Polytool: Proving Termination Automatically Based on Polynomial Interpretations

Manh Thang Nguyen^{*} and Danny De Schreye

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

1 Introduction

In this system description, we present Polytool, a fully automated system for proving left-termination of definite logic programs (LPs). The aim of Polytool is to extend the power of existing termination analysers by using well-founded orders based on polynomial interpretations. This is a direct extension of the well-founded orders based on (semi-)linear level mappings and norms that are used in most of the existing LP termination analysis systems.

Polytool is based on a termination condition that is rooted on acceptability [2]. More precisely, the system implements the constraint-based approach to termination analysis, presented in [6], but extended to non-linear, polynomial level mappings and norms.

The theoretical foundations of Polytool are formulated and proved in [9]. Space restrictions do not allow us to give a formal account of this theory. We will only present the main intuitions in order to make the paper reasonably self-contained.

In the next section we describe the approach, comment on the various components and illustrate them with fragments of a termination proof. In Section 3, we report on extensive experimentation with the system and comparison with several other systems. We conclude in Section 4.

2 The Polytool System

The main novelty of Polytool is that it uses polynomial interpretations. Instead of ordering atoms and terms by means of associated natural numbers, obtained as function-values under (semi-)linear level mappings and norms, it maps atoms and terms to polynomials. The polynomials are considered as functions $P : \mathbb{N} \rightarrow \mathbb{N}$, and coefficients are also in $\mathbb{N}$. We use the natural well-founded order over such polynomials: $P \geq_{\mathbb{N}} Q$ iff $P(a_1, \dots, a_n) \geq Q(a_1, \dots, a_m)$, for all $a_1, \dots, a_{\max(n,m)} \in \mathbb{N}$.

Acceptability-style termination proofs require 3 types of conditions (for a formal presentation, see [9]):

^{*} Manh Thang Nguyen is partly supported by GOA Inductive Knowledge Bases and partly by FWO Termination Analysis: Crossing Paradigm Borders.

1. For clauses that have intermediate body-atoms between the head and the recursive body-atom, a valid relation between the interpretations of the arguments of the successful instances of these atoms needs to be inferred (valid interargument relations). These can then be used as pre-conditions to prove the decreases in point 2 (to deal with existentially quantified variables).
2. For every clause, the interpretation of the head should be larger than the interpretation of each (mutually) recursive body-atom, given the valid interargument relations of the intermediate body-atoms as pre-conditions.
3. For every non-ground query of interest, we must impose conditions on the interpretation, such that the interpretation of the query cannot grow unboundedly w.r.t. the polynomial order due to instantiations caused by resolutions steps. In Polytool we use rigidity [3] constraints to impose this. Rigidity means that the interpretation of an atom/term should be invariant for any instance of the atom/term.

In the philosophy of the constraint-based approach in [6], we do not *choose* a particular interpretation for the atoms or terms. We introduce a general symbolic form for such interpretations and interargument relations. As an example and assuming that polynomials of degree 2 are selected for the interpretation, instead of assigning an interpretation: $I(p(x, y)) = x^2 + 3xy$, we would assign $I(p(x, y)) = p_1 x^2 + p_2 xy + p_3 y^2$, where p_1, p_2 and p_3 are symbolic coefficients ranging over $\mathbb{N}$. The strategy of the analysis is to:

- introduce symbolic versions of the interpretations (the polynomials associated with each function and predicate symbol),
- introduce symbolic versions of the valid interargument relations,
- express all conditions resulting from steps 1, 2 and 3 above as constraints on the coefficients (e.g. $p_1, p_2, p_3, \dots$),
- 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.

In Polytool, we implement these ideas as follows. On the level of the polynomial interpretations, we 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 predicate symbols and simple-mixed polynomials to function symbols. For more details on these classes of polynomials we refer to [13].

