Non-transformational Termination Analysis of Logic Programs, Based on General Term-Orderings

Alexander Serebrenik, Danny De Schreye

Department of Computer Science, K.U. Leuven
Celestijnenlaan 200A, B-3001, Heverlee, Belgium
Email: {Alexander.Serebrenik, Danny.DeSchreye}@cs.kuleuven.ac.be

Abstract. We present a new approach to termination analysis of logic programs. The essence of the approach is that we make use of general term-orderings (instead of level mappings), like it is done in transformational approaches to logic program termination analysis, but that we apply these orderings directly to the logic program and not to the term-rewrite system obtained through some transformation. We define some variants of acceptability, based on general term-orderings, and show how they are equivalent to LD-termination. We develop a demand driven, constraint-based approach to verify these acceptability-variants.
The advantage of the approach over standard acceptability is that in some cases, where complex level mappings are needed, fairly simple term-orderings may be easily generated. The advantage over transformational approaches is that it avoids the transformation step all together.
Keywords: termination analysis, acceptability, term-orderings.

1 Introduction

There are many different approaches to termination analysis of logic programs. One particular distinction is between *transformational* approaches and *"direct"* ones. A transformational approach first transforms the logic program into an "equivalent" term-rewrite system (or, in some cases, into an equivalent functional program). Here, equivalence means that, at the very least, the termination of the term-rewrite system should imply the termination of the logic program, for some predefined collection of queries[1]. Direct approaches do not include such a transformation, but prove the termination directly on the basis of the logic program.

Besides the transformation step itself, there is one other technical difference between these approaches. Direct approaches usually prove termination on the basis of a well-founded ordering over the natural numbers. More specifically, they use a *level mapping*, which maps atoms to natural numbers, and, they verify

[1] The approach of Arts [4] is exceptional in the sense that the termination of the logic program is concluded from a weaker property of *single-redex normalisation* of the term-rewrite system.

appropriate decreases of this level mapping on the atoms occuring in the clauses. On the other hand, transformational approaches make use of more general well-founded orderings over terms, such as reduction orders, or more specifically a simplification order, or others (see [11]).

At least for the direct approaches the systematic choice for level mappings and norms, instead of general term orders, seems arbitrary and ad hoc. This has been the main motivation for this paper. We present an initial study on the use of general well-founded term-orderings as a means of directly proving the termination of logic programs—without intermediate transformation. In particular,

- we study whether the theoretical results on acceptability can be reformulated on the basis of general term orders,
- we evaluate to what extent the use of the general term orderings (instead of level mappings) either improves or deteriorates the direct approaches.

To illustrate the latter point, consider the following program, that formulates some of the rules for computing the repeated derivative of a linear function in one variable u (see also [13]) :

Example 1.

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

Proving termination of this program on the basis of a level-mapping is hard. For this example, the required level-mapping is a non-linear function. In particular, a level mapping, such that: $|d(X, Y)| = \|X\|$, $|number(X)| = 0$, $\|der(X)\| = 2^{\|X\|}$, $\|X + Y\| = max(\|X\|, \|Y\|) + 1$, $\|X * Y\| = max(\|X\|, \|Y\|) + 1$, $\|u\| = 2$, $\|n\| = 2$, if n is a number,would be needed. No automatic system for proving termination on the basis of level mappings is able to generate such mappings. Moreover, we believe, that it would be very difficult to extend existing systems to support generation of appropriate non-linear mappings. □

Although we have not yet presented our general-well-founded term ordering approach, it should be intuitively clear, that we can capture the decrease in order between the $der(X)$ and DX by using an ordering on terms that gives the highest "priority" to the functor *der*.

As an example of the fact that moving to general ordering can also introduce deterioration, consider the following program from [7, 10].

Example 2. This program defines a predicate *conf* that decreases a list provided as an argument, by two elements, and then adds a new element to it.

$$conf(X) \leftarrow delete_2(X, Z), delete(U, Y, Z), conf(Y).$$

$$delete_2(X, Y) \leftarrow delete(U, X, Z), delete(V, Z, Y).$$
$$delete(X, [X|T], T).$$
$$delete(X, [H|T], [H|T1]) \leftarrow delete(X, T, T1).$$

Note that by reasoning in terms of sizes of terms, we can infer that the size decreases by 2 after the call to $delete_2$ predicate in the first clause and then increases by 1 in the subsequent call to the *delete* predicate. In total, sizes allow to conclude a decrease. Reasoning in terms of order relations only, however, does not allow to conclude the overall decrease from the inequalities $arg3 < arg2$ for the *delete* predicate and $arg1 > arg2$ for the $delete_2$ predicate. □

As can be expected, theoretically both approaches are essentially equivalent, that is existence of a level-mapping or an order is equivalent to termination. We will introduce a variant of the notion of acceptability, based on general term orders, which is again equivalent to termination in a similar way as in the level mapping based approach. On the more practical level, as illustrated in the two examples above, neither of the approaches is strictly better: the general term orders provide a larger set of orders to select from (in particular, note that orders based on level mappings and norms are a term order), the level mapping approach provides arithmetic, on top of mere ordering.

In the remainder of this paper, we will start off from a variant of the notion of *acceptability with respect to a set*, as introduced in [8], obtained by replacing level mappings by term orderings. We show how this variant of acceptability remains equivalent to termination under the left-to-right selection rule, for certain goals. Then, we illustrate how this result can be used to prove termination with some examples. We also provide a variant of the *acceptability* condition, as introduced in [3], and discuss advantages and disadvantages of each approach. Next, we discuss automation of the approach. We elaborate on a demand-driven method to set-up and verify sufficient preconditions for termination. In this method, the aim is to derive—in, as much as possible, a constructive way—a well-founded ordering over the set of all atoms and terms of the language underlying the program, that satisfies the termination condition.

