{rel}(H)$,
- for any substitution θ such that the arguments of the atoms in $(B_1\theta, \dots, B_{i-1}\theta)$ all satisfy their associated interargument relations $R_{\text{rel}(B_1)}, \dots, R_{\text{rel}(B_{i-1})}$:

$$H\theta \triangleright B_i\theta.$$

Proposition 1. [10] If P is rigid order-acceptable w.r.t. S , then P is order-acceptable w.r.t. S .

The stated condition of rigid order-acceptability is sufficient for acceptability, but is not necessary for it (see [10]). With Definition 7 and Proposition 1, proving

termination of a program now becomes verifying the rigidity of the call sets, the valid interargument conditions for predicates and the decrease conditions for the (mutually) recursive clauses. Since those conditions, as we discuss later on, are finite, Definition 7 and Proposition 1 are very useful and important for automating termination proofs in our approach.

3 Polynomial Interpretation of a Logic Program

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

3.1 Polynomial Interpretations

We start by recalling some basic notions in polynomial theory. A polynomial P is a function in one or more variables consisting of a sum of monomials, each of which is a product of natural powers in variables (a term) multiplied by a coefficient. In this paper, we only consider polynomials with integral coefficients. The sum of two polynomials P and Q is obtained by taking the sum of coefficients sharing the same term. Similarly, the subtraction of polynomial P to polynomial Q is done by subtracting each coefficient in P to a coefficient in Q which share the same term. The multiplication of two polynomials P and Q is obtained by multiplying each monomial in P by each monomial in Q and then summing the results.

Let $D \subseteq \mathbb{N}$ be an arbitrary non-empty set of natural numbers. A polynomial $P(X_1, \dots, X_n)$ with integral coefficients is called positive over D if and only if $P(x_1, \dots, x_n) \geq 0$ for all $x_1, \dots, x_n \in D$. $P(X_1, \dots, X_n)$ is called monotone over D if and only if $P(x_1, \dots, x_{i-1}, t, x_{i+1}, \dots, x_n) \geq P(x_1, \dots, x_{i-1}, s, \dots, x_{i+1}, x_n)$ for all $t, s, x_1, \dots, x_n \in D$, $t \geq s$. Let Ω_D be the set of all positive polynomials over D . The following defines orderings on Ω_D :

Definition 8. Let P and Q be two polynomials in Ω_D . Let X_1, \dots, X_n be all variables occurring in P or Q and $H(X_1, \dots, X_n) = P - Q$ be a polynomial which is the subtraction of P to Q . An ordering \geq_D on Ω_D is defined as $P \geq_D Q$ if and only if $H(x_1, \dots, x_n) \geq 0$ for all $x_1, \dots, x_n \in D$. A strict ordering $>_D$ associated with \geq_D is defined as $P >_D Q$ if and only if $H(x_1, \dots, x_n) > 0$. If $H(x_1, \dots, x_n) = 0$, we write $P \leq_{\geq_D} Q$. For the other cases, $P \parallel_{\geq_D} Q$.

Example 3. Let $P(X, Y) = XY + X$ and $Q(X, Y) = 2X + 2$.

For $T(X, Y) = P - Q = XY - X - 2$, we have the following cases:

- $D = \mathbb{N}$: Because $T(0, 0) = -2 < 0$ and $T(1, 4) = 1 > 0$, then $P \parallel_{\geq_D} Q$.
- $D = \{x \in \mathbb{N} | x \geq 2\}$: For all $x, y \in D$, $T(x, y) \geq 0$. Therefore, $P \geq_D Q$.
- $D = \{x \in \mathbb{N} | x \geq 3\}$: For all $x, y \in D$, $T(x, y) > 0$. We have $P >_D Q$.

□

Proposition 2. *The ordering $>_D$ defined in Definition 8 is a well-founded ordering.*

Proof. Proof is by contradiction. Suppose that $P_1 >_D P_2 >_D \dots$ is an infinite sequence of polynomials in Ω_D . Let θ be an assignment to each variable occurring in those polynomials of an element in D and let $P_1\theta, P_2\theta, \dots$ be the instantiations of P_1, P_2, \dots w.r.t. θ . Definition 8 implies that $P_1\theta > P_2\theta \dots$ is an infinite decreasing sequence of elements in \mathbb{N} , which contradicts that $>$ is a well-founded ordering on \mathbb{N} . \square

Let Σ_D be a set of polynomials, each having a fixed arity and integral coefficients, such that for any polynomial $P(X_1, \dots, X_n) \in \Sigma_D$, we have: $P(x_1, \dots, x_n) \in D$ for all $x_1, \dots, x_n \in D$. We call such condition ‘ D -closedness under evaluation’. It is clear that $\Sigma_D \subseteq \Omega_D$ and the orderings in Ω_D can be applied for the elements in Σ_D . The following notion of polynomial interpretations sets up an abstract version of each function and predicate symbol in the program.

Definition 9 (polynomial interpretation). *An polynomial interpretation I for a logic program P consists of:*

- A set of polynomials Σ_D , D -closed under evaluation, with D a non-empty set of natural numbers,
- An assignment ϕ associating each function and predicate symbol f/n in $\text{Fun}_P \cup \text{Pred}_P$ with a polynomial $P_f/n \in \Sigma_D$.

We denote a polynomial interpretation I by a tuple (Σ_D, ϕ) .

Definition 10 (polynomial norm). *The norm associated with a polynomial interpretation I is a mapping $\|\cdot\|_I : \text{Term}_P \rightarrow \Sigma_D$ which is defined recursively as:*

- $\|X\|_I = X$ if X is a variable,
- $\|f(t_1, \dots, t_n)\|_I = P_f(\|t_1\|_I, \dots, \|t_n\|_I)$ where $P_f/n = \phi(f/n)$ and $n \geq 0$.

Definition 11 (polynomial level mapping). *The level mapping associated with a polynomial interpretation I is a mapping $|\cdot|_I : \text{Atom}_P \rightarrow \Sigma_D$ which is defined as:*

$|q(t_1, \dots, t_n)|_I = P_q(\|t_1\|_I, \dots, \|t_n\|_I)$ where $P_q/n = \phi(q/n)$ and $n \geq 0$.

Example 4 (dist). Consider the following distributive program *dist*. This example was introduced in [10] (see also [36]):

$$\text{dist}(x, x).$$

$$\text{dist}(x * x, x * x).$$

$$\text{dist}(X + Y, U + V) : -\text{dist}(X, U), \text{dist}(Y, V). \quad (1)$$

$$\text{dist}(X * (Y + Z), T) : -\text{dist}(X * Y + X * Z, T). \quad (2)$$

$$\text{dist}((X + Y) * Z, T) : -\text{dist}(X * Z + Y * Z, T). \quad (3)$$

Let $I = (\Sigma_{\mathbb{N}}, \phi)$ be a polynomial interpretation of the program *dist* which consists of the following elements:

- The set $\Sigma_{\mathbb{N}}$ with $D = \mathbb{N}$, of all polynomials with natural coefficients,
- An assignment ϕ such that $\phi(X_1 * X_2) = P_*(X_1, X_2) = 2X_1 * X_2 + 2X_1 + 2X_2$,
 $\phi(X_1 + X_2) = P_+(X_1, X_2) = X_1 + X_2 + 1$, $\phi(x) = P_x = 0$,
and $\phi(\text{dist}(X_1, X_2)) = P_{\text{dist}}(X_1, X_2) = X_1$.

Obviously $\Sigma_{\mathbb{N}}$ is \mathbb{N} -closed under evaluation.

The size of the atom $A = \text{dist}(U * (X + Y), T)$ w.r.t. I is:

$$\begin{aligned} |\text{dist}(U * (X + Y), T)|_I &= \|U * (X + Y)\|_I = P_*(\|U\|_I, P_+(\|X\|_I, \|Y\|_I)) \\ &= P_*(U, X + Y + 1) = 2U * (X + Y + 1) + 2U + 2(X + Y + 1) \\ &= 2U * (X + Y) + 4U + 2X + 2Y + 2. \end{aligned} \quad \square$$

A quasi-ordering on $\text{Term}_P \cup \text{Atom}_P$ imposed by the ordering \geq_D on Ω_D is defined as follows:

Definition 12 (ordering on terms and atoms). *Let P be a program and I be a polynomial interpretation. We define \succeq_I a quasi-ordering on Term_P such that:*

- $t \succ_I s$ if and only if $\|t\|_I >_D \|s\|_I$ for any $t, s \in \text{Term}_P$,
- $t \preceq_I s$ if and only if $\|t\|_I \leq_D \|s\|_I$ for any $t, s \in \text{Term}_P$,

and \triangleright_I a proper extension of \succeq_I on $\text{Term}_P \cup \text{Atom}_P$ such that:

- $B \triangleright_I C$ if and only if $|B|_I >_D |C|_I$ for any $B, C \in \text{Atom}_P$,
- $B \trianglelefteq_I C$ if and only if $|B|_I \leq_D |C|_I$ for any $B, C \in \text{Atom}_P$.

Proposition 3. *The strict orderings \succ_I and \triangleright_I are well-founded orderings on Term_P and $\text{Term}_P \cup \text{Atom}_P$ respectively.*

Proof. Proof by contradiction: assume the ordering \succ_I is not well-founded. Therefore, there exists an infinite sequence of terms such that $t_1 \succ_I t_2 \succ_I \dots \succ_I t_n \dots$. However, Definition 12 also implies $\|t_1\|_I >_D \|t_2\|_I >_D \dots >_D \|t_n\|_I \dots$, which contradicts the fact that $>_D$ is a well-founded ordering on Ω_D . For the ordering \triangleright_I , the proof is similar. \square

Proposition 4. *Let P be a program and S be a set of atomic queries. If there exists a polynomial interpretation I such that*

- for any $A \in \text{Call}(P, S)$,
- for any clause $A' \leftarrow B_1, \dots, B_n$ in P such that $\text{mgu}(A, A') = \theta$ exists,
- for any atom B_i , such that $\text{rel}(B_i) \simeq \text{rel}(A)$,
- for any computed answer substitution σ for $\leftarrow(B_1, \dots, B_{i-1})\theta$:

$$|A|_I >_D |B_i \theta \sigma|_I,$$

then P left-terminates w.r.t. S .

Proof. Consider the ordering \triangleright_I of Proposition 3. By definition: $|A|_I >_D |B_i \theta \sigma|_I$ if and only if $A \triangleright_I B_i \theta \sigma$. Proposition 3 states that \triangleright_I is a well-founded ordering. Based on Theorem 1, we can conclude that the program P terminates w.r.t. the query set S . \square

Example 5. Reconsider Example 4. We prove termination of the program with the following set of queries $S = \{dist(t_1, t_2) | t_1 \text{ is a ground term and } t_2 \text{ is a free variable}\}$.

We choose the same polynomial interpretation I as in Example 4. Observe that the set $Call(P, S) = S$. Suppose $A = dist(t, s)$ is a selected atom in $Call(P, S)$. There are 3 cases to consider: clauses (1), (2) and (3). We present only the last one:

$$\begin{aligned}
& - A = dist((t_1 + t_2) * t_3, s) \text{ (} t_1, t_2, t_3 \text{ are ground terms) and clause (3) is selected. Let } \theta \text{ be a substitution such that } \theta = mgu(A, dist((X_1 + Y_1) * Z_1, T_1)). \\
& \text{It implies } X_1\theta = t_1, Y_1\theta = t_2, Z_1\theta = t_3. \text{ We denote } P_{t_1}, P_{t_2} \text{ and } P_{t_3} \text{ by the polynomial norms associated with } t_1, t_2 \text{ and } t_3 \text{ respectively. Therefore:} \\
& |dist((t_1 + t_2) * t_3, s)|_I = \|(t_1 + t_2) * t_3\|_I \\
& = 2\|t_1 + t_2\|_I * \|t_3\|_I + 2\|t_1 + t_2\|_I + 2\|t_3\|_I \\
& = 2(\|t_1\|_I + \|t_2\|_I + 1) * \|t_3\|_I + 2\|t_1\|_I + 2\|t_2\|_I + 2\|t_3\|_I + 2 \\
& = 2P_{t_1} * P_{t_3} + 2P_{t_2} * P_{t_3} + 4P_{t_3} + 2P_{t_1} + 2P_{t_2} + 2 \\
& >_D |dist(t_1 * t_3 + t_2 * t_3, s)|_I \\
& = \|t_1 * t_3\|_I + \|t_2 * t_3\|_I + 1 \\
& = 2\|t_1\|_I * \|t_3\|_I + 2\|t_2\|_I * \|t_3\|_I + 2\|t_1\|_I + 2\|t_2\|_I + 4\|t_3\|_I + 1 \\
& = 2P_{t_1} * P_{t_3} + 2P_{t_2} * P_{t_3} + 2P_{t_1} + 2P_{t_2} + 4P_{t_3} + 1.
\end{aligned}$$

With a similar verification for clauses (1) and (2), P is order-acceptable w.r.t. S and P terminates on S . \square

Next, we move to a termination condition with rigidity again.

Usually, when talking about rigidity, we are only interested in rigidity of a set of terms (or atoms) w.r.t. a particular norm (or level mapping). In [6], Bossi, Cocco and Fabris discuss rigidity of $Call(P, S)$ w.r.t. a (semi-)linear norm and a level mapping for some P and S . It is then generally extended to the case of rigidity of $Call(P, S)$ w.r.t. a general term ordering in [10]. In this paper, we discuss rigidity of terms (or atoms) w.r.t. a polynomial interpretation and show that it is also an extension of [6]. Let us extend some basic notions defined in [6].

