On Modular Termination Proofs

Jonathan C. Martin and Andy King

Technical Report No: 13-03
Date: October 2003

Published by the Computing Laboratory,
University of Kent, Canterbury, Kent CT2 7NF, UK.

Abstract

Reasoning about termination is a key issue in logic program development. One classic technique for proving termination is to construct a well-founded order on goals that decreases between successive goals in a derivation. In practise, this is achieved with the aid of a level mapping that maps atoms to natural numbers. This paper examines why it can be difficult to base termination proofs on natural level mappings that directly relate to the recursive structure of the program. The notions of bounded-recurrency and bounded-acceptability are introduced to alleviate these problems. These concepts are equivalent to the classic notions of recurrency and acceptability respectively, yet provide practical criteria for constructing termination proofs in terms of natural level mappings. Moreover, the construction is entirely modular in that termination conditions are derived in a bottom-up fashion by considering, in turn, each the strongly connected components of the program.

1 Introduction

The classes of recurrent and acceptable programs are, arguably, two of the most influential classes of logic program that occur in the termination literature. Acceptable programs are precisely those which, for ground input, terminate under the left-to-right selection rule of Prolog [2]. Programs which, for ground input, terminate under *any* selection rule are classified as being recurrent [5].

Whilst the notions of recurrency and acceptability provide a sound theoretical basis for studying termination, they do not provide much insight into the practicalities of deriving the level mappings which are needed to prove that a logic program is terminating or left-terminating. Instead, intuition has served as the guide in the development of automatic techniques. In particular, there has been a desire to derive natural level mappings based on the recursive structure of the program at hand. For example, given the program

$\mathsf{p}([\mathsf{H}|\mathsf{T}]) \leftarrow \mathsf{p}(\mathsf{T}).$

it is natural to define a level mapping $|.|$ to prove termination by $|\mathsf{p}(x)| = |x|_{\text{length}}$ where $|.|_{\text{length}}$ is the list-length norm because the predicate is inductively defined over the length of its argument which is a list. Other definitions, such as $|\mathsf{p}(x)| = |x|_{\text{length}} + 1$ and $|\mathsf{p}(x)| = 2|x|_{\text{length}}$ do not possess the same natural correspondence with the termination behaviour of the program.

This paper examines the reasons why termination proofs based on recurrency and acceptability are often difficult to obtain. The observations are not new in themselves [3, 6, 12, 18] and, by way of a solution, Apt and Pedreschi [3] define alternative characterisations of terminating and left-terminating programs which they call semi-recurrency and semi-acceptability respectively. This paper argues that these concepts do not, in fact, form an ideal basis for automatic termination analyses (though this approach has been followed by others [24, 27]); some difficulties complicate the construction of the level mapping that arises in the termination proof. To alleviate these problems, this paper introduces notions of bounded-recurrency and bounded-acceptability and shows that these concepts are equivalent to recurrency and acceptability respectively. These new characterisations of the two classes provide practical criteria for constructing termination proofs in terms of natural level mappings. The construction is entirely modular: termination conditions are derived in a bottom-up fashion by considering, in turn, each the strongly connected components of the predicate dependency graph. For program specialisation, and partial deduction in particular, this tactic is particularly promising since the overall computation is unlikely to be terminating but some sub-computations probably will be. More exactly, it is more useful to derive sufficient termination conditions for individual predicates rather than proving that a given top-level goal will terminate [10]. The notion of bounded-acceptability lends itself naturally to this process. Moreover, there has been much recent interest in the inference of level mappings [9, 15, 19, 20, 23, 26] in order to fully automate termination analysis. Thus the desire for natural level mappings is much more than an aesthetic predilection.

The paper is structured as follows. Section 2 introduces the concepts necessary for discussing termination, and in particular reviews the notions of recurrency and acceptability. Section 3 argues that level mappings have traditionally been overloaded in that they address two different termination issues. Sections 4 and 5 reviews the concepts of semi-recurrency and semi-acceptability, arguing that these notions also lead to artificial level mappings. Sections 6 and 7 explain how the concepts of bounded-recurrency and bounded-acceptability permit simpler, more natural level mappings to be used within termination proofs. Section 8 presents the concluding discussion, reflecting on other approaches to modularity [6, 18].

2 Preliminaries: the nuts and bolts of termination

2.1 Level mappings, norms and boundedness

The fundamental idea underlying all termination proofs is to define an order on the goals that can occur within a derivation. Given a program P and goal G_0, the finiteness of derivation $G_0, G_1, G_2, \ldots$ is in principle straightforward to demonstrate: it is sufficient to construct a well-founded order $<$ such that $G_{i+1} < G_i$ for all $i \geq 0$. The problem is to find such an order. To simplify the problem, it is convenient to define the order on abstractions of goals rather than on the goals themselves. Thus the order $<$ is defined such that $G' < G$ holds iff $\mathcal{A}(G') < \mathcal{A}(G)$ holds where $\mathcal{A}$ is an abstraction function. For example, $\mathcal{A}$ might be defined to map each goal G to a multiset of natural numbers, where each atom in G maps to a single number in the multiset. The idea of abstracting goals by mapping atoms to natural numbers leads to the concept of a level mapping.

Definition 2.1 (level mapping [11]) Let P be a program. A level mapping for P is a function $|.| : B_P \mapsto \mathbb{N}$ from the Herbrand base of P to the set of natural numbers $\mathbb{N}$. For an atom $A \in B_P$, $|A|$ denotes the level of A. $\square$

Example 2.1 Let P be the program

$\mathsf{p(a,\ X)} \leftarrow \mathsf{p(b,\ X)}.$
$\mathsf{p(b,\ a)}.$
$\mathsf{p(b,\ b)}.$

The function $|.| : \{\mathsf{p(a,\ a)}, \mathsf{p(a,\ b)}, \mathsf{p(b,\ a)}, \mathsf{p(b,\ b)}\} \mapsto \mathbb{N}$ defined by $|\mathsf{p(a,\ a)}| = 34$, $|\mathsf{p(a,\ b)}| = 12$, $|\mathsf{p(b,\ a)}| = 0$ and $|\mathsf{p(b,\ b)}| = 27$ is a level mapping for P. $\square$

Since a level mapping is defined over the Herbrand base it is not defined for non-ground atoms. (The reader is referred to Lloyd [21] for the standard definitions of the Herbrand base, Herbrand interpretations, Herbrand models, *etc.*) The lifting of the mapping to non-ground atoms was proposed in [4].

Definition 2.2 (bounded atom [4]) An atom A is bounded wrt a level mapping $|.|$ if $|.|$ is bounded on the set $[A]$ of variable free instances of A. If A is bounded then $|[A]|$ denotes the maximum that $|.|$ takes on $[A]$. $\square$

The importance of the notion of boundedness cannot be over stressed. Since goals which are ground cannot be used to compute values, they are the exception rather than the norm in logic programming. Thus practical termination proofs must be able to deal with non-ground goals and boundedness provides the basis for this.