2 Preliminaries

2.1 Term ordering

An *quasi-order* over a set S is a reflexive, asymmetric and transitive relation $\geq$ defined on elements of S. We define the associated equivalence relation $=_>$ as $s =_> t$, if and only if $s \geq t$ and $t \geq s$, and the associated strict *partial ordering* $>$ if and only if $s \geq t$ but not $t \geq s$. If neither $s \geq t$, nor $t \geq s$ we write $s \|_> t$. Sometimes, in order to distinguish between different orders we also use $\succeq$, $\succ$, $=_\succ$ and $\|_\succ$

An ordered set S is said to be *well-founded* if there are no infinite descending sequences $s_1 > s_2 > \ldots$ of elements of S. If the set S is clear from the context we will say that the order, defined on it, is well-founded. We'll also say that

a quasi-order is well-founded if the strict partial order associated with it, is well-founded.

Definition 1. *Let $\geq$ be a quasi-order on a set T. A quasi-order $\succeq$ defined on a set $S \supseteq T$ is called a* proper extension of $\geq$ *if*

- $t_1 \geq t_2$ *implies* $t_1 \succeq t_2$ *for all* $t_1, t_2 \in T$.
- $t_1 > t_2$ *implies* $t_1 \succ t_2$ *for all* $t_1, t_2 \in T$.

The study of termination of term-rewriting systems caused intensive study of term orderings. A number of useful properties of term orderings were established.

Definition 2. *Let $>$ be an ordering on terms.*

- $>$ *is called* monotonic *if* $s_1 > s_2$ *implies* $f(\bar{t}_1, s_1, \bar{t}_2) > f(\bar{t}_1, s_2, \bar{t}_2)$ *and* $p(\bar{t}_1, s_1, \bar{t}_2) > p(\bar{t}_1, s_2, \bar{t}_2)$ *for any terms* s_1 *and* s_2*, sequences of terms* $\bar{t}_1$ *and* $\bar{t}_2$*, function symbol* f *and predicate* p.
- $>$ *is said to have the* subterm property *if* $f(\bar{t}_1, s, \bar{t}_2) > s$ *holds for any term* $f(\bar{t}_1, s, \bar{t}_2)$.

These properties can be analogously defined for quasi-orders. The following are examples of strict orderings: $>$ on the set of numbers, the lexicographic order on the set of strings (this is the way entries are ordered in dictionaries), the multiset ordering and the recursive path ordering [11]. The following are examples of quasi-orders: $\geq$ on the set of numbers, $\supseteq$ on the power set of some set.

For our purposes monotonicity and subterm properties are too restrictive. Thus, we assign to each predicate or functor a subset of the argument positions, such that for the argument positions in this subset the specified properties hold. We will say that a predicate p (a functor f) is monotone (has a subterm property) on a specified subset of argument positions. The formal study of these weaker notions may be found in [20].

Example 3. Let f be a functor of arity two, and a, b two terms, such that $a > b$. Let f be monotone in the first argument position. Then, $f(a, c) > f(b, c)$ holds for any term c, but there might be some term c, such that $f(c, a) \not> f(c, b)$.

One of the strict orderings that is useful for proving termination is the recursive path ordering. We define this ordering formally, following [11].

We start with defining *multisets* that are similar to sets, but allow multiple occurrences of identical elements. An ordering $>$, defined on S, can be extended to an ordering on finite multisets of S, $M(S)$. This ordering is denoted $\gg$ and formally is defined as following.

Definition 3. *For a partially-ordered set $(S, >)$, the multiset ordering $\gg$ on $M(S)$ is defined as follows: $M \gg M'$ if, and only if, for some multisets $X, Y \in M(S)$, where X is a nonempty subset of M,*

$$M' = (M - X) \cup Y$$

and for all $y \in Y$ there is an $x \in X$, such that $x > y$.

Example 4. If S is a set of integers, then $\{\!\{1,1,2,2,-3,\}\!\}$ and $\{\!\{1,2,2,-3,-3\}\!\}$ are two different multisets on S. If $>$ is a usual order on integers and $\gg$ is its extension to $M(S)$, then $\{\!\{1,1,2,2,-3,\}\!\} \gg \{\!\{1,2,2,-3,-3\}\!\}$.

Now we are going to use this notion to define a recursive path ordering. Recursive path ordering starts with a partial ordering on a set of operators and based on it, defines an order on terms.

Definition 4. *Let $\succ$ be a partial order on a set of operators F. The* recursive path ordering $>$ *on the set of terms over F is defined recursively as follows:*

$$s = f(s_1, \ldots, s_m) > g(t_1, \ldots, t_n) = t$$

if

- *$s_i \geq t$ for some $i = 1, \ldots, m$*
- *$f \succ g$ and $s > t_j$ for all $j = 1, \ldots, n$*
- *f is identical to g and $\{s_1, \ldots, s_m\} \gg \{t_1, \ldots, t_n\}$, where $\gg$ is the extension of $>$ to multisets, and $\geq$ means $>$ or equivalent up to permutation of subterms.*

2.2 Logic Programs

We follow the standard notation for terms and atoms. A *query* is a finite sequence of atoms. Given an atom A, $rel(A)$ denotes the predicate occuring in A. $Term_P$ and $Atom_P$ denote, respectively, sets of all terms and atoms that can be constructed from the language underlying P. The extended Herbrand Universe U_P^E (the extended Herbrand base B_P^E) is a quotient set of $Term_P$ ($Atom_P$) modulo the variant relation.

