Polynomial Interpretations as a Basis for Termination Analysis of Logic Programs

Manh Thang Nguyen, Danny De Schreye
{ManhThang.Nguyen, Danny.DeSchreye}@cs.kuleuven.ac.be

Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001, Heverlee, Belgium

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 termination conditions.
Keywords: Termination analysis, acceptability, polynomial interpretations.

1 Introduction

In the last 20 years, the work on termination analysis has been most active for declarative programming languages, with an emphasis on two specific paradigms: logic programming (LP) and term rewriting systems (TRSs). In both areas, the work has been extensive and successful, with many powerful techniques developed and automated tools for these techniques available. However, termination analysis research has evolved very independently for these two paradigms. This has led to two collections of techniques and tools that co-exist without a reasonable level of cross-fertilization between them, nor an acceptable understanding of the portability of these techniques from one paradigm to the other.

Independent of the paradigm, almost every termination analysis is based on a mapping from computational states to some well-founded ordered set. A main difference between LP and TRS is the class of well-founded orderings that are being considered as a basis for the termination proof. For LP, computational states are usually mapped to a well-founded order on the natural numbers. This is usually done through "norms" and "level mappings", that respectively map terms and atoms to corresponding natural numbers (see [4]). In TRSs, a considerably wider range of well-ordered sets is being considered in the literature,

including polynomial interpretations, recursive and lexicographic path orders, and Knuth-Bendix orders (see [7, 18]).

On a most general, methodological level, cross-fertilization of techniques could be organized using two alternative routes: a transformational approach or a direct approach. A disadvantage of the transformational approach is that it somewhat obscures the intuitions regarding the termination argument. Often, one is not merely interested in finding a proof of termination as such, but it is more helpful if the proof - or the absence of it - helps us in better understanding the behaviour of the program. Another disadvantage is that this approach is only applicable to a class of transformable programs (i.e. well-moded logic programs).

In this paper we focus on a direct approach of porting techniques, in our case from TRS to LP. Within this context, an initial result to allow porting of more general orderings to the LP setting is presented in [5]. This work provides a new termination condition for definite logic programs based on general term orders. As such, it can be used as a framework in which different orderings considered in TRSs could be ported to LP directly and be evaluated. The current paper provides a first step in this study: the use of polynomial interpretations for LP termination analysis. Using polynomial interpretations as a basis for ordering terms in TRSs was first introduced by Lankford in [12]. It is currently one of the best known and most widely used techniques in TRS termination analysis.

In this paper, 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" [17]. The paper is organised as follows. In the next section, we present some preliminaries. In section 3, we introduce basic definitions of polynomial interpretations and show how this approach can be used to prove termination with some examples. In section 4, we discuss the automation of the approach. We end with a conclusion in section 5.

2 Preliminaries

2.1 Notations and Terminology

We assume familiarity with logic programming concepts and with the main results of logic programming [1, 13]. In the following, L denotes the language underlying a definite logic program P. We use Var_P, $Const_P$, Fun_P and $Pred_P$ to denote the set of variables, constant, function, and predicate symbols of L. 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, reflexive closure of the relation refer to. If p depends

on q and vice versa, p and q are called *mutually recursive*, denoted by $p \backsimeq q$. Let $Term_P$ and $Atom_P$ denote, respectively, the sets of all terms and atoms that can be constructed from L. The *extended Herbrand Universe* U_P^E, and the *extended Herbrand Base* B_P^E are the quotient sets of $Term_P$, and $Atom_P$ modulo the variant relation [9]. 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 [13]).

2.2 Norms and Level Mappings

Definition 1 (norm, level mapping). *A* norm *is a mapping* $\|.\| : U_P^E \rightarrow \mathbb{N}$. *A* level-mapping *is a mapping* $|.| : B_P^E \rightarrow \mathbb{N}$.

Several examples of norms can be found in literature [2]. 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.

Definition 2 (linear norm). *[15] A norm* $\|.\|$ *is a* linear norm *if it is recursively defined by means of the following schema:*

- $\|X\| = 0$ *for any variable* X,
- $\|f(t_1, ..., t_n)\| = f_0 + \sum_{i \in I_f} f_i \|t_i\|$ *where* $f_i \in \mathbb{N}$ *and the index set* $I_f \subseteq \{1, ..., n\}$ *depend only on the n-ary function symbol* f/n, $n \geq 0$.

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* $\preceq \succeq$ as $s \preceq \succeq t$ 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 $s_0 \succ s_1 \succ ...$ with $s_i \in S$. Let T be a set such that $S \subseteq T$. A quasi-ordering $\trianglerighteq$ defined on T is called a *proper extension of* $\succeq$ if

- $t_1 \succeq t_2$ implies $t_1 \trianglerighteq t_2$ 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 $\trianglerighteq$.