Example 2.2 Let P be the program and $|.|$ the level mapping of example 2.1. The atom $\mathsf{p(a,\ X)}$ is bounded since $|.|$ is bounded on the set $[\mathsf{p(a,\ X)}] = \{\mathsf{p(a,\ a)}, \mathsf{p(a,\ b)}\}$. Moreover, $|[\mathsf{p(a,\ X)}]| = \max(\{|\mathsf{p(a,\ a)}|, |\mathsf{p(a,\ b)}|\})$ and in particular $|[\mathsf{p(a,\ X)}]| = \max(\{34, 12\}) = 34$. $\square$

Level mappings are usually defined in terms of norms. Basically, a norm is a mapping from terms to natural numbers which provides some measure of the size of a term.

Example 2.3 The list-length norm $|.|_{\text{length}} : U_P \mapsto \mathbb{N}$ from the Herbrand universe to the natural numbers can be defined by

$$|t|_{\text{length}} = \begin{cases} 1 + |t_2|_{\text{length}} & \text{if } t = [t_1|t_2] \\ 0 & \text{otherwise} \end{cases}$$

Then, for example, $|[\mathsf{X}, \mathsf{Y}, \mathsf{Z}]|_{\text{length}} = 3$. □

Example 2.4 The term-size norm $|.|_{\text{size}} : U_P \mapsto \mathbb{N}$ is defined by

$$|f(t_1, \ldots, t_n)|_{\text{size}} = n + \sum_{i=1}^{n} |t_i|_{\text{size}}$$

Thus, for example, $|\mathsf{f}(\mathsf{a}, \mathsf{g}(\mathsf{b}))|_{\text{size}} = 2 + 1 = 3$. □

The next two lemmas follow easily from definition 2.2.

Lemma 2.1 Let $|.|$ be a level mapping and A a bounded atom. Then for every substitution θ, the atom $A\theta$ is also bounded and moreover $|[A]| \geq |[A\theta]|$. □

Proof 2.1 *Recall that $[A] = \{A\phi \mid \phi$ is a grounding substitution for $A\}$. Then $[A] \supseteq [A\theta]$, so $|[A]| \geq |[A\theta]|$.* □

Lemma 2.2 Let H be a bounded atom, B an atom and $|.|$ a level mapping. If for every grounding substitution θ for H and B, $|H\theta| > |B\theta|$, then B is also bounded and moreover $|[H]| > |[B]|$. □

Proof 2.2 *Recall that $[B] = \{B\theta \mid \theta$ is a grounding substitution for $B\}$. But $|H\theta| > |B\theta|$ for every grounding substitution θ for H and B, so $|.|$ is bounded on $[B]$. Let θ be any grounding substitution for H and B such that $|B\theta| = |[B]|$. Then, by lemma 2.1, $|[H]| \geq |[H\theta]| = |H\theta| > |B\theta| = |[B]|$.* □

2.2 Recurrency

In [4, 5], level mappings are used to define a class of terminating programs.

Definition 2.3 (recurrency [4, 5]) Let P be a definite logic program and $|.|$ a level mapping for P. A clause $H \leftarrow B_1, \ldots, B_n$ is recurrent (wrt $|.|$) iff for every grounding substitution θ and for all $i \in [1, n]$ it follows that $|H\theta| > |B_i\theta|$. P is recurrent (wrt $|.|$) iff every clause in P is recurrent (wrt $|.|$). □

Henceforth all logic programs are assumed to be definite, that is, each clause contains precisely one atom in its consequent (its head).

Example 2.5 Consider the append program below

app_1 append([], X, X).
app_2 append([U|X], Y, [U|Z]) ← append(X, Y, Z).

The clause app_2 is recurrent wrt to the level mapping $|\mathsf{append}(t_1, t_2, t_3)|_1 = |t_1|_{\mathrm{length}}$ since for every grounding substitution θ for app_2,

$$\begin{aligned}|\mathsf{append}([\mathsf{U}|\mathsf{X}],\ \mathsf{Y},\ [\mathsf{U}|\mathsf{Z}])\theta|_1 &= |[\mathsf{U}|\mathsf{X}]\theta|_{\mathrm{length}}\\ &= 1 + |\mathsf{X}\theta|_{\mathrm{length}}\\ &> |\mathsf{X}\theta|_{\mathrm{length}}\\ &= |\mathsf{append}(\mathsf{X},\ \mathsf{Y},\ \mathsf{Z})\theta|_1\end{aligned}$$

Similarly, it can be shown that the program is recurrent wrt $|.|_i$ for all $i \in [2, 4]$ where $|.|_2$, $|.|_3$ and $|.|_4$ are defined by

$$\begin{aligned}|\mathsf{append}(t_1, t_2, t_3)|_2 &= 3|t_1|_{\mathrm{length}} + 1\\ |\mathsf{append}(t_1, t_2, t_3)|_3 &= |t_3|_{\mathrm{length}}\\ |\mathsf{append}(t_1, t_2, t_3)|_4 &= \min(|t_1|_{\mathrm{length}}, |t_3|_{\mathrm{length}})\end{aligned}$$

Moreover, the clause app_1 is trivially recurrent wrt to any level mapping. □

Bezem formalised the concept of termination relating it to recurrency.

Definition 2.4 (termination [4]) Let P be a logic program and G a goal. Then G is terminating wrt P iff every SLD-derivation for $P \cup \{G\}$ is finite. P is terminating iff every variable-free goal is terminating wrt P. □

Theorem 2.1 (recurrency [4]) Every recurrent program is terminating.

The same result was also obtained independently by Cavedon [11] in the more general context of recurrent programs with negation (called locally ω-hierarchical programs in [11] and later renamed acyclic programs in [1]). The proof in [4] relies on the following definition.

Definition 2.5 (bounded goal [4]) A goal $G = \leftarrow A_1, \ldots, A_n$ is bounded wrt a level mapping $|.|$ iff every A_i is bounded wrt $|.|$. If G is bounded then $|[G]|$ denotes the finite multiset of natural numbers $\{\!|\, |[A_1]|, \ldots, |[A_n]| \,|\!\}$. □

The proof of theorem 2.1 applies the abstraction function $\mathcal{A} = |[.]|$ and as a result a well-founded order $<$ is defined over the set of bounded goals by taking $G' < G$ iff $|[G']| <_{mul} |[G]|$ where $<_{mul}$ is the multiset ordering over the natural numbers. Recall that this ordering is defined by $s_1 <_{mul} s_2$ iff there exists $n_1, \ldots, n_m \in s_1$ and $n \in s_2$ such that $s_1 = (s_2 / \{\!|n|\!\}) \cup \{\!|n_1, \ldots, n_m|\!\}$ and $n_i < n$ for all $i \in [1, m]$ [25]. The proof is completed by showing for every SLD-resolvent G' of a bounded goal G, that G' is bounded and $G' < G$. In fact, this proof suggests a stronger corollary (bounded goals are not necessarily variable-free, that is, ground).