Definition 13 (rigidity w.r.t. a polynomial interpretation).

A term $t \in Term_P$ is called rigid w.r.t. a polynomial interpretation I if and only if for any substitution θ , $\|t\|_I \leq_D \|t\theta\|_I$. An atom $A \in Atom_P$ is called rigid w.r.t. I if and only if for any substitution θ , $|A|_I \leq_D |A\theta|_I$. In this case, I is said to be rigid on, respectively, t and A .

The notion of rigidity on a term or an atom is naturally extended to the notion of rigidity on a set of terms or atoms. A polynomial interpretation I is called rigid on a set of atoms or a set of terms if and only if it is rigid on every element of the set. In particular, we are interested in polynomial interpretations that are rigid on a call set $Call(P, S)$ for some P and S .

Definition 14. Let I be a polynomial interpretation and t be a term (atom).

The i^{th} occurrence $X_{(i)}$ of a variable X in t is called relevant w.r.t. I if there exists a replacement $\{s \rightarrow X_{(i)}\}$ of a term s for $X_{(i)}$ in t such that $\|t\{s \rightarrow X_{(i)}\}\|_I \leq_D \|t\|_I$

is not true. We call $VREL_I(t)$ the set of all relevant occurrences of variables in t .

Example 6. Let $t = [X|X]$ and I be the interpretation imposed by the list-length norm $\|\cdot\|_I$, $\|[H|T]\|_I = 1 + \|T\|_I$. Then, $VREL_I(t) = \{X_{(2)}\}$. \square

Proposition 5. *Let I be a polynomial interpretation and t be a term. If $VREL_I(t) = \emptyset$, then t is rigid w.r.t. I .*

Proof. Obviously from Definition 14. If a term t is not rigid w.r.t. I , there must be some relevant occurrence of some variable in t . This contradicts the hypothesis $VREL_I(t) = \emptyset$. Thus t is rigid w.r.t. I . \square

We also need the following notion of polynomial interargument relations.

Definition 15 (polynomial interargument relation). *Let P be a program, p/n be a predicate in P and I be a polynomial interpretation. A polynomial interargument relation for p/n is a relation $R_{p/n} = \{(t_1, \dots, t_n) \mid t_i \in \text{Term}_P \wedge \varphi_p(\|t_1\|_I, \dots, \|t_n\|_I)\}$, where:*

- $\varphi_p(\|t_1\|_I, \dots, \|t_n\|_I)$ is a formula of an arbitrary combination of inequalities,
- each inequality in φ_p is either $\|s_i\|_I \geq_D \|s_j\|_I$, $\|s_i\|_I >_D \|s_j\|_I$, $\|s_i\|_I \leq_D \|s_j\|_I$ or $\|s_i\|_I \leq_{>_D} \|s_j\|_I$, where s_i, s_j are constructed from t_1, \dots, t_n by applying functors of P .

$R_{p/n}$ is a valid polynomial interargument relation for p/n w.r.t. I if and only if for every $p(t_1, \dots, t_n) \in \text{Atom}_P : P \models p(t_1, \dots, t_n)$ implies $(t_1, \dots, t_n) \in R_{p/n}$.

Using the notions of rigidity and polynomial interargument relations w.r.t. a polynomial interpretation integrated with Definition 7, Proposition 1 and Definition 12, we obtain:

Proposition 6. *Let S be a set of atomic queries, P be a program and I be a polynomial interpretation. For each predicate p/n in P , let $R_{p/n}$ be a valid polynomial interargument relation w.r.t. I . If I is rigid on $\text{Call}(P, S)$ such that*

- for any clause $H \leftarrow B_1, \dots, B_n$,
- for any atom B_i in its body, such that $\text{rel}(B_i) \preceq \text{rel}(H)$,
- for any substitution θ , such that the arguments of the atoms $B_1\theta, \dots, B_{i-1}\theta$ satisfy their associated polynomial interargument relations $R_{\text{rel}(B_1)}, \dots, R_{\text{rel}(B_{i-1})}$:

$$\|H\theta\|_I >_D \|B_i\theta\|_I,$$

then P left-terminates w.r.t. S .

Proof. Similar to the proof of Proposition 4. \square

Proposition 6 can be applied to verify termination of a logic program w.r.t. a call set. More particularly, we have to check that all following conditions are satisfied w.r.t. a given polynomial interpretation I :

1. The given polynomial interpretation should be valid: All polynomials associated with function and predicate symbols are in Σ_D . This can be done by verifying the 'D-closedness under evaluation' for all those polynomials:
For any polynomial $P_f/n = \phi(f/n)$ with f/n a function or predicate symbol in the program, $a_1, \dots, a_n \in D$ implies $P_f(a_1, \dots, a_n) \in D$.
2. I is rigid on the call set $Call(P, S)$:
For any query Q in the call set, the set of all relevant occurrences of variables $VREL_I(Q)$ is empty.
3. For a clause that has intermediate body-atoms between the head and a (mutually) recursive body-atom, a valid polynomial interargument relations of those atoms needs to be inferred. Intuitively, the condition of the valid interargument relations can be stated as follows: For any success answer of the program w.r.t. the query, the corresponding polynomial interargument relation among the arguments of this answer must hold. We can verify that by first checking if it is correct for the facts in the program. Then for every clause, if the relations holds for all body-atoms, the relation for the head should also holds.
4. 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 w.r.t. the ordering, given that interargument relations for intermediate body-atoms hold.

Example 7. Reconsider Example 1. We are interested in proving termination of the program w.r.t. the query set $S = \{d(t_1, t_2) | t_1 \text{ is a ground term and } t_2 \text{ is a free variable}\}$. Observe that $Call(P, S)$ coincides with S .

$$d(der(u), 1). \tag{4}$$

$$d(der(X + Y), DX + DY) : -d(der(X), DX), d(der(Y), DY). \tag{5}$$

$$d(der(X * Y), X * DY + Y * DX) : -d(der(X), DX), d(der(Y), DY). \tag{6}$$

$$d(der(der(X)), DDX) : -d(der(X), DX), d(der(DX), DDX). \tag{7}$$

Let I be a polynomial interpretation that consists of following elements:

- A set of polynomials Σ_D with $D = \{x \in \mathbb{N} | x \geq 2\}$,
- An assignment ϕ such that $\phi(X_1 + X_2) = P_+(X_1, X_2) = X_1 + X_2$,
 $\phi(der(X)) = P_{der}(X) = X^2$, $\phi((X_1 * X_2)) = P_*(X_1, X_2) = X_1 * X_2$,
 $\phi(u) = \phi(1) = 2$, and $\phi(d(X_1, X_2)) = P_d(X_1, X_2) = X_1$.

Let $R_{d/2} = \{(t_1, t_2) | t_1, t_2 \in Term_P, \|t_1\|_I >_D \|t_2\|_I\}$ be a polynomial interargument relation w.r.t. the predicate $d/2$.

It is clear that all polynomials associated with function and predicate symbols satisfy the 'D-closedness under evaluation':

- $\forall X \geq 2 \Rightarrow P_{der}(X) = X^2 \geq 2$,
- $\forall X_1, X_2 \geq 2 \Rightarrow P_+(X_1, X_2) = X_1 + X_2 \geq 2$,
- $\forall X_1, X_2 \geq 2 \Rightarrow P_*(X_1, X_2) = X_1 * X_2 \geq 2$,
- $\phi(u) = \phi(1) = 2 \geq 2$,
- $\forall X_1, X_2 \geq 2 \Rightarrow P_d(X_1, X_2) = X_1 \geq 2$,

It is also clear that for any $Q \in \text{Call}(P, S)$, we have $\text{VREL}_I(Q) = \emptyset$. It implies I is rigid on $\text{Call}(P, S)$.

Checking the validity of $R_{d/2}$ w.r.t. I is equivalent to verifying the correctness of the following conditions w.r.t. any substitution θ :

$$\begin{aligned} & \| \text{der}(u)\theta \|_{I >_D} \| (1)\theta \|_I \\ & \| \text{der}(X)\theta \|_{I >_D} \| DX\theta \|_I \text{ and } \| \text{der}(Y)\theta \|_{I >_D} \| DY\theta \|_I \text{ implies} \\ & \quad \| \text{der}(X + Y)\theta \|_{I >_D} \| (DX + DY)\theta \|_I \\ & \| \text{der}(X)\theta \|_{I >_D} \| DX\theta \|_I \text{ and } \| \text{der}(Y)\theta \|_{I >_D} \| DY\theta \|_I \text{ implies} \\ & \quad \| \text{der}(X * Y)\theta \|_{I >_D} \| (X * DY + Y * DX)\theta \|_I \\ & \| \text{der}(X)\theta \|_{I >_D} \| DX\theta \|_I \text{ and } \| \text{der}(DX)\theta \|_{I >_D} \| DDX\theta \|_I \text{ implies} \\ & \quad \| \text{der}(\text{der}(X))\theta \|_{I >_D} \| DDX\theta \|_I. \end{aligned}$$

And we also need the following decrease conditions hold w.r.t. any substitution θ :

$$\begin{aligned} & |d(\text{der}(X + Y), DX + DY)\theta|_I >_D |d(\text{der}(X), DX)\theta|_I \\ & \quad d(\text{der}(X), DX)\theta \text{ satisfies } R_{d/2} \text{ implies} \\ & |d(\text{der}(X + Y), DX + DY)\theta|_I >_D |d(\text{der}(Y), DY)\theta|_I \\ & |d(\text{der}(X * Y), X * DY + Y * DX)\theta|_I >_D |d(\text{der}(X), DX)\theta|_I \\ & \quad d(\text{der}(X), DX)\theta \text{ satisfies } R_{d/2} \text{ implies} \\ & |d(\text{der}(X * Y), X * DY + Y * DX)\theta|_I >_D |d(\text{der}(Y), DY)\theta|_I \\ & |d(\text{der}(\text{der}(X)), DDX)\theta|_I >_D |d(\text{der}(X), DX)\theta|_I \\ & \quad d(\text{der}(X), DX)\theta \text{ satisfies } R_{d/2} \text{ implies} \\ & |d(\text{der}(\text{der}(X)), DDX)\theta|_I >_D |d(\text{der}(DX), DDX)\theta|_I \end{aligned}$$

They are equivalent to the following inequalities on variables X, Y, DX, DY, DDX for the valid interargument relation conditions:

$$\begin{aligned} & 4 > 2 \\ & \forall X, Y, DX, DY \in D : X^2 > DX \wedge Y^2 > DY \Rightarrow (X + Y)^2 > DX + DY \\ & \forall X, Y, DX, DY \in D : X^2 > DX \wedge Y^2 > DY \Rightarrow (X * Y)^2 > YDX + XDY \\ & \forall X, DX, DDX \in D : X^2 > DX \wedge DX^2 > DDX \Rightarrow X^4 > DDX \end{aligned}$$

And for the decrease conditions:

$$\begin{aligned} & \forall X, Y \in D : (X + Y)^2 > X^2 & \forall X, Y, DX \in D : X^2 > DX \Rightarrow (X * Y)^2 > Y^2 \\ & \forall X, Y, DX \in D : X^2 > DX \Rightarrow (X + Y)^2 > Y^2 & \forall X \in D : X^4 > X^2 \\ & \forall X, Y \in D : (X * Y)^2 > X^2 & \forall X, DX \in D : X^2 > DX \Rightarrow X^4 > DX^2 \end{aligned}$$

Since $D = \{x \in \mathbb{N} \mid x \geq 2\}$, the above inequalities are easily verified and the program left-terminates. \square

4 Automating the Termination Proof

A key question is how to automate the search for a polynomial interpretation and polynomial interargument relations (or the coefficients of the polynomial associated with each function/predicate symbol and the coefficients of the polynomial interargument relations) which derives a termination proof of a given logic program. In the philosophy of the constraint-based approach in [12], we do not choose a particular polynomial interpretation and polynomial interargument relation. We introduce a general symbolic form for the polynomial associated with each function/predicate symbol in the polynomial interpretation and for the polynomial interargument relations. As an example and assuming that polynomials of degree 2 are selected for the interpretation, instead of assigning the polynomial $P_q(X, Y) = X^2 + 3XY$ to a predicate symbol $q(X, Y)$, we would assign $P_q(X, Y) = q_1 X^2 + q_2 XY + q_3 Y^2$, where q_1 , q_2 and q_3 are unknown symbolic coefficients ranging over \mathbb{N} . The strategy of the analysis is to:

- introduce symbolic versions of the polynomial interpretation (the polynomials associated with each function and predicate symbol),
- introduce symbolic versions of the valid interargument relations,
- express all conditions resulting from Proposition 6 as constraints on the coefficients (e.g. q_1, q_2, q_3, \dots),
- 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 for all atoms and terms and a concrete valid interargument relation for all intermediate body-atoms that respect the termination condition. Therefore, each solution gives a termination proof.