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.[11]).

Definition 4 (order-acceptability w.r.t. a set). *[5] 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 $A' \leftarrow B_1, ..., B_n$, such that $mgu(A,A') = \theta$ exists,*
- *for any atom B_i, such that $rel(B_i) \preceq rel(A)$,*
- *for any computed answer substitution σ for $\leftarrow (B_1, ..., B_{i-1})\theta$:*

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

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

Theorem 1. *[5] 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 U_P^E. An* interargument relation for *p is a relation $R_p = \{(t_1, ..., t_n) | t_i \in Term_P \wedge \varphi_p(t_1, ..., t_n)\}$, where:*

- *$\varphi_p(t_1, ..., t_n)$ is a formula in a disjunctive normal form,*
- *each conjunct in φ_p is either $s_i \succeq s_j$, $s_i \succ s_j$, $s_i \preceq \succeq s_j$ or $s_i \|_{\succeq} s_j$, where s_i, s_j are constructed from $t_1, ..., t_n$ by applying functors of P.*

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

The concept of rigidity is also generalized to general orderings.

Definition 6 (rigidity). *[5] A term or atom $A \in U_P^E \cup B_P^E$ is called* rigid w.r.t. a quasi-ordering $\succeq$ *if $\forall \sigma \in Subs$, $A \preceq \succeq A\sigma$. 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 1. 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 $\|.\|_l$, i.e. $t_1 \succeq t_2$ if and only if $\|t_1\|_l \geq \|t_2\|_l$, $t_1 \succ t_2$ if and only if $\|t_1\|_l > \|t_2\|_l$. For any substitution σ, $\|[X|t]\sigma\|_l = 1 + \|t\|_l = \|[X|t]\|_l$. Therefore, $[X|t]\sigma \preceq \succeq [X|t]$. However, this list is not rigid w.r.t. the quasi-ordering $\trianglerighteq$ imposed by the term-size norm $\|.\|_t$, i.e. $t_1 \trianglerighteq t_2$ if and only if $\|t_1\|_t \geq \|t_2\|_t$, $t_1 \triangleright t_2$ if and only if $\|t_1\|_t > \|t_2\|_t$. For instance, with $\sigma_1 = \{X/a_1\}$, a_1 is a constant, $\|[X|t]\sigma_1\|_t = 1 + \|t\|_t$, while with $\sigma_2 = \{X/[a_1, a_2]\}$ a_1, a_2 are constants, $\|[X|t]\sigma_2\|_t = 3 + \|t\|_t$. That implies $[X|t]\sigma_2 \triangleright [X|t]\sigma_1$. □

The following notion of rigid order-acceptability w.r.t. a set of atoms no longer forces us to reason on $Call(P,S)$. 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). *[5] Let S be a set of atomic queries and P be a program. Let $\succeq$ be a well-founded quasi-ordering on U_P^E and for each predicate p in P, let R_p be a valid interargument relation for p w.r.t. $\succeq$. P is* rigid order-acceptable w.r.t. *S if there exists a proper extension $\trianglerighteq$ of $\succeq$ on $U_P^E \cup B_P^E$, which is rigid on $Call(P,S)$ such that*

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

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

Theorem 2. *[5] 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 [5]).

3 Polynomial Interpretations in Logic Programming

Recall that our objective is to develop and discuss the basic definitions and properties of polynomial interpretations, and apply them to prove termination of a program. Here terms and atoms are mapped to polynomials, instead of natural numbers. This will allow to solve a class of problems that the traditional approach can not solve. To illustrate this point, consider the following program, *Der*, that formulates rules for computing the repeated derivative of a function in some variable u. This example was introduced in [5] (see also [8]).

Example 2 (Der).

$$d(der(u), 1).$$
$$d(der(A), 0) : -number(A).$$
$$d(der(X + Y), DX + DY) : -d(der(X), DX), d(der(Y), DY).$$
$$d(der(X * Y), X * DY + Y * DX) : -d(der(X), DX), d(der(Y), DY).$$
$$d(der(der(X)), DDX) : -d(der(X), DX), d(der(DX), DDX).$$

We are interested in proving termination of this program w.r.t. the query set S=$\{d(t_1, t_2) | t_1$ is a ground term, and t_2 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 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 linear norm $\|.\|$ and level mapping $|.|$ of general forms such that: $\|u\| = 0$, $\|t_1 + t_2\| = f_0^+ + f_1^+\|t_1\| + f_2^+\|t_2\|$, $\|t_1 * t_2\| = f_0^* + f_1^*\|t_1\| + f_2^*\|t_2\|$, $\|der(t)\| = f_0^d + f_1^d\|t\|$, $|d(t_1, t_2)| = d_0 + d_1\|t_1\| + d_2\|t_2\|$, $|number(t)| = n_0 + n_1\|t\|$ where

t, t_1, t_2 are terms and 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 [6] shows a contradiction: the system of inequalities that is set up from the acceptability condition is unsolvable. A complete proof can be found in [14]. Of course this only proves that one particular approach is unable to prove termination on the basis of linear mappings. □