We refer to an SLD-tree constructed using the left-to-right selection rule of Prolog, as an LD-tree. We will say that a goal G *LD-terminates* for a program P, if the LD-tree for (P, G) is finite.

The following definition is borrowed from [1].

Definition 5. *Let P be a program and p, q be predicates occuring in it.*

- *We say that p* refers to *q in P if there is a clause in P that uses p in its head and q in its body.*
- *We say that p* depends on *q in P and write $p \sqsupseteq q$, if (p, q) is in the transitive, reflexive closure of the relation* refers to.
- *We say that p* and *q* are mutually recursive *and write $p \simeq q$, if $p \sqsupseteq q$ and $q \sqsupseteq p$. We also write $p \sqsupset q$ when $p \sqsupseteq q$ and $q \not\sqsupseteq p$.*

3 Term-acceptability with respect to a set

In this section we present and discuss some of the theory we developed to extend acceptability to general term orders. In the literature, there are different

variants of acceptability. The most well-known of these is the acceptability as introduced by Apt and Pedreschi [3]. This version is defined and verified on the level of ground instances of clauses, but draws its practical power mostly from the fact that termination is proved for *any bounded* goal. Here, boundedness is a notion related to the selected level mapping and requires that the set $\{|G\theta| \mid \theta \text{ is a grounding substitution for goal } G\}$ is bounded in the natural numbers, where $|\cdot| : B_P \rightarrow \mathcal{N}$ denotes the level mapping.

Another notion of acceptability is the "acceptability with respect to a set of goals", introduced by De Schreye et. al. in [8]. This notion allows to prove termination with respect to any set of goals. However, it relies on procedural concepts, such as calls and computed answer substitution. It was designed to be verified through global analysis, for instance through abstract interpretation.

A variant of acceptability w.r.t. a set that avoids the drawbacks of using procedural notions and that can be verified on a local level was designed in [10]. This variant required that the goals of interest are *rigid* under the given level mapping. Here, rigidity means that $|G\theta| = |G|$, for any substitution θ, where $|\cdot| : B_P^E \rightarrow \mathcal{N}$ now denotes a generalised level mapping, defined on the extended Herbrand base.

Comparing the notions of boundedness and rigidity in the context of a level mapping based approach, it is clear that boundedness is more general than rigidity. If the level mapping of a goal is invariant under substitution, then the level mapping is bounded on the set of instances of the goal, but not conversely.

Given the latter observation and given that acceptability of [3] is a more generally known and accepted notion, we started our work by generalising this variant.

However, generalising the concept of boundedness to general term orders proved to be very difficult. We postpone the discussion on this issue until after we formulated the results, but because of these complications, we only arrived at generalised acceptability conditions that are useful in the context of well-moded, simply moded programs and goals.

Because of this, we then turned our attention to acceptability with respect to a set. Here, the generalisation of rigidity was less complicated, so that in the end we obtained the strongest results for this variant of acceptability. Therefore, we first present term-acceptability with respect to a set of goals. We need the following notion.

Definition 6. *[9] Let P be a definite 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 a selected atom in some derivation for $P \cup \{\leftarrow Q\}$, for some $Q \in S$ and under the left-to-right selection rule.*

To illustrate this definition recall the following example [1, 10].

Example 5.

$$perm([],[]).$$
$$perm(L,[El|T]) \leftarrow del(El,L,L1), perm(L1,T).$$

$$del(X,[X|T],T).$$
$$del(X,[H|T],[H|T1]) \leftarrow del(X,T,T1).$$

Let S be $\{perm(t_1,t_2) \mid t_1$ is a nil-terminated list and t_2 is a free variable$\}$. Then, $Call(P,S) = S \cup \{\, del(t_1,t_2,t_3) \mid t_1$ and t_3 are free variables and t_2 is a nil-terminated list$\}$. Such information about S could for instance be expressed in terms of the rigid types of [16] and $Call(P,S)$ could be computed using the type inference of [16]. □

The following definition generalises the notion of acceptability w.r.t. a set [9] in two ways: 1) it generalises it to general term orders, 2) it generalises it to mutual recursion, using the standard notation of mutual recursion [1]—the original definition of acceptability required decrease only for calls to the predicate that appears in the head of the clause. This restriction limited the approach to programs with direct recursion only.

Definition 7. *Let S be a set of atomic queries and P a definite program. P is* term-acceptable w.r.t. *S if there exists a well-founded order $>$, such that*

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

$$A > B_i\theta\sigma$$

The following establishes the connection between term-acceptability w.r.t. a set S and LD-termination for queries in S.

Theorem 1. *Let P be a program. P is term-acceptable w.r.t. a set of atomic queries S if and only if P is LD-terminating for all queries in S.*

Proof. For all proofs we refer to [20].

This theorem is similar to Proposition 2.8 [10]. However, since the definition of term-acceptability deals correctly with mutual recursion, termination is established not only for directly recursive programs.

We postpone applying the Theorem 1 to Example 5 until a more syntactic way of verifying term-acceptability w.r.t. a set is developed.

To do this, we extend the sufficient condition of [8], that impose the additional requirement of rigidity of the level mapping on the call set, to the case of general term orders.

First we adapt the notion of rigidity to general term orders.

