Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

K.R. Apt, D. Pedreschi

Studies in pure Prolog: termination

Computer Science/Department of Software Technology Report CS-R9048 September

---

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Studies in Pure Prolog: Termination

Krzysztof R. Apt
Centre for Mathematics and Computer Science
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands

Dino Pedreschi
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56125 Pisa, Italy

Abstract

We provide a theoretical basis for studying termination of logic programs with the Prolog selection rule. To this end we study the class of *left terminating* programs. These are logic programs that terminate with the Prolog selection rule for all ground goals. First we show that various ways of defining semantics coincide for left terminating programs. Then we offer a characterization of left terminating programs that provides us with a practical method of proving termination. The method is proven to be complete and is illustrated by giving simple proofs of termination of the quicksort, permutation and mergesort programs for the desired class of goals.

Keywords and Phrases: Prolog programs, termination, declarative semantics, left terminating programs, acceptable programs.
1985 Mathematics Subject Classification: 68Q40, 68T15,
CR Categories: F.3.2., F.4.1, H.3.3, I.2.3.

Notes. This research was partly done during the authors' stay at the Department of Computer Sciences, University of Texas at Austin, Austin, Texas, U.S.A. . First author's work was partly supported by ESPRIT Basic Research Action 3020 (Integration). Second author's work was partly supported by ESPRIT Basic Research Action 3012 (Compulog) and by the Italian National Research Council – C.N.R.. This paper will appear in Proceedings of Symposium on Computional Logic (J.W. Lloyd editor), Lecture Notes in Computer Science, Springer-Verlag, 1990.

1 Introduction

Background

Algorithms are designed for two types of problems - decidable ones and semi-decidable ones. In the latter case we cannot claim termination for all inputs. In the former case we usually can and only in few cases - like interactive programs (game playing programs, editors, ...) or operating systems, we choose not to do so.

In this paper we study termination of Prolog programs and, naturally, confine our attention to the category of programs that terminate for all inputs. By termination we mean here finiteness of *all* possible Prolog derivations starting in the initial goal. However, in the case of Prolog programs one is confronted with the problem that an apparently correct program may fail to

terminate in this sense for certain forms of inputs. For example, the append program fails to terminate in this sense for a goal with all arguments being variables. To cope with this complication we only require that the program terminates for all *ground* inputs. In such cases only "yes" or "no" answer can be given. We call such programs *left terminating*. Then to show that a Prolog program exhibits a proper termination behaviour we first show that it is left terminating and then that it terminates for certain types of non-ground inputs. Our method of showing the former will also allow us to establish the latter.

When studying Prolog programs from the point of view of termination it is useful to notice that some programs terminate for all ground goals for *all* selection rules. Such programs are extensively studied in Bezem [Bez89] where they are called *terminating programs*. These are usually programs whose termination depends on a simple reduction in one or more arguments. Examples of terminating programs are append, member, N queens, various tree insertion and deletion programs and several others.

However, some Prolog programs satisfy such a strong termination property but fail to terminate for certain desired forms of inputs for some selection rules.

An example is the following append3 program in which the append program is used:

```
append3(Xs, Ys, Zs, Us) ←
  append(Xs, Ys, Vs),
  append(Vs, Zs, Us).
```

Then append3 is a terminating program which terminates for the goal ← append3(xs, ys, zs, Us), where xs, ys, zs are lists and Us a variable, when the Prolog selection rule is used but fails to terminate when the rightmost selection rule is used.

Worse yet, some programs fail to be terminating even though they terminate for the Prolog selection rule for the desired class of inputs. An example is the flatten program which collects all the nodes of a tree in a list:

```
flatten(nil, []) ←.
flatten(t(L, X, R), Xs) ←
  flatten(L, X1s),
  flatten(R, X2s),
  append(X1s, [X | X2s], Xs).
```

flatten is not a terminating program but it terminates for the goal ← flatten(x, Xs), where x is a ground term and Xs a variable, when the Prolog selection rule is used.

In general, the problem arises due to the use of local variables, i.e. variables which appear in the body of a clause but not in its head. Several left terminating Prolog programs use local variables in an essential way and consequently fail to be terminating. Examples of such programs are various sorting and permutation programs and graph searching programs. Programs which fall into this category are usually of the form "generate and test" or "divide and conquer".

In this paper we provide a framework to study left terminating programs. To this end we refine the ideas of Bezem [Bez89] and Cavedon [Cav89] and use the concept of a level mapping. This is a function assigning natural numbers to ground atoms. Our main tool is the concept of an *acceptable program*. Intuitively, a program is acceptable if for some level mapping, for all ground instances of the clauses of the program, the level of the head is smaller than the level of atoms in a certain prefix of the body. Which prefix is considered is determined by some model of the program.

The main result of the paper is that the notions of left termination and acceptability coincide. The proof of this fact uses an iterated multiset ordering. This equivalence result provides us

with a method of proving left termination. Moreover, it allows us to prove termination of a left terminating Prolog program for a class of non-ground goals. The method is easy to use and is illustrated by proving termination of the quicksort, permutation and mergesort programs.

Plan of the paper

This paper is organized as follows. In the next section we introduce the concept of a left terminating program. This is a program that terminates for all ground goals w.r.t. Prolog selection rule. We show that left terminating programs satisfy an elegant semantic property: the least Herbrand model of a left terminating program P is a unique fixpoint of the immediate consequence operator T_P associated with P, can be identified with the unique fixpoint of the 3-valued immediate consequence operator associated with P and can be characterized in terms of the completion of P, $comp(P)$.

In Section 3 we provide a useful characterization of left terminating programs by introducing the notion of an acceptable program and proving that the notions of acceptability and left termination coincide. The crucial concept here is that of a bounded goal. It allows us to characterize terminating goals.

Finally, in Section 4 we prove left termination of the quicksort, permutation and mergesort programs by providing in each case a simple proof of acceptability. Using the concept of boundedness we show that each program terminates w.r.t. a desired class of non-ground goals.

Preliminaries

We use standard notation and terminology of Lloyd [Llo87] or Apt [Apt88]. In particular, we use the following abbreviations for a logic program P (or simply a *program*):
B_P for the Herbrand Base of P,
T_P for the immediate consequence operator of P,
M_P for the least Herbrand model of P,
$ground(P)$ for the set of all ground instances of clauses from P,
$comp(P)$ for Clark's completion of P.

Also, we use Prolog's convention identifying in the context of a program each string starting with a capital letter with a variable, reserving other strings for the names of constants, terms or relations. So, for example Xs stands for a variable whereas xs stands for a term.

In the programs we use the usual list notation. The constant [ ] denotes the empty list and [. | .] is a binary function which given a term x and a list xs produces a new list [x | xs] with head x and tail xs. By convention, identifiers ending with "s", like xs, will range over lists. The standard notation [$x_1, \ldots, x_n$], for $n \geq 0$, is used as an abbreviation of [x_1 | [$\ldots$ [x_n | [ ]] $\ldots$]]. In general, the Herbrand Universe will also contain "impure" elements that contain [ ] or [. | .] but are not lists - for example $s([\,])$ or $[s(0) \mid 0]$ where 0 is a constant and s a unary function symbol. They will not cause any complications.

Given an operator T on a complete partial ordering L with the least element $\bot$, we define the *upward ordinal powers* of T starting at $\bot$ in the standard way and denote them by $T \uparrow \alpha$ where α is an ordinal. If L has the greatest element, say $\top$, (this is the case when for example L is a complete lattice) we define the *downward ordinal powers* of T starting at $\top$ in the standard way and denote them by $T \downarrow \alpha$.

Throughout the paper we consider SLD-resolution with one selection rule only – namely that of Prolog, usually called the leftmost selection rule. As S in SLD stands for "selection rule", we denote this form of resolution by LD (Linear resolution for Definite clauses). The

concepts of LD-derivation, LD-refutation, LD-tree, etc. are then defined in the usual way. By "pure Prolog" we mean in this paper the LD-resolution combined with the depth first search in the LD-trees.