4.1 Generating a Symbolic Form for Termination Concepts

On the level of the polynomial interpretations, we need to restrict to a fixed type of set D associated with the set of polynomials Σ_D , from which the ‘D-closedness under evaluation’ condition is established. An obvious option for this set is $D = \{x \in \mathbb{N} \mid x \geq \mu\}$, with μ a symbolic parameter, $\mu \in \mathbb{N}, \mu \geq 0$. We also need to restrict to fixed types of polynomials, since there does not exist a finite symbolic representation for all possible polynomials. Specifically, we will associate linear polynomials to predicates and linear, simple, or simple-mixed polynomials to functors.

- *The linear class*: each monomial of a polynomial in this class contains at most one variable of at most degree 1:

$$P(X_1, \dots, X_n) = p_0 + \sum_{k=1}^n p_k * X_k,$$

- *The simple class*: the class contains polynomials of at most degree 1 in each variable:

$$P(X_1, \dots, X_n) = \sum_{j_k \in \{0,1\}} p_{j_1 \dots j_n} X_1^{j_1} \dots X_n^{j_n},$$

- *The simple-mixed class*: each monomial of a polynomial in this class consists of either a single variable with degree of 2 or several variables with degree of at most 1:

$$P(X_1, \dots, X_n) = \sum_{j_k=\{0,1\}} p_{j_1 \dots j_n} X_1^{j_1} \dots X_n^{j_n} + \sum_{k=1}^n p_k X_k^2.$$

For more details on these classes of polynomials we refer to [32, 9].

Theoretically, the selection of μ does not influence the correctness of the termination proof and we can give μ an arbitrary fixed value. The work in [9] states that if there exists a sufficient polynomial interpretation w.r.t. a $\mu = \mu_1 \in \mathbb{N}$ that derives a proof, there also exists another sufficient polynomial interpretation w.r.t. $\mu = 0$. Although this is a proof for termination analysis of TRS, it can be done in a similar way for LP and can be extended to any value of μ .

Practically, it is important. A reason is that given a particular value of μ , the search space for other symbolic parameters (i.e. coefficients of polynomials associated with function and predicate symbols) may be too large to be able to find a suitable polynomial interpretation. Of course, if we consider μ an unknown parameter that needs to be found, we also increase the complexity of the problem. In this paper, μ is considered unknown.

The condition of valid interargument relations is also symbolised. It is often abstracted as pre-conditions of the relations among the size of the arguments of the intermediate body atoms that implies the decrease between the size of the head and that of an recursive body atom in the clause. There are a number of works on inferring valid interargument relations of predicates. In [12], it is formulated as an inequality between a linear combination of the ‘inputs’ and a linear combination of the ‘outputs’, in which ‘inputs’ are the arguments of the predicate which are never called with free variables, and ‘outputs’ are the remaining arguments. In [8], interargument relations are formulated as a conjunction of monotonicity constraints of the form $u \geq v + b$, where u and v are variables occurring in the head and body atoms, $b = 0$ or 1 .

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

$$R_{p/n} = \{(t_1, \dots, t_n) \mid P(\|t_1\|_I, \dots, \|t_n\|_I) \geq_D Q(\|t_1\|_I, \dots, \|t_n\|_I)\} \quad (8)$$

where Q/n and P/n are polynomials with non-negative integer coefficients.

The form of interargument relations applied in [12] can be considered a special case of Form (8) where $P(\|t_1\|_I, \dots, \|t_n\|_I)$ is constructed from the ‘input’ arguments and $Q(\|t_1\|_I, \dots, \|t_n\|_I)$ is constructed from the ‘outputs’.

Since the approach in [12] only considers the relation between the ‘input’ and ‘output’ arguments of the predicates, it has some limitations. In some cases, there is no relation between the ‘inputs’ and ‘outputs’, but among the ‘inputs’ and the ‘output’ themselves. Especially, if all arguments in a predicate are ‘inputs’ (or ‘outputs’), the approach in [12] fails to infer any relation among them. The following example, DIV, 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 8 (DIV).

$$\begin{aligned}
div(X, s(Y), 0) &: -less(X, s(Y)). \\
div(X, s(Y), s(Z)) &: -sub(X, s(Y), R), div(R, s(Y), Z). \\
sub(s(X), s(Y), Z) &: -sub(X, Y, Z). \quad sub(X, 0, X). \\
less(s(X), s(Y)) &: -less(X, Y). \quad less(0, s(Y)).
\end{aligned} \tag{9}$$

with the query pattern $div(g,g,f)$.

This program terminates w.r.t. to the query pattern. If we look at clause (9), the decrease in size between the head and the recursive body atom can be established if we can infer a valid interargument relation of sub/β such that the size of its first argument is greater than that of its third argument. However, if we apply the approach in [12], where a symbolic form of valid interargument relations based on modes is used, inferring such kind of interargument relation for sub/β is impossible. Since the mode of the predicate sub/β is $sub(g, g, f)$ (the call pattern to the predicate sub/β in clause (9) is $sub(g, g, f)$), the approach can only infer a valid interargument relation for sub/β such that the linear combination of the sizes of the first and second arguments is greater or equal than the size of the third argument. It is not sufficient that for any success answer for the call to $sub(X, s(Y), R)$ in clause (9), the size of the first argument X is greater than the size of the third argument R .

If we use the interargument relation of Form (8), it is possible to infer a valid interargument relation for $sub(X, Y, Z)$ that has the following form:

$$\|X\|_I \geq \|Y\|_I + \|Z\|_I$$

This valid interargument relation guarantees that for any success answer of the call to $sub(X, s(Y), R)$ in clause (9), we can infer $\|X\|_I > \|R\|_I$ if $\|s(Y)\|_I \geq 1$. \square .

There are also a number of other examples in the Termination Problem Database [2] that cannot be proved terminating by the technique in [12] 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 valid interargument relations. Specifically, we may use the symbolic forms of the linear, simple, or simple-mixed polynomials for the left- and right-hand side polynomials P/n and Q/n . For the inference of valid interargument relations, we use the technique proposed in [12]. Based on the definition of a given predicate, sufficient conditions for its valid interargument relation are generated. Any tuple of terms which satisfies these conditions belongs to the relation.

Example 9. For Example 1, we define the symbolic form for the polynomial interpretation as follows:

- A set of polynomials Σ_D with $D = \{x \in \mathbb{N} \mid x \geq \mu\}$,

- An assignment ϕ such that:
 - $\phi(X_1 + X_2) = p_1 X_1^2 + p_2 X_2^2 + p_{11} X_1 X_2 + p_{10} X_1 + p_{01} X_2 + p_{00}$,
 - $\phi(X_1 * X_2) = m_1 X_1^2 + m_2 X_2^2 + m_{11} X_1 X_2 + m_{10} X_1 + m_{01} X_2 + m_{00}$,
 - $\phi(\text{der}(X)) = \text{der}_2 X^2 + \text{der}_1 X + \text{der}_0$,
 - $\phi(u) = c_u$, $\phi(I) = c_1$,
 - and $\phi(d(X_1, X_2)) = d_{10} X_1 + d_{01} X_2 + d_{00}$.

The symbolic form for the linear polynomial interargument relation of the predicate $d(X_1, X_2)$ is:

$$R_{d/2} = \{(t_1, t_2) \mid dl_0 + dl_1 \|t_1\theta\|_I + dl_2 \|t_2\theta\|_I \geq_D dr_0 + dr_1 \|t_1\theta\|_I + dr_2 \|t_2\theta\|_I\},$$

where $P(X, Y) = dl_0 + dl_1 X + dl_2 Y$ corresponds to the left-hand side polynomial of the interargument relation and $Q(X, Y) = dr_0 + dr_1 X + dr_2 Y$ corresponds to the right-hand side polynomial of the relation. \square

4.2 Reformulating the Termination Conditions

In this section, we reformulate all termination conditions in Proposition (6), including the rigidity property, ‘D-closedness under evaluation’, the valid interargument relations and the decrease conditions, in the form of symbolic constraints, based on the symbolic forms of the polynomial interpretations and interargument relations. First, we reconsider the rigidity property.

Rigidity Conditions

Proposition 6 expresses that all elements of $\text{Call}(P, S)$ must be rigid w.r.t. I . Proposition 5 characterises rigidity of a term t by verifying if $\text{VREL}_I(t) = \emptyset$. Since there may be an infinite number of elements in the call set, it is impossible to verify the rigidity condition of $\text{Call}(P, S)$ by checking every element in the set. We present a sufficient method for checking the emptiness of $\text{VREL}_I(t)$ syntactically, resulting in a finite set of constraints on symbolic coefficients.

The call set $\text{Call}(P, S)$ can be specified as a (finite) set of call patterns, each presented as a rigid type graph [20]. Also in [20], Janssens and Bruynooghe propose a technique to infer the set of rigid type graphs given the set $\text{Call}(P, S)$. In the following, we recall and extend some basic definitions in [20] which are based on (semi-)linear norms and level-mappings to the case of general polynomial interpretations. We also give some examples to illustrate the idea.

There are a number of primitive types, each is used to represent a particular subset of constants in the language (e.g. **INT**, **REAL**). Let \mathbb{P} denote the set of all primitive types in the language. We assume that there exists a function *Denote*: $\mathbb{P} \rightarrow 2^{C^{const_P}}$. We recall the notion of *rigid type graphs*, which are an instance of directed graphs, as follows:

Definition 16 (rigid type graph). [20] *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* $\subseteq \text{Nodes} \times \text{Nodes}$ such that $(\text{Nodes}, \text{ForArcs})$ is a tree,
3. *BackArcs* $\subseteq \text{Nodes} \times \text{Nodes}$ such that for arc $(m, n) \in \text{BackArcs}$, node n is an ancestor of node m in *ForArcs*,
4. *Label* is a function $\text{Nodes} \rightarrow \mathbb{P} \cup \text{Func}_P \cup \text{Pred}_P \cup \{\mathbf{MAX}, \mathbf{OR}\}$,
5. *ArgPos* is a function: $\bigcup_{k>0} (\{m \in \text{Nodes} \mid \text{Label}(m) = f/k \in \text{Func}_P \cup \text{Pred}_P\} \times \{1, \dots, k\}) \rightarrow \text{Nodes} \setminus \{\text{root}\}$, such that for each $m \in \text{Nodes}$, with $\text{Label}(m) = f/k$, $\text{ArgPos}(m, \cdot): \{1, \dots, k\} \rightarrow \text{Nodes}$ is a bijection from $\{1, \dots, k\}$ onto $\{n \in \text{Nodes} \mid (m, n) \in \text{ForArcs} \cup \text{BackArcs}\}$.

The following example shows how to use rigid type graphs to represent a call set:

Example 10 (delete). Let us consider the following ‘delete’ program:

$$\begin{aligned} & \text{del}(H, [H|T], T). \\ & \text{del}(X, [H|T], [H|T_1]) : \text{-del}(X, T, T_1). \end{aligned} \quad (10)$$

and a query set $Q = \{\text{delete}(t_1, t_2, t_3) \mid t_2 \text{ is a nil-terminated list, } t_1, t_3 \text{ are free variables}\}$. The call set corresponding to the query set is $S = Q$. This set is specified by one call pattern which is represented by the rigid type graph in Fig. 1, where the left and right branches of the root ‘del/3’ correspond to the first

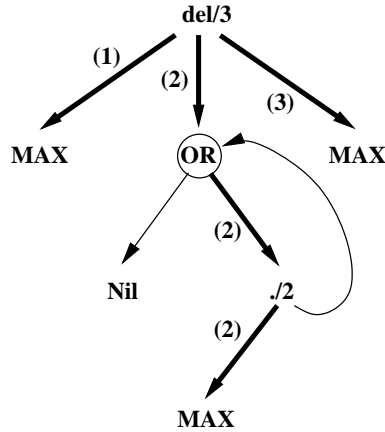


Fig. 1. Call set and critical paths for the delete/3 example

and third arguments of free variables t_1, t_3 . The middle branch connected to the node ‘OR’ shows that t_2 can be an empty list ‘Nil’ or a list containing a head with a free variable and a tail of a nil-terminated list. \square

For each rigid type graph representing an atom A , each node MAX in the graph corresponds to a possible occurrence of a free variable in A . The atom is rigid

w.r.t. the polynomial interpretation I if all these occurrences are not relevant occurrences w.r.t. I . In the following, we formulate this rigidity condition syntactically based on the rigid type graph.

Definition 17 (critical path [12]). Let $T = (Nodes, ForArcs, BackArcs, Label, ArgPos)$ be a rigid type graph. A critical path in T is a path of arcs in $ForArcs$ from the root node to a node labeled MAX .

Example 11 (delete-continued). In Fig. 1, there are three critical paths: (1): $((del, MAX))$, (2): $((del, OR), (OR, ./2), (./2, MAX))$, and (3): $((del, MAX))$. \square

The following proposition, extended from [11] (where in [11] each function and predicate symbol is associated with a semi-linear norm or level mapping), provides a method to generate sufficient constraints for rigidity from rigidity conditions.

Proposition 7. Let P be a program and $T = \{Nodes, ForArcs, BackArcs, Label, ArgPos\}$ be a rigid type graph of an atom $t \in Atom_P$. Let I be a polynomial interpretation of P and for any function and predicate symbol g/k , let $P_g(X_1, \dots, X_k) = \sum_{0 \leq j_1, \dots, j_k \leq M_g} g_{j_1 \dots j_k} X_1^{j_1} \dots X_k^{j_k}$ be the polynomial associated with g/k where $M_g \geq 0$ is the imposed largest power of a variable in $P_g(X_1, \dots, X_k)$. If on a critical path P to a MAX node of T there exists an arc (n_1, n_2) , with $Label(n_1) = f/k$ and $ArgPos(n_1, i) = n_2$ such that $\sum_{j_i > 0} f_{j_1 \dots j_k} = 0$, then the variable occurrence in the atom t corresponding to the MAX node is not a relevant occurrence w.r.t. I .