Example 1 (permute).

$$\begin{array}{ll} perm([], []). & perm(L, [H|T]) : -del(H, L, L_1), perm(L_1, T). \\ del(H, [H|T], T). & del(X, [H|T], [H|T_1]) : -del(X, T, T_1). \end{array}$$

We use an interpretation in which:

$$\begin{array}{ll} I(perm(x, y)) = p_{10}x + p_{01}y + p_{00} & I(del(x, y, z)) = d_{100}x + d_{010}y + d_{001}z + d_{000} \\ I(.(x, y)) = ._{11}xy + ._{10}x + ._{01}y + ._{00} & P_{[]} = c_{[]} \end{array}$$

□

The first component of Polytool (see Fig. 1) does a combined mode and type analysis of the given program. For the given description of the set of (atomic) queries of interest - in terms of modes and types - we need to infer similar descriptions for all calls that may occur during a derivation. We use the rigid types of [8] to represent mode/type-information of the queries and use the type-inference system of [8] to collect the descriptions of all other calls.

Example 2 (permute-continued). Let the query set Q_1 be $\{perm(t_1, t_2) | t_1 \text{ is a nil-terminated list, } t_2 \text{ is a free variable}\}$. The call set corresponding to Q_1 , computed by the type-inference engine, is $S = Q_1 \cup Q_2$, where $Q_2 = \{del(t_1, t_2, t_3) | t_2 \text{ is a nil-terminated list, } t_1 \text{ and } t_3 \text{ are free variables}\}$. $\square$

In the next component of Polytool, we use the computed call patterns to derive the rigidity constraints on the interpretation. The rigidity constraint generator (Fig. 1) derives this set of Diophantine constraints.

Example 3 (permute-continued). Based on the polynomial interpretation in Example 1 and computed call set S in Example 2, the rigidity constraint generator derives the following Diophantine constraints with coefficients as variables:

$$\begin{array}{lll} p_{10} * (._{11} + ._{10}) = 0 & p_{01} = 0 & \\ d_{100} = 0 & d_{010} * (._{11} + ._{10}) = 0 & d_{001} = 0 \end{array}$$

$\square$

In the following component, the polynomial constraint generator translates the other termination conditions, consisting of the valid interargument relations and the head-body decreases, into polynomial constraints.

Example 4 (permute-continued). With the polynomial interpretation of Example 1, for the clause

$$del(X, [H|T], [H|T_1]) : -del(X, T, T_1)$$

the polynomial constraint generator produces the inequation:

$$\begin{aligned} \forall X, H, T, T_1 \in \mathbb{N} : & \quad (1) \\ d_{100}X + d_{010}(._{10}H + ._{01}T + ._{11}HT + ._{00}) + d_{001}(._{10}H + ._{01}T_1 + ._{11}HT_1 + ._{00}) + d_{000} \\ > d_{100}X + d_{010}T + d_{001}T_1 + d_{000} \end{aligned}$$

$\square$

In the next phase of the system, the generated polynomial constraints are transformed into Diophantine constraints. The point is to eliminate the variables (e.g. X, H, T, T_1 in the example) and to obtain constraints on the coefficients (e.g. $d_{100}, d_{010}, ._{10}, \dots$) only. This component was by far the hardest one to develop. In general, the generated polynomial constraints are considerably more complex than (1) in Example 4. They most often take the form of implications: a decrease between the polynomial interpretation of a head-atom versus a recursive body-atom holds *if* the valid interargument relations for the intermediate body-atoms hold. As far as we know, no complete solver for such systems of constraint

Fig. 1. Overall Structure

exists. A main effort in our work was to provide a set of transformation rules that generate a sufficient system of Diophantine constraints on the coefficients in the polynomial constraints. Here *sufficient* means that any solution of the Diophantine constraints is a solution to the given polynomial constraints (but not conversely). In this component of Polytool, we apply a number of techniques, including the substitution and evaluation rules of [6] but extended to polynomial interpretations, and the technique for testing positiveness of polynomials of [7].

