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

Extended abstract

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 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 simplification order, or others (see [10]). Although transformational approaches have been quite successful (see [15] for an overview), they have one disadvantage: Due to the transformation, they provide little understanding of the termination proof in terms of the original program. This is not a problem if the goal is only to achieve (automatic) verification. However, in a context

* Extended Abstracts of LOPSTR 2000, Tenth International Workshop on Logic-based Program Synthesis and Transformation, 24-28 July 2000, London, UK. Technical Report Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-00-6-1. http://www.cs.man.ac.uk/cstechrep/titles00.html

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

of understanding and reasoning about the termination behaviour of the logic program, the direct approach may be better suited.

In this paper, we present an initial study on how to combine the two approaches. We investigate the use of well-founded term-orderings as a means of directly proving the termination of logic programs—without intermediate transformation. Our motivation for doing this is both:

- to improve our understanding of the relation between these approaches, and
- to 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 t (see also [12]) :

$$
\begin{array}{l}
d(der(t), 1).\\
d(der(A), 0) \leftarrow number(A).\\
d(der(X+Y), DX+DY) \leftarrow d(der(X), DX), d(der(Y), DY).\\
d(der(X*Y), X*DY+Y*DX) \leftarrow d(der(X), DX), d(der(Y), DY).\\
d(der(der(X)), DDX) \leftarrow d(der(X), DX), d(der(DX), DDX).
\end{array}
$$

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 $|d(X,Y)| = \|X\|$, $|number(X)| = 0$, $\|der(X)\| = 2^{\|X\|}$, $\|X+Y\| = max(\|X\|, \|Y\|) + 1$, $\|X*Y\| = max(\|X\|, \|Y\|) + 1$, $\|t\| = 2$, $\|n\| = 2$, (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*.

In the remainder of this paper, we begin with presenting a variant of the notion of *acceptability* and showing that it 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 and discuss automation of the approach

2 Preliminaries

2.1 Term ordering

An *order* over a set S is an irreflexive, asymmetric and transitive relation $>$ defined on elements of S. As usual, $s \geq t$ denotes that either $s > t$ or $s = t$. A 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.

Definition 1. *Let $>$ be an ordering on terms.*

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

The following are examples of order relations: $>$ on the set of numbers, lexicographic order on the set of strings (this is a way the entries are ordered in dictionaries), multiset ordering and recursive path ordering [10].

2.2 Logic Programs

We follow the standard notation for terms and atoms. $Term_P$ and $Atom_P$ denote, respectively, sets of all terms and atoms that can be constructed from the language underlying a program 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.

For a predicate p/n a *mode* is an atom $p(m_1, \ldots, m_n)$, where $m_i \in \{i, o\}$ for $1 \leq i \leq n$. Positions with i are called *input positions*, and positions with o are called *output positions* of p. We assume that a fixed mode is associated with each predicate in a program. To simplify the notation, an atom written as $p(\mathbf{s}, \mathbf{t})$ means: $\mathbf{s}$ is the vector of terms filling the input positions, and $\mathbf{t}$ is the vector of terms filling the output positions.

A *query* is a finite sequence of atoms. $Var(A)$ denotes a set of the variables appearing in A. Superscripts are used to denote the restriction of the atom only to its input (output) positions, i.e., $p(\mathbf{s}, \mathbf{t})^{inp}$ denotes $p(\mathbf{s})$.

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.