Proof. The condition $\sum_{j_i > 0} f_{j_1 \dots j_k} = 0$ assures that all coefficients of the monomials containing X_i with its maximum power > 0 are zero. Actually, it means X_i is not involved in $P_f(X_1, \dots, X_n)$. Therefore, for the considered MAX node, there is at least one functor on the critical path to this MAX node for which the argument position corresponding to the path is not involved in P_f . It implies that the MAX node is not relevant w.r.t. I . \square

Note that this proposition applies for arbitrary polynomial interpretations. Therefore, it is also applicable for the polynomial interpretations where each function symbol is associated with a linear, simple, or simple-mixed polynomial and each predicate symbol is associated with a linear polynomial, as imposed in this section.

Corollary 1. If on every critical path of T there exists an arc (n_p, n_{p+1}) , with $Label(n_p) = f^{(p)}/k_p$ and $ArgPos(n_p, i_p) = n_{p+1}$, such that $\sum_{j_{i_p} > 0} f_{j_1 \dots j_{k_p}}^{(p)} = 0$, then any atom t associated with T is rigid w.r.t. I .

Proof. Let $X_{(i)}$ be a variable occurrence of an arbitrary variable X in the atom t . This variable occurrence corresponds with a node MAX in the associated rigid type graph T of t . Since there exists an arc (n_p, n_{p+1}) with $Label(n_p) = f^{(p)}/k_p$ and $ArgPos(n_p, i_p) = n_{p+1}$ on the critical path of T to the MAX node such that $\sum_{j_{i_p} > 0} f_{j_1 \dots j_{k_p}}^{(p)} = 0$, Proposition 7 implies that $X_{(i)}$ is not a relevant occurrence

w.r.t. I .

Therefore, all variable occurrences in t are not relevant and $VREL_I(t) = \emptyset$, which implies that t is rigid w.r.t. I . \square

Corollary 2. *Let CP be a critical path of T . Let $(n_{t_1}, n_{t_1+1}), \dots, (n_{t_m}, n_{t_m+1})$ be all arcs in CP such that $Label(n_{t_p}) = f^{(t_p)}/k_p$ is a function or predicate symbol and $ArgPos(n_{t_p}, i_p) = n_{t_p+1}$. If for any such CP we have:*

$$\prod_{p=1}^m \left(\sum_{j_{i_p} > 0} f_{j_1 \dots j_{k_p}}^{(t_p)} \right) = 0 \quad (11)$$

then any atom t associated with T is rigid w.r.t. I .

Example 12 (delete-continued). Reconsider Example 11. From Fig. 1, $Call(P, S)$ is rigid w.r.t. I if the following constraints hold:

$$C_{rigid} = \begin{cases} del_{100} = 0 \\ del_{010} * (l_{11} + l_{10}) = 0 \\ del_{001} = 0 \end{cases}$$

\square

In the remaining of this subsection, we reconsider the other symbolic constraints, derived from the ‘D-closedness under evaluation’, the valid interargument relations, and the decrease conditions. We will show that they take one of the following two forms:

$$\forall \bar{X} \in D : P \gtrsim 0 \quad (12)$$

$$\forall \bar{X} \in D : P_1 \geq Q_1 \wedge \dots \wedge P_n \geq Q_n \Rightarrow P_{n+1} \gtrsim Q_{n+1} \quad (n \geq 1) \quad (13)$$

where P, P_i, Q_i are polynomials with non-negative integer coefficients and \gtrsim is \geq or $>$.

‘D-closedness under Evaluation’

Let us consider the ‘D-closedness under evaluation’ condition required for the polynomial interpretation. For a polynomial $P/n \in \Sigma_D$, the requirement is:

$$X_1 \geq \mu, \dots, X_n \geq \mu \Rightarrow P(X_1, \dots, X_n) \geq \mu.$$

We can rewrite the condition as:

$$\forall X_1, \dots, X_n \in D : P(X_1, \dots, X_n) - \mu \geq 0 \text{ or } P(X_1, \dots, X_n) - \mu \geq_D 0$$

which is of Form (12).

Valid Interargument Relations

For the valid interargument relations, we clarify the following two cases:

1. For the ‘fact’ clauses which contain only a head atom, a valid relation among arguments of the atom takes the form $P \geq Q$, as in Form (8), where both sides of the inequality are polynomials with non-negative integer symbolic coefficients. This is already in Form (12).
2. For any clause which contains both head and body parts, the condition for valid interargument relations can be stated as: if the interargument relation for any body-atom is valid, the interargument relation for the head atom is also valid. This gives the following type of condition for the clause:

$$\forall \bar{X} \in D : R_{p_1} \wedge \dots \wedge R_{p_n} \Rightarrow R_{p_{n+1}},$$

in which p_1, \dots, p_n are the predicates of the body-atoms, p_{n+1} is the predicate of the head atom and R_{p_k} is of the form $P \geq Q$, as in Form (8). It is clear that this formula is in Form (13).

Decrease Conditions

Finally, suppose that the rigidity and valid polynomial interargument conditions are satisfied, then the decrease condition between the head and any (mutually) recursive body-atom in any (mutually) recursive clause is generated. This condition is written as:

$$\forall \bar{X} \in D : R_{p_1} \wedge \dots \wedge R_{p_n} \Rightarrow H > B, \quad (n \geq 0)$$

where $R_{p_1} \dots R_{p_n}$ are the valid polynomial interargument relations for the intermediate body-atoms between the head and the (mutually) recursive body-atom, and H, B are polynomials corresponding to the head and the (mutually) recursive body-atom. Obviously, if there is no intermediate body atom in the clause ($n = 0$), the formula is in Form (12). Otherwise it takes Form (13). In both cases, the inequality sign \gtrsim is $>$.

Example 13. Reconsider Example 9. We want to reformulate all termination conditions for this example in symbolic forms. We will not write all polynomial constraints. In order to illustrate the main ideas of this section, we present only one constraint for each type of conditions instead.

- The rigidity condition of the call set $Call(P, S) = \{d(t_1, t_2) | t_1 \text{ is a ground term and } t_2 \text{ is a free variable}\}$ w.r.t. I : The rigid type graph corresponding to the call pattern of $Call(P, S)$ is represented in Fig. 2. From the figure, the following rigidity condition is generated:

$$d_{01} = 0.$$

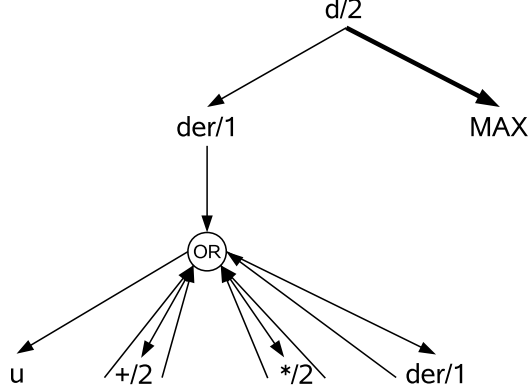


Fig. 2. Critical path $d/2$

- ‘D-closedness under evaluation’: For the polynomial associated with the function symbol $der/1$, we have the following constraint:

$$\forall X \in D : X \geq \mu \Rightarrow der_2 X^2 + der_1 X + der_0 \geq \mu$$

- Interargument relations: There are four clauses from which valid interargument conditions are inferred. We only present the condition on the last clause. The other conditions can be derived similarly.

$$\forall X, DX, DDX \in D :$$

$$dl_0 + dl_1(de_2 X^2 + de_1 X + de_0) + dl_2 DX \geq$$

$$dr_0 + dr_1(de_2 X^2 + de_1 X + de_0) + dr_2 DX$$

\wedge

$$dl_0 + dl_1(de_2 DX^2 + de_1 DX + de_0) + dl_2 DDX \geq$$

$$dr_0 + dr_1(de_2 DX^2 + de_1 DX + de_0) + dr_2 DDX$$

\Rightarrow

$$dl_0 + dl_1(de_2(de_2 X^2 + de_1 X + de_0)^2 + de_1(de_2 X^2 + de_1 X + de_0) + de_0) + dl_2 DDX \geq$$

$$dr_0 + dr_1(de_2(de_2 X^2 + de_1 X + de_0)^2 + de_1(de_2 X^2 + de_1 X + de_0) + de_0) + dr_2 DDX.$$

- Decrease conditions: There are 3 recursive clauses 5, 6 and 7 where decrease conditions can be inferred. In this example, we present the condition for the recursive body atom $d(der(DX), DDX)$ of the last clause:

$$\forall X, DX, DDX \in D :$$

$$dl_0 + dl_1(de_2 X^2 + de_1 X + de_0) + dl_2 DX \geq$$

$$dr_0 + dr_1(de_2 X^2 + de_1 X + de_0) + dr_2 DX$$

\Rightarrow

$$d_{10}(de_2(de_2 X^2 + de_1 X + de_0)^2 + de_1(de_2 X^2 + de_1 X + de_0) + de_0)$$

$$+ d_{01} DDX + d_{00} > d_{10}(de_2 DX^2 + de_1 DX + de_0) + d_{01} DDX + d_{00}.$$

□

4.3 From Symbolic Conditions to Constraints on Coefficients

Note that our mission is to find a polynomial interpretation such that all symbolic conditions generated in the previous section are satisfied. One way to do it is to transform all those conditions into a set of Diophantine constraints, in which the old coefficients in the conditions become the new variables in the transformed constraints. A solution for these Diophantine constraints is a sufficient condition for a termination proof of the program. This section explains how to do this.

As we analysed in Section 4.2, all generated rigidity constraints are in Form (11), which are already Diophantine constraints with symbolic coefficients as variables. The other symbolic constraints generated from the ‘closedness under evaluation’ property, the valid interargument relations and the decrease conditions, take Form (12):

$$\forall \bar{X} \in D : P \succcurlyeq 0$$

or Form (13):

$$\forall \bar{X} \in D : P_1 \geq Q_1 \wedge \dots \wedge P_n \geq Q_n \Rightarrow P_{n+1} \succcurlyeq Q_{n+1} \quad (n \geq 1)$$

where P, P_i, Q_i are polynomials with non-negative integer coefficients and \succcurlyeq is \geq or $>$.

In the following, we introduce a two-phases transformation method to transform all constraints of Forms (12) and (13) into sufficient Diophantine constraints on symbolic coefficients.

First Phase

In the first phase, all constraints of Form (13) are transformed into new symbolic constraints of Form (12) such that if there is any assignment for the unknown coefficients in the new constraints of Form (12) such that they hold, then it also guarantees that the original constraints of Form (13) hold. We have the following proposition:

Proposition 8. *Let Ω/n and $\Phi/1$ be two polynomials with non-negative integer coefficients where $\Phi/1$ is a strictly monotone polynomial. Then we have:*

$$\begin{aligned} \forall \bar{X} \in D : \Phi(P_{n+1}) - \Phi(Q_{n+1}) \succcurlyeq \Omega(P_1, \dots, P_n) - \Omega(Q_1, \dots, Q_n) \\ \Rightarrow \\ \forall \bar{X} \in D : P_1 \geq Q_1 \wedge P_2 \geq Q_2 \wedge \dots \wedge P_n \geq Q_n \Rightarrow P_{n+1} \succcurlyeq Q_{n+1}. \end{aligned}$$

where \succcurlyeq is \geq or $>$.

Proof. Let \bar{x} be an assignment to natural numbers in D for all variables \bar{X} in the constraint:

$$\Phi(P_{n+1}) - \Phi(Q_{n+1}) \succcurlyeq \Omega(P_1, \dots, P_n) - \Omega(Q_1, \dots, Q_n)$$

such that the constraint holds. We consider the following two cases:

1. There exists an index j , $0 \leq j \leq n$ such that: $\bar{x} : P_j \leq Q_j$. It implies:
 $\bar{x} : P_1 \geq Q_1 \wedge P_2 \geq Q_2 \wedge \dots \wedge P_n \geq Q_n$ false and
 $\bar{x} : P_1 \geq Q_1 \wedge P_2 \geq Q_2 \wedge \dots \wedge P_n \geq Q_n \Rightarrow P_{n+1} \gtrsim Q_{n+1}$ holds.
2. For all indexes j , $0 \leq j \leq n$, $\bar{x} : P_j \geq Q_j$. Because Ω is a polynomial with non-negative coefficients:
 $\bar{x} : \Phi(P_{n+1}) - \Phi(Q_{n+1}) \gtrsim \Omega(P_1, \dots, P_n) - \Omega(Q_1, \dots, Q_n) \geq 0$. It implies
 $\bar{x} : P_{n+1} \gtrsim Q_{n+1}$, so that again the implication holds.