By choosing variables of the input clauses and the used mgu's in a fixed way we can assume that for every program P and goal G there exists exactly one LD-tree for $P \cup \{G\}$.

2 Left Termination

Our interest here is in terminating Prolog programs. This motivates the following concept.

Definition 2.1 A program P is called *left terminating* if all LD-derivations of P starting in a ground goal are finite. □

In other words, a program is left terminating if all LD-trees for P with a ground root are finite. When studying Prolog programs, one is actually interested in proving termination of a given program not only for all ground goals but also for a class of non-ground goals constituting the intended queries. Our method of proving left termination will allow us to identify for each program such a class of non-ground goals.

But first let us see some simple consequences of the above definition. Following Blair [Bla86] a program is called *determinate* if $T_P \uparrow \omega = T_P \downarrow \omega$.

Theorem 2.2 *Every left terminating program is determinate.*

Proof. By the results of Apt and Van Emden [AvE82] (see also Lloyd [Llo87]) for every program P

$T_P \uparrow \omega = \{A \in B_P \mid \text{there exists a successful } SLD\text{-tree for } P \cup \{\leftarrow A\}\}$,
$T_P \downarrow \omega = \{A \in B_P \mid \text{there does not exist a finitely failed } SLD\text{-tree for } P \cup \{\leftarrow A\}\}$.

We always have $T_P \uparrow \omega \subseteq T_P \downarrow \omega$, since T_P is monotonic. To prove the converse inclusion for a left terminating program P, take some $A \in T_P \downarrow \omega$. By the second equality the LD-tree for $P \cup \{\leftarrow A\}$ is not finitely failed. But by the choice of P it is finite, so it is successful. Thus by the first equality $A \in T_P \uparrow \omega$.

□

The converse of the above theorem does not hold - it suffices to take $P = \{A \leftarrow A, B\}$. Then $T_P \uparrow \omega = \emptyset$ and $T_P \downarrow \omega = \emptyset$ but P is not left terminating.

The determinate programs, and consequently left terminating programs, enjoy some pleasing semantic properties it is useful to record.

Theorem 2.3 *For a determinate program P, M_P is the unique fixpoint of T_P.*

Proof. We prefer to give a more general proof of this fact. To this end consider a monotonic operator T on a complete lattice. Then by monotonicity
(i) for every fixpoint Y of T

$$T \uparrow \omega \subseteq Y \subseteq T \downarrow \omega,$$

(ii) $T \uparrow \omega \subseteq T \uparrow (\omega + 1) \subseteq T \downarrow \omega$.

Suppose now that $T \uparrow \omega = T \downarrow \omega$. Then by (i) T has at most one fixpoint and by (ii) $T \uparrow \omega$ is a fixpoint of T, since by definition $T \uparrow (\omega + 1) = T(T \uparrow \omega)$.
The claim of the theorem now follows, since T_P is monotonic and by the result of Apt and Van Emden [AvE82] $M_P = T_P \uparrow \omega$. □

The other property of determinate programs is based on the theory of 3-valued models for logic programs developed by Fitting [Fit85]. We recall first the relevant definitions and results. Fitting [Fit85] uses a 3-valued logic due to Kleene [Kle52].

In Kleene's logic there are three truth values: **t** for true, **f** for false and **u** for undefined. Every connective takes the value **t** or **f** if it takes that value in 2-valued logic for all possible replacements of **u**'s by **t** or **f**; otherwise it takes value **u**.

A Herbrand interpretation for this logic (called a 3-*valued* Herbrand interpretation) is defined as a pair (T, F) of disjoint sets of ground atoms. Given such an interpretation $I = (T, F)$ a ground atom A is true in I if $A \in T$, false in I if $A \in F$ and undefined otherwise. Given $I = (T, F)$ we denote T by I^+ and F by I^-. Thus $I = (I^+, I^-)$. If $I^+ \cup I^- = B_P$, we call I a *total* 3-valued Herbrand interpretation for the program P.

Every (2-valued) Herbrand interpretation I for a program P determines a total 3-valued Herbrand interpretation $(I, B_P - I)$ for P. This allows us to identify every 2-valued Herbrand interpretation I with its 3-valued counterpart $(I, B_P - I)$.

Given a program P, the 3-valued Herbrand interpretations for P form a complete partial ordering with the ordering $\subseteq$ defined by

$$I \subseteq J \text{ iff } I^+ \subseteq J^+ \wedge I^- \subseteq J^-$$

and with the least element $(\emptyset, \emptyset)$. Note that in this ordering every total 3-valued Herbrand interpretation is $\subseteq$-maximal.

Following Fitting [Fit85], given a program P we define an operator Φ_P on the complete partial ordering of 3-valued Herbrand interpretations for P as follows:

$$\Phi_P(I) = (T, F),$$

where

$$T = \{A \mid \text{there exists } A \leftarrow B_1, \ldots, B_k \text{ in } \mathit{ground}(P) \text{ with } B_1 \wedge \ldots \wedge B_k \text{ true in } I\},$$
$$F = \{A \mid \text{for all } A \leftarrow B_1, \ldots, B_k \text{ in } \mathit{ground}(P),\ B_1 \wedge \ldots \wedge B_k \text{ is false in } I\}.$$

It is easy to see that T and F are disjoint, so $\Phi_P(I)$ is indeed a 3-valued Herbrand interpretation. Φ_P is a natural generalization of the operator T_P to the case of 3-valued logic. Φ_P is easily seen to be monotonic. The following observation of Fitting [Fit85] is of relevance here.

Lemma 2.4 *For every program P and ordinal α*

$$\Phi_P \uparrow \alpha = (T_P \uparrow \alpha,\ B_P - T_P \downarrow \alpha).$$

□

This implies the following results.

Lemma 2.5 *For a determinate program P, $M_P = \Phi_P \uparrow \omega$.* □

Proof. By Lemma 2.4 and the fact that $M_P = T_P \uparrow \omega$.

Corollary 2.6 *For a determinate program P, M_P is the unique fixpoint of Φ_P.*

Proof. Let Y be a fixpoint of Φ_P. By the monotonicity of Φ_P, $\Phi_P \uparrow \omega \subseteq Y$, so by Lemma 2.5, $M_P \subseteq Y$. But M_P is a total 3-valued Herbrand interpretation so it is $\subseteq$-maximal and consequently $M_P = Y$. □

The final characterization of the model M_P for determinate programs is in terms of the completion $comp(P)$.

Theorem 2.7 *For a determinate program P, for all ground atoms $A \in B_P$*

$$\begin{aligned} M_P &\models A \quad \text{iff } comp(P) \models A, \\ M_P &\models \neg A \quad \text{iff } comp(P) \models \neg A. \end{aligned}$$

Proof. Combining various completeness and characterization results (see Lloyd [Llo87] or Apt [Apt88]) we have for every logic program P,

$$\begin{aligned} T_P \uparrow \omega \models A \quad &\text{iff} \quad comp(P) \models A, \\ T_P \downarrow \omega \models \neg A \quad &\text{iff} \quad comp(P) \models \neg A. \end{aligned}$$

But for a determinate program P, $M_P = T_P \uparrow \omega = T_P \downarrow \omega$. □

Corollary 2.8 *For a determinate program P*

$$\begin{aligned} M_P &= \{A \in B_P \mid comp(P) \models A\}, \\ M_P &= \{A \in B_P \mid comp(P) \not\models \neg A\}. \end{aligned}$$

□

Thus for determinate programs, and a fortiori for left terminating programs, three most common approaches to semantics coincide and result in a simple declarative semantics in the form of a unique fixpoint of the T_P operator which coincides with the unique fixpoint of the Φ_P operator and which can be characterized by means of the completion $comp(P)$.

3 Proving Left Termination

Let us consider now how to prove that a program is left terminating. Starting from Floyd [Flo67] the classical proofs of program termination have been based on the use of well-founded orderings. This approach has been successfully used in the area of logic programming (see e.g. Bezem [Bez89], Cavedon [Cav89]) but with no attention paid to Prolog programs. The notable exception is Deville [Dev90].

We obtain the desired method by a modification of the ideas of Bezem [Bez89] and Cavedon [Cav89].