3.1 Polynomial Interpretations

Let $\mathbb{N}$ be the set of all natural numbers and $A \subseteq \mathbb{N}$. We denote by $\mathbb{P}^A_{Var_P}$ the set of all polynomials in Var_P over A, with coefficients in $\mathbb{N}$. The following definition establishes an ordering on $\mathbb{P}^A_{Var_P}$.

Definition 8 (polynomial ordering). *Let P be a program and $A \subseteq \mathbb{N}$. Let $\mathbb{P}^A_{Var_P}$ be a set of all polynomials in Var_P over A. For polynomials $H, Q \in \mathbb{P}^A_{Var_P}$ let $X_1, ..., X_n$ be the variables occurring in H or Q. We define an ordering $\geq_A$ on $\mathbb{P}^A_{Var_P}$ as $H \geq_A Q$ if and only if $H - Q \geq 0$ for all instantiations $a_1, ..., a_n \in A$ of $X_1, ..., X_n$ respectively. A strict ordering $>_A$ associated with $\geq_A$ is defined as $H >_A Q$ if and only if $H - Q > 0$ for all $a_1, ..., a_n \in A$. If $H - Q = 0$ for all $a_1, ..., a_n \in A$, we write $H \leq \geq_A Q$. For any other cases, $H \|_{\geq_A} Q$.*

We usually require that A is an infinite set. Under this condition, $H \leq \geq_A Q$ if and only if the two polynomials are identical, denoted by $H \equiv Q$, i.e. all their corresponding coefficients are equal.

Example 3. Let H, Q be two polynomials in $Var_P = \{X_1, X_2, X_3\}$ over A such that: $H = 2X_1^2 + 3X_2X_3 + 5X_3$ and $Q = X_1^2 + 3X_2 + 2X_3 + 4$. We define a function $F(X_1, X_2, X_3) = H - Q = X_1^2 + 3X_2X_3 - 3X_2 + 3X_3 - 4$. Consider the following cases:

- $A = \mathbb{N} \backslash \{0\}$. For all $a_1, a_2, a_3 \in A$, $F(a_1, a_2, a_3) \geq 0$. Hence, $H \geq_A Q$.
- $A = \mathbb{N} \backslash \{0, 1\}$. For all $a_1, a_2, a_3 \in A$, $F(a_1, a_2, a_3) > 0$. Hence, $H >_A Q$.
- $A = \mathbb{N}$. For $a_1 = a_2 = a_3 = 0$, $F(a_1, a_2, a_3) = -4 < 0$. For $a_1 = a_2 = a_3 = 2$, $F(a_1, a_2, a_3) = 12 > 0$. Hence, $H \|_{\geq_A} Q$. □

Theorem 3. *Let $A \neq \emptyset$. The ordering $>_A$ on $\mathbb{P}^A_{Var_P}$ defined in definition 8 is a well-founded ordering.*

Proof. See [14].

Definition 9 (polynomial pre-interpretation).
A polynomial pre-interpretation J for a language of terms L *consists of:*

- *a set of natural numbers A, $A \subseteq \mathbb{N}$,*
- *an assignment that associates each n-ary function symbol f, $n \geq 0$, in L with a polynomial $P_f(X_{f_1}, ..., X_{f_m})$ from A^m to A, where the coefficients of the polynomial P_f/m are in $\mathbb{N}$ and the index set $I_f = \{f_1, ..., f_m\} \subseteq \{1, ..., n\}$ is determined by f/n.*

Note that each constant c in L can be considered an 0-ary function symbol and is assigned to an element c_I of A. Another issue is that the set A should be *closed under evaluating the polynomials*, i.e. for all $f{\in}Fun_P$ and $a_1,...,a_n{\in}A$, $P_f(a_1,...,a_n){\in}A$. This extra condition is required in the definition because of the fact that terms are recursively defined from their subterms. Thus, when selecting a polynomial pre-interpretation, we not only select an appropriate polynomial associated with each function symbol but also an appropriate set A such that the above closure property is guaranteed.

Definition 10 (polynomial norm). *The* polynomial norm associated with a polynomial pre-interpretation J *is a mapping* $\|.\|_J : Term_P{\rightarrow}\mathbb{P}^A_{Var_P}$ *which is defined recursively as:*

- $\|X\|_J = X$ *if* X *is a variable,*
- $\|f(t_1,...,t_n)\|_J = P_f(\|t_{f_1}\|_J,...,\|t_{f_m}\|_J)$,

where $P_f(X_1,...,X_m)$ *and* $I_f = \{f_1,...,f_m\}$ *are the same as in the definition of the polynomial pre-interpretation* J.