Definition 2. *[1] A query $p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called* well moded *if for all $i \in [1, n]$ holds $Var(\mathbf{s_i}) \subseteq \bigcup_{j=1}^{i-1} Var(\mathbf{t_j})$. A program clause $p_0(\mathbf{t_0}, \mathbf{s_{n+1}}) \leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called* well moded *if for all $i \in [1, n+1]$ $Var(\mathbf{s_i}) \subseteq \bigcup_{j=0}^{i-1} Var(\mathbf{t_j})$. A program is called* well moded *if every clause of it is.*

Definition 3. *[2] A query $p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called* simply moded *if $\mathbf{t_1}, \ldots, \mathbf{t_n}$ is a linear family of variables and for all $i \in [1, n]$ $Var(\mathbf{s_i}) \cap (\bigcup_{j=i}^{n} Var(\mathbf{t_j})) = \emptyset$. A clause $p_0(\mathbf{s_0}, \mathbf{t_0}) \leftarrow p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is called* simply moded *if the query $p_1(\mathbf{s_1}, \mathbf{t_1}), \ldots, p_n(\mathbf{s_n}, \mathbf{t_n})$ is simply moded and $Var(\mathbf{s_0}) \cap (\bigcup_{j=1}^{n} Var(\mathbf{t_j})) = \emptyset$. A program is called simply moded if every clause of it is.*

3 A necessary and sufficient termination condition

In this section we present a termination condition based on orders on terms. Our first definition is motivated by the following example:

Example 1.

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

We can always define $>$, such that $p(f(X)) > p(X)$. However, this information can be used for proving termination only if we refer to the argument position of p, as to an input position.

Guided by this example we are going to define an order on terms that will refer only to their input positions. More precisely, we start by defining the equivalence relation on the set of atoms and terms, that will "ignore" the output positions, and then define an order on these equivalence classes.

Definition 4. *Let P be a moded program, and $A_1, A_2 \in B_P^E$. We will say that $A_1 \sim A_2$ if $(A_1)^{inp}$ and $(A_2)^{inp}$ are identical.*

Definition 5. *Let P be a well-moded program, $>$ a well-founded order on $U_P^E \cup B_P^E/\sim$ and I a model for P. The program P is called* term-acceptable w.r.t. $>$ and I *if for all $A \leftarrow B_1, \ldots, B_n$ in P and all substitutions θ, such that $(A\theta)^{inp}$ and $B_1\theta, \ldots, B_{i-1}\theta$ are ground and $I \models B_1\theta \wedge \ldots B_{i-1}\theta$ holds: $A\theta > B_i\theta$.*

P is called *term-acceptable* if it is term-acceptable w.r.t. some well-founded order and some model. The following theorem states that term-acceptability of a well-moded program is sufficient for termination of well-moded goals w.r.t. this program.

Theorem 1. *Let P be a well-moded program, that is term-acceptable w.r.t. a well-founded order $>$ and a model I. Let G be a well-moded goal. Then G LD-terminates.*

Note that if the requirement of well-modedness is relaxed the theorem does not hold anymore.

Example 2.

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

We assume the modes $p(i)$ and $q(i)$ to be given. This program is not well-moded w.r.t. the given modes, but it is term-acceptable with respect to the following order $>$ on terms $p(a) > \ldots > q(f(f(f(a)))) > q(f(f(a))) > q(f(a)) > q(a)$ and a model $I = \{p(a), q(a), q(f(a)), q(f(f(a))), \ldots\}$. However, note that the well-founded 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 3. Consider the following well-moded program (the mode $p(o)$).

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

The well-moded goal $p(X)$ terminates w.r.t. this program, but it is not term-acceptable, since $p(f(X)) \sim p(g(X))$.

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 the second direction of the equivalence holds as well.

Theorem 2. *Let P be a well-moded and simply moded program, LD-terminating for any well-moded and simply-moded goal. Then there exists a model I and a well-founded order $>$, such that P is term-acceptable w.r.t. I and $>$.*

Example 4. The following example is common in proving termination ([1, 9]).

$$\begin{array}{l}
permute([], []). \\
permute(L, [El|T]) \leftarrow delete(El, L, L1), permute(L1, T). \\
delete(X, [X|T], T). \\
delete(X, [H|T], [H|T1]) \leftarrow delete(X, T, T1).
\end{array}$$