Recurrent Programs

It is useful to recall first some concepts and results from Bezem [Bez89]. A *level mapping* for a program P is a function $|\ |\ : B_P \rightarrow N$ of ground atoms to natural numbers. For $A \in B_P$, $|A|$ is the level of A. Following Bezem [Bez89] (see also Cavedon [Cav89]), a program is called *recurrent* if for some level mapping $|\ |$, for every clause $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$

$$|A| > |B_i| \text{ for } i \in [1, n].$$

Another relevant concept is that of *boundedness*: an atom A is bounded with respect to a level mapping $|\ |$ if $|\ |$ is bounded on the set $[A]$ of ground instances of A. A goal is bounded if all its atoms are. Bezem [Bez89] showed that every SLD-derivation of a recurrent program starting in a bounded goal terminates.

A program is called *terminating*, if all its SLD-derivations starting in a ground goal are finite. Hence, terminating programs have the property that the SLD-trees of ground goals are finite, and any search procedure in such trees will always terminate, independently from the adopted selection rule.

One of the main results in Bezem [Bez89] is that a program is recurrent if and only if it is terminating. Because of this result recurrent programs and bounded goals are too restrictive concepts to deal with Prolog programs, as a larger class of programs and goals is terminating when adopting a specific selection rule, e.g. Prolog selection rule.

Example 3.1
(i) Consider the following program `even` which defines even numbers and the "less than or equal" relation:

```
even(0) ←.
even(s(s(X))) ← even(X).

lte(0,Y) ←.
lte(s(X),s(Y)) ← lte(X,Y).
```

`even` is recurrent with $|even(s^n(0))| = n$ and $|lte(s^n(0), s^m(0))| = \min\{n, m\}$. Now consider the goal:

$$G = \leftarrow lte(x, s^{100}(0)), even(x)$$

which is supposed to compute the even numbers not exceeding 100. The LD-tree for G is finite, whereas there exists an infinite SLD-derivation when the rightmost selection rule is used. As a consequence of Bezem's result, the goal G is not bounded, although it can be evaluated by a finite Prolog computation.

Actually, most "generate and test" Prolog programs are not recurrent, as they heavily depend on the left-to-right order of evaluation, like the example above.

(ii) Consider the following naive `reverse` program:

```
reverse([], []) ←.
reverse([X | Xs], Ys) ←
  reverse(Xs, Zs),
```

```
  append(Zs, [X], Ys).

append([], Ys, Ys) ←.
append([X | Xs], Ys, [X | Zs]) ← append(Xs, Ys, Zs).
```

The ground goal $\leftarrow$ *reverse*(xs, ys), for arbitrary lists xs and ys, has an infinite SLD-derivation, obtained by using the selection rule which selects the leftmost atom at the first two steps, and the second leftmost atom afterwards. By Bezem's result, **reverse** is not recurrent.

(iii) Consider the following program DC, representing a (binary) "divide and conquer" schema; it is parametric with respect to the *base, conquer, divide* and *merge* predicates.

```
dc(X,Y) ←
  base(X),
  conquer(X,Y).
dc(X,Y) ←
  divide(X,X1,X2),
  dc(X1,Y1),
  dc(X2,Y2),
  merge(Y1,Y2,Y).
```

Many programs naturally fit into this schema, or its generalization to non fixed arity of the `divide/merge` predicates. Unfortunately, DC is not recurrent: it suffices to take a ground instance of the recursive clause with $X = a$, $X1 = a$, $Y = b$, $Y1 = b$, and observe that the atom *dc(a,b)* occurs both in the head and in the body of such a clause. In this example, the leftmost selection rule is needed to guarantee that the input data is divided into subcomponents before recurring on such subcomponents. □

Acceptable Programs

To cope with these difficulties we modify the definition of a recurrent program as follows.

Definition 3.2 Let P be a program, $||$ a level mapping for P and I a (not necessarily Herbrand) model of P. P is called *acceptable with respect to* $||$ *and* I if for every clause $A \leftarrow B_1, \ldots, B_n$ in *ground*(P)

$$|A| > |B_i| \text{ for } i \in [1, \bar{n}],$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models B_i\}).$$

Alternatively, we may define $\bar{n}$ by

$$\bar{n} = \begin{cases} n & \text{if } I \models B_1 \wedge \ldots \wedge B_n, \\ i & \text{if } I \models B_1 \wedge \ldots \wedge B_{i-1} \text{ and } I \not\models B_1 \wedge \cdots \wedge B_i. \end{cases}$$

P is called *acceptable* if it is acceptable with respect to some level mapping and a model of P. □

Thus, given a level mapping $|\ |$ for P and a model I of P, in the definition of acceptability w.r.t. $|\ |$ and I for every clause $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$ we only require that the level of A is higher than the level of B_i's in a certain prefix of $B_1, \ldots, B_n$. Which B_i's are taken into account is determined by the model I. If $I \models B_1 \wedge \ldots \wedge B_n$ then all of them are considered and otherwise only those whose index is $\leq \bar{n}$, where $\bar{n}$ is the least index i for which $I \not\models B_i$.

The idea underlying the above definition can be illustrated by the following example. Consider a program P containing the clause

$$p(X) \leftarrow q(X,Y), r(Y)$$

and a model I of P. Consider two ground instances

(c_1) $p(a) \leftarrow q(a,b), r(b)$,
(c_2) $p(a) \leftarrow q(a,c), r(c)$

of this clause (assuming that the constants a, b, c are in the Herbrand Universe of P) and suppose that $q(a,b) \in I$ but $q(a,c) \notin I$. To prove acceptability, a level mapping $|\ |$ is supposed to satisfy

$$|p(a)| > |q(a,b)| \text{ and } |p(a)| > |r(b)|$$

for clause (c_1), but only

$$|p(a)| > |q(a,c)|$$

for clause (c_2). Intuitively, the condition $q(a,c) \notin I$ excludes (by the soundness of the SLD-resolution) the existence of a refutation for $q(a,c)$ and consequently there is no point in checking that the level mapping decreases from $p(a)$ to $r(c)$, since the Prolog interpreter will never reach $r(c)$ during the execution starting with the goal $\leftarrow p(a)$.

The following observation is immediate.

Lemma 3.3 *Every recurrent program is acceptable.*

Proof. Take $I = B_P$. Then for every $A \leftarrow B_1, \ldots, B_n$ in $ground(P)$, $\bar{n} = n$. □

Our aim is to prove that the notions of acceptability and left termination coincide.

Multiset ordering

To prove one half of this statement we use the multiset ordering. A *multiset*, sometimes called *bag*, is an unordered sequence. Given a (non-reflexive) ordering $<$ on a set W, the *multiset ordering over* $(W, <)$ is an ordering on finite multisets of the set W. It is defined as the transitive closure of the relation in which X is smaller than Y if X can be obtained from Y by replacing an element a of Y by a finite (possibly empty) multiset each of whose elements is smaller than a in the ordering $<$.

In symbols, first we define the relation $\prec$ by

$$X \prec Y \text{ iff } X = Y - \{a\} \cup Z \text{ for some } Z \text{ such that } b < a \text{ for } b \in Z,$$

where X, Y, Z are finite multisets of elements of W, and then define the multiset ordering over $(W, <)$ as the transitive closure of the relation $\prec$.

It is well-known (see e.g. Dershowitz [Der87]) that multiset ordering over a well-founded ordering is again well-founded. Thus it can be iterated while maintaining well-foundedness.

What we need in our case is two fold iteration. We start with the set of natural numbers N ordered by $<$ and apply the multiset ordering twice. We call the first iteration multiset ordering and the second *double multiset ordering*. Both are well-founded. The double multiset ordering is defined on the finite *multisets* of finite multisets of natural numbers, but we shall use it only on the finite *sets* of finite multisets of natural numbers. The following lemma will be of help when using the double multiset ordering.

Lemma 3.4 *Let X and Y be two finite sets of finite multisets of natural numbers. Suppose that*

$$\forall x \in X \, \exists y \in Y \, (y \, majorizes \, x),$$

where y majorizes x means that x is smaller than y in the multiset ordering.