Definition 8. *(see also [6]) The term or atom $A \in U_P^E \cup B_P^E$ is called* rigid *w.r.t. a quasi-order $\geq$ if for any substitution θ, $A =_> A\theta$. In this case $\geq$ is said to be* rigid on *A.*

The notion of the rigidity on a term (an atom) is naturally extended to the notion of rigidity on a set of atoms (terms). In particular, we will be interested in orders that are rigid on $Call(P, S)$ for some P and S.

We also need interargument relations based on general term orders.

Definition 9. *Let P be a definite program, p/n a predicate in P. An* interargument relation *is a relation $R_p \subseteq \{p(t_1, \ldots, t_n) \mid \ t_i \in Term_P\}$. R_p is a* valid interargument relation for p/n *if and only if for every $p(t_1, \ldots, t_n) \in Atom_P$:* if $P \models p(t_1, \ldots, t_n)$ *then $p(t_1, \ldots, t_n) \in R_p$.*

Usually, the interargument relation will be defined based on a quasi-order used for proving termination. However, in general, this needs not be the case.

Example 6. Consider the following program.

$$p(0, []). \quad p(f(X), [X|T]) \leftarrow p(X, T)$$

The following interargument relations can be considered for p: $\{p(t_1, t_2) \mid t_2 > t_1 \vee t_1 =_> t_2\}$, valid if $>$ is a part of a quasi-order imposed by a list-length norm, $\| \cdot \|_l$. Recall, that for lists $\|[t_1|t_2]\|_l = 1 + \|t_2\|_l$, while the list-length of other terms is considered to be 0. On the other hand, $\{p(t_1, t_2) \mid t_1 > t_2 \vee t_1 =_> t_2\}$ is valid, if $>$ and $=_>$ are parts of a quasi-order imposed by a term-size norm.

Using general (non-norm based) quasi-orders, $\{p(t_1, t_2) \mid t_1 > t_2\}$ is valid, for example, for the recursive path ordering [11] with the following order on functors: $f/1 \succ ./2$ and $0 \succ []$. Alternatively, $\{p(t_1, t_2) \mid t_2 > t_1\}$ is valid, for example, for the recursive path ordering with the following order on functors: $./2 \succ f/1$ and $[] \succ 0$. □

Using the notion of rigidity a sufficient condition for term-acceptability w.r.t. a set is presented. We call this condition *rigid term-acceptability w.r.t. a set of queries*.

Theorem 2. *Let S be a set of atomic queries and P be a definite program. Let $\geq$ be a quasi-order on U_P^E and for each predicate p in P, let R_p be a valid interargument relation for p. If there exists a well-founded proper extension $\succeq$ of $\geq$ to $U_P^E \cup B_P^E$, which is rigid on $Call(P, S)$ such that*

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

$$H\theta \succ B_i\theta$$

then P is term-acceptable w.r.t. S

We continue the analysis of Example 5.

Example 7. Let $\succeq$ be a well-founded quasi-order on $U_P^E \cup B_P^E$, such that:

- for all terms t_1, t_{21} and t_{22}: $perm(t_1, t_{21}) =_\succ perm(t_1, t_{22})$.
- for all terms $t_{11}, t_{12}, t_2, t_{31}, t_{32}$: $del(t_{11}, t_2, t_{31}) =_\succ del(t_{12}, t_2, t_{32})$.
- for all terms t_{11}, t_{12} and t_2: $[t_{11}|t_2] =_\succ [t_{12}|t_2]$.

That is, we impose that the ordering is invariant on predicate argument positions and functor argument positions that may occur with a free variable in $Call(P, S)$. Furthermore, we impose that $\succeq$ has the subterm and monotonicity properties at all remaining predicate or functor argument positions.

First we investigate the rigidity of $\succeq$ on $Call(P, S)$, namely: $G\theta =_\succ G$ for any $G \in Call(P, S)$ and any θ. Now any effect that the application of θ to G may have on G needs to be through the occurrence of some variable in G. However, because we imposed that $\succeq$ is invariant on all predicate and functor argument positions that may possibly contain a variable in some call, $G\theta =_\succ G$.

Associate with del the interargument relation $R_{del} = \{del(t_1, t_2, t_3) \mid t_2 \succ t_3\}$. First, we verify that this interargument relationship is valid. Note, that an interargument relationship is valid whenever it is a model for its predicate. Thus, to check whether R_{del} is valid, $T_P(R_{del}) \subseteq R_{del}$ is checked. For the non-recursive clause of del the inclusion follows from the subset property of $\succeq$, while for the recursive one, from the monotonicity of it.

Then, consider the recursive clauses of the program.

- $perm$. If $del(El, L, L1)\theta$ satisfies R_{del}, then $L\theta \succ L1\theta$. By the monotonicity, $perm(L, T)\theta \succ perm(L1, T)\theta$ and, by the first condition on the quasi-order $\succeq$, $perm(L, [El|T])\theta =_\succ perm(L, T)\theta$. Thus, the decrease $perm(L, [El|T])\theta \succ perm(L1, T)\theta$ holds.
- del. By the properties of $\succ$: $del(X, [H|T], [H|T1]) \succ del(X, T, [H|T1])$ and $del(X, T, [H|T1]) =_\succ del(X, T, T1)$, i.e., $del(X, [H|T], [H|T1]) \succ del(X, T, T1)$.

We have shown that all the conditions of Theorem 2 are satisfied, and thus, P is term-acceptable w.r.t. S. By Theorem 1, P terminates for all queries in S.