In any case, we always have:

$$\bar{x} : P_1 \geq Q_1 \wedge P_2 \geq Q_2 \wedge \dots \wedge P_n \geq Q_n \Rightarrow P_{n+1} \gtrsim Q_{n+1}.$$

That is the requirement for the proof. \square

Proposition 9. Let $P = \Phi(P_{n+1}) - \Phi(Q_{n+1}) - \Omega(P_1, \dots, P_n) + \Omega(Q_1, \dots, Q_n)$. Then:

$$\forall \bar{X} \in D : \Phi(P_{n+1}) - \Phi(Q_{n+1}) \gtrsim \Omega(P_1, \dots, P_n) - \Omega(Q_1, \dots, Q_n)$$

if and only if

$$\forall \bar{X} \in D : P \gtrsim 0.$$

If we select a symbolic form for the polynomials Ω/n and Φ/n , we can reformulate the condition on Φ/n to be strictly monotonic using the following proposition:

Proposition 10. Let $\Phi(X_1, \dots, X_n) = \sum_{i_1, \dots, i_n=0}^M a_{i_1, \dots, i_n} X_1^{i_1} \dots X_n^{i_n}$ be a symbolic form for the polynomial Φ/n with unknown non-negative coefficients where M is the highest power of a variable. Φ/n is strictly monotonic on the variable position k , $0 \leq k \leq n$, if and only if:

$$\sum_{i_1, \dots, i_n=0, i_k > 0}^M a_{i_1, \dots, i_n} > 0$$

An easy choice for the symbolic form of Ω/n and Φ/m are the linear or simple-mixed polynomials.

By applying proposition (9) and (10), we can transform all generated implication constraints to inequality constraints such that if the latter constraints hold, the formers also hold.

Example 14. We continue with Example 9. We choose decrease condition (14) as an example showing how we transform an implication rule to a polynomial constraint.

Since there is only one constraint in the left-hand side of the implication rule, we choose a linear symbolic form for the polynomial Φ and a simple-mixed form for Ω :

$$\Omega(X) = om_0 + om_1X + om_2X^2 \qquad \Phi(X) = ph_0 + ph_1X.$$

The condition for the strict monotonicity of the polynomial $\Phi/1$ is:

$$ph_1 > 0$$

The new inferred generated polynomial constraint is:

$\forall \bar{X} \in D :$

$$\begin{aligned} & ph_0 + ph_1(d_{10}(de_2(de_2X^2 + de_1X + de_0)^2 + de_1(de_2X^2 + de_1X + de_0) + de_0) + \\ & d_{01}DDX + d_{00}) - ph_0 - ph_1(d_{10}(de_2DX^2 + de_1DX + de_0) + d_{01}DDX + d_{00}) \\ & - om_0 - om_1(dl_0 + dl_1(de_2X^2 + de_1X + de_0) + dl_2DX) - om_2(dl_0 + dl_1(de_2X^2 + \\ & + de_1X + de_0) + dl_2DX)^2 + om_0 + om_1(dr_0 + dr_1(de_2X^2 + de_1X + de_0) + \\ & dr_2DX) + om_2(dr_0 + dr_1(de_2X^2 + de_1X + de_0) + dr_2DX)^2 > 0. \end{aligned}$$

It is equivalent to:

$\forall \bar{X} \in D :$

$$\begin{aligned} & ph_1d_{10}(de_2(de_2X^2 + de_1X + de_0)^2 + de_1(de_2DX^2 + de_1X + de_0)) - ph_1d_{10} \\ & (de_2DX^2 + de_1DX + de_0) - om_1(dl_0 + dl_1(de_2X^2 + de_1X + de_0) + dl_2DX) - \\ & om_2(dl_0 + dl_1(de_2X^2 + de_1X + de_0) + dl_2DX)^2 + om_1(dr_0 + dr_1(de_2X^2 + \\ & de_1X + de_0) + dr_2DX) + om_2(dr_0 + dr_1(de_2X^2 + de_1X + de_0) + dr_2DX)^2 > 0 \end{aligned}$$

If we rewrite the left-hand side of the above constraint as a polynomial over variable X, DX in the distributive form, we get a new constraint of the following form:

$$M_1X^4 + M_2X^3 + M_3X^2 + M_4X + M_5DX^2 + M_6DX + M_7 > 0 \quad (15)$$

where M_1, \dots, M_7 are expressions containing only coefficients:

$ph_1, om_1, om_2, de_0, de_1, de_2, dl_0, dl_1, dl_2, dr_0, dr_1, dr_2.$ □

Second Phase

In this phase, we transform any constraint of Form (12): $\forall \bar{X} \in D : P \gtrsim \theta$, to a set of Diophantine constraints on symbolic coefficients. This transformation guarantees that if there is a solution for the set of Diophantine constraints (an assignment for each symbolic coefficient and constant in the constraints a non-negative integer), then the original constraint holds.

Note that any constraint of Form (12) can be easily transformed to an equivalent constraint of the form:

$$\forall \bar{X} \in D : Q \geq 0 \quad (16)$$

with Q a polynomial (i.e. if \gtrsim is $>$, we rewrite the constraint as $\forall \bar{X} \in D : P - 1 \geq 0$). Therefore, in stead of starting with the original constraints of Form (12), we can work with the equivalent constraints of Form (16).

In the context of termination analysis of TRS, this problem has been studied by several authors [5, 16, 19, 32]. In this paper, we apply the method in [19].

Proposition 11. *Let $P(X_1, \dots, X_n)$ be a polynomial with integer coefficients. We have $\forall \bar{X} \in D : P(X_1, \dots, X_n) \geq 0$, where $D = \{x \in \mathbb{N} | x \geq \mu\}$, if and only if the polynomial $Q(X_1, \dots, X_n) = P(X_1 + \mu, \dots, X_n + \mu)$ has non-negative integer coefficients only.*

Hence, to prove that there exists a set D and an assignment for each symbolic coefficient a non-negative integer such that $\forall \bar{X} \in D : P(X_1, \dots, X_n) \geq 0$, a heuristic is to first generate the polynomial Q w.r.t. a unknown bound value μ and then find μ and the symbolic coefficients such that all coefficients of Q are non-negative. The final problem becomes solving a set of Diophantine constraints with μ and the symbolic coefficients as variables.

Example 15. Constraint (15) in Example 14 can be rewritten as:

$$\forall \bar{X} \in D : M_1X^4 + M_2X^3 + M_3X^2 + M_4X + M_5DX^2 + M_6DX + M_7 - 1 \geq 0$$

Applying Proposition 11, we first calculate the polynomial $Q(X, DX)$:

$$Q(X, DX) = M_1(X + \mu)^4 + M_2(X + \mu)^3 + M_3(X + \mu)^2 + M_4(X + \mu) + M_5(DX + \mu)^2 + M_6(DX + \mu) + M_7 - 1$$

or:

$$Q(X, Y) = M_1X^4 + X^3(4M_1\mu + M_2) + X^2(6M_1\mu^2 + 3M_2\mu + M_3) + X(4M_1\mu^3 + 3M_2\mu^2 + 2M_3\mu + M_4) + M_5DX^2 + DX(2M_5\mu + M_6) + (M_1\mu^4 + M_2\mu^3 + M_3\mu^2 + M_4\mu + M_5\mu^2 + M_6\mu + M_7 - 1).$$

Then we derive the following set of constraints which contains symbolic coefficients $ph_1, om_1, om_2, de_0, de_1, de_2, dl_0, dl_1, dl_2, dr_0, dr_1, dr_2$ and μ as variables:

$$\begin{cases} M_1 \geq 0 \\ 4M_1\mu + M_2 \geq 0 \\ 6M_1\mu^2 + 3M_2\mu + M_3 \geq 0 \\ 4M_1\mu^3 + 3M_2\mu^2 + 2M_3\mu + M_4 \geq 0 \\ M_5 \geq 0 \\ 2M_5\mu + M_6 \geq 0 \\ M_1\mu^4 + M_2\mu^3 + M_3\mu^2 + M_4\mu + M_5\mu^2 + M_6\mu + M_7 - 1 \geq 0 \end{cases}$$

□

4.4 Solving Diophantine Constraints

The previous section shows that we can formulate all termination conditions in symbolic forms and transform them to a sufficient set of Diophantine constraints. The problem then becomes solving a system of nonlinear Diophantine constraints with the coefficients and μ as variables such that a solution for the Diophantine constraints implies termination of the logic program. It can be done

by using any available Diophantine solver.

There are a number of works dealing with solving Diophantine constraints (e.g. [9, 14]). The approach proposed in [9] provides an effective solution for this problem. The main idea of the approach is to put an arbitrary bound on the values of each variable in the constraints (e.g. $[0, B]$ where B is a non-negative integer), to turn it into an instance of the **FDCSPs**³, which is studied intensively, especially in the context of constraint logic programming. Alternatively, in [14], the constraint set is first encoded as a SAT-problem, and then a back-end SAT solver is used to solve the latter. For more details, we refer to [9, 14].

5 Experimental Evaluation

We have implemented a system (Polytool)⁴ for automated termination proof based on the approach. It is integrated in the system implementing the constraint-based approach of [12] and consists of four parts. The first part is the type inference engine of Janssens and Bruynooghe [20], coded in MasterProlog (IT Masters 2000). Based on this system, given a program and a set of queries, the call set is computed and the rigid type graph for each call pattern of the call set is generated. If we only consider query patterns with modes, we can replace this module by a groundness analysis tool (e.g. a groundness analysis based on a simple subdomain of Pos [18], a groundness analysis using Pos-based abstract interpretation [7]). The second part, the core of the system, which generates the set of all polynomial conditions, has been done in SICStus Prolog (SICS 3.12.2). Also the third part, which normalises the polynomial conditions and transforms them into Diophantine constraints, based on the approach discussed in Section 4.3, is implemented in SICS 3.12.2. We have also implemented an alternative transformation for this part which is mainly based on the work in [12] with an extension to nonlinear polynomial interpretations. A description for this transformation is in [27]. The final part is a back-end Diophantine constraint solver. There are two alternatives for it. The first tool that we have used is the *CiME* 2.02 of Contejean, Marché, Tomás and Urbain [9]. This tool is written in Objective CAML (CAML 3.0.9). The second one is the AProVE-SAT implemented by Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann and Harald Zankl [14]. After performing a thorough testing, we selected AProVE-SAT, since it is much faster in comparison with *CiME* 2.02. There are two versions of Polytool (Polytool-V1 and Polytool-V2). In the first version, we use interargument relations with modes and apply the transformation approach in [27]. In the second version, which was implemented later, we use the form of interargument relations and the transformation which are described in Sections 4.1 and 4.3. We have tested the performance of both versions of the system on a number of examples, including the benchmarks for Logic Programming (version 2006) in the Termination Problems Database [2] (Tables

³ Finite Domain Constraint Satisfaction Problems

⁴ For the source code, please refer to: <http://www.cs.kuleuven.be/~manh/polytool>

6, 4, 8, 5 and 3), and examples collected from other sources (Table 7)⁵. The strategy is as follows: first, we search for a polynomial interpretation with the form of linear polynomials. If we can not find such interpretation, we continue the search for an interpretation with a simple-mixed polynomial form. The domain of all symbolic coefficients in the generated Diophantine constraints is fixed to the set $D_{cof} = \{0, 1, 2\}$. The experiments have been performed using SICS 3.12.2, running on Intel Pentium IV 2.80 MHz, 1Gb RAM, Debian Linux 2.6.8. We have also performed an experimental evaluation with other systems, namely: the Hasta La Vista [31], the cTI [24], the TALP [28], and the AProVE analysers [17]. For Hasta La Vista and cTI, the whole set of benchmarks are tested. For TALP and AProVE, the test results of the benchmarks in the Termination Problems Database are collected from the 2006 Termination Competition for the Logic Programming category [1] and the results for the remaining benchmarks are done offline in the same machine. The maximum execution time for each benchmark on each termination tool is 300 seconds. For Polytool and TALP, polynomial interpretations are chosen. For cTI, we choose the ‘default’ option (with TNA = ‘enable’, NTI = ‘yes’ and Comp = ‘shown’). For AProVE, the ‘fully automatic’ mode is chosen. In the resulting tables, we do not provide the time for analysing the benchmarks. Only the success or failure of these systems w.r.t. the benchmarks is provided. In the tables, the following abbreviations are used:

- ‘*EXAMPLE*’ and ‘*QUERY*’ refer to the tested program and the set of atomic queries. Here, ‘g’ and ‘f’ denote a ground term and a free variable respectively.
- ‘*AP*’, ‘*TA*’, ‘*CI*’, ‘*PO1*’, ‘*PO2*’, and ‘*HA*’ refer to the results given by AProVE, TALP, cTI, Polytool-V1, Polytool-V2, and Hasta La Vista. It contains the symbol ‘*E*’ if the system gives an error during the analysis, ‘*N*’ if the benchmarks contains arithmetic expressions and the system is not appropriate for that, the symbol ‘+’ if the system reports termination, and the symbol ‘-’ if the system fails to do so. We use ‘+*’ if proving termination of an example requires a non-linear polynomial interpretation.