Corollary 2.1 (recurrency [4]) Let P be a logic program, G a goal and $|.|$ a level mapping. If P is recurrent wrt $|.|$ and G is bounded wrt $|.|$ then G is terminating wrt P.

Example 2.6 Reconsider append and the level mappings of example 2.5. Then

$$\begin{aligned}&\leftarrow \mathsf{append}([\mathsf{U},\ \mathsf{V},\ \mathsf{W}],\ \mathsf{Y},\ \mathsf{Z}) \text{ is bounded wrt } |.|_1, |.|_2 \text{ and } |.|_4,\\ &\leftarrow \mathsf{append}(\mathsf{X},\ \mathsf{Y},\ [\mathsf{U},\ \mathsf{V},\ \mathsf{W}]) \text{ is bounded wrt } |.|_3 \text{ and } |.|_4\end{aligned}$$

Hence these goals are terminating wrt append. Also, for a goal G observe that G is bounded wrt $|.|_1$ iff G is bounded wrt $|.|_2$. Moreover, G is bounded wrt $|.|_4$ if (*not iff*) G is bounded wrt $|.|_1$ or G is bounded wrt $|.|_3$. Thus by proving recurrency of append wrt $|.|_4$ a larger class of goals can be proven terminating than by proving recurrency wrt $|.|_1$, $|.|_2$ or $|.|_3$. This illustrates that the choice of the level mapping effects the set of goals which can be shown to be terminating. □

As a final remark, Bezem also proved the converse of theorem 2.1.

Theorem 2.2 (recurrency [4]) A logic program is recurrent iff it is terminating.

2.3 Acceptability

The notion of recurrency is a theoretical one and is not of much use in proving termination of Prolog programs. Many Prolog programs only terminate under a left-to-right selection rule. This observation led Apt and Pedreschi [3] to refine the notion of termination as follows.

Definition 2.6 (left-termination [3]) Let P be a logic program and G a goal. Then G is left-terminating wrt P iff every LD-derivation for $P \cup \{G\}$ is finite. P is left-terminating iff every variable-free goal is left-terminating wrt P. □

Example 2.7 Consider the permute program below

$perm_1$ permute([], []).
$perm_2$ permute([H|T], [A|P]) $\leftarrow$ delete(A, [H|T], L), permute(L, P).

del_1 delete(X, [X|Y], Y).
del_2 delete(X, [Y|Z], [Y|W]) $\leftarrow$ delete(X, Z, W).

The goal $\leftarrow$ permute([1], [1]) is terminating wrt permute and as a consequence is left-terminating also. The goal $\leftarrow$ permute([1, 2], [1, 2]) is left-terminating but *not* terminating, since there exists a computation rule which results in the following infinite derivation

$$\begin{array}{ll}
\leftarrow & \mathsf{permute}([1,2],[1,2]), \\
\leftarrow & \mathsf{delete}(1,[1,2],\mathsf{L}), \mathsf{permute}(\mathsf{L},[2]), \\
\leftarrow & \mathsf{delete}(1,[1,2],\mathsf{L}), \mathsf{delete}(2,\mathsf{L},\mathsf{L}'), \mathsf{permute}(\mathsf{L}',[]), \\
\leftarrow & \mathsf{delete}(1,[1,2],\mathsf{L}), \mathsf{delete}(2,\mathsf{Z},\mathsf{W}), \mathsf{permute}(\mathsf{L}',[]), \\
\leftarrow & \mathsf{delete}(1,[1,2],\mathsf{L}), \mathsf{delete}(2,\mathsf{Z}',\mathsf{W}'), \mathsf{permute}(\mathsf{L}',[]), \\
\leftarrow & \ldots
\end{array}$$

By theorem 2.2, it follows that the program is not recurrent. However, the program can be proven to be left-terminating. □

The class of recurrent programs was extended in [3] to the class of acceptable programs in order to provide a theoretical basis for proving termination of left-terminating programs.

Definition 2.7 (acceptability [3]) Let $|.|$ be a level mapping and I an interpretation for a logic program P. A clause $c : H \leftarrow B_1, \ldots, B_n$ is acceptable wrt $|.|$ and I iff

1. I is a model for c and
2. for all $i \in [1, n]$ and for every grounding substitution θ for c such that $I \models \{B_1, \ldots, B_{i-1}\}\theta$ it follows that $|H\theta| > |B_i\theta|$.

P is acceptable (wrt $|.|$ and I) iff every clause in P is acceptable (wrt $|.|$ and I). □

Analogous results to those for recurrent programs (theorem 2.1, corollary 2.1 and theorem 2.2) have been proven for acceptable programs. The abstraction function used, however, is rather more complicated than that which is applied in the proof of recurrency. First, observe that if a goal $G = \leftarrow A_1, \ldots, A_n$ terminates under a left-to-right computation rule, then each atom A_i is not necessarily bounded, but should be once the atoms to its left have been resolved. This idea forms the basis of the following definitions.

Definition 2.8 (maximum function) The maximum function $\max : \wp(\mathbb{N}) \mapsto \mathbb{N} \cup \{\infty\}$ is defined as

$$\max S \;=\; \begin{cases} 0 & \text{if } S = \emptyset \\ n & \text{else if } S \text{ is finite and } n \text{ is the maximum of } S \\ \infty & \text{otherwise} \end{cases}$$

Then $\max S \neq \infty$ iff the set S is finite. □

Definition 2.9 (left-bounded goal [3]) Let $|.|$ be a level mapping, I an interpretation and $G = \leftarrow A_1, \ldots, A_n$ a goal. Then G is left-bounded wrt $|.|$ and I iff the set

$$|[G]_I^i| = \left\{ |A_i\theta| \;\middle|\; \begin{array}{l} \theta \text{ is a grounding substitution for } G \\ I \models \{A_1, \ldots, A_{i-1}\}\theta \end{array} \right\}$$

is finite for each $i \in [1, n]$. If G is left-bounded wrt $|.|$ and I then $|[G]_I|$ denotes the finite multiset $\{\!| \max |[G]_I^1|, \ldots, \max |[G]_I^n| |\!\}$. □

Note that the term left-bounded is introduced here to avoid confusion with definition 2.5.

Using the abstraction function $\mathcal{A} = |[.]_I|$ allows one to prove that for a goal G which is left-bounded wrt $|.|$, any SLD-resolvent G' of G is left-bounded and furthermore $|[G']_I| <_{mul} |[G]_I|$. The result is the analogue of corollary 2.1.