Example 5 (permute-continued). We continue with Example 4 by deriving a set of Diophantine constraints from the constraint (1). Because there is no implication in this constraint, only the technique in [7] is applied. First we need to transform the constraint into the normal form:

$$(d_{100} - d_{100})X + (d_{010} + d_{001})_{\cdot 10}H + d_{010}(\cdot_{01} - 1)T + d_{010 \cdot 11}HT \\ + d_{001}(\cdot_{01} - 1)T_1 + d_{001 \cdot 11}HT_1 + (d_{001 \cdot 00} + d_{010 \cdot 00} - 1) \geq_{\mathbb{N}} 0$$

Applying the technique for testing positiveness of polynomials (a polynomial $P(x, y, \dots, z) \geq_{\mathbb{N}} 0$ iff all its coefficients are not negative), the above constraint holds iff the following Diophantine constraints hold:

$$\begin{array}{lll} d_{100} - d_{100} \geq 0 & (d_{010} + d_{001})_{\cdot 10} \geq 0 & \\ d_{010}(\cdot_{01} - 1) \geq 0 & d_{010 \cdot 11} \geq 0 & \\ d_{001}(\cdot_{01} - 1) \geq 0 & d_{001 \cdot 11} \geq 0 & d_{001 \cdot 00} + d_{010 \cdot 00} - 1 \geq 0 \end{array}$$

□

In the final component, the rigidity and the above generated Diophantine constraints become the input for the CiME 2.02, a Diophantine solver implemented by Contejean, Marché, Tomás and Urbain [4]. A solution for the constraint set implies the existence of a polynomial interpretation, valid interargument relations and a termination proof.

3 Experimental Evaluation

We have implemented a system (Polytool)¹ for automated termination proof based on the approach. It is integrated in the system implementing the constraint-based approach of [6] and consists of four parts. The first part is the type inference engine of Janssens and Bruynooghe [8], 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. The second part, the core of the system, which generates the set of all polynomial conditions, has been done in SICS 3.12.2. Also the third part, which normalises the polynomial conditions and transforms them to Diophantine constraints, is implemented in SICS 3.12.2. The final part is the Diophantine constraint solver (CiME 2.02) of Contejean, Marché, Tomás and Urbain [4]. This part is written in Objective CAML (CAML 3.0.9). We have tested the performance of the system on a number of examples, including benchmarks for LP in Termination Problems Database [1] (Tables 1(b), 2(a), 2(b), 3(a) and 3(b)), and examples collected from other sources (Table 1(a))². The domain of all variables in the generated Diophantine constraints is fixed to the set $D = \{0, 1, 2\}$. The experiments have been performed using SICS 3.12.2, running on Intel Pentium IV 2.80 MHz, 1Gb RAM. We have also performed an experimental evaluation on these examples with other systems, namely: Hasta La Vista [12], TALP [10], and TerminWeb [15]. We do not provide the running times of TALP and TerminWeb on the benchmarks because the tests have been done via the HTTP protocol and there is no information about the configuration of the servers on which these systems are installed. Only the success or failure of these systems w.r.t. the examples is provided. For TALP, polynomial interpretations are chosen. For TerminWeb, all provided semi-linear norms, i.e., node-size, edge-size and list-length norms, are selected. The other options for TALP and TerminWeb remain the same as in their web interfaces. In the tables, the following abbreviations are used:

- *Prog* and *Query* refer to the tested program and the query pattern. For the query pattern, '*g*' and '*f*' denote a ground term and a free variable respectively.
- T_1 , T_2 refer to the running time of Polytool and Hasta La Vista.
- R_1 , R_2 , R_3 and R_4 refer to the results given by the Polytool, Hasta La Vista, TALP and TerminWeb. It contains the symbol '+' if the system reports termination, or the symbol '-' if the system fails to do so.

From the tables, Polytool seems to be quite powerful since it can prove termination of 66 out of 83 terminating cases, in comparison with the performance of Hasta La Vista (53/83), TALP (59/83) and TerminWeb (50/83). For the running times, Polytool is slower than Hasta La Vista in a number of cases.