Since the aim of this work is to analyse termination of logic programs with only symbolic computation, we leave the examples containing arithmetic and built-in predicates out of the resulting tables. The remaining results of 330 benchmarks are summarised in tables 1 and 2. These tables show that AProVE is the most powerful system since it can prove termination of 234 benchmarks. It seems reasonable because in AProVE, a number of powerful termination techniques such as dependency pairs, semantic labelling, matchbox, A-transformation, polynomial interpretations, . . . are applied. Polytool-V2 is in second position, with 206 positive proofs. It is an interesting result since we use only polynomial interpretations as the technique for termination analysis. TALP (179 positive proofs),

⁵ In the table, examples ‘dist’, ‘der’ were collected from [30], example ‘tausky’ was introduced in [33]. The source of all these examples can be found in http://www.cs.kuleuven.be/~manh/polytool/new_examples.zip

	AProVE	TALP	cTI	Polytool-V1	Polytool-V2	Hasta-La-Vista
Positive	222	166	172	158	190	132
Negative	86	142	136	150	118	110
Error	0	0	0	0	0	68

Table 1. The result over 308 benchmarks in the Termination Problems Database

	AProVE	TALP	cTI	Polytool-V1	Polytool-V2	Hasta-La-Vista
Positive	12	13	6	13	16	5
Negative	10	9	16	9	6	17

Table 2. The result over 22 benchmarks in the other test set

cTI (178 positive proofs), and Polytool-V1 (171 positive proofs) are next. The final one is Hasta La Vista, with 137 positive proofs. The cause for poor performance of Hasta La Vista may come from the fact that the type analyser in this system is quite old and generates errors for a number of test examples.

5.1 Comparison between Polytool-V1 and Polytool-V2

It is clear from the tables that Polytool-V2 is much better than Polytool-V1 in term of precision. As we discussed in Section 4.1, the use of interargument relations based on modes in Polytool-V1 is the main reason for the precision lost. Proving termination of programs such as Example 8 in the paper, or ‘BCGGV05/transpose-fb.pl’, ‘lpexamples/log2a.pl’, ‘talp/plumer/pl7.6.2c.pl’,... in the Termination Problem Database is impossible for Polytool-V1.

However, there is also a trade-off. In Polytool-V2, we loose efficiency. Polytool-V2 is slower than Polytool-V1 in most examples. This is due to the use of a more general form for the interargument relations and multiplication of two levels of symbolic coefficients in the transformation of the termination conditions to Diophantine constraints, which does not appear in Polytool-V1.

5.2 Comparison between Hasta La Vista and Polytool-(V1,V2)

Let us first compare the precision and efficiency between Polytool and Hasta La Vista since these systems have a similar framework of the constraint-based approach. From a theoretical point of view, for benchmarks without meta-predicates or arithmetic in them, termination analysis of Polytool is at least as precise as the analysis of Hasta La Vista. The claim comes from the fact that the approach based on polynomial interpretations used in Polytool can be considered as a generalisation of the (semi-)linear norm based approach used in Hasta La Vista. The results in Table 7 show that there is a class of examples (e.g. ‘dist’, ‘der’, ‘SK90_1’, ‘taussky’, ...) which can not be solved by Hasta La Vista, but can be solved by Polytool. For those examples, nonlinear polynomial interpretations are required.

Example 16. Consider again Example 4. With the analysis of Polytool, the following nonlinear polynomial interpretation is produced: $P_{dist}(X_1, X_2) = X_1$, $P_*(X_1, X_2) = 2X_1X_2$, $P_+(X_1, X_2) = X_1 + X_2 + 1$, $c_x = 0$, and a set $D = \mathbb{N}$. Termination of Example 1 is also successfully proved by Polytool and the following polynomial interpretation and valid interargument relation are generated: A polynomial interpretation I with the set $D = \mathbb{N}$ and $P_d(X_1, X_2) = X_1 + 2$, $P_{der}(X) = X^2 + 2X + 2$, $P_{plus}(X_1, X_2) = X_1 + 2X_2 + 1$, $P_{mul}(X_1, X_2) = 2X_1 + X_2 + 2$, $c_u = 2$, $c_l = 0$ and a valid interargument relation $R_{d/2} = \{(t_1, t_2) \mid \|t_1\|_I \geq_D \|t_2\|_I + 2\}$. Hasta La Vista, in contrast, cannot prove termination of these examples. \square

Observe that independent of whether we choose (semi-)linear norms and level mappings or polynomial interpretations, it may still give rise to nonlinear Diophantine constraints in the final step. Therefore, the requirement for a fast and effective nonlinear Diophantine constraint solver is necessary and AProVE-SAT seems to be a good selection. For CiME, when the maximum degrees of variables or the domain of each variable in the constraints increase, the performance of the solver decreases considerably.

Example 17. Consider the program ‘normal’ with the query set ‘norm(g, f)’ in Table 8:

$$\begin{aligned} norm(F, N) &: -rewrite(F, F1), norm(F1, N). \\ norm(a, a). & \quad rewrite(op(op(A, B), C), op(A, op(B, C))). \\ rewrite(op(A, op(B, C)), op(A, L)) &: -rewrite(op(B, C), L). \end{aligned}$$

For this example, both Polytool and Hasta La Vista produce nonlinear Diophantine constraints, but only Polytool succeeds. If we take the constraints generated by Hasta La Vista as an input for AProVE-SAT, it also gives a positive result. This shows that the constraint solver used in Hasta La Vista, **CLPFD**⁶, which mostly works with linear Diophantine constraints, is not powerful enough to solve such constraint sets. \square

In Hasta La Vista, all constant symbols in the input program are mapped to a same value (zero). Polytool, in contrast, maps different constant symbols to different constants in D . This property allows it to solve examples where constant symbols play an important role in termination behavior of the program. E.g. Termination of the Example ‘pl2.3.1’ in Table 8:

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

with the query set ‘ $p(g, f)$ ’ can be verified by Polytool, but can not by Hasta La Vista.

Another issue is efficiency. Overall, Hasta La Vista is faster than Polytool on

⁶ Constraint Logic Programming over Finite Domain

a large number of benchmarks. The reason is that termination analysis based on polynomial interpretations increases the number of coefficients of the polynomials associated with predicate and function symbols in the interpretations, which are variables in the generated Diophantine constraints. This leads to less efficiency of Polytool.

5.3 Comparison between AProVE, TALP and Polytool-(V1,V2)

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, it is applied indirectly in TALP and AProVE: given a logic program and a query set, they first transform them to a TRS, while preserving the termination behavior of the original program. Then, termination analysis is applied to the target TRS. Based on it, a number of other termination techniques developed for TRS becomes usable for the analysis. In fact, both TALP and AProVE are powerful systems with a large number of successful termination strategies. A limitation of the transformation approach in TALP is that only well-moded logic programs are considered [4, 15]. The results in the tables show that there are a number of examples, which are not well-moded for a specific query pattern, solvable by Polytool, but not solved by TALP (E.g., examples ‘*pl1.1*’, ‘*binary4*’ and ‘*pl4.5.3c*’ in Table 8, examples ‘*append-bff*’, ‘*delete-bff*’, ‘*flatlength-bfb*’ in Table 6). AProVE instead applies a quite strong transformational approach which can also deal with non well-moded logic programs [29]. However, this technique requires a variable condition that may cause the failure of the analysis on the transformed TRS (E.g. examples ‘*delete-bff*’, ‘*less-bf*’, and ‘*member-fb*’ in Table 6).

Example 18. Consider the following example ‘*less-bf*’ with the query set ‘*less(b,f)*’ in Table 6:

$$\text{less}(0, s(X)). \quad \text{less}(s(X), s(Y)) : \neg \text{less}(X, Y).$$

This example is very simple and can be proved terminating by most termination systems which use direct termination proving techniques (e.g. cTI, Hasta La Vista and Polytool). However, both TALP and AProVE fail to prove it. For TALP, this example is not well-moded. For AProVE, the dependency pairs method applied in the approach fails to prove termination of the transformed TRS. \square

Another problem is the efficiency of the transformational approaches used in TALP and AProVE. They may generate a very complex term rewriting system from even a simple logic program, such that most current termination analysers are unable to prove its termination. For example, within 300 seconds, Polytool can easily prove termination of Example 1. However, in the same time, both AProVE and TALP fail to do it and report a ‘timeout’ message.

5.4 Is $\mu = 0$ enough?

As we mentioned in the previous section, theoretically, the selection of μ in the set $D = \{x \in \mathbb{N} | x \geq \mu\}$ does not influence the correctness of the termination proof [9]. However, a question is: ‘Is the selection of $\mu = 0$ practically enough?’. The answer is ‘no’ for Polytool and the following example illustrates this point:

Example 19. Let us consider the example ‘taussky’ with the query set ‘ $q(b,f)$ ’ in Table 7 (see also [21]).

$$\begin{aligned} q(\text{mul}(X, \text{mul}(Y, Z)), T) &: -q(\text{mul}(\text{mul}(X, Y), Z), T). \\ q(\text{mul}(e, e), e). \quad q(\text{mul}(X, i(X)), e). \\ q(i(\text{mul}(X, Y)), Z) &: -q(\text{mul}(i(Y), i(X)), Z). \\ q(g(\text{mul}(X, Y), Y), Z) &: -q(f(\text{mul}(X, Y), X), Z). \quad q(f(e, Y), Y). \end{aligned}$$

For this example, if we fix the set D of Σ_D in the polynomial interpretation $D = \mathbb{N}$ ($\mu = 0$) and the domain for symbolic coefficients of the polynomials $D_{cof} = 0, 1, 2$, Polytool produces no solution. If we relax $D = \{x \in \mathbb{N} | x \geq \mu\}$ with μ as unknown coefficient, the system successfully proves termination of the example and generates the polynomial interpretation: $P_q(X_1, X_2) = X_1$, $P_{\text{mul}}(X_1, X_2) = 2X_1X_2 + X_2$, $P_i(X) = 2X^2$, $P_g(X_1, X_2) = 2$, $P_f(X_1, X_2) = 1$, and a set $D = \mathbb{N} \setminus \{0\}$. The reason is that if we fix $\mu = 0$, the domain $D_{cof} = 0, 1, 2$ may not be large enough to be able to find suitable values for the coefficients. \square

Of course, when fixing $\mu = 0$, we already reduce the number of variables in the Diophantine constraints by one and the execution time of the Diophantine constraint solver may become quite shorter.

EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA	EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA
BW/append	app(b,b,f)	+	+	+	+	+	+	old/reach	reach(b,b,b,b)	+	+	+	-	+	+
BS/app-ooi	app(f,f,b)	+	+	+	+	+	+	old/rotate	rotate(b,f)	+	+	+	+	+	+
BS/app	app(b,b,f)	+	+	+	+	+	+	old/sleaves	sleaves(b,b)	+	+	+	+	+	-
BS/ways	ways(b,b,f)	+	-	+	-	+	E	old/sublist0	sublist(b,b)	+	+	+	+	+	+
COM/der	der(b,f)	-	-	-	-	+	E	old/slist-bad	sublist(b,f)	-	-	-	-	-	-
cti/som	som3(b,f,f)	+	-	+	+	+	+	old/sublist	sublist(f,b)	+	+	+	+	+	-
old/ack	ack(b,b,f)	+	+	-	-	-	-	old/subset-no	subset(f,b)	-	-	-	-	-	+
old/app3-bis	app3-b(b,b,b,f)	+	+	+	+	+	+	old/subset	subset(b,b)	+	+	+	+	+	+
old/app3	app3-a(b,b,b,f)	+	+	+	+	+	+	old/u-bal-tree	balance(b,f)	+	-	+	-	+	-
old/bal-tree2	balance(f,b)	-	-	-	-	-	E	SC/NJ1	rev(b,f)	+	-	+	+	+	+
old/bal-tree	balance(b,f)	+	-	+	-	+	E	SC/NJ2	f(b,b,f)	+	+	+	+	+	-
old/inorder	inorder(b,f)	+	+	+	+	+	+	SC/NJ3	ack(b,b,f)	+	+	-	-	-	-
old/intleave	intleave(b,b,f)	+	+	+	+	+	+	SC/NJ4	p(b,b,b,f)	+	+	+	+	+	+
old/merge	merge(b,f)	+	-	-	-	-	-	SC/NJ5	f(b,b,f)	+	+	-	+	+	-
old/perm1	perm(b,f)	+	+	+	-	-	-	SC/NJ6	f(b,b,f)	+	+	+	+	+	+
old/perm2	perm(b,f)	+	+	+	+	+	+	TB/append	app(b,b,f)	+	+	+	+	+	+
old/queens	queens(b,f)	+	+	+	N	N	E	TB/transpose	transpose(b,f)	-	-	+	+	+	E
old/qsort	qs(b,f)	+	+	+	+	+	E	UN/preorder-dl	preorder(b,f)	+	-	+	+	+	+

Table 3. Termination Problem Database 2006: Terminweb benchmarks

EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA	EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA
ack-oi	ack(b,f,b)	-	-	-	-	-	-	log2a-oi	log2(f,b)	-	-	-	-	-	-
ack	ack(b,b,f)	+	+	-	-	-	-	log2a	log2(b,f)	+	+	+	-	+	-
average-oi	av(b,f,b)	+	+	+	+	+	-	log2b-oi	log2(f,b)	-	-	-	-	-	-
average	av(b,b,f)	+	+	+	+	+	-	log2b	log2(b,f)	+	+	+	+	+	+
factorial	fact(b,f)	-	-	-	N	N	E	mapcolor	cl-map(f,b)	-	-	-	-	-	-
fib-oi	fib1(f,b)	-	-	-	N	N	E	mergesort-oi	merge(f,b)	-	-	-	-	-	-
fib	fib1(b,f)	-	-	-	N	N	E	mergesort	merge(b,f)	+	-	-	-	-	E
hanoi	mov(b,b,b,b)	-	-	-	N	N	E	shapes	shapes(b,f)	+	-	-	-	-	E
kay4	kay4(b)	-	-	-	N	N	E	tautology	tautolog(b)	-	+	-	-	-	E