We do not need to construct the actual order, but only to prove that there is one, that meets all the requirements. In our case, the requirement (subterm and monotonicity on the remaining argument positions) is satisfiable. □

4 The results for standard acceptability

In this section we briefly discuss some of the results we obtained in generalising the acceptability notions of [3, 14]. Since these results are weaker than those presented in the previous section, we do not elaborate on them in full detail. In particular, we do not recall the definitions of well-moded programs and goals, nor those of simply moded programs and goals, that we use below, but instead refer to [1], respectively [2]. Below, we assume that in-output modes for the program and goal are given. For any atom A and a mode m_A for A, we denote by A^{inp} the atom obtained from A by removing all output arguments. E.g., let $A = p(f(2), 3, X)$ and $m_A = p(in, in, out)$, then $A^{inp} = p(f(2), 3)$.

Definition 10. *Let $\geq$ be a quasi-order on B_P^E. We call* $\geq$ output-independent *if for any two moded atoms A and B: $A^{inp} = B^{inp}$ implies $A =_> B$.*

For well-moded programs, term-acceptability in the style of [3] can now be defined as follows.

Definition 11. *Let P be a well-moded program, $\geq$ an output-independent well-founded order and I a model for P. The program P is called* term-acceptable w.r.t. $\geq$ and I *if for all $A \leftarrow B_1, \ldots, B_n$ in P, all i, $1 \leq i \leq n$, and all substitutions θ, such that $(A\theta)^{inp}$ and $B_1\theta, \ldots, B_{i-1}\theta$ are ground and $I \models B_1\theta \wedge \ldots \wedge B_{i-1}\theta$ holds: $A\theta > B_i\theta$.*

P is called *term-acceptable* if it is term-acceptable with respect to some output-independent well-founded quasi-order and some model. Note the similarity and the difference with the notion of *well-acceptability* introduced by Etalle, Bossi and Cocco [14]—both notions relay on "ignoring" the output positions. However, the approach suggested in [14] measures atoms by level-mappings, while our approach is based on general term orders. In addition [14] requires a decrease only between atoms of mutually recursive predicates. Similarly, one might use the notion of term-acceptability that requires a decrease only between atoms of mutually recursive predicates. This definition will be equivalent to the one we used, since for atoms of non-mutually recursive predicates the dependency relation, $\sqsupset$, can always be used to define an order. Since every level mapping naturally gives rise to the order on atoms, that is $A_1 \succ A_2$ if $\mid A_1 \mid > \mid A_2 \mid$, we conclude that *every well-acceptable program is term-acceptable.*

The following theorem states that term-acceptability of a well-moded program is sufficient for LD-termination of all well-moded goals w.r.t. this program. Etalle, Bossi and Cocco [14] call such a program *well-terminating.*

Theorem 3. *Let P be a well-moded program, that is term-acceptable w.r.t. an output-independent well-founded quasi-order $\geq$ and a model I. Let G be a well-moded goal, then G LD-terminates.*

If the requirement of well-modedness is relaxed the theorem ceases to hold.

Example 8.

$$p(a) \leftarrow q(X) \qquad q(f(X)) \leftarrow q(X)$$

with the modes $p(in), q(in)$. This program is not well-moded w.r.t. the given modes, but satisfies the remaining conditions of term-acceptability with respect to the following quasi-order: $p(a) > q(t)$ and $q(f(t)) > q(t)$ for any term t and $t =_> s$ only if t and s are syntactically identical, and the following model $\{p(a), q(a), q(f(a)), \ldots\}$. However, well-moded goal $p(a)$ is non-terminating. □

Unfortunately, well-modedness is not sufficient to make the converse to hold. That is, there is a well-moded program P and a well-moded goal G, such that G is LD-terminating w.r.t. P, but P is not term-acceptable.

Example 9. Consider the following program:

$$p(f(X)) \leftarrow p(g(X))$$

with the mode $p(out)$. This program is well-moded, the well-moded goal $p(X)$ terminates w.r.t. this program, but it is not term-acceptable, since the required decrease $p(f(X)) > p(g(X))$ violates output-independence of $>$. □

Intuitively, the problem in the example occured, since some information has been passed via the output positions, i.e, P is not simply moded. Indeed, if P is simply-moded,[2], the second direction of the theorem holds as well. This was already observed in [14] in the context of well-acceptability and well-termination. The following is an immediate corollary to Theorem 5.1 in [14]. As that theorem states for well-moded simply moded programs well-termination implies well-acceptability. Therefore, well-terminating programs are term-acceptable.

Corollary 1. *Let P be a well-moded and simply moded program, LD-terminating for any well-moded goal. Then there exists a model I and an output-independent well-founded quasi-order $\geq$, such that P is term-acceptable w.r.t. I and $>$.*

To conclude, we briefly discuss why it is difficult to extend the notions of term-acceptability to the non well-moded case, using a notion of boundedness, as it was done for standard acceptability [3]. In acceptability based on level mappings, boundedness ensures that the level mapping of a (non-ground) goal can only increase up to some finite bound when the goal becomes more instantiated. Observe that every ground goal is trivially bounded.

One particular possible generalisation of boundedness to term-orderings, which is useful for maintaining most of our results, is:

An atom A is *bounded* with respect to an ordering $>$, if there exists an atom C such that for all ground instances $A\theta$ of A: $C > A\theta$, and $\{B \in B_P^E \mid C > B\}$ is finite.

Such a definition imposes constraints which are very similar to the ones imposed by standard boundedness in the context of level mappings. However, one thing we lose is that it is no longer generalisation of groundness. Consider an atom $p(a)$ and assume that our language contains a functor $f/1$ and a constant b. Then one particular well-founded ordering is

$$p(a) > \ldots > p(f(f(b))) > p(f(b)) > p(b).$$

So, $p(a)$ is not bounded with respect to this ordering.