Corollary 2.2 (acceptability [3]) Let P be a logic program, G a goal, $|.|$ a level mapping and I an interpretation for P. If P is acceptable wrt $|.|$ and I and G is left-bounded wrt $|.|$ and I then G is left-terminating wrt P. □

Sufficient and necessary conditions for left-termination are characterised by the following theorem.

Theorem 2.3 (acceptability) A logic program is acceptable iff it is left-terminating.

Example 2.8 Considering the $\mathsf{permute}$ program again, let $|.|$ be the level mapping defined by

$$\begin{aligned} |\mathsf{permute}(t_1, t_2)| &= |t_1|_{\text{length}} + 1 \\ |\mathsf{delete}(t_1, t_2, t_3)| &= |t_2|_{\text{length}} \end{aligned}$$

and I be the interpretation

$$\begin{array}{rl} \{\mathsf{delete}(t_1, t_2, t_3) \mid & |t_2|_{\text{length}} = |t_3|_{\text{length}} + 1\} \cup \\ \{\mathsf{permute}(t_1, t_2) \mid & |t_1|_{\text{length}} = |t_2|_{\text{length}}\} \end{array}$$

Now I is a model for the program and, in particular, for the clause $perm_2$, and for every grounding substitution θ for $perm_2$,

$$\begin{aligned} |\mathsf{permute([H|T],\ [A|P])}\theta| &= |\mathsf{[H|T]}\theta|_{\text{length}} + 1 \\ &> |\mathsf{[H|T]}\theta|_{\text{length}} \\ &= |\mathsf{delete(A,\ [H|T],\ L)}\theta| \end{aligned}$$

and for every grounding substitution θ for $perm_2$ with $I \models \mathsf{delete(a,\ [H|T],\ L)}\theta$,

$$\begin{aligned}|\mathsf{permute([H|T],\ [A|P])}\theta| &= |\mathsf{[H|T]}\theta|_{\text{length}} + 1\\ &= (|\mathsf{L}\theta|_{\text{length}} + 1) + 1\\ &> |\mathsf{L}\theta|_{\text{length}} + 1\\ &= |\mathsf{permute(L,\ P)}\theta|\end{aligned}$$

Hence $perm_2$ is acceptable wrt $|.|$ and I. The clauses $perm_1$ and del_1 are trivially acceptable wrt $|.|$ and I since I is a model for them, while the clause del_2 can easily be shown to be acceptable wrt $|.|$ and I in the same way as for $perm_2$. This proves the program permute is left-terminating. □

3 The recurrent problem

The main problem with recurrency, as noted by [3] and [12], is that it does not intuitively relate to recursion, the principal cause of non-termination in a logic program. Definition 2.3 requires that, for every ground instance of a clause, the level of its head atom is greater than the level of every body atom irrespective of the recursive relation between the two. There is a temptation to address this issue by using a modified definition of recurrency which only requires a decrease for mutually recursive body atoms. The following example, from [12], shows that this revision, by itself, is too weak to prove termination.

Example 3.1 Using the weaker form of recurrency suggested above, the following program would be classed as recurrent.

p([H|T]) ← append(X, Y, Y), p(T).

append([U|X], Y, [U|Z]) ← append(X, Y, Z).
append([], X, X).

Using the left-to-right computation rule and the top-down search rule, however, the goal ← p([1,2]) admits an infinite computation. Of course, the clause defining the predicate p should not be classified as recurrent. The reason is that, while append is truly recurrent, only bounded goals are guaranteed to terminate and the predicate p contains an unbounded call to append. □

This example shows that the level mapping decrease between the head and the non-recursive atoms of a clause implied by definition 2.3, is required to ensure that all subcomputations are initiated from a bounded goal. Enforcing boundedness in this way, however, complicates the derivation of level mappings. The following example, illustrating this, also comes from [12].

Example 3.2 Consider the following program

p_1 p([]).
p_2 p([H|T]) ← q([H|T]), p(T).

q_1 q([]).
q_2 q([H|T]) ← q(T).

It is clear that this program is terminating for any goal ← p(x) where x is a rigid list, that is, $|x|_{\text{length}} = |x\theta|_{\text{length}}$ for every grounding substitution θ. To construct an automatic proof of termination one would like to use the level mapping $|.|$ defined by

$$|\mathsf{p}(x)| = |x|_{\text{length}} \quad |\mathsf{q}(x)| = |x|_{\text{length}}$$

The problem is that the clause p_2 is not recurrent wrt this level mapping since it is not the case that $|\mathsf{p}([\mathsf{H}|\mathsf{T}])\theta| > |\mathsf{q}([\mathsf{H}|\mathsf{T}])\theta|$ for all grounding substitutions θ. For the inequality to hold, an unnatural offset must be included in the level mapping definition by taking for example $|\mathsf{p}(x)| = |x|_{\text{length}} + 1$. $\square$

These examples show that the strict decrease in level between the head and body atoms of a recurrent clause is required for two distinct purposes:

1. To ensure that the levels of mutually recursive calls are strictly decreasing.
2. To ensure that subcomputations are initiated from a bounded goal.

4 Semi-recurrency

Apt and Pedreschi observed that, for termination, while it is necessary for the level mapping to decrease between the head of a clause and each mutually recursive body atom, a strict decrease is not required for the non-recursive body atoms. To distinguish between recursive and non-recursive body atoms the notion of predicate dependency is introduced.

Definition 4.1 (predicate dependency) Let $p, q \in \Pi$ where Π is the set of predicate symbols in a logic program P. Then p directly depends on q iff

$$p(t_1, \ldots, t_{n_p}) \leftarrow B_1, \ldots, B_n \in P$$

and $B_i = q(s_1, \ldots, s_{n_q})$ for some $i \in [1, n]$. The depends on relation, denoted $\sqsupseteq$, is defined as the reflexive, transitive closure of the directly depends on relation. If $p \sqsupseteq q$ and $q \sqsupseteq p$ then p and q are mutually dependent and this is denoted by $p \simeq q$. Furthermore, let $p \sqsupset q$ iff $p \sqsupseteq q$ and $p \not\simeq q$ and finally let $p \sqsubset q$ iff $q \sqsupset p$. $\square$

Apt and Pedreschi then introduced the notion of semi-recurrency to exploit the observation that a strict decrease in level in only required for the mutually recursive body atoms. In what follows $rel(A)$ denotes the predicate symbol of the atom A.