Table 4. Termination Problem Database: LPEXAMPLES Benchmarks

EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA	EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA
ag01	h(b)	+	+	-	-	-	E	incomp	p(b)	-	-	-	-	-	-
applast	goal(b,f,f)	+	-	-	-	-	E	incomp-var	p(f)	+	-	-	+	+	+
at	at(f,f)	+	+	-	-	-	E	intlist	int(b,b,f)	+	+	-	-	-	-
avg-bfb	avg(b,f,b)	+	+	+	+	+	E	lleans	lleaves(b,b)	+	+	+	+	+	-
avg	avg(b,b,f)	+	+	+	+	+	E	log	log(b,f)	+	+	+	+	+	-
baby91	f(b,f)	-	-	-	-	-	E	mapcolor	cl-map(f,b)	-	-	-	-	-	-
bappend	goal(g)	+	-	-	-	-	E	palind	palind(b)	+	+	+	+	+	+
blist	goal(b)	+	-	-	-	-	E	paper1	p(b,f)	+	-	-	-	-	-
btappend	goal(g)	+	-	-	-	-	E	paper2	p(b,f)	+	-	-	-	-	-
btapplast	goal(b,f,f)	+	-	-	-	-	E	parse	parse(b,f)	+	+	-	-	+	+
btree	goal(b)	+	-	-	-	-	E	perm	perm1(b,b)	+	+	+	+	+	+
cconfdel	conf(b)	+	-	-	-	-	E	plus	plus(b,f,f)	+	-	+	+	+	+
cnfequiv	cnfequiv(b,f)	-	-	-	-	-	-	p-nonlin	p(b)	+	+	-	-	-	+
confdel	conf(b)	+	-	-	-	-	E	p	p(b)	+	+	-	+	+	-
convert	conv(b,b,f)	+	+	+	+	+	+	pplus2	plus(b,f,f)	+	-	+	+	+	+
cstack	cstack(b,f)	+	+	-	+	+	E	pplus	plus(b,f,f)	+	-	+	+	+	+
csnake	tsnake(b,b,b)	-	-	-	-	-	E	preorder	preorder(b,f)	+	-	+	+	+	+
d-halfpred	f(b)	-	-	-	-	-	-	prime	prime(b)	+	-	-	-	-	-
d	d(b,b,f)	+	+	-	-	-	E	quot	div(b,b,f)	+	+	-	+	+	-
evenodd	even(b)	+	+	+	+	+	-	rev	rev(b,f)	+	+	-	-	+	-
factor	factor(b,f)	+	+	-	+	+	+	samefringe	sfringe(b,b)	+	+	-	+	+	-
flatten-phd	flat(b,f)	+	+	-	-	+	-	shuffle	query(b)	+	-	-	-	-	-
flatten	flat(b,f)	+	+	-	+	+	-	snake	tsnake(b,b,b)	-	-	-	-	-	E
giesl97	f(b,f,f)	+	-	+	+	+	+	times2	times(b,b,f)	+	-	+	+	+	-
gopher	gopher(b,f)	+	+	-	+	+	+	times	times(b,b,f)	+	+	-	-	-	-
hbal-tree	hbtree(b,f)	+	-	-	+	+	-	toyama	f(f,f,b)	+	-	-	-	-	-
ifdiv	div(b,b,f)	-	-	-	-	-	-	transpose2	tranaux(f,b,f)	-	-	-	-	-	-
ifminus	minus(b,b,f)	+	-	-	+	+	-	transpose-fb	tran(f,b)	+	+	-	-	+	E
incomp2	f(b)	+	-	+	-	-	-	weight	weight(b,f)	+	+	-	-	-	-
ack	ack(b,b,f)	+	+	-	-	-	-								

Table 5. Termination Problem Database: SGST06 Benchmarks

EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA	EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA
ackerman	ack(b,b,f)	+	+	-	-	-	-	merge	ms(f,b)	-	-	-	-	-	E
append-bff	app(b,f,f)	+	-	+	+	+	+	min-bf	min(b,f)	+	+	+	+	+	+
append-ffb	app(f,f,b)	+	+	+	+	+	+	min-fb	min(f,b)	-	-	-	-	-	-
delete-bbf	delete(b,b,f)	-	-	+	+	+	+	mult	mult(b,b,f)	+	+	+	+	+	E
delete-bfb	delete(b,f,b)	-	-	+	+	+	+	nrev-bf	rev(b,f)	+	+	+	+	+	+
delete-bff	delete(b,f,f)	-	-	-	-	-	E	nrev-fb	rev(f,b)	-	-	-	-	-	-
delete-ffb	delete(f,b,f)	-	-	+	-	-	E	numeral	num(b)	+	+	+	+	+	+
delete-ffb	delete(f,f,b)	-	-	+	-	-	E	ordered	ordered(b)	+	+	+	+	+	+
delmin-bff	delmin(b,f,f)	+	-	+	+	+	E	palind	palind(b)	+	+	+	+	+	+
delmin-ffb	delmin(f,f,b)	+	-	+	+	+	E	parse	parse(b,f)	+	+	-	+	+	E
der-bf	p(b,f)	+	+	-	-	+	E	perm1-fb	perm1(b,f)	+	+	+	+	+	-
der-fb	p(f,b)	-	-	-	-	-	E	perm-bf	perm(b,f)	+	+	+	-	-	E
factor	factor(b,f)	+	+	-	-	+	+	perm-fb	perm(f,b)	-	-	-	-	-	+
flat-bf	flat(b,f)	+	+	+	+	+	-	p-nonlin	p(b)	+	+	-	-	-	+
flat-fb	flat(f,b)	-	-	-	-	-	-	p	p(b)	+	+	-	+	+	-
flatlen-bbf	fl(b,b,f)	+	+	+	+	+	+	prefix-bf	prefix(b,f)	+	-	+	+	+	+
flatlen-bfb	fl(b,f,b)	+	-	+	+	+	-	prefix-fb	prefix(f,b)	+	+	+	+	+	+
flatlen-bff	fl(b,f,f)	+	-	+	+	+	-	qsort-bf	qs(b,f)	+	+	+	+	+	+
flatlen-ffb	fl(f,b,f)	-	-	-	-	-	-	qsort-fb	qs(f,b)	-	-	-	-	-	E
flatlen-ffb	fl(f,f,b)	-	-	-	-	-	-	reverse-bf	reverse(b,f)	+	+	+	+	+	+
frontier-bf	front(b,f)	+	+	+	+	+	+	reverse-fb	reverse(f,b)	-	-	-	-	-	-
frontier-fb	front(f,b)	-	-	-	-	-	-	s-tree	s-tree(b)	+	+	+	+	+	+
g	g(f)	-	-	-	-	-	E	select-bff	select(b,f,f)	-	-	-	-	-	-
in-bf	in(b,f)	-	-	-	-	-	E	select-fbf	select(f,b,f)	+	+	+	+	+	+
in-fb	in(f,b)	-	-	+	-	-	E	select-ffb	select(f,f,b)	+	-	+	+	+	+
inorder-bf	inorder(b,f)	+	+	+	+	+	+	slsort-bb	ss(b,b)	+	+	+	+	+	+
inorder-fb	inorder(f,b)	-	-	-	-	-	-	slsort-bf	ss(b,f)	+	+	+	-	+	E
insert-bbf	insert(b,b,f)	+	+	+	+	+	+	slsort-fb	ss(f,b)	-	-	-	-	-	E
insert-bfb	insert(b,f,b)	+	+	+	+	+	+	sublist-bf	sublist(b,f)	-	-	-	-	-	-
insert-bff	insert(b,f,f)	-	-	-	-	-	E	sublist-fb	sublist(f,b)	+	+	+	+	+	+
insert-ffb	insert(f,b,f)	-	-	+	-	-	E	subset-bf	subset(b,f)	-	-	-	-	-	-
insert-ffb	insert(f,f,b)	-	-	+	-	-	E	subset-fb	subset(f,b)	-	-	-	-	-	-
length1	len1(b,f)	+	+	+	+	+	+	suffix-bf	suffix(b,f)	-	-	-	-	-	-
length	len(b,f)	+	+	+	+	+	+	suffix-fb	suffix(f,b)	-	-	+	+	+	+
less-bf	less(b,f)	-	-	+	+	+	E	sum-bfb	sum(f,b,f)	+	+	+	+	+	+
less-fb	less(f,b)	+	+	+	+	+	+	sum-ffb	sum(f,f,b)	+	+	+	+	+	+
list	list(b)	+	+	+	+	+	+	t	t(b)	+	-	-	-	-	E
map-color	map(f)	+	-	+	+	+	+	transp-bb	tran(b,b)	+	+	+	+	+	E
max-bff	max(b,f,f)	-	-	+	+	+	E	transp-bf	tran(b,f)	-	-	+	+	+	E
max-bfb	max(f,b,f)	-	-	+	+	+	E	transp-fb	tran(f,b)	+	+	-	-	-	E
max-ffb	max(f,f,b)	+	+	+	+	+	+	treemem-bf	treem(b,f)	-	-	-	-	-	-
member-bf	member(b,f)	-	-	-	-	-	-	treemem-fb	treem(f,b)	+	+	+	+	+	+
member-fb	member(f,b)	+	+	+	+	+	+	tree	bin-tree(b)	+	+	+	+	+	+

Table 6. Termination Problem Database: BCGGV05 Benchmarks

EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA	EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA
SK90_4	p(g,f)	+	+	-	-	-	-	dist	dist(g,f)	+	+	-	+	+	-
taussky	q(g,f)	+	+	-	+	+	-	der	d(g,f)	-	-	-	+	+	-
addmul	p(g,f)	+	-	-	+	+	+	boolexp	cequiv(g)	-	-	-	-	-	-
fibonacci	p(g,f)	+	+	-	-	-	-	car_1	div(g,g,f)	+	-	-	-	+	-
lamdacal	g(g,g,f)	-	-	-	-	-	+	car_13	in(g,g,f)	+	+	-	-	-	-
flat	flat(g,f)	-	+	-	+	+	-	SK90_3	sum(g,f)	+	+	-	+	+	+
log-1	log(g,f)	-	+	+	-	+	-	fac_TRS	fac(g,f)	+	-	+	-	+	-
treecomp	less_leaves(g,g,f)	+	+	+	+	+	-	fward_ins	f(f,f,f)	-	-	-	-	-	-
SK90_2	p(g,f)	+	-	-	-	+	-	AC.04	div(g,g,f)	-	+	+	+	+	+
average1	av(g,g,f)	-	+	+	+	+	-	SK90	d(g,f)	+	+	-	+	+	+
average2	av(g,f,g)	-	+	+	+	+	-	SK90_1	p(g,f)	-	-	-	+	+	-