¹ For the source code, please refer to: <http://www.cs.kuleuven.be/~manh/polytool>

² In the table, examples *dist*, *der* were collected from [11], example *taussky* was introduced in [14]. The source of all examples in the tables can be found in http://www.cs.kuleuven.be/~manh/polytool/new_examples.zip

3.1 Comparison Between Hasta La Vista and Polytool

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 the theoretical point of view, for the benchmarks without meta-predicates or arithmetic expressions in them, termination analysis of Polytool is at least as precise as the analysis of Hasta La Vista. The claim could come from the fact that the approach based on polynomial interpretations used in Polytool can be considered as a generalization of the semi-linear norm based approach used in Hasta La Vista. The results in Table 1(a) 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, non-linear polynomial interpretations are required.

Observe that independently of whether we choose (semi-)linear norms and level mappings or polynomial interpretations, it still gives rise to nonlinear Diophantine constraints in the final step. Therefore, the requirement for a fast and effective nonlinear Diophantine constraint solver is necessary and CiME 2.02 seems to be a good selection. The only problem is, when the maximum degrees of variables or the domain of each variable in the constraints increases, the performance of the solver decreases considerably. A possible solution is to first apply the **INCLP(R)**³ implemented by De Koninck, Schrijvers and Demoen [5] to narrow the domain of each variable in the constraint set and then use CiME 2.02 to solve it over the narrowed domains in the following step.

Example 6. Consider the program *normal* with the query pattern *norm(g,f)* in Table 1(b):

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

Table 1. Results on Termination Benchmarks

(a) Variously collected examples								(b) TALP examples							
Prog	Query	T ₁	R ₁	T ₂	R ₂	R ₃	R ₄	Prog	Query	T ₁	R ₁	T ₂	R ₂	R ₃	R ₄
dist	dist(g,f)	0.47	+	0.26	-	+	-	ex1	p(f,g)	0.03	+	0.02	+	+	-
der	d(g,f)	20.02	+	0.25	-	-	-	ex4	p1(g)	0.02	+	0.01	+	+	+
boolexp	cequiv(g)	43.54	-	0.27	-	-	-	nat	isNat(g)	0.02	+	0.04	+	+	+
car_1	div(g,g,f)	0.18	-	0.06	-	-	-	nat	nEq(g,g)	0.03	+	0.03	+	+	+
car_13	in(g,g,f)	0.06	-	0.04	+	+	-	nat	gt(f,g)	0.02	+	0.04	+	+	+
fac_TRS	fac(g,f)	0.12	-	0.05	-	-	+	nat	odd(g)	0.03	+	0.03	+	+	+
fward_ins	f(f,f,f)	0.03	-	0.01	-	-	-	nat	fac(g,f)	0.06	+	0.07	+	+	+
SK90_1	p(g,f)	0.22	+	0.16	-	-	-	normal	norm(g,f)	0.15	+	0.09	-	+	-
SK90_2	p(g,f)	0.68	-	0.35	-	-	-	perm	perm(g,f)	0.25	+	0.13	+	+	+
SK90_3	sum(g,f)	0.13	+	0.44	+	+	-	permute	perm1(g,f)	0.08	+	0.08	+	+	+
SK90_4	p(g,f)	0.48	-	0.44	-	+	-	permute	perm2(f,g)	0.09	+	0.1	+	+	+
taussky	p(g,f)	0.16	+	0.07	-	+	-	qsort	qs(g,f)	2.37	+	0.14	+	+	-
addmul	p(g,f)	0.11	+	0.07	+	-	-	t_closure	tc(g,f)	0.03	-	0.02	-	+	-
fib	p(g,f)	0.76	-	0.22	-	+	-	simple	p(f,g)	0.02	+	0.03	+	-	-
lamdacal	g(g,g,f)	0.17	+	0.16	+	-	-	gcd	gcd(g,g,f)	0.03	-	0.03	-	-	+
log-1	log(g,f)	1.33	-	0.16	-	+	-	palind	palind(g)	0.05	+	0.05	+	+	+
average1	av(g,g,f)	0.31	+	0.04	-	+	-	slowsort	sort(g,f)	0.1	+	0.08	+	+	+
average2	av(g,f,g)	0.11	+	0.04	-	+	+	flat	flat(g,f)	0.73	+	0.08	-	+	-
flat	flat(g,f)	0.5	+	0.05	-	+	-	div	div(g,g,f)	0.09	+	0.09	+	+	+
queens	queens(f)	7.21	+	0.65	+	-	+	remind	rem(g,g,f)	11.35	-	0.08	-	-	+