The following modes are assumed to be given: $delete(o, i, o), permute(i, o)$ This program is well-moded w.r.t. these modes. Thus, in order to prove termination of well-moded goals w.r.t. it, we have to find a well-founded order $>$ on $U_P^E \cup B_P^E / \sim$ and a model I, such that the program will be term-acceptable w.r.t. them. Let $>$ be any ordering having the subterm and monotonicity properties and such that for all $t_2, s_1 \in U_P^E$: $delete(t_2) < permute(s_1)$[2] . Let $I = \{delete(t_1, t_2, t_3) \in B_P \mid t_2 > t_3\} \cup \{permute(t_1, t_2) \mid t_1, t_2 \in U_P\}$

First, we verify that I is a model. This can be done by checking that $T_P(I) \subseteq I$, where T_P is the immediate consequence operator. For $permute/2$ clauses, this is trivially fulfilled. For the non-recursive clause of $delete/3$, it requires verifying that for all $t_1, t_2 \in U_P$ we have $delete(t_1, [t_1|t_2], t_2) \in I$. Thus, we need to check that $[t_1|t_2] > t_2$, which holds due to the subterm property of $>$.

To verify the recursive clause of $delete/3$, "if $delete(t_1, t_2, t_3) \in I$ then also $delete(t_1, [t|t_2], [t|t_3]) \in I$" has to be checked. This requires verifying that $t_2 > t_3$ implies $[t|t_2] > [t|t_3]$, which follows from the monotonicity of $>$.

Next we verify the term-acceptability condition. For the recursive clause of $delete/3$, we need that $delete([H|T])\theta > delete(T)\theta$, for each grounding θ. This follows from the subterm and monotonicity properties of $>$.

[2] For the sake of simplicity we write A^{inp} instead of A in the requirements for $>$ on elements of $B_P^E / \sim$, since $>$ respects $\sim$, thus, cannot depend on the output positions

For the recursive clause for *permute*/2, first note that the decrease w.r.t. the *delete*/3-body atom is fulfilled, because we imposed it on $>$. For the recursive body atom, we need that, if $I \models delete(El, L, L1)\theta$, then $permute(L)\theta > permute(L1)\theta$. Due to the monotonicity of $>$, it is sufficient that $L\theta > L1\theta$ is implied by $I \models delete(El, L, L1)\theta$, which holds because of I.

The following observation is readily generalised from the example above. Assume that we have a clause $p(t_1, \ldots, t_n) \leftarrow B_1, \ldots, B_{i-1}, q(s_1, \ldots, s_m), \ldots, B_k$, such that p/n and q/m are not mutually dependent on each other (such as the *delete* atom in the recursive clause for *permute*). Then, we can always impose that $p^{inp}(u_1, \ldots, u_{n'}) > q^{inp}(v_1, \ldots, v_{m'})$, for all $u_1, \ldots, u_{n'}, v_1, \ldots, v_{m'} \in U_P$. Such constraints on $>$ will be orthogonal to all other constraint we may impose on it. Therefore, in the remainder of the paper, we will only check the acceptability decrease $A\theta > B_i\theta$ for body atoms B_i with a predicate symbol that is mutually dependent on that of A.

Note that, comparing with standard acceptability, this is not surprising. In standard acceptability, the decrease for non-mutually dependent predicates is only needed to ensure that a boundedness property is inherited from head to body-atoms in the clauses. Since we are focusing on well-moded programs and queries only, boundedness follows from groundedness of the input and is always inherited by body atoms.

4 Towards automation of the approach

In this section we present an approach to automatic verification of the term-acceptability condition. The basic idea for the approach is inspired on the "constraint based" termination analysis proposed in [9]. 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 order $>$ and the model I.

This approach has been applied successfully to a number of examples that appear in the literature on termination, such as different versions of *permute* [5, 15, 9], *dis-con* [6], *transitive closure* [15], *add-mult* [16], *combine*, *reverse*, *odd-even*, *at_least_double* and *normalization* [9], *quicksort* program [17, 1], *derivative* [12], *distributive law* [11], *boolean ring* [14], *flatten* [4].