Because of such complications, we felt that the rigidity-based results of the previous section are the preferred generalisations to general term orders.

5 Towards automation of the approach

In this section we present an approach leading towards automatic verification of the term-acceptability condition. The basic idea for the approach is inspired

on the "constraint based" termination analysis proposed in [10]. We start off from the conditions imposed by term-acceptability, and systematically reduce these conditions to more explicit constraints on the objects of our search: the quasi-order $\geq$ and the interargument relations, R_p, or model I.

The approach presented below has been applied successfully to a number of examples that appear in the literature on termination, such as different versions of *permute* [5, 17, 10], *dis-con* [7], *transitive closure* [17], *add-mult* [19], *combine*, *reverse*, *odd-even*, *at_least_double* and *normalisation* [10], *quicksort* program [21, 1], *derivative* [13], *distributive law* [12], *boolean ring* [15], *flatten* [4].

In the remainder of the paper, we explain the approach using some of these examples. We start by showing how the analysis of Example 5, presented before, can be performed systematically. We stress the main points of the methodology.

Example 10. $\geq$ should be rigid on $Call(P, S)$. To enforce the rigidity, $\geq$ should ignore all argument positions in atoms in $Call(P, S)$ that might be occupied by free variables, i.e., the second argument position of *perm* and the first and the third argument positions of *del*. Moreover, since for the elements of $Call(P, S)$ the first argument of *perm* and the second argument of *del* are general nil-terminated lists, the first argument of ./2 should be ignored as well.

The $>$-decreases imposed in the term-acceptability w.r.t. a set S are:

$$del(X, [H|T], [H|T1])\theta > del(X, T, T1)\theta$$
$$del(El, L, L_1)\theta \text{ satisfies } R_{del} \text{ implies } perm(L, [El|T])\theta > perm(L_1, T)\theta$$

Each of these conditions we simplify by replacing the predicate argument positions that should be ignored by some arbitrary term—one of $v_1, v_2, \ldots$. The following conditions are obtained:

$$del(v_1, [H|T]\theta, v_2) > del(v_3, T\theta, v_4) \quad (1)$$
$$del(El, L, L_1)\theta \text{ satisfies } R_{del} \text{ implies } perm(L\theta, v_1) > perm(L_1\theta, v_2) \quad (2)$$

Observe that this only partially deals with the requirements that the rigidity conditions expressed above impose: rigidity on functor arguments (the first argument of ./2 should be invariant w.r.t. the order) is not expressed. We keep track of such constraints implicitly, and only verify them at a later stage when additional constraints on the order are derived.

For each of the conditions we have two options on how to enforce it:

Option 1): The decrease required in the condition can be achieved by imposing some property on $\geq$, which is consistent with the constraints that were already imposed on $\geq$ before.

In our example, condition (1) is satisfied by imposing the subterm property for the second argument of ./2 and monotonicity on the second argument of *del*. The second argument of ./2 does not belong to a set of functor argument positions that should be ignored. Then, $[t_1|t_2] > t_2$ holds for any terms t_1 and t_2, and by the monotonicity of $>$ in the second argument of *del* (1) holds.

In general we can select from a bunch of term-order properties, or even specific term-orders, that were proposed in the literature.

Option 2): The required decrease is imposed as a constraint on the interargument relation(s) R of the preceding atoms.

In the *perm* example, the decrease $perm(L\theta, v_1) > perm(L_1\theta, v_2)$ cannot directly be achieved by imposing some constraint on $\geq$. Thus, we impose that the underlying decrease $L\theta > L_1\theta$ should hold for the intermediate body atoms $(del(El, L, L_1)\theta)$ that satisfy the interargument relation R_{del}.

Thus, in the example, the constraint is that R_{del} should be such that for all $del(t_1, t_2, t_3)$ that satisfy R_{del}: $t_2 > t_3$. Recall that the interargument relation is valid if it forms a model for its predicate. Thus, one way to verify that a valid interargument relation R_{del} as required exists, is to impose that $M = \{del(t_1, t_2, t_3) \mid t_2 > t_3\}$ itself is a model for the *del* clauses in the program, i.e., $T_P(M) \subseteq M$. As shown in [20], this reduces to "$[t_1|t_2] > t_2$" and "$t_2 > t_3$ implies $[t|t_2] > [t|t_3]$". These are fed into our Option 1) step, imposing a monotonicity property on the second argument of $./2$ for $>$, completing the proof. □

The previous example does not illustrate the approach in full generality. It might happen that more than one intermediate goal preceded the recursive atom in the body of the clause. In this case we refer to the whole conjunction as to "one" subgoal. Formally, given a sequence of intermediate body atoms $B_1, \ldots, B_n$ a (generalised) clause $B_1, \ldots, B_n \leftarrow B_1, \ldots, B_n$ is constructed and one step of unfolding is performed on each atom in its body, producing a generalised program P'.

Example 11. The following version of the *perm* program appeared in [17].

$$
\begin{array}{ll}
perm([],[]). & ap_1([],L,L). \\
perm(L,[H|T]) \leftarrow & ap_1([H|L1],L2,[H|L3]) \leftarrow \\
\quad ap_2(V,[H|U],L), & \quad ap_1(L1,L2,L3). \\
\quad ap_1(V,U,W), & ap_2([],L,L). \\
\quad perm(W,T). & ap_2([H|L1],L2,[H|L3]) \leftarrow \\
 & \quad ap_2(L1,L2,L3).
\end{array}
$$

This example is analysed, based on Theorem 3 for the well-moded case. We would like to prove termination of the goals $perm(t_1, t_2)$, where t_1 is a ground list and t_2 a free variable. Assume the modes $perm(in, out)$, $ap_1(in, in, out)$, $ap_2(out, out, in)$.