³ Interval-based Nonlinear Constraint Logic Programming over the Reals.

Table 2. Examples from Apt and Plumer

(a) Apt								(b) Plumer							
Prog	Query	T ₁	R ₁	T ₂	R ₂	R ₃	R ₄	Prog	Query	T ₁	R ₁	T ₂	R ₂	R ₃	R ₄
list	list(g)	0.01	+	0.01	+	+	+	merge_t	merge(g,f)	0.36	+	0.42	+	-	-
fold	fold(f,g,f)	0.03	+	0.01	+	+	+	pl1.1	append(g,f,f)	0.03	+	0.02	+	-	-
lte	goal	0.04	+	0.04	+	+	+	pl1.1	append(f,f,g)	0.03	+	0.03	+	+	+
map	map(g,f)	0.03	+	0.01	+	+	+	pl2.3.1	p(g,f)	0.02	+	0.01	-	+	-
member	mem(f,g)	0.03	+	0.01	+	+	+	pl2.3.1	p(f,f)	0.01	-	0.01	-	-	-
merg	merg(g,f)	1.01	+	0.37	+	-	-	pl3.5.6a	p(f)	0.02	+	0.03	+	-	+
merg_ap	merg(g,f,g)	300	-	0.82	-	+	-	pl4.4.3	merge(g,g,f)	0.08	+	0.02	+	+	+
naiv_rev	reverse(g,f)	0.04	+	0.03	+	+	+	pl4.4.6a	perm(g,f)	0.04	+	0.03	+	+	+
ordered	ordered(g)	0.05	+	0.05	+	+	+	pl6.1.1	qsort(g,f)	1.12	+	0.19	+	+	-
overlap	o_lap(g,g)	0.03	+	0.03	+	+	+	pl7.2.9	mult(g,g,f)	0.03	+	0.03	+	+	+
perm	perm(g,f)	0.45	+	0.11	+	+	+	pl7.6.2c	reach(g,g,g,g)	4.87	-	0.22	-	+	+
qsort	qs(g,f)	0.07	+	0.13	+	+	+	pl8.2.1a	merge(g,f)	0.43	+	0.26	-	+	-
select	select(f,g,f)	0.01	+	0.02	+	+	+	pl8.3.1	minsort(g,f)	0.06	+	0.05	+	-	-
subset	subset(g,g)	0.06	+	0.05	+	+	+	pl8.3.1a	minsort(g,f)	0.05	+	0.07	+	-	+
sum	sum(f,f,g)	0.01	+	0.02	+	+	+	pl8.4.2	e(g,f)	0.1	+	0.09	+	+	+