Definition 4.2 (semi-recurrency [3]) Let P be a logic program and $|.|$ a level mapping for P. A clause $H \leftarrow B_1, \ldots, B_n$ is semi-recurrent (wrt $|.|$) iff for every grounding substitution θ and for all $i \in [1, n]$ it follows that

1. $|H\theta| > |B_i\theta|$ if $rel(H) \simeq rel(B_i)$,
2. $|H\theta| + 1 > |B_i\theta|$ if $rel(H) \not\simeq rel(B_i)$.

P is semi-recurrent (wrt $|.|$) if every clause in P is semi-recurrent (wrt $|.|$). $\square$

Whilst this definition now admits a simple termination proof of example 3.2 using the level mapping of that example, it is not hard to construct examples where it is inadequate.

Example 4.1 Consider the following program

c_1 $\mathsf{p}([])$.
c_2 $\mathsf{p}([\mathsf{H}|\mathsf{T}]) \leftarrow \mathsf{q}([\mathsf{H}, \mathsf{H}|\mathsf{T}]), \mathsf{p}(\mathsf{T})$.

c_3 $\mathsf{q}([])$.
c_4 $\mathsf{q}([\mathsf{H}|\mathsf{T}]) \leftarrow \mathsf{q}(\mathsf{T})$.

To prove that the above program is semi-recurrent requires the following unnatural level mapping: $|\mathsf{p}(x)| = |x|_{\text{length}} + 1$ and $\mathsf{q}(x)| = |x|_{\text{length}}$. □

It seems that very little has actually been gained from this revised definition of recurrency which still insists that there is not an increase from the level of the head to the level of all body atoms. In fact, it does not matter if the level of a non-recursive atom is greater than the level of the head provided that such an atom is bounded whenever it is selected.

To be fair, the notion of semi-recurrency was introduced to facilitate modular termination proofs and does indeed, in some cases, allow proofs to be based on simpler level mappings than those used in proofs of recurrency. In the above example, however, this is not the case.

Example 4.2 Reconsider the program of example 4.1. According to the methodology of [3] a modular termination proof can be constructed in a bottom-up fashion on the recursive cliques of the predicate dependency graph. First q is proven to be (semi) recurrent wrt $|.|_q$ defined by

$$|\mathsf{q}(x)|_q = |x|_{\text{length}}$$

Second, p is proven to be (semi) recurrent wrt $|.|_p$ defined by

$$|\mathsf{p}(x)|_p = |x|_{\text{length}} \quad |\mathsf{q}(x)|_p = 0$$

The final step in the construction requires the derivation of a level mapping $|.|'$ such that

$$|\mathsf{p}([t_1|t_2])|' \geq |\mathsf{q}([t_1, t_1|t_2])|_q \quad \text{and} \quad |\mathsf{p}([t_1|t_2])|' \geq |\mathsf{p}(t_2)|'$$

for all ground terms t_1 and t_2. Providing the level mapping $|.|'$ exists, theorem 4.9 of [3] can be used to conclude that the program is semi-recurrent and hence terminating. In terms of automation, this existence proof is achieved through defining $|.|'$ so that the above inequalities are satisfied. However, the most likely choice of a definition for $|.|'$ is

$$|\mathsf{p}(x)|' = |x|_{\text{length}} + 1$$

which, of course, this is no easier to derive than the original mapping $|.|$ of example 4.1. □

What is most conspicuous about the definition of semi-recurrency, is that the difference in levels between a non-recursive body atom and the head atom of a clause is limited to be at most zero, whereas it could be arbitrarily large, though still finite. Indeed, a simple termination proof for the program of example 4.1 can be obtained using a more natural level mapping if condition 2 of definition 4.2 is replaced by $|H\theta| + k > |B_i\theta|$ if $rel(H) \not\simeq rel(B_i)$, where k is some large constant. It is easy to prove that this revised definition of semi-recurrency is equivalent to recurrency. In addition, theorems 4.6, 4.8 and 4.9 of [3], which are used for constructing modular termination proofs, all still hold with this alternative definition.

Note that the problem with the termination proofs above arises because the atoms in the body of a clause contain extra function symbols which raise the levels of those atoms to the level of the head. Since it is fairly unlikely that such a body atom will contain, say, a million function symbols or more, by taking $k = 1000000$ the vast majority of recurrent programs which occur in practise could be proven terminating by focusing solely on their recursive structure and employing the appropriately weakened forms of the theorems of Apt and Pedreschi.

5 Semi-acceptability

Similar remarks to those of section 3 can be made about the definition of acceptability. The notion of semi-acceptability was introduced as an analogous concept to semi-recurrency for left-terminating programs.

Definition 5.1 (semi-acceptability [3]) Let $|.|$ be a level mapping and I an interpretation for a logic program P. A clause $c : H \leftarrow B_1, \ldots, B_n$ is semi-acceptable wrt $|.|$ and I iff

1. I is a model for c and
2. for all $i \in [1, n]$ and for every grounding substitution θ for c such that $I \models \{B_1, \ldots, B_{i-1}\}\theta$ it follows that
   (a) $|H\theta| > |B_i\theta|$ if $rel(H) \simeq rel(B_i)$,
   (b) $|H\theta| + 1 > |B_i\theta|$ if $rel(H) \not\simeq rel(B_i)$.

P is semi-acceptable (wrt $|.|$ and I) iff every clause in P is semi-acceptable (wrt $|.|$ and I). □

Not surprisingly, termination proofs based on semi-acceptability suffer from similar problems to those encountered in examples 4.1 and 4.2. The definition could be adjusted in the manner prescribed above for semi-recurrency, but the result is not as satisfactory as the following example shows.

Example 5.1 Consider the following program

```
doubleSquare(0, []).
doubleSquare(s(X), [D|Ds]) ←
    square(X, 0, Y), doublePlus(Y, 0, D), doubleSquare(X, Ds).

square(0, Y, Y).
square(s(X), Acc, Y) ←
    doublePlus(X, s(Acc), Acc1), square(X, Acc1, Y).

doublePlus(0, X, X).
doublePlus(s(X), Y, s(s(Z))) ←
    doublePlus(X, Y, Z).
```

The function symbol s is interpreted as the successor function. Let $|0|_s = 0$ and $|\mathsf{s}(x)|_s = 1 + |x|_s$ and I be the interpretation

$$\begin{array}{rl}\{\mathsf{doublePlus}(t_1, t_2, t_3) \mid & |t_3|_s = 2|t_1|_s + |t_2|_s\} \cup \\ \{\mathsf{square}(t_1, t_2, t_3) \mid & |t_3|_s = |t_1|_s^2 + |t_2|_s\} \cup \\ \{\mathsf{doubleSquare}(t, [t_1, t_2, \ldots, 0]) \mid & |t_i|_s = 2(|t|_s - i)^2\}\end{array}$$