Similarly, we define the notion of a polynomial interpretation that sets up an abstract version of each predicate symbol.

Definition 11 (polynomial interpretation). *A* polynomial interpretation I for a language L underlying a program P *consists of a polynomial pre-interpretation* J *for the language of terms defined by* L *extended by*

- *an assignment to each predicate symbol* p/n, $n{\geq}0$, *in* L *of a polynomial* $P_p(X_{p_1},...,X_{p_m})$ *from* A^m *to* A, *where the coefficients of the polynomial* P_p/m *are in* $\mathbb{N}$ *and the index set* $I_p = \{p_1,...,p_m\}{\subseteq}\{1,...,n\}$ *is determined by* p/n.

Definition 12 (polynomial level-mapping). *The* polynomial level-mapping associated with a polynomial interpretation I *is a mapping* $|.|_I : Atom_P{\rightarrow}\mathbb{P}^A_{Var_P}$ *which is defined as:* $|p(t_1,...,t_n)|_I = P_p(\|t_{p_1}\|_J,...,\|t_{p_m}\|_J)$ *where* $P_p(X_1,...,X_m)$ *and* $I_p = \{p_1,...,p_m\}$ *are as in the definition of the polynomial interpretation* I.

For each term t and atom A, we denote by $P_t = \|t\|_J$ and $P_A = |A|_I$ as the polynomial interpretations of respectively t and A in I.

Example 4 (Dist). Consider the following distributive program *Dist*. This example was introduced in [5] (see also [18]):

$$dist(x,x).$$
$$dist(x*x, x*x).$$
$$dist(X+Y, U+V) : -dist(X,U), dist(Y,V). \quad (1)$$
$$dist(X*(Y+Z), T) : -dist(X*Y + X*Z, T). \quad (2)$$
$$dist((X+Y)*Z, T) : -dist(X*Z + Y*Z, T). \quad (3)$$

Let I be a polynomial interpretation that consists of a set $A \subseteq \mathbb{N}$, an assignment that associates the function symbol $*/2$ with the polynomial $P_* = X_1 * X_2$, the function symbol $+/2$ with the polynomial $P_+ = X_1 + X_2 + 1$, the constant x with a constant $c_x \in A$, and an assignment that associates the predicate symbol $dist/2$ with the polynomial $P_{dist} = X$, where the variable X corresponds to the first argument position of *dist/2*. The polynomial interpretation of the atom $A = dist(U*(X+Y), T)$ in I is: $P_A = |dist(U*(X+Y), T))|_I = \|U*(X+Y)\|_J = P_*(\|U\|_J, \|X+Y\|_J) = P_*(\|U\|_J, P_+(\|X\|_J, \|Y\|_J)) = \|U\|_J * (\|X\|_J + \|Y\|_J + 1) = U * (X + Y + 1)$. □

We define a quasi-ordering on $U_P^E \cup B_P^E$ imposed by the ordering $>_A$ on $\mathbb{P}^A_{Var_P}$ as follows:

Definition 13 (ordering on terms and atoms). *Let P be a program and I be a polynomial interpretation. We define $\succeq_I$ a quasi-ordering on U_P^E such that:*

- *$t \succ_I s$ if and only if $P_t >_A P_s$ for any $t, s \in U_P^E$,*
- *$t \preceq \succeq_I s$ if and only if $P_t \leq \geq_A P_s$ for any $t, s \in U_P^E$,*

and $\trianglerighteq_I$ a proper extension of $\succeq_I$ on $U_P^E \cup B_P^E$ such that:

- *$B \triangleright_I C$ if and only if $P_B >_A P_C$ for any $B, C \in B_P^E$,*
- *$B \trianglelefteq \trianglerighteq_I C$ if and only if $P_B \leq \geq_A P_C$ for any $B, C \in B_P^E$,*

where P_t, P_s, P_B, P_C are polynomial interpretations of t, s, B and C.