Then X is smaller than Y in the double multiset ordering.

Proof. We call an element $y \in Y$ *majorizing* if it majorizes some $x \in X$. X can be obtained from Y by first replacing each majorizing $y \in Y$ by the multiset M_y of elements of X it majorizes and then removing from Y the non-majorizing elements. This proves the claim. □

Below we use the notation bag $(a_1, \ldots, a_n)$ to denote the multiset consisting of the unordered sequence $a_1, \ldots, a_n$.

Boundedness

Another important concept is boundedness. It allows us to identify goals from which no divergence can arise. Recall that an atom A is called *bounded* w.r.t. a level mapping $||$ if $||$ is bounded on the set $[A]$ of ground instances of A. If A is bounded, then $|[A]|$ denotes the maximum that $||$ takes on $[A]$. Note that every ground atom is bounded.

Our concept of a bounded goal differs from that of Bezem [Bez89] in that it takes into account the model I. This results in a more complicated definition.

Definition 3.5 Let P be a program, $||$ a level mapping for P, I a model of P and $k \geq 0$.

(i) With each ground goal $G = \leftarrow A_1, \ldots, A_n$ we associate a finite multiset $|G|_I$ of natural numbers defined by

$$|G|_I = \text{bag}\,(|A_1|, \ldots, |A_{\bar{n}}|),$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models A_i\}).$$

(ii) With each goal G we associate a set of multisets $|[G]|_I$ defined by

$$|[G]|_I = \{|G'|_I \mid G' \text{ is a ground instance of } G\}.$$

(iii) A goal G is called *bounded by* k w.r.t. $||$ and I if $k \geq \ell$ for $\ell \in \cup|[G]|_I$.

A goal is called *bounded* w.r.t. $||$ and I if it is bounded by some $k \geq 0$ w.r.t. $||$ and I.

□

It is useful to note the following.

Lemma 3.6 *Let P be a program, $||$ a level mapping for P and I a model of P. A goal G is bounded w.r.t. $||$ and I iff the set $|[G]|_I$ is finite.*

Proof. Consider a goal G that is bounded by some k. Suppose that G has n atoms. Then each element of $|[G]|_I$ is a multiset of at most n numbers selected from $[0, k]$. The number of such multisets is finite.

The other implication is obvious. □

The following lemma is an analogue of Lemma 2.5 of Bezem [Bez89].

Lemma 3.7 *Let P be a program that is acceptable w.r.t. a level mapping $||$ and a model I. Let G be a goal that is bounded (w.r.t. $||$ and I) and let H be an LD-resolvent of G from P. Then*

(i) H is bounded,

(ii) $|[H]|_I$ is smaller than $|[G]|_I$ in the double multiset ordering.

Proof. Let $G = \leftarrow A_1, \ldots, A_n$ $(n \geq 1)$. For some input clause $C = A \leftarrow B_1, \ldots, B_k$ $(k \geq 0)$ and mgu θ of A and A_1, $H = \leftarrow (B_1, \ldots, B_k, A_2, \ldots, A_n)\theta$.

First we show that for every ground instance H_0 of H there exists a ground instance G' of G such that $|H_0|_I$ is smaller that $|G'|_I$ in the multiset ordering.

So let H_0 be a ground instance of H. For some substitution δ

$$H_0 = \leftarrow B'_1, \ldots, B'_k, A'_2, \ldots, A'_n$$

and A'_1 is ground, where for brevity for any atom, clause or goal B, B' denotes $B\theta\delta$. Note that

$$C' = A'_1 \leftarrow B'_1, \ldots, B'_k$$

and

$$G' = \leftarrow A'_1, \ldots, A'_n,$$

since $A' = A'_1$ as $A\theta = A_1\theta$.

Case 1 For $i \in [1, k]$ $I \models B'_i$.
Then

$$|H_0|_I = \text{ bag}\,(|B'_1|, \ldots, |B'_k|,\ |A'_2|, \ldots, |A'_{\bar{n}}|)$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [2, n] \mid \not\models A'_i\}).$$

Additionally, since I is a model of P, $I \models A'_1$. Thus

$$|G'|_I = \text{ bag}\,(|A'_1|,\ |A'_2|, \ldots, |A'_{\bar{n}}|).$$

This means that $|H_0|_I$ is obtained from $|G'|_I$ by replacing $|A'_1|$ by $|B'_1|, \ldots, |B'_k|$. But by the definition of acceptability

$$|B'_i| < |A'_1|$$

for $i \in [1, k]$, so $|H_0|_I$ is smaller than $|G'|_I$ in the multiset ordering. □

Case 2 For some $i \in [1,k]$ $I \not\models B'_i$.
Then

$$|H_0|_I = \text{bag}(|B'_1|, \ldots, |B'_{\bar{k}}|)$$

where

$$\bar{k} = \min(\{i \in [1,k] \mid I \not\models B'_i\}).$$

Also by the definition of acceptability

$$|B'_i| < |A'_1|$$

for $i \in [1,\bar{k}]$, so $|H_0|_I$ is smaller than $|G'|_I$ in the multiset ordering. □

This implies claim (i) since G is bounded. By Lemma 3.6 $|[H]|_I$ is finite and claim (ii) now follows by Lemma 3.4. □

Corollary 3.8 *Let P be an acceptable program and G a bounded goal. Then all LD-derivations of $P \cup \{G\}$ are finite.*

Proof. The double multiset ordering is well-founded. □

Corollary 3.9 *Every acceptable program is left terminating.*

Proof. Every ground goal is bounded. □

LD-trees

To prove the converse of Corollary 3.9 we analyze the size of finite LD-trees. To this end we need the following lemma, where $nodes_P(G)$ for a program P and a goal G denotes the number of nodes in the LD-tree for $P \cup \{G\}$.

Lemma 3.10 (LD-**tree**) *Let P be a program and G a goal such that the LD-tree for $P \cup \{G\}$ is finite. Then*

(i) for all substitutions θ, $nodes_P(G\theta) \leq nodes_P(G)$,

(ii) for all prefixes H of G, $nodes_P(H) \leq nodes_P(G)$,

(iii) for all non-root nodes H in the LD-tree for $P \cup \{G\}$, $nodes_P(H) < nodes_P(G)$.

Proof. (i) By an application of a variant of the Lifting Lemma (see e.g. Lloyd [Llo87]) to LD-derivations we conclude that to every LD-derivation of $P \cup \{G\theta\}$ with input clauses $C_1, C_2, \ldots$, there corresponds an LD-derivation of $P \cup \{G\}$ with input clauses $C_1, C_2, \ldots$ of the same of larger length. This implies the claim.
(ii) Consider a prefix $H = \leftarrow A_1, \ldots, A_k$ of $G = \leftarrow A_1, \ldots, A_n$ $(n \geq k)$. By an appropriate renaming of variables (formally justified by the Variant Lemma 2.8 in Apt [Apt88]) we can assume that all input clauses used in the LD-tree for $P \cup \{H\}$ have no variables in common with G. We can now transform the LD-tree for $P \cup \{H\}$ into an initial subtree of the LD-tree

for $P \cup \{G\}$ by replacing in it a node $\leftarrow B_1, \ldots, B_l$ by $\leftarrow B_1, \ldots, B_l, A_{k+1}\theta, \ldots, A_n\theta$, where θ is the composition of the mgu's used on the path from the root H to the node $\leftarrow B_1, \ldots, B_l$. This implies the claim.
(iii) Immediate by the definition. □

As stated at the beginning of Section 2, we are interested in proving not only left termination of a program, but also its termination for a class of non-ground goals. We now show that the concepts of acceptability and boundedness provide us with a complete method for proving both properties.

Theorem 3.11 *Let P be a left terminating program. Then for some level mapping $|\ |$ and a model I of P*

(i) P is acceptable w.r.t. $|\ |$ and I,

(ii) for every goal G, G is bounded w.r.t. $|\ |$ and I iff all LD-derivations of $P \cup \{G\}$ are finite.