In the remainder of the paper, we explain the approach using two of these examples, presented above—permutation and derivation.

The analysis starts off from the $>$-decreases imposed in the term-acceptability. For the *permute* example, these are

$$delete([H|T])\theta > delete(T)\theta \qquad (1)$$

$$I \models delete(El, L, L1)\theta \text{ implies } permute(L)\theta > permute(L1)\theta \qquad (2)$$

For each such condition, 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 $>$, which is consistent with the constraints that were already imposed on $>$ before.

In the *permute* example, condition (1) is satisfied by imposing the subterm property on $>$.

In general we can select from a bunch of term-order properties, or even specific term-orders, that were proposed in the literature. Orderings that may be considered are general orderings, having subterm and monotonicity properties; recursive path ordering with status [13]; lexicographic ordering; etc.

Option 2): The required decrease is imposed as a constraint on the model I.

In the *permute* example, the decrease $permute(L)\theta > permute(L1)\theta$ cannot directly be achieved by imposing some constraint on $>$. Thus, we impose that the underlying decrease $L\theta > L1\theta$ should hold for the intermediate body atoms $(delete(El, L, L1)\theta)$[3] that are in the model I.

Thus, in the example, the constraint is that I should be such that for all $delete(t_1, t_2, t_3) \in I$: $t_2 > t_3$. One way to constructively verify that an I exists such that the property $t_2 > t_3$ holds for its $delete(t_1, t_2, t_3)$ atoms is to simply impose that $S = \{delete(t_1, t_2, t_3) \mid t_2 > t_3\}$ itself is a model for the *delete* clauses in the program.

So our new constraint on I is that it should include S. Practically we can enforce this by imposing that $T_P(S) \subseteq S$ should hold. As shown in the previous section, this reduces to the constraints "$[t_1|t_2] > t_2$" and "$t_2 > t_3$ implies $[t|t_2] > [t|t_3]$". These are again fed into our Option 1) step, imposing monotonicity and subterm properties on $>$. At this point the proof is complete.

Example 5. Our last example is the motivating example from the Introduction, i.e., computing the repeated derivative of a linear function in one variable t. We assume that the moding information $d(i, o)$ is given. We start by constructing the $>$-decreases that follow from the term-acceptability.

$$d(der(X+Y))\theta > d(der(X))\theta \tag{3}$$

$$I \models d(der(X), DX)\theta \text{ implies } d(der(X+Y))\theta > d(der(Y))\theta \tag{4}$$

$$d(der(X*Y))\theta > d(der(X))\theta \tag{5}$$

$$I \models d(der(X), DX)\theta \text{ implies } d(der(X*Y))\theta > d(der(Y))\theta \tag{6}$$

$$d(der(der(X)))\theta > d(der(X))\theta \tag{7}$$

$$I \models d(der(X), DX)\theta \text{ implies } d(der(der(X))\theta > d(der(DX))\theta \tag{8}$$

Conditions (3)-(7) impose monotonicity and subset properties to hold on $>$. In order to satisfy condition (8), it is sufficient to prove that in I, the first

[3] It might happen that more than one intermediate goal preceded the recursive atom in the body 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 program P' that is further analysed.

argument of d is greater than the second one, that is that if $S = \{d(t_1, t_2) \mid t_1 > t_2\}$ then $T_P(S) \subseteq S$ and $S \subseteq I$. This may be reduced to the following conditions:

$$der(t) > 1 \tag{9}$$
$$I \models number(t_1) \text{ implies } der(t_1) > 0 \tag{10}$$
$$der(t_1) > t_2 \;\&\; der(t_3) > t_4 \text{ implies } der(t_1 + t_3) > t_2 + t_4 \tag{11}$$
$$der(t_1) > t_2 \;\&\; der(t_3) > t_4 \text{ implies } der(t_1 * t_3) > t_1 * t_4 + t_2 * t_3 \tag{12}$$
$$der(t_1) > t_2 \;\&\; der(t_2) > t_3 \text{ implies } der(der(t_1)) > t_3 \tag{13}$$

Condition (13) follows from monotonicity and transitivity of $>$. However, (9)-(12) 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 [10] with *der* having the highest priority.