The term-acceptability imposes, among the others, the following $>$-decrease: $I \models ap_2(V, [H|U], L)\theta \wedge ap_1(V, U, W)\theta$ implies $perm(L)\theta > perm(W)\theta$. The underlying decrease $L\theta > W\theta$ cannot be achieved by reasoning on $ap_1/3$ or $ap_2/3$ alone. Therefore, we construct a following generalised program P':

$$
\begin{array}{l}
ap_2([],[t_1|t_2],[t_1|t_2]), ap_1([],t_2,t_2). \\
ap_2([t_6|t_1],[t_5|t_2],[t_6|t_3]), ap_1([t_6|t_1],t_2,[t_6|t_4]) \leftarrow \\
\qquad ap_2(t_1,[t_5|t_2],t_3), ap_1(t_1,t_2,t_4).
\end{array}
$$

Now, we need to very that $M = \{ap_2(a_1, a_2, a_3), ap_1(b_1, b_2, b_3) \mid a_3 > b_3\}$ satisfies $T_{P'}(M) \subseteq M$. Using the 2 clauses, this is reduced to "$[t_1|t_2] > t_2$" and "$t_3 > t_4$

implies $[t_6|t_3] > [t_6|t_4]$", imposing monotonicity and subterm properties on $>$. The proof is completed analogously to the previous example. □

As a last example, we return to the motivating Example 1, on computing higher derivatives of polynomial functions in one variable.

Example 12. We are interested in proving termination of the queries that belong to $S = \{d(t_1, t_2) \mid t_1$ is a repeated derivative of a function in a variable u and t_2 is a free variable$\}$. So S consists of atoms of the form $d(der(u), X)$ or $d(der(u * u + u), Y)$ or $d(der(der(u + u)), Z)$, etc. Observe, that $Call(P, S) = S$.

We start by analysing the requirements that imposes the rigidity of $\geq$ on $Call(P, S)$. First, the second argument position of d should be ignored, since it might be occupied by a free variable. Second, the first argument position of d is occupied by a ground term. Thus, rigidity does not pose any restrictions on functors argument positions.

Then, we construct the $>$-decreases implied by the rigid term-acceptability. The arguments that should be ignored are replaced by anonymous terms $v_1, v_2, \ldots$ For the sake of brevity we omit most of these $>$-decreases, since the only requirement they pose is monotonicity and subterm properties on the first argument of d. However, in order to satisfy the following $>$-decrease:

$$d(der(X), DX)\theta \text{ satisfies } R_d \text{ implies } d(der(der(X))\theta, v_1) > d(der(DX)\theta, v_2)$$

more complicated analysis should be performed. It is sufficient to prove that for any $(t_1, t_2) \in R_d$ holds that $t_1 > t_2$. That is if $M = \{d(t_1, t_2) \mid t_1 > t_2\}$ then $T_P(M) \subseteq M$. This may be reduced to the following conditions:

$$der(u) > 1 \tag{3}$$
$$t_1 \in R_{number} \text{ implies } der(t_1) > 0 \tag{4}$$
$$der(t_1) > t_2 \ \& \ der(t_3) > t_4 \text{ implies } der(t_1 + t_3) > t_2 + t_4 \tag{5}$$
$$der(t_1) > t_2 \ \& \ der(t_3) > t_4 \text{ implies } der(t_1 * t_3) > t_1 * t_4 + t_2 * t_3 \tag{6}$$
$$der(t_1) > t_2 \ \& \ der(t_2) > t_3 \text{ implies } der(der(t_1)) > t_3 \tag{7}$$

Condition (7) follows from monotonicity and transitivity of $>$. However, (4)-(6) are not satisfied by general properties of $>$ and we choose, to specify the order. The order that meets these conditions is the recursive path ordering [11] with der having the highest priority, i.e. $der \succ +$ and $der \succ *$. □

This example demonstrates the main points of our methodology. First, given a program P and a set S of goals, *compute the set of calls* $Call(P, S)$ (for instance through the abstract interpretation of [16]). Second, *enforce the rigidity of* $\geq$ *on* $Call(P, S)$, i.e., ignore all predicate or functor argument positions that might be occupied by free variables in $Call(P, S)$. Third, repeatedly *construct* $>$*-decreases*, such that rigid term-acceptability condition will hold and check if those can be verified by some of the predefined orders.

6 Conclusion

We have presented a non-transformational approach to termination analysis of logic programs, based on general term-orderings. The problem of termination was studied by a number of authors (see [7] for the survey). More recent work on this topic can be found in [18, 9, 14, 10].

Our approach gets it power from integrating the traditional notion of acceptability [3] with the wide class of term-orderings that have been studied in the context of the term-rewriting systems. In theory, such an integration is unnecessary: acceptability (based on level mappings only) is already equivalent to LD-termination. In practice, the required level mappings may sometimes be very complex (such as for Example 1 or *distributive law* [12], *boolean ring* [15] or *flattening of a binary tree* [4]), and automatic systems for proving termination are unable to generate them. In such cases, generating an appropriate term-ordering, replacing the level mapping, may often be much easier, especially since we can reuse the impressive machinery on term-orders developed for term-rewrite systems. In some other cases, such as *turn* [6], simple level mappings do exist (in the case of *turn*: a norm counting the number of 0s before the first occurrence of 1 in the list is sufficient), but most systems based on level mappings will not even find this level mapping, because they only consider mappings based on term-size or list-length norms. Again, our approach is able to deal with such cases.