Proof. Define the level mapping by putting for $A \in B_P$

$$|A| = nodes_P(\leftarrow A).$$

Since P is left terminating, this level mapping is well defined. Next, choose

$$I = \{A \in B_P \mid \text{there is an } LD\text{-refutation of } P \cup \{\leftarrow A\}\}.$$

By the strong completeness of SLD-resolution, $I = M_P$, so I is a model of P.

First we prove one implication of (ii).

(ii1) Consider a goal G such that all LD-derivations of $P \cup \{G\}$ are finite. We prove that G is bounded by $nodes_P(G)$ w.r.t. $|\ |$ and I.

To this end take $\ell \in \cup|[G]|_I$. For some ground instance $\leftarrow A_1, \ldots, A_n$ of G and $i \in [1, \bar{n}]$, where

$$\bar{n} = \min(\{n\} \cup \{i \in [1, n] \mid I \not\models A_i\}),$$

we have $\ell = |A_i|$. We now calculate

$$\begin{array}{ll} & nodes_P(G) \\ \geq & \quad \{\text{Lemma 3.10 (i)}\} \\ & nodes_P(\leftarrow A_1, \ldots, A_n) \\ \geq & \quad \{\text{Lemma 3.10 (ii)}\} \\ & nodes_P(\leftarrow A_1, \ldots, A_{\bar{n}}) \\ \geq & \quad \{\text{Lemma 3.10 (iii), noting that for } j \in [1, \bar{n}-1] \\ & \quad \text{there is an } LD\text{-refutation of } P \cup \{\leftarrow A_1, \ldots, A_j\}\} \\ & nodes_P(\leftarrow A_i, \ldots, A_{\bar{n}}) \\ \geq & \quad \{\text{Lemma 3.10 (ii)}\} \\ & nodes_P(\leftarrow A_i) \\ = & \quad \{\text{definition of } |\ |\} \\ & |A_i| \\ = & \ \ell. \end{array}$$

(i) We now prove that P is acceptable w.r.t. $|\ |$ and I. Take a clause $A \leftarrow B_1, \ldots, B_n$ in P and its ground instance $A\theta \leftarrow B_1\theta, \ldots, B_n\theta$. We need to show that

$$|A\theta| > |B_i\theta| \text{ for } i \in [1, \bar{n}],$$

where

$$\bar{n} = \min(\{n\} \cup \{i \in [1,n] \mid I \not\models B_i\theta\}).$$

We have $A\theta\theta \equiv A\theta$, so $A\theta$ and A unify. Let $\mu = \mathrm{mgu}(A\theta, A)$. Then $\theta = \mu\delta$ for some δ. By the definition of LD-resolution, $\leftarrow B_1\mu, \ldots, B_n\mu$ is an LD-resolvent of $\leftarrow A\theta$.

Then for $i \in [1, \bar{n}]$

$$\begin{array}{ll} & |A\theta| \\ = & \quad \{\text{definition of } |\ |\} \\ & nodes_P(\leftarrow A\theta) \\ > & \quad \{\text{Lemma 3.10 (iii), } \leftarrow B_1\mu, \ldots, B_n\mu \text{ is a resolvent of } \leftarrow A\theta\} \\ & nodes_P(\leftarrow B_1\mu, \ldots, B_n\mu) \\ \geq & \quad \{\text{part (ii1), noting that } B_i\theta \in \cup|[\leftarrow B_1\mu, \ldots, B_n\mu]|_I\} \\ & |B_i\theta|. \end{array}$$

(ii2) Consider a goal G which is bounded w.r.t. $|\ |$ and I. Then by (i) and Corollary 3.8 all LD-derivations of $P \cup \{G\}$ are finite. □

Corollary 3.12 A program is left terminating iff it is acceptable.

Proof. By Corollary 3.9 and Theorem 3.11. □

4 Applications

The equivalence between the left terminating and acceptable programs provides us with a method of proving termination of Prolog programs. The level mapping and the model used in the proof of Theorem 3.11 were quite involved and relied on elaborate information about the program at hand which is usually not readily available. However, in practical situations much simpler constructions suffice. The level mapping can be usually defined as a simple function of the terms of the ground atom and the model takes into account only some straightforward information about the program. We illustrate it by means of three examples.

First, we define by structural induction a function $|\ |$ on ground terms by putting:

$$\begin{array}{l} |[x|xs]| = |xs| + 1, \\ |f(x_1, \ldots, x_n)| = 0 \text{ if } f \neq [\,.\,|\,.\,]. \end{array}$$

It is useful to note that for a list xs, $|xs|$ equals its length. The function $|\ |$ is called *listsize* in Ullman and Van Gelder [UvG88]. It will be used in the examples below.

Quicksort

Consider the following program QS (for quicksort):

```
(qs1)  qs([], []) ←.
(qs2)  qs([X | Xs], Ys) ←
         f(X, Xs, X1s, X2s),
         qs(X1s, Y1s),
         qs(X2s, Y2s),
         a(Y1s, [X | Y2s], Ys).

(f1)   f(X, [], [], []) ←.
(f2)   f(X, [Y | Xs], [Y | Y1s], Y2s) ←
         X > Y,
         f(X, Xs, Y1s, Y2s).
(f3)   f(X, [Y | Xs], Y1s, [Y | Y2s]) ←
         X ≤ Y,
         f(X, Xs, Y1s, Y2s).

(a1)   a([], Ys, Ys) ←.
(a2)   a([X | Xs], Ys, [X | Zs]) ←
         a(Xs, Ys, Zs).
```

We assume that QS operates on the domain of natural numbers over which the builtin relations $>$ and $\leq$, written in infix notation, are defined. This domain can be incorporated into the Herbrand universe of QS by adding to the language of QS the constant 0 and the successor function s (for example by adding to QS the clause `s(0) > 0` ←.).

Denote now the program consisting of the clauses $(f_1), (f_2), (f_3)$ by `filter`, and the program consisting of the clauses $(a_1), (a_2)$ by append.

Lemma 4.1 `filter` *is recurrent with* $|f(x, xs, x1s, x2s)| = |xs|$. □

We adopted here the simplifying assumption that builtins $>$ and $\leq$ are recurrent with the level mapping $|s > t| = 0$ and $|s \leq t| = 0$.

Lemma 4.2 append *is recurrent with* $|a(xs, ys, zs)| = |xs|$. □

Lemma 4.3 QS *is not recurrent.*

Proof. Consider clause (qs_2) instantiated with the ground substitution

$$\{X/a,\ Xs/b,\ Ys/c,\ X1s/[a|b],\ Y1s/c\}.$$

Then the ground atom $qs([a|b], c)$ appears both in the head and the body of the resulting clause. □

To prove that QS is left terminating we show that it is acceptable. We define an appropriate level mapping $|\ |$ by extending the ones given in Lemma's 4.1 and 4.2 with

$$|qs(xs, ys)| = |xs|.$$

Next, we define a Herbrand interpretation of QS by putting

$$\begin{array}{rcl} I & = & \{qs(xs,ys) \mid |xs| = |ys|\} \\ & \cup & \{f(x,xs,y1s,y2s) \mid |xs| = |y1s| + |y2s|\} \\ & \cup & \{a(xs,ys,zs) \mid |xs| + |ys| = |zs|\} \\ & \cup & [X > Y] \\ & \cup & [X \leq Y]. \end{array}$$

Recall that $[A]$ for an atom A stands for the set of all ground instances A.

Lemma 4.4 *I is a model of* QS.

Proof. First, note that $|[]|+|ys| = |ys|$ and that $|xs|+|ys| = |zs|$ implies $|[x|xs]|+|ys| = |[x|zs]|$. This implies that I is a model of append.

Next, note that $|[]|+|[]| = |[]|$ and that $|xs| = |y1s|+|y2s|$ implies $|[y|xs]| = |[y|y1s]|+|y2s|$ and $|[y|xs]| = |y1s|+|[y|y2s]|$. This implies that I is a model of `filter`.

Finally, note that $|[\,]| = |[\,]|$ and that $|xs| = |x1s|+|x2s|$, $|x1s| = |y1s|$, $|x2s| = |y2s|$ and $|y1s|+|[x|y2s]| = |ys|$ imply $|[x|xs]| = |ys|$. This implies that I is a model of QS. □

We now prove the desired result.