Thus, for example, the goal $\leftarrow$ doubleSquare(s(s(s(0))), L) will succeed with L = [s(s(s(s(s(s(s(s(0)))))))),s(s(0)),0]. Observe that I is a model for the program. Now let the level mapping $|.|$ be defined by

$$|\mathsf{doubleSquare}(x, y)| = |\mathsf{square}(x, y, z)| = |\mathsf{doublePlus}(x, y, z)| = |x|_s$$

The predicates square and $\mathsf{doublePlus}$ are both semi-recurrent (and hence semi-acceptable) wrt $|.|$ and I. Now consider $\mathsf{doubleSquare}$ and in particular the inequality $|\mathsf{doubleSquare(s(X),[D|Ds])}\theta| + k > |\mathsf{doublePlus(Y,0,D)}\theta|$ where θ is any grounding substitution. Since $I \models \mathsf{square(X,0,Y)}\theta$ it follows that $|\mathsf{Y}\theta|_s = |\mathsf{X}\theta|_s^2$, hence there exists no k for which $1 + |\mathsf{X}\theta|_s + k > |\mathsf{Y}\theta|_s$ always holds. Therefore the inequality $|\mathsf{doubleSquare(s(X),[D|Ds])}\theta| + k = 1 + |\mathsf{X}\theta|_s + k > |\mathsf{Y}\theta|_s = |\mathsf{doublePlus(Y,0,D)}\theta|$ cannot always hold. Hence the predicate $\mathsf{doubleSquare}$ is not semi-acceptable wrt $|.|$ and I under the revised definition suggested above even though the level mapping is natural.

It is easy to prove semi-acceptability of the program, however, wrt the level mapping $|.|'$ where $|.|'$ is defined exactly as for $|.|$ except that

$$|\mathsf{doubleSquare}(x, y)|' = |x|_s^2$$

Note that a goal is bounded wrt $|.|$ iff it is bounded wrt $|.|'$ and all such goals are left-terminating. It seems reasonable then to base a proof of termination on the former level mapping since it more closely relates to the recursion and as a result is easier to derive automatically. However, no automatic termination analysis has yet been devised which can manipulate quadratic level mappings such as $|.|_s^2$. □

Observe that the k above acts as an upper bound on the difference between the level of any body atom and the level of the head atom. Of course, this ad hoc approach falls down when there is no upper bound as in example 5.1.

In summary, although semi-recurrency and semi-acceptability are more flexible notions than their predecessors, they still enforce a dependence between the level of a head atom and the levels of non-recursive body atoms. This dependence is counter intuitive and forces one to use artificial level mappings to obtain termination proofs.

6 Bounded-recurrency

Recall from section 3 that there are two conditions which must be fulfilled to ensure that a program is terminating.

1. The levels of mutually recursive calls are strictly decreasing.
2. All subcomputations are initiated from a bounded goal.

It is possible to define what constitutes a terminating program directly from these two requirements.

Definition 6.1 (bounded-recurrency) Let $|.|$ be a level mapping for a logic program P. A clause $c : H \leftarrow B_1, \ldots, B_n$ is bounded-recurrent (wrt $|.|$) iff for every substitution θ for c such that $H\theta$ is bounded and for all $i \in [1, n]$ it follows that

1. $B_i\theta$ is bounded and
2. $|[H\theta]| > |[B_i\theta]|$ whenever $rel(H) \simeq rel(B_i)$.

P is bounded-recurrent (wrt $|.|$) iff every clause in P is bounded-recurrent (wrt $|.|$). □

Observe that no decrease is enforced between the level of the head of a clause and the levels of the non-recursive body atoms. All that is required is that each atom is bounded whenever the head is bounded. While this is more intuitively appealing, observe that boundedness of non-recursive atoms still influences the definition of the level mapping in a non-modular way.

Example 6.1 Consider the following program for Curry's type assignment taken from [3].

typ_1 type(E, var(X), T) $\leftarrow$ in(E, X, T).
typ_2 type(E, apply(M, N), T) $\leftarrow$ type(E, M, arrow(S, T)), type(E, M, S).
typ_3 type(E, lambda(X, M), arrow(S, T)) $\leftarrow$ type([(X, S) | E], M, T).

in_1 in([(X, T) | E], X, T).
in_2 in([(Y, T) | E], X, T) $\leftarrow$ X $\neq$ Y, in(E, X, T).

One may observe that the predicate in is inductively defined over the length of its first argument which is a list. The predicate type is inductively defined on the size of its second argument which is a λ-term. As a result, one would hope to base a termination proof on the level mapping $|.|$ defined by

$$|\mathsf{in}(x, y, z)| = |x|_{\text{length}} \quad |\mathsf{type}(x, y, z)| = |y|_{\text{size}}$$

The problem, of course, is that any call type(E, var(X), t) which is bounded wrt $|.|$ can give rise to a call in(E, X, T) which is not bounded wrt $|.|$. Clearly this can lead to non-termination. Definition 6.1, therefore, insists that for the clause typ_1 the body atom in(E, X, T) is bounded whenever the head is. Unfortunately this entails that the level mapping must now be modified to take the first argument of type into account. This in turn leads to problems with the clause typ_3 since the first argument is increasing in the recursive call. Eventually, one arrives at a level mapping definition such as

$$|\mathsf{in}(x, y, z)|_1 = |x|_{\text{length}} \quad |\mathsf{type}(x, y, z)|_1 = |x|_{\text{length}} + 2|y|_{\text{size}}$$

which bears no immediate relation to the program structure. As a result such a mapping is likely to be difficult to derive automatically. □

Clearly there is an interdependence between ensuring non-recursive atoms are bounded wrt $|.|$ and ensuring that the levels of recursive calls are decreasing wrt $|.|$. This plainly arises out of the use of the one level mapping. It seems therefore that the obvious way to break the dependence is to use *two* level mappings. One holds the responsibility for ensuring the recursive decrease in levels, while the other assures that non-recursive atoms are bounded. This idea is captured in the following definition.

Definition 6.2 (bounded-recurrency) Let $|.|_1$ and $|.|_2$ be level mappings for a logic program P. A clause $c : H \leftarrow B_1, \ldots, B_n$ is bounded-recurrent (wrt $|.|_1$ and $|.|_2$) iff for every substitution θ for c such that $H\theta$ is bounded wrt $|.|_1$ and $|.|_2$ and for all $i \in [1, n]$ it follows that

1. $B_i\theta$ is bounded wrt $|.|_1$ and $|.|_2$ and
2. $|[H\theta]|_1 > |[B_i\theta]|_1$ whenever $rel(H) \simeq rel(B_i)$.

P is bounded-recurrent (wrt $|.|_1$ and $|.|_2$) iff every clause in P is bounded-recurrent (wrt $|.|_1$ and $|.|_2$). □

It is informally understood that a goal G is bounded wrt $|.|_1$ and $|.|_2$ iff G is bounded wrt $|.|_1$ and G is bounded wrt $|.|_2$. Note that, when the two level mappings coincide, that is when $|.|_1 \equiv |.|_2$, then definition 6.2 is equivalent to definition 6.1.