5 Discussion and conclusions

We have presented a non-transformational approach to termination analysis of logic programs, based on general term-orderings. The 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, 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.

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.

However, in its current version, the proposed technique is not a fully conservative extension of standard acceptability. For the moment, our approach is limited to well-moded programs and goals only.

Another line of future work is to develop an extension to term-orderings for the notion of "acceptability with respect to a set of atoms" of [7, 8]. We expect that this extension may be much easier in the non-ground case, because "acceptability with respect to a set" does not require any notion of boundedness.

6 Acknowledgement

Alexander Serebrenik is supported by GOA: "LP^+: a second generation logic programming language". Danny De Schreye is a senior research associate of FWO Flanders, Belgium.

References

1. K. R. Apt. *From Logic Programming to Prolog.* Prentice-Hall International 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 International Symposium on Mathematical Foundations of Computer Science*, pages 1–19. Springer Verlag, 1993. Lecture Notes in Computer Science, volume 711.
3. K. R. Apt and D. Pedreschi. Studies in Pure Prolog: Termination. In J. W. Lloyd, editor, *Proceedings Esprit Symposium on Computational Logic*, pages 150–176. Springer Verlag, 1990.
4. T. Arts. *Automatically proving termination and innermost normalisation of term rewriting systems.* PhD thesis, Faculteit Wiskunde & Informatica, Universiteit Utrecht, 1997.
5. T. Arts and H. Zantema. Termination of logic programs using semantic unification. In M. Proietti, editor, *Logic Programming Synthesis and Transformation, 5th International Workshop. LOPSTR'95*, pages 219–233. Springer Verlag, 1995. Lecture Notes in Computer Science, volume 1048.
6. D. De Schreye and S. Decorte. Termination of logic programs: The never-ending story. *The Journal of Logic Programming*, 19/20:199–260, May/July 1994.
7. 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, *Fifth Generation Computer Systems '92: Proceedings of the International Conference on Fifth Generation Computer Systems.*, pages 481–488. IOS Press, 1992.
8. S. Decorte and D. De Schreye. Termination analysis: some practical properties of the norm and level mapping space. In J. Jaffar, editor, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming*, pages 235–249. MIT Press, June 1998.
9. 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.
10. N. Dershowitz. Termination. In C. Kirchner, editor, *Proceedings of the First International Conference on Rewriting Techniques and Applications*, pages 180–224. Springer Verlag, 1985. Lecture Notes in Computer Science, volume 202.
11. N. Dershowitz and C. Hoot. Topics in termination. In C. Kirchner, editor, *Rewriting Techniques and Applications, 5th International Conference*, pages 198–212. Springer Verlag, 1993. Lecture Notes in Computer Science, volume 690.
12. N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM (CACM)*, 22(8):465–476, August 1979.
13. M. C. Ferreira and H. Zantema. Well-foundedness of term orderings. In N. Dershowitz, editor, *Conditional Term Rewriting Systems, proceedings fourth international workshop CTRS-94*, pages 106–123. Springer Verlag, 1995. Lecture Notes in Computer Science, volume 968.

14. J. Hsiang. Rewrite method for theorem proving in first order theory with equality. *Journal of Symbolic Computation*, 8:133–151, 1987.
15. M. Krishna Rao, D. Kapur, and R. Shyamasundar. Transformational methodology for proving termination of logic programs. *The Journal of Logic Programming*, 34:1–41, 1998.
16. L. Plümer. *Termination Proofs for Logic Programs.* Lecture Notes in Artificial Intelligence, volume 446. Springer Verlag, 1990.
17. L. Sterling and E. Shapiro. *The Art of Prolog.* The MIT Press, 1994.