Theorem 4.5 QS *is acceptable w.r.t.* $|\ |$ *and* I.

Proof. As `filter` and append are recurrent w.r.t. $|\ |$, we only need to consider clauses (qs_1) and (qs_2). (qs_1) satisfies the appropriate requirement voidly.

Consider now a ground instance C of (qs_2). C is of the form $A \leftarrow B_1, B_2, B_3, B_4$. We now prove three facts which obviously imply that C satisfies the appropriate requirement.

Fact 1 $|A| > |B_1|$.

Proof. Note that

$$|qs([x|xs],ys)|| = |[x|xs]| > |xs| = |f(x,xs,x1s,x2s)|.$$

□

Fact 2 Suppose $I \models B_1$. Then $|A| > |B_2|$ and $|A| > |B_3|$.

Proof. By assumption $|xs| = |x1s| + |x2s|$, so

$$|qs([x|xs],ys)| > |xs| \geq |x1s| = |qs(x1s,y1s)|$$

and analogously

$$|qs([x|xs],ys)| > |qs(x2s,y2s)|.$$

□

Fact 3 Suppose $I \models B_1$ and $I \models B_2$. Then $|A| > |B_4|$.

Proof. By Fact 2 $|qs([x|xs],ys)| > |qs(x1s,y1s)| = |x1s|$ and by assumption $|x1s| = |y1s|$, so

$$|qs([x|xs],ys)| > |y1s| = |a(y1s,[x|y2s],ys)|.$$

□
□

So far we only proved that QS is left terminating. We now prove that it terminates for a large class of goals.

Lemma 4.6 *For all terms $t, t_1, \ldots, t_k$, $k \geq 0$, a goal of the form*

$$\leftarrow qs([t_1, \ldots, t_k], t)$$

is bounded w.r.t. $||$ and I.

Proof. Let A be a ground instance of $qs([t_1, \ldots, t_k], t)$. Then $|A| = |[t_1, \ldots, t_k]| = k$, so $| \leftarrow A|_I = \text{bag}(k)$. Hence $\leftarrow qs([t_1, \ldots, t_k], t)$ is bounded by k w.r.t. $||$ and I. □

It is worth noting that every "ill typed" goal $\leftarrow qs(s, t)$, where s is a non-variable, non-list term is also bounded w.r.t. $||$ and I, as $|s'| = 0$ for every ground instance s' of s.

Corollary 4.7 *For all terms $t, t_1, \ldots, t_k$, $k \geq 0$, all LD-derivations of* QS $\cup\{\leftarrow qs([t_1, \ldots, t_k], t)\}$ *are finite.*

Proof. By Corollary 3.8. □

Permutation

Consider now the following program PERM (for permutation) studied in Plümer [Plü90b] :

```
(p1)  p([], []).
(p2)  p(Xs, [X | Ys]) :-
        a(X1s, [X|X2s], Xs),
        a(X1s, X2s, Zs),
        p(Zs, Ys).
```

augmented by the clauses (a_1) and (a_2) which form the append program defining the relation a.

The intention is to invoke p with its first argument instantiated. Clause (p_1) states that the empty list is a permutation of itself. Clause (p_2) takes care of a non-empty list xs - one should first split it into two sublists $x1s$ and $[x|x2s]$ and concatenate $x1s$ and $x2s$ to get zs. If now ys is a permutation of zs, $[x|ys]$ is a permutation of xs.

Lemma 4.8 PERM *is not recurrent.*

Proof. By Theorem 2.8 of Bezem [Bez89] every recurrent program P is terminating, which means that all SLD-derivations of P starting with a ground goal are finite. But the SLD-derivation of PERM $\cup\{\leftarrow p(xs, [x|ys])\}$ with xs, x, ys ground, in whose second goal the middle atom $a(x1s, x2s, zs)$ is selected, diverges when clause (a_2) is repeatedly used. Thus PERM is not terminating and so it is not recurrent. □

We now prove that PERM is acceptable. First, we define a level mapping by putting

$$|p(zs, ys)| = |zs| + 1,$$
$$|a(x1s, x2s, zs)| = \min(|x1s|, |zs|).$$

Next, we define a Herbrand interpretation I by putting

$$I = [p(Zs, Ys)] \cup \{a(x1s, x2s, zs) \mid |x1s| + |x2s| = |zs|\}.$$

Lemma 4.9 *I is a model of* PERM.

Proof. I is trivially a model of (p_1) and (p_2). In the proof of Lemma 4.4 we showed that I is also a model of append. □

We can now prove the desired result.

Theorem 4.10 PERM *is acceptable w.r.t. $||$ and I.*

Proof. It is easy to see that append is recurrent w.r.t. $||$, so we only need to consider clause (p_2). Let $C = A \leftarrow B_1, B_2, B_3$ be a ground instance of (p_2). The required condition for C is implied by the following three facts.

Fact 1 $|A| > |B_1|$.

Proof. Note that

$$|p(xs, [x|ys])| = |xs| + 1 > |xs| \geq \min(|x1s|, |xs|) = |a(x1s, [x|x2s], xs)|.$$

□

Fact 2 Suppose $I \models B_1$. Then $|A| > |B_2|$.

Proof. By assumption $|x1s| + |[x|x2s]| = |xs|$, so

$$|p(xs, [x|ys])| = |xs| + 1 > |x1s| \geq \min(|x1s|, |zs|) = |a(x1s, x2s, zs)|.$$

□

Fact 3 Suppose $I \models B_1$ and $I \models B_2$. Then $|A| > |B_3|$.

Proof. By assumption $|x1s| + |[x|x2s]| = |xs|$ and $|x1s| + |x2s| = |zs|$, so

$$|p(xs, [x|ys])| = |xs| + 1 > |xs| = |x1s| + |x2s| + 1 = |zs| + 1 = |p(zs, ys)|.$$

□
□

Also, we have the following.

Lemma 4.11 *For all terms $t, t_1, \ldots, t_k$, $k \geq 0$, a goal of the form*

$$\leftarrow p([t_1, \ldots, t_k], t)$$

is bounded w.r.t. $||$ and I.

Proof. The same as that of Lemma 4.6. □

Corollary 4.12 *For all terms $t, t_1, \ldots, t_k$, $k \geq 0$, all LD-derivations of* PERM $\cup\{\leftarrow p([t_1, \ldots, t_k], t)\}$ *are finite.*

Proof. By Corollary 3.8. □

It is useful to note that we had to use here for **append** a different level mapping than the one used in the proof of acceptability of QS. With the original level mapping for **append**, PERM is not acceptable w.r.t. any model. Indeed, consider a ground instance A of the head of (p_2). Let $C = A \leftarrow B_1, B_2, B_3$ be a ground instance of (p_2) in which the variable $X1s$ is instantiated to some ground term t with $|t| = |A|$. Then with the original level mapping for **append** we have $|A| = |t| = |B_1|$.

In contrast, the level mapping for **append** used in Theorem 4.10 can also be used in the proof of acceptability of QS.

Mergesort

Finally, consider the following program MS (for mergesort) taken from Ullman and Van Gelder [UvG88]:

```
(ms1)  ms([], []) ←.
(ms2)  ms([X], [X]) ←.
(ms3)  ms([X | [Y | Xs]], Ys) ←
         s([X | [Y | Xs]], X1s, X2s),
         ms(X1s, Y1s),
         ms(X2s, Y2s),
         m(Y1s, Y2s, Ys).

(s1)   s([], [], []) ←.
(s2)   s([X | Xs], [X | Ys], Zs) ←
         s(Xs, Zs, Ys).

(m1)   m([], Xs, Xs) ←.
(m2)   m(Xs, [], Xs) ←.
(m3)   m([X | Xs], [Y | Ys], [X | Zs]) ←
         X ≤ Y,
         m(Xs,[Y | Ys], Zs).
(m4)   m([X | Xs], [Y | Ys], [Y | Zs]) ←
         X > Y,
         m([X | Xs], Ys, Zs).
```

We assume that MS operates on the same domain as QS. The intention is to invoke ms with its first argument being an unsorted list. Clause (ms_3) takes care of non-empty list of length at least 2. The idea is first to split the input list in two lists of roughly equal length (note the reversed order of parameters in the recursive call of s), then mergesort each sublist and finally merge the resulting sorted sublists.