Table 3. Examples from TerminWeb and Taboch

(a) TerminWeb								(b) Taboch							
Prog	Query	T ₁	R ₁	T ₂	R ₂	R ₃	R ₄	Prog	Query	T ₁	R ₁	T ₂	R ₂	R ₃	R ₄
som	som(g,g,f)	0.04	+	0.03	+	+	+	bad_list	sublist(g,g)	0.04	-	0.04	-	-	-
NJ1	rev(g,f)	0.04	+	0.05	+	+	+	quicksort	qs(g,f)	112.7	+	0.35	+	+	+
NJ2	f(g,g,f)	0.07	+	0.06	-	+	+	queens	queens(g,f)	0.25	+	0.5	+	-	+
NJ3	ack(g,g,f)	0.16	-	0.1	-	-	+	rotate	rotate(g,f)	0.05	+	0.05	+	+	+
NJ4	p(g,g,g,f)	0.05	+	0.03	+	+	+	sameleaves	s_leaves(g,g)	0.09	+	0.13	+	+	-
NJ5	f(g,g,f)	0.04	+	0.05	-	+	+	sublist	sublist(g,g)	0.05	+	0.06	+	+	+
NJ6	f(g,g,f)	0.1	+	0.07	+	+	+	sublist1	sublist(g,g)	0.04	+	0.03	+	+	+

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 C_iME 2.02, it also gives a positive result. This shows that the constraint solver used in Hasta La Vista, **CLPFD**⁴, 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 $\mathbb{N}$. 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 2(b):

$$p(X, Z) : -q(X, Y), p(Y, Z). \quad p(X, X). \quad q(a, b).$$

with the query pattern $Q = p(g, f)$ can be verified by Polytool, but not by Hasta La Vista.

Another issue is the efficiency. Overall, Hasta La Vista is faster than Polytool on a number of benchmarks. A reason could be that termination analysis based on polynomial interpretations increases the number of coefficients of the

⁴ Constraint Logic Programming over Finite Domain.

polynomials associated with predicates and functors which are variables in the generated Diophantine constraints. This leads to less efficiency of Polytool.

3.2 Comparison Between TALP and Polytool

A point of similarity between Polytool and TALP is that both systems use polynomial interpretations as a basis for the termination proof. However, it is applied indirectly in TALP: given a logic program and a query set, it first transforms them to a TRS. This transformation is termination preserving. Then, a polynomial interpretation technique is applied to the target TRS. A limitation of TALP is that only well-moded logic programs are considered [10]. 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, not solvable by TALP (e.g., *pl1.1* with query set *append(g,f,f)* in Table 2(b)).

4 Conclusions

We have presented the development of an automated tool for termination proof of LP based on polynomial interpretations. It is a further extension of the previous work in [9] as we aim at the implementation phase. 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 [9], the type inference engine in [8], the constraint-based approach in [6] and the Diophantine constraint solver in [4] 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, TerminWeb and TALP. It is shown from the evaluation that Polytool is powerful enough to solve a number of terminating benchmarks. It can verify termination of a class of examples in which nonlinear norms are required. In comparison with other tools, the result shows that Polytool has a higher success rate.

References

1. The termination problems database, <http://www.lri.fr/~marche/wst2004-competition/tpdb.html>, viewed march 2006
2. K. R. Apt and D. Pedreschi. Studies in pure prolog: Termination. In J. W. Lloyd, editor, *Proceedings Symposium in Computational Logic*, pages 150–176. Springer Verlag, Berlin, Heidelberg, 1990.
3. 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.

4. E. Contejean, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 2005.
5. L. De Koninck, T. Schrijvers, and B. Demoen. INCLP(R) - Interval-based non-linear constraint logic programming over the reals. In M. Fink, H. Tompits, and S. Woltran, editors, *Workshop on Logic Programming*, pages 91–100, 2006.
6. 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.
7. H. Hong and D. Jakus. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
8. 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.
9. 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.
10. 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.
11. A. Serebrenik. *Termination Analysis of Logic Programs*. PhD thesis, Department of Computer Science, K.U.Leuven, Belgium, 2003.
12. 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.
13. 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.
14. J. Steinbach. On the complexity of simplification orderings. Technical Report SR-93-18 (SFB), SEKI University of Kaiserslautern, 1993.
15. C. Taboch, S. Genaim, and M. Codish. Terminweb: Semantic based termination analyser for logic programs, <http://www.cs.bgu.ac.il/~mcodish/terminweb>, 2002