Theorem 4. *The strict orderings $\succ_I$ and $\triangleright_I$ are well-founded orderings on U_P^E and $U_P^E \cup B_P^E$ respectively.*

Integrated with definition 4 and theorem 1 we obtain:

Proposition 1. *Let P be a program and S be a set of atomic queries. If there exists a polynomial interpretation I such that*

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

$$P_A >_A P_{B_i\theta\sigma}$$

where P_A denotes the polynomial interpretation of the atom A,

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

Example 5. Reconsider example 4. We prove termination of the program with the following set of queries $S = \{dist(t_1, t_2) | t_1$ is a ground term and t_2 is a free variable$\}$. We choose the polynomial interpretation I of example 4 except that $A = \mathbb{N} \setminus \{0, 1\}$. Then, $\forall t \in Term_P$, $\|t\|_J >_A 1$. 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:

- $A = dist((t_1+t_2)*t_3, s)$ (t_1, t_2, t_3 are ground terms) and clause (3) is selected. There exists a substitution θ such that $\theta = mgu(A, dist((X_1+Y_1)*Z_1, T_1))$. That implies $X_1\theta = t_1, Y_1\theta = t_2, Z_1\theta = t_3$. Therefore, $|dist((t_1 + t_2) * t_3, s)|_I = \|(t_1 + t_2) * t_3\|_J = \|t_1 + t_2\|_J * \|t_3\|_J = \|t_1\|_J * \|t_3\|_J + \|t_2\|_J * \|t_3\|_J + \|t_3\|_J >_A \|t_1\|_J * \|t_3\|_J + \|t_2\|_J * \|t_3\|_J + 1 = \|t_1 * t_3 + t_2 * t_3\|_J = |dist(X_1 * Z_1 + Y_1 * Z_1, T_1)\theta|_I$.

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

Next, we study rigidity of a call set w.r.t. a polynomial interpretation and use it to verify rigid order acceptability.

3.2 Rigidity

First we present the classical notion of strictly monotone polynomials. This class of polynomials is discussed in [18]. Next we study the rigidity of a set of (terms) atoms w.r.t. a polynomial (pre-)interpretation that maps (terms) atoms to polynomials.

Definition 14 (strictly monotone polynomials). *Let $A \subseteq \mathbb{N}$. A polynomial $P(X_1, ..., X_n)$, $n > 0$, over A is called* strictly monotone *if and only if $t > s \Rightarrow P(a_1, ..., a_{i-1}, t, a_{i+1}..., a_n) > P(a_1, ..., a_{i-1}, s, a_{i+1}..., a_n)$ holds for all* i, $1 \leq i \leq n$, *and all $s, t, a_1, ..., a_{i-1}, a_{i+1}..., a_n \in A \backslash \{0\}$.*

Example 6. Reconsider example 3. Let $A = \mathbb{N}\backslash\{0\}$. Obviously, both H and Q are monotone polynomials. □

Definition 15 (monotone polynomial (pre-)interpretation). *A polynomial pre-interpretation is called* monotone *if it associates each function symbol f/n, $n > 0$ in Fun_P with a strictly monotone polynomial. A polynomial interpretation is* monotone *if it consists of a* monotone polynomial pre-interpretation *and an assignment that associates each predicate symbol p/n, $n > 0$, in $Pred_P$ with a strictly monotone polynomial.*

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 [2], Bossi, Cocco and Fabris discussed 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 [5]. 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 [2]. Let us recall and extend some basic notions defined in [2].

Definition 16 (rigidity w.r.t. a polynomial (pre-)interpretation). *A term $t \in U_P^E$ is called* rigid w.r.t. a polynomial pre-interpretation J *if and only if for any substitution θ, $\|t\|_J \leq \geq_A \|t\theta\|_J$. An atom $A \in B_P^E$ is called* rigid w.r.t. a polynomial interpretation I *if and only if for any substitution θ, $|A|_I \leq \geq_A |A\theta|_I$. In this case, J and I are 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. In particular, we are interested in polynomial interpretations that are rigid on a call set $Call(P, S)$ for some P and S.

Definition 17. *Let J be a polynomial pre-interpretation and t be a term. The i^{th} occurrence $X_{(i)}$ of a variable X in t is called* relevant w.r.t. J *if there exists a replacement $\{s \rightarrow X_{(i)}\}$ of a term s for $X_{(i)}$ such that $\|t\{s \rightarrow X_{(i)}\}\|_J \not\equiv \|t\|_J$. We call $VREL(t)$ the set of all relevant occurrences of variables in t.*

Obviously from definition 17, if a term t is not rigid w.r.t. J, there must be some relevant occurrence of some variable in t.

Example 7. Let $t = [X|X]$ and J be the polynomial pre-interpretation imposed by the list-length norm $\|.\|_l$, $P_{[H|T]} = 1 + P_T$. Then, $VREL(t) = \{X_{(2)}\}$. □

Proposition 2. *Let J be a polynomial pre-interpretation and t be a term. If $VREL(t) = \emptyset$, then t is rigid w.r.t. J. For the reverse direction, if J is monotone and t is rigid w.r.t. J, then $VREL(t) = \emptyset$.*

Proof. See [14].

The following proposition shows that monotone polynomial pre-interpretations characterize relevant subterms in a purely syntactic way.