Additional extensions of acceptability [3], such as semi-acceptability [1], well-acceptability [14] appeared in the literature. For the well-moded programs term-acceptability implies acceptability (or, equivalently, semi-acceptability). As illustrated by Example 9, not every acceptable program is term-acceptable. Similarly, well-acceptability implies term-acceptability. Moreover, for well-moded simply moded programs well-acceptability is equivalent to term-acceptability. In [20] we have proved that, under certain conditions, term-acceptability w.r.t. a set implies term-acceptability (w.r.t. the least Herbrand model).

Unlike transformational approaches, that establish the termination results for logic programs by the reasoning on termination of term-rewriting systems, we apply the term-orderings directly to the logic programs, thus, avoiding transformations. This could both be regarded as an advantage and as a drawback of our approach. It may be considered as a drawback, because reasoning on successful instances of intermediate body-atoms introduces an additional complication in our approach, for which there is no counterpart in transformational methods (except for in the transformation step itself). On the other hand, we consider it as an advantage, because it is precisely this reasoning on intermediate body atoms that gives more insight in the property of *logic program termination* (as opposed to *term-rewrite system termination*).

So, in a sense our approach provides the best of both worlds: a means to incorporate into 'direct' approaches the generality of general term-orderings.

We consider as a future work a full implementation of the approach. Although we already tested very many examples manually, an implementation will allow us to conduct a much more extensive experimentation, comparing the technique also in terms of efficiency with other systems. Since we apply a demand-driven

approach, systematically reducing required conditions to more simple constraints on the ordering and the model, we expect that the method can lead to very efficient verification.

7 Acknowledgement

Alexander Serebrenik is supported by GOA: "LP^+: a second generation logic programming language". We thank anonymous referees for very useful comments.

References

1. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall Int. Series in Computer Science. Prentice Hall, 1997.
2. K. R. Apt and S. Etalle. On the unification free Prolog programs. In A. M. Borzyszkowski and S. Sokolowski, editors, *18th Int. Symp. on Mathematical Foundations of Computer Science*, pages 1–19. Springer Verlag, 1993. LNCS 711.
3. K. R. Apt and D. Pedreschi. Studies in Pure Prolog: Termination. In J. W. Lloyd, editor, *Proc. Esprit Symp. on Comp. Logic*, pages 150–176. Springer Verlag, 1990.
4. T. Arts. *Automatically proving termination and innermost normalisation of term rewriting systems.* PhD thesis, Universiteit Utrecht, 1997.
5. T. Arts and H. Zantema. Termination of logic programs using semantic unification. In M. Proietti, editor, *5th Int. Workshop on Logic Programming Synthesis and Transformation*, pages 219–233. Springer Verlag, 1995. LNCS 1048.
6. A. Bossi, N. Cocco, and M. Fabris. Norms on terms and their use in proving universal termination of a logic program. *Theoretical Computer Science*, 124(2):297–328, February 1994.
7. D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *J. Logic Programming*, 19/20:199–260, May/July 1994.
8. D. De Schreye, K. Verschaetse, and M. Bruynooghe. A framework for analyzing the termination of definite logic programs with respect to call patterns. In I. Staff, editor, *Proc. of the Int. Conf. on Fifth Generation Computer Systems.*, pages 481–488. IOS Press, 1992.
9. S. Decorte and D. De Schreye. Termination analysis: some practical properties of the norm and level mapping space. In J. Jaffar, editor, *Proc. of the 1998 Joint Int. Conf. and Symp. on Logic Programming*, pages 235–249. MIT Press, June 1998.
10. S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based termination analysis of logic programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(6):1137–1195, November 1999.
11. N. Dershowitz. Termination. In C. Kirchner, editor, *First Int. Conf. on Rewriting Techniques and Applications*, pages 180–224. Springer Verlag, 1985. LNCS 202.
12. N. Dershowitz and C. Hoot. Topics in termination. In C. Kirchner, editor, *Rewriting Techniques and Applications, 5th Int. Conf.*, pages 198–212. Springer Verlag, 1993. LNCS 690.
13. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM (CACM)*, 22(8):465–476, August 1979.
14. S. Etalle, A. Bossi, and N. Cocco. Termination of well-moded programs. *J. Logic Programming*, 38(2):243–257, February 1999.

15. J. Hsiang. Rewrite method for theorem proving in first order theory with equality. *Journal of Symbolic Computation*, 8:133–151, 1987.
16. G. Janssens and M. Bruynooghe. Deriving descriptions of possible values of program variables by means of abstract interpretation. *J. Logic Programming*, 13(2&3):205–258, July 1992.
17. M. Krishna Rao, D. Kapur, and R. Shyamasundar. Transformational methodology for proving termination of logic programs. *J. Logic Programming*, 34:1–41, 1998.
18. N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In L. Naish, editor, *Proc. of the Fourteenth Int. Conf. on Logic Programming*, pages 63–77. MIT Press, July 1997.
19. L. Plümer. *Termination Proofs for Logic Programs.* LNAI 446. Springer Verlag, 1990.
20. A. Serebrenik and D. De Schreye. Termination analysis of logic programs using acceptability with general term orders. Technical Report CW 291, Departement Computerwetenschappen, K.U.Leuven, Leuven, Belgium, 2000. Available at http://www.cs.kuleuven.ac.be/publicaties/rapporten/CW2000.html.
21. L. Sterling and E. Shapiro. *The Art of Prolog.* The MIT Press, 1994.