Table 7. Other termination tests

EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA	EXAMPLE	QUERY	AP	TA	CI	PO1	PO2	HA
apt/append	app2(f,b,b)	+	+	+	+	+	+	PL/merge-t	merge(b,f)	+	-	+	-	+	+
apt/curry-ap	type(b,b,f)	+	-	-	+	+	E	PL/pl1.1	app(f,f,f)	-	-	-	-	-	-
apt/dcschem	dcsolve(b,f)	-	-	+	+	+	+	PL/pl1.2	perm(b,f)	+	+	+	-	+	+
apt/fold	fold(b,b,f)	+	+	+	+	+	+	PL/pl1.2-t	perm(b,f)	+	+	+	-	+	+
apt/gtsolve	gtsolve(b,f)	+	-	+	+	+	+	PL/pl2.3.1	p(b,f)	+	+	-	+	+	+
apt/list	list(b)	+	+	+	+	+	+	PL/pl3.1.1	a	-	-	-	-	-	-
apt/lte	goal	+	-	+	+	+	+	PL/pl3.5.6a	p(f)	-	-	-	-	-	-
apt/map1	map(b,f)	+	-	+	-	+	E	PL/pl3.5.6	p(f)	-	-	-	-	-	-
apt/map	map(b,f)	+	+	+	+	+	+	PL/pl4.0.13oi	ap3(f,f,f,b)	-	-	-	-	-	-
apt/member	member(f,b)	+	+	+	+	+	+	PL/pl4.0.1	ap3(b,b,b,f)	+	+	+	+	+	+
apt/merge-ap	merge(b,f,f)	+	-	-	-	-	-	PL/pl4.4.3	merge(b,b,f)	+	+	+	+	+	+
apt/merge-av	merge(b,f,f)	-	-	-	-	-	-	PL/pl4.4.6a	perm(b,f)	+	+	+	+	+	+
apt/merge	merge(b,f)	+	-	-	-	-	-	PL/pl4.5.2	s2(b,f)	-	-	-	-	-	-
apt/nai-revoi	reverse(f,b)	-	-	-	-	-	-	PL/pl4.5.3a	p(b)	-	-	-	-	-	-
apt/nai-rev	reverse(b,f)	+	+	+	+	+	+	PL/pl4.5.3b	p(b)	-	-	-	-	-	-
apt/ordered	ordered(b)	+	+	+	+	+	+	PL/pl4.5.3c	goal(b)	-	-	-	-	-	-
apt/overlap	overlap(b,b)	+	+	+	+	+	+	PL/pl5.2.2	turin(b,b,b,f)	-	-	-	-	-	-
apt/perm	perm(b,f)	+	+	+	+	+	-	PL/pl6.1.1	qsort(b,f)	+	+	+	+	+	+
apt/qsor-oi	qs(f,b)	-	-	-	-	-	-	PL/pl7.2.9	mult(b,b,b)	+	+	+	+	+	+
apt/qsor	qs(b,f)	+	+	+	+	+	-	PL/pl7.6.2a	reach(b,b,b)	-	-	-	-	-	-
apt/select1	select(b,b,f)	+	+	+	+	+	+	PL/pl7.6.2b	reach(b,b,b,b)	-	-	-	-	-	-
apt/select	select(f,b,f)	+	+	+	+	+	+	PL/pl7.6.2c	reach(b,b,b,b)	+	+	+	-	+	-
apt/SS-mapo	color-map(f,b)	-	-	-	-	-	-	PL/pl8.2.1a	merge(b,f)	+	-	+	+	+	-
apt/SS-map	color-map(f,b)	-	-	-	-	-	-	PL/pl8.2.1	merge(b,f)	+	-	-	-	-	+
apt/SS-mapt	color-map(f,b)	-	-	-	-	-	-	PL/pl8.3.1a	minsort(b,f)	+	+	+	-	+	+
apt/subset1	subset1(f,b)	-	-	-	-	-	-	PL/pl8.3.1	minsort(b,f)	-	-	-	-	-	+
apt/subset	subset(b,b)	+	+	+	+	+	+	PL/pl8.4.1	even(b)	+	+	+	+	+	+
apt/sum	sum(f,f,b)	+	+	+	+	+	+	PL/pl8.4.2	e(b,f)	+	+	+	+	+	-
az/flat-oi	flat(f,b)	-	-	-	-	-	-	TAB/b-sblist	sublist(b,b)	-	-	-	-	-	-
az/flat	flat(b,f)	+	+	-	+	+	-	TAB/merge	merge(b,f)	+	-	-	-	-	-
az/perm	perm(b,f)	+	+	+	-	+	-	TAB/perm1	perm(b,f)	+	+	+	-	+	-
az/p	p(b)	+	+	+	+	+	+	TAB/perm2	perm(b,f)	+	+	+	+	+	-
dds/append	append(b,b,f)	+	+	+	+	+	+	TAB/qicksort	qs(b,f)	+	+	+	+	+	+
dds/dis-comb	con(b)	+	+	+	+	+	-	TAB/queens	queens(b,f)	+	+	+	N	N	+
dds/dis-con	dis(b)	+	+	+	+	+	E	TAB/rotate	rotate(b,f)	+	+	+	+	+	+
dds/dup	duplicate(b,f)	+	+	+	+	+	+	TAB/sleaves	sleaves(b,b)	+	+	+	+	+	+
dds/merge	merge(b,b,f)	+	+	+	+	+	+	TAB/sublist1	sublist(b,b)	+	+	+	+	+	+
dds/permute	permute(b,f)	+	+	+	+	+	+	TAB/sublist	sublist(b,b)	+	+	+	+	+	+
dds/rev-ii	reverse(b,b,f)	+	+	+	+	+	+	talp/append	app3(f,f,b)	+	+	+	+	+	+
dds/rev	reverse(b,f,b)	+	+	+	+	+	+	talp/binary2	add(f,f,b)	+	+	-	+	+	E
dds/sum-oi	sum(b,f,b)	+	+	+	+	+	+	talp/binary3	times(b,b,f)	+	+	-	+	+	E
dds/sum	sum(b,b,f)	+	+	+	+	+	+	talp/binary4	times(f,f,b)	-	-	-	-	-	E
MA/bid	bid(b,f,f,f)	-	-	+	N	N	E	talp/binary	add(b,b,f)	+	+	-	+	+	E
MA/derivoii	d(f,b,b)	-	-	+	N	N	E	talp/div	div(b,b,f)	+	+	+	+	+	+
MA/deriv	d(b,b,f)	-	-	+	N	N	E	talp/evaluate	myis(f,b)	+	+	+	-	+	E
MA/fib	fib(b,f)	-	-	-	N	N	E	talp/ex1	p(b,f)	+	-	-	-	-	-
MA/fib-t	fib(b,f)	+	+	+	-	+	+	talp/ex4-2	p2(f)	-	-	-	-	-	-
MA/gram2	parse(b,f)	-	-	+	N	+	E	talp/ex4	p1(b)	+	+	+	+	+	+
MA/gramr	goal	-	-	+	N	+	E	talp/flat	flat(b,f)	+	+	-	+	+	-
MA/hanoi	shan(b,b,b,b,f)	-	-	-	N	N	E	talp/gcd	gcd(b,b,b,f)	+	-	+	-	+	-
MA/hanoi.s	shan(b,b,b,b,b,f)	+	+	+	N	N	E	talp/nat	fact(b,f)	+	+	+	+	+	+
MA/matrix	mmult(b,b,b,f)	-	+	+	-	-	-	talp/normal	normal(b,f)	+	+	-	+	+	-
MA/money	m(f,f,f,f,f,f,f,f)	+	+	+	N	N	E	talp/palind	palind(b)	+	+	+	+	+	+
MA/progeom	pds(b,f)	-	-	-	N	N	-	talp/perm	perm(b,f)	+	+	+	-	+	+
MA/qsorap	qsor(b,f)	+	+	+	-	+	+	talp/permute	perm(b,f)	+	+	+	+	+	+
MA/query	query(b)	-	-	+	N	+	E	talp/qsor	qs(b,f)	+	+	+	+	+	+
MA/tak	tak(b,b,b,f)	-	-	-	N	N	E	talp/rem-oi	rem(b,b,b)	-	-	+	-	+	-
MA/zebra	zeb(f,f,f,f,f,f,f,f)	+	-	+	+	+	+	talp/rem	rem(b,b,b)	+	-	+	-	+	-
NAO/ack	ack(b,b,f)	+	+	-	-	-	-	talp/simple	p(b,f)	+	-	-	-	-	-
NAO/queens	queens(f)	+	+	+	+	+	E	talp/slsort-oi	sort(f,b)	-	-	-	-	-	E
NAO/reverse	reverse(b,f)	+	+	+	+	+	+	talp/slsort	sort(b,f)	+	+	+	+	+	E
NAO/sicstus	rev-conc(b,b,f)	+	+	+	+	+	+	talp/t-closure	tc(b,f)	+	+	-	+	+	E
								talp/vangeld	q(b,b)	+	-	-	-	-	E

Table 8. Termination Problem Database: TALP benchmarks

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). These workshops have raised a considerable interest in gaining a better understanding of each others approaches. It soon became clear that there has to be a close relationship between one of the most popular techniques in TRS, polynomial interpretations, and one of the key techniques in LP, acceptability with (semi-)linear norms and level mappings. However, partly because of the distinction between orderings over the natural numbers (LP) versus orderings over polynomials (TRS), 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. Although termination conditions in LP are formulated in terms of mappings to natural numbers, the actual termination proofs do not reason on natural numbers. They are formulated in terms of linear inequalities. In fact, LP termination analysis systems never work on the basis of the norm and the level mapping; they work on the level of the *abstract norm* and *abstract level mapping* (see [35]). As such, one outcome of the work is that, indeed, the polynomial interpretations of TRS are a direct generalization of the current LP practice.

On the more technical level, the contribution of this paper is that we provide a complete theoretical framework for polynomial interpretations in LP termination analysis and an approach to find such polynomial interpretations automatically. Part of this builds strongly on the results in [10] on order-acceptability, another part extends the results of Bossi et al. [6] on syntactic characterization of rigidity.

We have also developed an automated tool (Polytool) for termination proof of LP based on polynomial interpretations. It has required an intensive work in coding, especially the construction for the symbolic form of the polynomial constraints from the acceptability conditions w.r.t. polynomial interpretations and the transformation from the polynomial constraints to the Diophantine constraints.

Our main contribution is the integration of a number of techniques including the termination framework in [26], the call pattern inference tools in [20, 18, ?], the constraint-based approach in [12] and the Diophantine constraint solver in [14], to provide a completely automated termination analyser.

We have also done an intensive experimental evaluation of Polytool and other termination analysers such as Hasta La Vista, cTI, TALP and AProVE. It is shown from the evaluation that Polytool is powerful enough to solve a large number of terminating benchmarks. It can verify termination of a class of examples in which nonlinear norms are required.

7 Acknowledgements

Manh Thang Nguyen is partly supported by GOA Inductive Knowledge Bases and partly by FWO Termination Analysis: Crossing Paradigm Borders. We thank Gerda Janssens for making her type inference engine available, Frederic Mesnard and Roberto Bagnara for providing us the Cti system and the Parma Polyhedra Library.

References

1. The termination competition 2006, <http://www.lri.fr/~marche/termination-competition/2006/>, viewed september 2006.
2. The termination problems database 3.2, <http://www.lri.fr/~marche/tpdb/>, viewed november 2006.
3. K. R. Apt. Logic programming. In *Handbook of theoretical computer science*, volume B, pages 493–574. MIT Press, 1990.
4. T. Arts and H. Zantema. Termination of logic programs using semantic unification. In *Proceedings of International Symposium in Logic Program Synthesis and Transformation (LOPSTR'95)*, pages 219–233, 1995.
5. A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.
6. A. Bossi, N. Cocco, and M. Fabris. Proving termination of logic programs by exploiting term properties. In S. A. T. Maibaum, editor, *Proceedings TAPSOFT, volume 494 of Lecture Notes in Computer Science*, pages 153–180. Springer Verlag, 1991.
7. M. Codish, S. Genaim, H. Søndergaard, and P. J. Stuckey. Higher-precision groundness analysis. In *ICLP*, pages 135–149, 2001.
8. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
9. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 2005.
10. D. De Schreye and A. Serebrenik. Acceptability with general orderings. In *Computational Logic: Logic Programming and Beyond*, pages 187–210. 2002.
11. S. Decorte, D. De Schreye, and M. Fabris. Automatic inference of norms: a missing link in automatic termination analysis. In D. Miller, editor, *Proceedings of the International Symposium on Logic Programming*, pages 420–436, 1993.
12. 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.
13. N. Dershowitz. 33 examples of termination. In *Proceedings of French Spring School in Theoretical Computer Science*, volume 909 of *LNCS*, pages 16–26, 1995.
14. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Sat solving for termination analysis with polynomial interpretations. In *Tenth International Conference on Theory and Applications of Satisfiability Testing*, 2007.
15. H. Ganzinger and U. Waldmann. Termination proofs of well-moded logic programs via conditional rewrite systems. In *Proceedings of the Third International Workshop on Conditional Term Rewriting Systems, CTRS '92*, pages 430–437, London, UK, 1993.

16. J. Giesl. Generating polynomial orderings for termination proofs. In *6th International Conference on Rewriting Techniques and Applications*, pages 426–431, 1995.
17. J. Giesl, P. Schneider-Kamp, and R. Thiemann. Aprove 1.2: Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR '06)*, pages 281–286, 2006.
18. A. Heaton, M. Abo-Zaed, M. Codish, and A. King. A simple polynomial groundness analysis for logic programs. *J. Log. Program.*, 45(1-3):143–156, 2000.
19. H. Hong and D. Jakus. 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. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 342–376. 1970.
22. D. S. Lankford. On proving term rewriting systems are noetherian. Technical report, Mathematics Department, Louisiana Tech. University, Ruston, LA, 1979.
23. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, Berlin, 1987.
24. 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.
25. M. T. Nguyen and D. De Schreye. Polynomial interpretations as a basis for termination analysis of logic programs. Technical report, Department of Computer Science, K.U.Leuven, Belgium, 2005.
26. M. T. Nguyen and D. De Schreye. Polynomial interpretations as a basis for termination analysis of logic programs. In G. G. M. Gabbrielli, editor, *Proceedings of the 21st International Conference on Logic Programming (ICLP'05)*, volume 3668 of *LNCS*, pages 311–325. Springer Verlag, 2005.
27. M. T. Nguyen and D. De Schreye. Polytool: Proving termination automatically based on polynomial interpretations. Technical report, Department of Computer Science, K.U.Leuven, Belgium, 2006.
28. E. Ohlebusch, C. Claves, and C. Marché. Talp: A tool for the termination analysis of logic programs. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications*, volume 1833 of *LNCS*, pages 270–273. Springer Verlag, 2000.
29. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In *Proceedings of the 16th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'06)*, 2006.
30. A. Serebrenik. *Termination Analysis of Logic Programs*. PhD thesis, Department of Computer Science, K.U.Leuven, Belgium, 2003.
31. A. Serebrenik and D. De Schreye. Hasta-La-Vista: Termination analyser for logic programs. In F. Mesnard and A. Serebrenik, editors, *6th International Workshop on Termination (WLPE'03)*, pages 60–74, 2003.
32. J. Steinbach. Proving polynomials positive. In R. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'92)*, volume 652 of *LNCS*, pages 18–20, 1992.
33. J. Steinbach. On the complexity of simplification orderings. Technical Report SR-93-18 (SFB), SEKI University of Kaiserslautern, 1993.
34. C. Taboch, S. Genaim, and M. Codish. Terminweb: Semantic based termination analyser for logic programs, <http://www.cs.bgu.ac.il/~mcodish/terminweb>, 2002.

35. K. Verschaeetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'91)*, pages 301–315, 1991.
36. H. Zantema. *Termination*, In *Terese, Term Rewriting Systems*, chapter 6. Cambridge Univ. Press, 2003.