Denote the program consisting of the clauses $(s_1), (s_2)$ by **split**, and the program consisting of the clauses $(m_1), (m_2), (m_3), (m_4)$ by **merge**.

Lemma 4.13 split *is recurrent with* $|s(xs, x1s, x2s)| = |xs|$. □

Lemma 4.14 merge *is recurrent with* $|m(xs, ys, zs)| = |xs| + |ys|$. □

Lemma 4.15 MS *is not recurrent.*

Proof. Analogous to that of Lemma 4.3. □

We now show that MS is acceptable. We define an appropriate level mapping $||$ by extending the ones given in Lemma's 4.13 and 4.14 with

$$|ms(xs, ys)| = |xs| + 1.$$

Next, we define a Herbrand interpretation of MS by putting

$$\begin{array}{rcl} I & = & \{ms(xs, ys) \mid |xs| = |ys|\} \\ & \cup & \{s(xs, y1s, y2s) \mid |y1s| = \lceil |xs|/2 \rceil, |y2s| = \lfloor |xs|/2 \rfloor\} \\ & \cup & \{m(xs, ys, zs) \mid |xs| + |ys| = |zs|\} \\ & \cup & [X > Y] \\ & \cup & [X \leq Y]. \end{array}$$

Lemma 4.16 I *is a model of* MS.

Proof. First, note that $|[\,]| + |xs| = |xs|$, $|xs| + |[\,]| = |xs|$, $|xs| + |[y|ys]| = |zs|$ implies $|[x|xs]| + |[y|ys]| = |[x|zs]|$, and that $|[x|xs]| + |ys| = |zs|$ implies $|[x|xs]| + |[y|ys]| = |[y|zs]|$. This implies that I is a model of **merge**.

Next, note that $|[\,]| = |[\,]|$ and $|[x]| = |[x]|$ imply that I is a model of (ms_1) and (ms_2). Moreover, $|x1s| = \lceil |[x|[y|xs]]|/2 \rceil$ and $|x2s| = \lfloor |[x|[y|xs]]|/2 \rfloor$ imply $|[x|[y|xs]]| = |x1s| + |x2s|$, which, together with $|x1s| = |y1s|, |x2s| = |y1s|$ and $|y1s| + |y2s| = |ys|$, imply $|[x|[y|xs]]| = |ys|$. This implies that I is a model of (ms_3).

Next, note that $|[\,]| = \lceil |[\,]|/2 \rceil$ and $|[\,]| = \lfloor |[\,]|/2 \rfloor$ imply that I is a model of (s_1). Finally, to see that I is a model of (s_2), consider an atom $s(xs, zs, ys) \in I$. The following two cases arise.
Case 1 $|xs| = 2k, k \geq 0$. By assumption, $|zs| = k$ and $|ys| = k$. This implies $|[x|ys]| = k + 1 = \lceil |[x|xs]|/2 \rceil$ and $|zs| = k = \lfloor |[x|xs]|/2 \rfloor$.
Case 2 $|xs| = 2k + 1, k \geq 0$. By assumption, $|zs| = k + 1$ and $|ys| = k$. This implies $|[x|ys]| = k + 1 = \lceil |[x|xs]|/2 \rceil$ and $|zs| = k + 1 = \lfloor |[x|xs]|/2 \rfloor$.
In both cases we conclude that $s([x|xs], [x|ys], zs) \in I$, i.e. I is a model of (s_2). □

We now prove the desired result.

Theorem 4.17 MS *is acceptable w.r.t.* $||$ *and* I.

Proof. As split and merge are recurrent w.r.t. $|\ |$, we only need to consider clauses (ms_1), (ms_2) and (ms_3). (ms_1) and (ms_2) satisfy the appropriate requirement voidly.

Consider now a ground instance $C = \leftarrow B_1, B_2, B_3, B_4$ of (ms_3). We prove three facts which imply that C satisfies the appropriate requirement.

Fact 1 $|A| > |B_1|$.

Proof. Note that

$$|ms([x|[y|xs]], ys)]| = |[x|[y|xs]]| + 1 > |[x|[y|xs]]| = |s([x|[y|xs]], x1s, x2s)|.$$

□

Fact 2 Suppose $I \models B_1$. Then $|A| > |B_2|$ and $|A| > |B_3|$.

Proof. By assumption $|x1s| = \lceil |[x|[y|xs]]|/2 \rceil$ and $|x2s| = \lfloor |[x|[y|xs]]|/2 \rfloor$, which implies $|[x|[y|xs]]| > |x1s|$ and $|[x|[y|xs]]| > |x2s|$, as $|[x|[y|xs]]| > 1$. Hence

$$|ms([x|[y|xs]], ys)| = |[x|[y|xs]]| + 1 > |x1s| + 1 = |ms(x1s, y1s)|$$

and analogously

$$|ms([x|[y|xs]], ys)| > |ms(x2s, y2s)|.$$

□

Fact 3 Suppose $I \models B_1$, $I \models B_2$ and $I \models B_3$. Then $|A| > |B_4|$.

Proof. By assumption $|ms([x|[y|xs]], ys)| > |[x|[y|xs]]| = |x1s| + |x2s|$ and $|x1s| = |y1s|$, $|x2s| = |y2s|$, so

$$|ms([x|[y|xs]], ys)| > |y1s| + |y2s| = |m(y1s, y2s, ys)|.$$

□

□

Additionally, we have the following.

Lemma 4.18 *For all terms $t, t_1, \ldots, t_k$, $k \geq 0$, a goal of the form*

$$\leftarrow ms([t_1, \ldots, t_k], t)$$

is bounded w.r.t. $|\ |$ and I.

Proof. The same as that of Lemma 4.6. □

Corollary 4.19 *For all terms $t, t_1, \ldots, t_k$, $k \geq 0$, all LD-derivations of* MS $\cup \{\leftarrow ms([t_1, \ldots, t_k], t)\}$ *are finite.*

Proof. By Corollary 3.8. □

5 Conclusions

Assessment of the method

Our approach to termination is limited to the study of left terminating programs, so it is useful to reflect how general this class of programs is. The main result of Bezem [Bez89] states that every total recursive function can be computed by a recurrent program. As recurrent programs are left terminating, the same property is shared by left terminating programs.

For a further analysis of left terminating programs we first introduce the following notions, essentially due to Dembinski and Maluszynski [DM85]. We follow here the presentation of Plümer [Plü90a]. Given an n-ary relation symbol p, by a *mode* for p we mean a function d_p from $\{1,\ldots,n\}$ to the set $\{+,-\}$. We write d_p in a more suggestive form $p(d_p(1),\ldots,d_p(n))$.

Modes indicate how the arguments of a relation should be used. If $d_p(i) = '+'$, we call i the *input position* of p and if $d_p(i) = '-'$, we call i the *output position* of p (both w.r.t. d_p) . The input positions should be replaced by ground terms and the output positions by variables. This motivates the following notion.

Given a mode d_p for a relation p, we say that an atom $A = p(t_1,\ldots,t_n)$ *respects* d_p if for $i \in [1,n]$, t_i is ground if i is an input position of p w.r.t. d_p and t_i is a variable if i is an output position of p w.r.t. d_p.

A *mode* for a program P is a function which assigns to each relation symbol of P a non-empty set of modes. Given a mode for a program P, we say that an atom A *respects moding* if A respects some mode in the set of modes associated with the relation p used in A.

As an example consider the mode for the program append represented by the following set:

$$\{\mathtt{append}(+,+,-),\ \ \mathtt{append}(-,-,+)\}.$$

It indicates that append should be called either with its first two arguments ground and the third being a variable, or with its first two arguments being a variable and the third argument ground. Then any atom $append(xs,ys,zs)$, where either xs,ys are ground and zs is a variable, or xs,ys are variables and zs is ground, respects moding.