Example 6.2 Returning to the program of example 6.1, recall that the stumbling block in the derivation of a natural level mapping arose because any call type(E, var(X), T) which is bounded wrt $|.|$ can give rise to a call in(E, X, T) which is not bounded wrt $|.|$. At this point, one intuitively reasons that if the first argument of a call to type is a rigid list then the first argument of all subsequent calls to type will also be a rigid list. So define a second level mapping $|.|'$ by

$$|\mathsf{in}(x,y,z)|' = |x|_{\mathrm{length}} \quad |\mathsf{type}(x,y,z)|' = |x|_{\mathrm{length}}$$

The program is bounded-recurrent wrt $|.|$ and $|.|'$. Indeed, any call to type or in which is bounded wrt $|.|$ and $|.|'$ only gives rise to calls which are bounded wrt $|.|$ and $|.|'$. Combine this with the fact that recursive calls are decreasing wrt $|.|$ and termination can be proven in a very intuitive manner. Furthermore, the level mappings $|.|$ and $|.|'$ follow directly from the structure of the program, facilitating their automatic derivation. $\Box$

Lemma 6.1 and corollary 6.1 below establish that bounded-recurrent programs are indeed terminating. Proof of this relies on orderings which not only take into account the levels of atoms but also their relation to each other in the predicate dependency graph. For a level mapping $|.|$ and goal $G = \leftarrow A_1, \ldots, A_n$, if G is bounded wrt $|.|$ then let $|[G]|$ denote the finite multiset of pairs $\{\!|\langle rel(A_1), |[A_1]|\rangle, \ldots, \langle rel(A_n), |[A_n]|\rangle|\!\}$. Let $\prec$ be the lexicographical ordering on $\Pi(\sqsubset) \times \mathbb{N}(<)$ and let $\prec_{mul}$ be the multiset ordering induced from $\prec$. Observe that $\prec_{mul}$ is well founded.

Lemma 6.1 Let $|.|_1$ and $|.|_2$ be level mappings for a logic program P. Let P be bounded-recurrent wrt $|.|_1$ and $|.|_2$ and let G be a goal which is bounded wrt $|.|_1$ and $|.|_2$. Let G' be an SLD-resolvent of G from P. Then

1. G' is bounded wrt $|.|_1$ and $|.|_2$,
2. $|[G']|_1 \prec_{mul} |[G]|_1$, and
3. every SLD-derivation of $P \cup \{\leftarrow G\}$ is finite.

Proof 6.1 *Assume A_j is the selected literal in $G = \leftarrow A_1, \ldots, A_m$ and the used clause is $c : H \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$. Then $G' = \leftarrow (A_1, \ldots, A_{j-1}, B_1, \ldots, B_n, A_{j+1}, \ldots, A_m)\theta$ where $\theta \in mgu(A_j, H)$.*

1. *Since G is bounded wrt $|.|_1$ and $|.|_2$, it follows that A_k and $A_k\theta$ are bounded wrt $|.|_1$ and $|.|_2$ for all $k \in [1, m]$. In particular, $A_j\theta = H\theta$ is bounded wrt $|.|_1$ and $|.|_2$. It follows, by definition 6.2, that $B_i\theta$ is bounded wrt $|.|_1$ and $|.|_2$ for all $i \in [1, n]$ and hence G' is bounded wrt $|.|_1$ and $|.|_2$.*
2. *Moreover, $|[A_k]|_1 \geq |[A_k\theta]|_1$ for all $k \in [1, m]$ by lemma 2.1. Finally, for all $i \in [1, n]$*
   (a) *$|[A_j\theta]|_1 > |[B_i\theta]|_1$ if $rel(A_j) = rel(H) \simeq rel(B_i)$, by definition 6.2, and*
   (b) *$rel(B_i\theta) \sqsubset rel(A_j)$ otherwise.*

   Hence $\langle rel(B_i\theta), |[B_i\theta]|_1\rangle \prec \langle rel(A_j), |[A_j]|_1\rangle$ for all $i \in [1, n]$ and also $\langle rel(A_k\theta), |[A_k\theta]|_1\rangle \preceq \langle rel(A_k), |[A_k]|_1\rangle$ for all $k \in [1, m]$ thereby proving $|[G']|_1 \prec_{mul} |[G]|_1$.
3. *Since $\prec_{mul}$ is well-founded the result follows immediately.*

Corollary 6.1 Every bounded-recurrent program is terminating.

Theorem 6.1 Let $|.|_1$ and $|.|_2$ be level mappings for a logic program P. The following hold.

1. If P is recurrent wrt $|.|_1$ then P is bounded-recurrent wrt $|.|_1$ and $|.|_1$.

2. If P is bounded-recurrent wrt $|.|_1$ and $|.|_2$, then there exists a level mapping $|.|_3$ such that P is recurrent wrt $|.|_3$. Moreover, for any atom A, A is bounded wrt $|.|_3$ if A is bounded wrt $|.|_1$ and $|.|_2$.

Proof 6.2 *Let $c : H \leftarrow B_1, \ldots, B_n$ be a clause in P. Suppose P is recurrent wrt $|.|_1$. Let θ be a substitution such that $H\theta$ is bounded wrt $|.|_1$. Then $B_i\theta$ is bounded and $|[H\theta]|_1 > |[B_i\theta]|_1$ for all $i \in [1, n]$ by recurrency. The second part follows by lemma 6.1 and theorem 2.2 and corollary 2.2 of [5].*

7 Bounded-acceptability

The definition of bounded-recurrency is easily adapted to obtain a characterisation of left-terminating programs.

Definition 7.1 (bounded-acceptability) Let $|.|_1$ and $|.|_2$ be level mappings and I an interpretation for a logic program P. A clause $c : H \leftarrow B_1, \ldots, B_n$. is bounded-acceptable (wrt $|.|_1$, $|.|_2$ and I) iff

1. I is a model for c and

2. for all $i \in [1, n]$ and for every substitution θ such that $H\theta$ is bounded wrt $|.|_1$ and $|.|_2$, $\{B_1, \ldots, B_{i-1}\}\theta$ is ground and $I \models \{B_1, \ldots, B_{i-1}\}\theta$ it follows that

   (a) $B_i\theta$ is bounded wrt $|.|_1$ and $|.|_2$ and

   (b) $|[H\theta]|_1 > |[B_i\theta]|_1$ whenever $rel(H) \simeq rel(B_i)$.

P is bounded-acceptable (wrt $|.|_1$, $|.|_2$ and I) iff every clause in P is bounded-acceptable (wrt $|.|_1$, $|.|_2$ and I). □

Lemma 7.1 asserts that every bounded-acceptable program is left-terminating. The proof of this follows along the same lines as that for acceptable programs.