Proposition 3. *For any polynomial pre-interpretation J, for any term t, the following property holds:*

(i) $VREL(t) = \{t\}$ *if t is a variable,*
(ii) $VREL(t) \subseteq \cup_{j=1,...,m} VREL(t_{fj})$, *if $t = f(t_1, ..., t_n)$ and $P_t = P_f(P_{t_{f_1}}, ..., P_{t_{f_m}})$ is the polynomial interpretation of t ($t_{fj}, 1 \leq j \leq m$, are the selected subterms of t under J),*
(iii) If J is monotone, then the inclusion in the conclusion of ii) becomes an equality.

Proof. The proof is similar to the proof in [2] except that it is extended to the case of polynomial pre-interpretations. □

The major advantage of monotone polynomial pre-interpretations is that we can check the rigidity of a term t w.r.t. a given monotone polynomial pre-interpretation in a syntactic way: namely to verifying emptiness of $VREL(t)$. In principle, another way of verifying that t is rigid under J is to compute P_t and check that it is variable-free. However, this is computationally more expensive.

3.3 Applying Rigid Order Acceptability to Polynomial Interpretations

First, we need the following notion of polynomial interargument relations.

Definition 18 (polynomial interargument relation). *Let P be a program, p/n be a predicate in P and I be a polynomial interpretation for the language L underlying P. A* polynomial interargument relation *for p is a relation $R_p = \{(t_1, ..., t_n) | t_i \in Term_P \wedge \varphi_p(P_{t_1}, ..., P_{t_n})\}$, where:*

- *$\varphi_p(P_{t_1}, ..., P_{t_n})$ is a formula in a disjunctive normal form,*
- *each conjunct in φ_p is either $P_{s_i} \geq_A P_{s_j}$, $P_{s_i} >_A P_{s_j}$, $P_{s_i} \leq \geq_A P_{s_j}$ or $P_{s_i} \|_{\geq_A} P_{s_j}$, where s_i, s_j are constructed from $t_1, ..., t_n$ by applying functors of P.*

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

Using the notions of rigidity and polynomial interargument relations w.r.t. a polynomial interpretation integrated with definition 7, theorem 2 and definition 13 we obtain:

Proposition 4. *Let S be a set of atomic queries, P be a program and I be a polynomial interpretation for the language L underlying P. For each predicate p in P, let R_p be a valid polynomial interargument relation for p w.r.t. I. If I is rigid on $Call(P, S)$ such that*

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

$$P_{H\theta} >_A P_{B_i\theta},$$

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

Example 8. Reconsider example 2. We are interested in proving termination of the program w.r.t. the query set $S = \{d(t_1, t_2) | t_1$ is a ground term and t_2 is a free variable$\}$. Observe that $Call(P, S)$ coincides with S.

Let I be a polynomial interpretation that consists of a set $A = N \backslash \{0, 1\}$, an assignment that associates the function symbol $der/1$ with the polynomial $P_{der} = X^2$, $+/2$ with $P_+ = X_1 + X_2$, $*/2$ with $P_* = X_1 * X_2$, the constant u with a constant $c_u \in A$ and an assignment that associates the predicate symbol $d/2$ with $P_d = X$, where the variable X corresponds to the first argument position of $d/2$. Let $R_d = \{(t_1, t_2) | t_1, t_2 \in Term_P$ and $P_{t_1} \geq_A P_{t_2}\}$ be a polynomial interargument relation w.r.t. the predicate $d/2$.

It is easy to verify that I is rigid on $Call(P, S)$ and R_d is valid w.r.t. I. Then, the program terminates if the following holds:

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

$$|d(der(X*Y), X*DY + Y*DX)\theta|_I >_A |d(der(Y), DY)\theta|_I$$
$$|d(der(der(X)), DDX)\theta|_I >_A |d(der(X), DX)\theta|_I$$
$$d(der(X), DX)\theta \text{ satisfies } R_d \text{ implies}$$
$$|d(der(der(X)), DDX)\theta|_I >_A |d(der(DX), DDX)\theta|_I$$

They are equivalent to the following inequalities on $X, Y, DX \in Var_p$:

$$(X+Y)^2 >_A X^2 \qquad X^2 >_A DX \Rightarrow (X*Y)^2 >_A Y^2$$
$$X^2 >_A DX \Rightarrow (X+Y)^2 >_A Y^2 \qquad X^4 >_A X^2$$
$$(X*Y)^2 >_A X^2 \qquad X^2 >_A DX \Rightarrow X^4 >_A DX^2$$

Since $A = \mathbb{N}\backslash\{0,1\}$, the above inequalities are easily verified and the program left-terminates. □

4 Automation: the general idea

For automation of the approach, two sources of ideas and techniques are important:

- the generalisation of the constraint-based approach to termination analysis of [6] from linear norms and level mappings to polynomials.
- the integration of a number of useful results and heuristics from TRSs ([3, 7, 10, 12, 16]).