The following simple theorem shows that the property of left termination is quite natural.

Theorem 5.1 *Let P be a program with a mode such that for all atoms A which respect moding, all LD-derivations of $P \cup \{\leftarrow A\}$ are finite. Then P is left terminating.*

Proof. Consider a ground atom A. A is a ground instance of some atom B which respects moding. By a variant of the Lifting Lemma applied to the LD-resolution we conclude that all LD-derivations of $P \cup \{\leftarrow A\}$ are finite. This implies that P is left terminating. □

The assumptions of the above theorem are satisfied by an overwhelming class of Prolog programs.

As Theorem 3.11 shows, the method presented in this paper is a complete method for proving termination of Prolog programs. We believe that it is also a useful method, since it allows us to factore termination proofs into simpler, separate proofs, which consist of checking the guesses for the level mapping $|\ |$ and the model I. Moreover, the method is modular, because termination proofs provided for subprograms can be reused in later proofs.

In this paper, the method is used as an "a posteriori" technique for verifying termination of existing Prolog programs. However, it could also provide a guideline for the program de-

velopment, if the program is constructed together with its termination proof. A specific level mapping and a model could suggest, in particular, a specific ordering of atoms in clause bodies.

It is worth noting that some fragments of the proof of accceptability can be automated, at least in the case of the applications presented in Section 4. In our examples, where the function *listsize* is used, the task of checking the guesses for both the level mapping $|\ |$ and the model I can be reduced to checking the validity of universal formulas in Presburger arithmetic, which is a decidable theory. To illustrate this point, consider the following guess I for a model for the program PERM:

$$\begin{aligned} I \;=\; & \{p(zs, ys) \mid |zs| = |ys|\} \\ \cup\; & \{a(x1s, x2s, zs) \mid |x1s| + |x2s| = |zs|\}. \end{aligned}$$

To show that I is a model of, say, clause (p_2), we have to prove the following implication:

$$\{a(x1s, [x|x2s], xs), a(x1s, x2s, zs), p(zs, ys)\} \subseteq I \Rightarrow p(xs, [x|ys]) \in I.$$

By homomorphically mapping lists onto their lengths, i.e. by mapping $[\,]$ to 0 and $[:\ |\ .]$ to the successor function $s(.)$, we get the following formula of Presburger arithmetic:

$$n_1 + n_2 + 1 = n \ \wedge\ n_1 + n_2 = k \ \wedge\ k = m \Rightarrow n = m + 1$$

where $n_1 = |x1s|, n_2 = |x2s|, n = |xs|, k = |zs|, m = |ys|$.

Analogous considerations apply to the verification of the level mapping.

Finally, it is useful to notice a simple consequence of our approach to termination. By proving that a program P is acceptable and a goal G is bounded, we can conclude by Corollary 3.8 that the LD-tree for $P \cup \{G\}$ is finite. Thus, for the leftmost selection rule, the set of computed answer substitutions for $P \cup \{G\}$ is finite and consequently, by virtue of the strong completeness of SLD-resolution, we can use the LD-resolution to compute the set of all correct answer substitutions for $P \cup \{G\}$. In other words, query evaluation of bounded goals can be implemented using pure Prolog.

Related work

Of course the subject of termination of Prolog programs has been studied by others. Without aiming at completeness we mention here the following related work.

Vasak and Potter [VP86] identified two forms of termination for logic programs – existential and universal one and characterized the class of universal terminating goals for a given program with selected selection rules. However, this characterization cannot be easily used to prove termination. Using our terminology, given a program P, a goal G is universally terminating w.r.t. a selection rule R if the SLD-tree for $P \cup \{G\}$ via R is finite.

Baudinet [Bau88] presented a method for proving termination of Prolog program in which with each program a system of equations is associated whose least fixpoint is the meaning of the program. By analyzing this least fixpoint various termination properties can be proved. The main method of reasoning is fixpoint or structural induction.

Ullman and Van Gelder [UvG88] considered the problem of automatic verification of termination of a Prolog program and a goal. In their approach first some sufficient set of inequalities between the sizes of the arguments of the relation symbols are generated, and then it is verified if they indeed hold. Termination of the programs studied in the previous section is beyond the scope of their method.

This approach was improved in Plümer [Plü90b], [Plü90a], who allowed a more general form of the inequalities and the way sizes of the arguments are measured. This resulted in a more powerful method. Both the quicksort and the permutation programs studied in the previous section can be handled using Plümer's method. However, the mergesort remains beyond its scope.

Deville [Dev90] also considers termination in his proposal of systematic program development. In his framework, termination proofs exploit well-founded orderings together with mode and multiplicity informa-tion, the latter representing an upper bound to the number of answer substitutions for goals which respect a given mode. For instance, a termination proof of the program DC of Example 3.1(iii) for the goal $\leftarrow dc(x, Y)$ would involve verification of the following statements (assuming that x is a ground term):

1. the goal $\leftarrow divide(x, X1, X2)$ respects moding, and both $X1$ and $X2$ are bound to ground terms, $x1$ and $x2$ respectively, by any computed answer substitution for such a goal;

2. both $x1$ and $x2$ are smaller than x w.r.t. some well-founded ordering;

3. the mode $divide(+, -, -)$ has a finite multiplicity.

Our approach seems to be simpler as it relies on fewer concepts. Also, it suggests a more uniform methodology. On the other hand, in Deville's approach more information about the program is obtained.

References

[Apt88] K. R. Apt. Introduction to logic programming. Technical Report CS-R8826, Centre for Mathematics and Computer Science, 1988. To appear in "Handbook of Theoretical Computer Science", North Holland (J. van Leeuwen, ed.).

[AvE82] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *J. ACM*, 29(3):841–862, 1982.

[Bau88] M. Baudinet. Proving termination properties of PROLOG programs. In *Proceedings of the 3rd Annual Symposium on Logic in Computer Science (LICS)*, pages 336–347, Edinburgh, Scotland, 1988.

[Bez89] M. Bezem. Characterizing termination of logic programs with level mappings. In E. L. Lusk and R. A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 69–80. The MIT Press, 1989.

[Bla86] H. A. Blair. Decidability in the Herbrand Base. Manuscript (presented at the Workshop on Foundations of Deductive Databases and Logic Programming, Washington D.C., August 1986), 1986.

[Cav89] L. Cavedon. Continuity, consistency, and completeness properties for logic programs. In G. Levi and M. Martelli, editors, *Proceedings of the Sixth International Conference on Logic Programming*, pages 571–584. The MIT Press, 1989.

[Dev90] Y. Deville. *Logic Programming. Systematic Program Development.* International Series in Logic Programming. Addison-Wesley, 1990.

[DM85] P. Dembinski and J. Maluszynski. AND-parallelism with intelligent backtracking for annotated logic programs. In *Proceedings of the International Symposium on Logic Programming*, pages 29–38, Boston, 1985.

[Fit85] M. Fitting. A Kripke-Kleene semantics for general logic programs. *Journal of Logic Programming*, 2:295–312, 1985.

[Flo67] R. W. Floyd. Assigning meanings to programs. In *Proceedings Symposium on Applied Mathematics, 19, Math. Aspects in Computer Science*, pages 19–32. American Society, 1967.

[Kle52] S. C. Kleene. *Introduction to Metamathematics.* van Nostrand, New York, 1952.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, second edition, 1987.

[Plü90a] L. Plümer. *Termination Proofs for Logic Programs.* Lecture Notes in Artificial Intelligence 446, Springer-Verlag, Berlin, 1990.

[Plü90b] L. Plümer. Termination proofs for logic programs based on predicate inequalities. In D. H. D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference on Logic Programming*, pages 634–648. The MIT Press, 1990.

[UvG88] J. D. Ullman and A. van Gelder. Efficient tests for top-down termination of logical rules. *J. ACM*, 35(2):345–373, 1988.

[VP86] T. Vasak and J. Potter. Characterization of terminating logic programs. In *Proceedings of the 1986 IEEE Symposium on Logic Programming*, 1986.