The idea of the approach in [6] is to set up a symbolic form for all concepts involved in the termination conditions: in our case, the polynomial interpretation of each function and predicate symbol, the polynomial interargument relations and polynomial ordering conditions in proposition 4. Note that if we do not put a limit on the maximal degree of the polynomial, then there can be no finite general form of the polynomial associated with a term (there are infinitely many monomials $a_i X^{i_1} X^{i_2} ... X^{i_k}$ to consider). This is why we associate each function and predicate symbol with a *simple-mixed* polynomial, which is either a multivariate polynomial with all variables of at most degree 1 or a unary polynomial of at most degree 2.

From TRSs we borrow a sufficient condition for monotonicity of the polynomials:

Proposition 5. *(see also [18]) Let* $P = \sum_{i=1}^{r} a_i X_1^{k_{i,1}} X_2^{k_{i,2}} ... X_m^{k_{i,m}}$ *be a polynomial from* A^m *to* A *for which* $A = \mathbb{N}\backslash\{0\}, m > 0$ *and* $a_i \geq 0$ *for all* $i = 1, ..., r, r > 0$. *P is strictly monotone if* $\sum_{i=1}^{r} a_i k_{i,j} > 0$ *for every* $j = 1, ..., m$.

Example 9. Reconsider example 4. Let the first and the second argument positions of the predicate $dist/2$ be, respectively, the input and output positions. For all other function symbols, let all arguments be the input arguments. Let I be a polynomial interpretation such that the constant x is associated with $x_I \in A$, the function symbol $+/2$ is associated with the polynomial $P_+(X, Y) =$

$f_0^+ + f_1^+X + f_2^+Y + f_3^+XY$, the function symbol $*/2$ is associated with the polynomial $P_*(X,Y) = f_0^* + f_1^*X + f_2^*Y + f_3^*XY$, and the predicate symbol $dist/2$ is associated with the polynomial $P_{dist}(X) = f_0^d + f_1^dX + f_2^dX^2$. I is monotone if $f_1^d + f_2^d * 2 > 0$, $f_1^+ + f_3^+ > 0$, $f_2^+ + f_3^+ > 0$, $f_1^* + f_3^* > 0$, $f_2^* + f_3^* > 0$. □

For the interargument relations, we only allow the linear interargument relations of [6], i.e. $R_{p/n} = \{(t_1,...,t_n) | \sum_{i \in p_{inp}} p_i^e P_{t_i} \geq_A \sum_{j \in p_{out}} p_j^e P_{t_j} + p_0^e\}$, with $p_i^e \in \mathbb{N}$, $i \in \{1,...,n\}$, p_{inp} and p_{out} respectively the sets of input and output argument positions of p/n. But because these are applied to polynomial interpretations of terms, they still give rise to non-linear conditions in general.

Example 9 (continued). As an example, the condition for a valid interargument relation R_{dist} applied to clause 1 is of the following form:

$$(d_1^eX \geq_A d_2^eU + d_0^e) \wedge (d_1^eY \geq_A d_2^eV + d_0^e) \Rightarrow$$
$$d_1^e(f_0^+ + f_1^+X + f_2^+Y + f_3^+XY) \geq_A d_2^e(f_0^+ + f_1^+U + f_2^+V + f_3^+UV) + d_0^e. \quad \square$$

Next all other polynomial inequality conditions from proposition 4 are translated into constraints on the introduced symbols.

Example 9(continued). As an example, for recursive clause 1, the following constraints are imposed:

$$f_0^d + f_1^d(f_0^+ + f_1^+X + f_2^+Y + f_3^+XY)$$
$$+ f_2^d(f_0^+ + f_1^+X + f_2^+Y + f_3^+XY)^2 >_A f_0^d + f_1^dX + f_2^dX^2$$
$$d_1^eX \geq d_2^eU + d_0^e \Rightarrow f_0^d + f_1^d(f_0^+ + f_1^+X + f_2^+Y + f_3^+XY)$$
$$+ f_2^d(f_0^+ + f_1^+X + f_2^+Y + f_3^+XY)^2 >_A f_0^d + f_1^dY + f_2^dY^2 \quad \square$$

After normalisation, all the above constraints are transformed into the form: $P(X_1,...,X_n) \geq_A 0 \Rightarrow Q(X_1,...,X_m) \geq_A 0$ or the form $P(X_1,...,X_n) \geq_A 0$. In [6] techniques are proposed to transform the constraints of the first type into constraints of the second type. These can be extended to polynomials.

The following step is to transform all those constraints into constraints which contain only coefficients as variables. It can be done by applying one of the following approaches from TRS:

In the first approach of [10], the first step is to move from $A \subseteq \mathbb{N}$ to $\mathbb{R}^+$. Let a be $min\{c_I | c_I \in A$ is a polynomial interpretation of a constant c$\}$. Then instead of demanding that any of these constraints should hold (i.e. $P(X_1,...,X_n) \geq 0$ for all $X_1,...,X_n \in A$), it is sufficient to prove that $P(X_1,...,X_n) \geq 0$ for all $X_1,...,X_n \in A_R$, $A_R = \mathbb{R}^+ \backslash [0,a)$. The next step is to apply repeatedly the following differentiation rules to transform all polynomial constraints to constraints containing only coefficients as variables:

$$\frac{P(...,X_i,...) > 0}{P(...,a,...) > 0, \frac{\partial P(...,X_i,...)}{\partial X_i} \geq 0} \qquad \frac{P(...,X_i,...) \geq 0}{P(...,a,...) \geq 0, \frac{\partial P(...,X_i,...)}{\partial X_i} \geq 0}$$

Note the introduction of the inequations on the derivatives, which are actually extra constraints. Within TRS it has been argued that imposing these extra

constraints is most often reasonable as it allows to eliminate all variables X_i and because, if a solution to the original problem exists, the solution space is usually large enough to also contain an element that respects the extra constraints. There are a number of heuristics that can be applied to solve these constraints.

In the second approach of [3], all constraints are transformed to Diophantine inequalities. Then, if we put an arbitrary bound on the values of variable coefficients (e.g., $[0, B]$), the problem becomes solving a *finite domain constraint satisfaction problem* for a finite set of variables. Here finite-domain constraint solvers provide a variety of techniques to solve the remaining inequalities.

5 Conclusions

Since a few years ago, 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 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 [17]). 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. Part of this builds strongly on the results in [5] on order acceptability, another part extends the results of Bossi et al. [2] on syntactic characterization of rigidity.

In the paper we only provide two examples of the class of programs for which the extension from linear to polynomial interpretations is important. Note that typical examples in LP termination analysis are often deliberately chosen to be linear, to remain in the scope of the designed techniques. Non-linear polynomial functions are present in many real world problems and programs encoding these problems are bound to require polynomial interpretations for their termination proofs.

It remains to be studied how we can benefit from the huge amount of work that people in TRS termination analysis have spent on automating proofs with

polynomial interpretations and how integration of these techniques with the best approaches of LP termination analysis can lead to even more powerful techniques. We expect that this will lead to the development of a powerful new termination analyzer in the near future.

6 Acknowledgements

Manh Thang Nguyen is supported by GOA/2003/08. We thank the referees for useful comments.

References

1. K. R. Apt. Logic programming. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 493–574. MIT Press, 1990.
2. A. Bossi, N. Cocco, and M. Fabris. Proving termination of logic programs by exploiting term properties. In *TAPSOFT, Vol.2*, pages 153–180, 1991.
3. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *J. Auto. Reason.*, 2005.
4. D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *J. Log. Program.*, 19-20:199–260, 1994.
5. D. De Schreye and A. Serebrenik. Acceptability with general orderings. In *Computational Logic: Logic Programming and Beyond*, pages 187–210. Springer Verlag, 2002.
6. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint based automatic termination analysis of logic programs. *ACM Trans. Program. Lang. Syst*, 21(6):1137–1195, November 1999.
7. N. Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1-2):69–116, 1987.
8. N. Dershowitz. 33 examples of termination. *LNCS*, 909:16–26, 1995.
9. M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modeling of the operational behaviour of logic languages. *Theor. Comput. Sci.*, 63(3):289–318, 1989.
10. J. Giesl. Generating polynomial orderings for termination proofs. In *RTA*, pages 426–431, 1995.
11. G. Janssen and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *J. Log. Program.*, 13(2&3):205–258, 1992.
12. D. S. Lankford. On proving term rewriting systems are noetherian. Technical report, Mathematics Department, Louisiana Tech. University, Ruston, LA, 1979.
13. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, Berlin, 1987.
14. 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.
15. A. Serebrenik. *Termination Analysis of Logic Programs*. PhD thesis, Department of Computer Science, K.U.Leuven, Belgium, 2003.
16. J. Steinbach. Generating polynomial orderings. *Inf. Process. Lett.*, 49(2):85–93, 1994.
17. K. Verschaetse and D. De Schreye. Deriving termination proofs for logic programs, using abstract procedures. In *Proceedings 8th ICLP*, pages 301–315, 1991.
18. H. Zantema. *Termination, In Terese, Term Rewriting Systems*, chapter 6. Cambridge Univ. Press, 2003.