Lemma 7.1 Let $|.|_1$ and $|.|_2$ be level mappings and I an interpretation for a program P. Let P be bounded-acceptable wrt $|.|_1$, $|.|_2$ and I, and let G be a goal which is left-bounded wrt $|.|_1$ and I, and wrt $|.|_2$ and I. Let G' be an LD-resolvent of G from P. Then

1. G' is left-bounded wrt $|.|_1$ and I, and wrt $|.|_2$ and I,

2. $|[G']_I|_1 \prec_{mul} |[G]_I|_1$ and

3. every LD-derivation of $P \cup \{\leftarrow G\}$ is finite.

Proof 7.1 *Let $G = \leftarrow A_0, A_1, \ldots, A_m$ $(m > 0)$ and assume $c : H \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$ is the program clause used. Then $G' = \leftarrow (B_1, \ldots, B_n, A_1, \ldots, A_m)\theta$ where $\theta \in mgu(A_0, H)$.*

1. *It is necessary to show for all $j \in [1,2]$, $i \in [1, n+m]$ that $|[G']_I^i|_j$ is finite. Firstly, for all $j \in [1,2]$, $i \in [1,n]$*

$$\begin{array}{rcl}
|[G']_I^i|_j & = & |[\leftarrow (B_1,\ldots,B_n,A_1,\ldots,A_m)\theta]_I^i|_j \\
& = & \left\{ |B_i\theta\phi|_j \;\middle|\; \begin{array}{l} \phi \text{ grounds } G' \\ I \models \{B_1,\ldots,B_{i-1}\}\theta\phi \end{array} \right\} \\
& = & \left\{ |B_i\theta\phi\sigma|_j \;\middle|\; \begin{array}{l} \phi \text{ grounds } \{B_1,\ldots,B_{i-1}\}\theta \\ I \models \{B_1,\ldots,B_{i-1}\}\theta\phi \\ \sigma \text{ grounds } B_i\theta\phi \end{array} \right\}
\end{array}$$

Now by definition 7.1, for all $i \in [1,n]$, for every substitution ϕ such that $H\theta\phi$ is bounded wrt $|.|_1$ and $|.|_2$, $\{B_1,\ldots,B_{i-1}\}\theta\phi$ is ground and $I \models \{B_1,\ldots,B_{i-1}\}\theta\phi$

   (a) *$B_i\theta\phi$ is bounded wrt $|.|_1$ and $|.|_2$ and*

   (b) *$|[H\theta\phi]|_1 > |[B_i\theta\phi]|_1$ whenever $rel(H) \simeq rel(B_i)$.*

Hence, $|[G']_I^i|_j$ is finite for all $i \in [1,n]$, $j \in [1,2]$. Now for all $j \in [1,2]$, $k \in [1,m]$

$$\begin{array}{rcl}
|[G']_I^{n+k}|_j & = & |[\leftarrow (B_1,\ldots,B_n,A_1,\ldots,A_m)\theta]_I^{n+k}|_j \\
& = & \left\{ |A_k\theta\phi|_j \;\middle|\; \begin{array}{l} \phi \text{ grounds } G' \\ I \models \{B_1,\ldots,B_n,A_1,\ldots,A_{k-1}\}\theta\phi \end{array} \right\} \\
& \subseteq & \left\{ |A_k\theta\phi|_j \;\middle|\; \begin{array}{l} \phi \text{ grounds } \{H,A_1,\ldots,A_m\}\theta \\ I \models \{H,A_1,\ldots,A_{k-1}\}\theta\phi \end{array} \right\} \\
& = & |[\leftarrow (A_0,A_1,\ldots,A_m)\theta]_I^{k+1}|_j \\
& \subseteq & |[\leftarrow (A_0,A_1,\ldots,A_m)]_I^{k+1}|_j
\end{array}$$

Since G is left-bounded wrt $|.|_1$ and I, and wrt $|.|_2$ and I, then $|[G']_I^{n+k}|_j$ is finite for all $k \in [1,m]$, $j \in [1,2]$.

2. *It follows directly that for all $k \in [1,m]$, $j \in [1,2]$, $\max|[G']_I^{n+k}|_j \leq \max|[G]_I^{k+1}|_j$ and for all $i \in [1,n]$, whenever $rel(A_0) = rel(H) \simeq rel(B_i)$*

$$\begin{array}{rcl}
\max|[G']_I^i|_1 & < & \max\{|H\theta\phi|_1 \mid \phi \text{ grounds } H\theta\} \\
& = & \max\{|A_0\theta\phi|_1 \mid \phi \text{ grounds } A_0\theta\} \\
& = & \max|[\leftarrow A_0\theta]_I^1|_1 \\
& \leq & \max|[\leftarrow A_0]_I^1|_1 \\
& = & \max|[G]_I^1|_1
\end{array}$$

Hence $\langle rel(B_i\theta), \max|[G']_I^i|_1\rangle \prec \langle rel(A_0), \max|[G]_I^1|_1\rangle$ for all $i \in [1,n]$ and $\langle rel(A_k\theta), \max|[G']_I^{n+k}|_1\rangle \preceq \langle rel(A_k), \max|[G]_I^{k+1}|_1\rangle$ for all $k \in [1,m]$ thereby proving $|[G']_I|_1 \prec_{mul} |[G]_I|_1$.

3. *Since $\prec_{mul}$ is well-founded the result follows immediately.*

Corollary 7.1 Every bounded-acceptable program is left-terminating.

Theorem 7.1 Let $|.|_1$ and $|.|_2$ be level mappings and I an interpretation for a program P. The following hold.

1. If P is acceptable wrt $|.|_1$ then P is bounded-acceptable wrt $|.|_1$ and $|.|_1$.

2. If P is bounded-acceptable wrt $|.|_1$ and $|.|_2$, then there exists a level mapping $|.|_3$ such that P is acceptable wrt $|.|_3$. Moreover, for any atom A, A is bounded wrt $|.|_3$ if A is bounded wrt $|.|_1$ and $|.|_2$. □

Proof 7.2 *Let $c : H \leftarrow B_1, \ldots, B_n$ be a clause in P. Suppose P is acceptable wrt $|.|_1$. Then I is a model for c. Let θ be a substitution such that $H\theta$ is bounded wrt $|.|_1$, $\{B_1, \ldots, B_{i-1}\}\theta$ is ground and $I \models \{B_1, \ldots, B_{i-1}\}\theta$. Then $B_i\theta$ is bounded wrt $|.|_1$ and $|[H\theta]|_1 > |[B_i\theta]|_1$ by acceptability and lemma 2.2. The second part follows by lemma 7.1, theorem 2.2 and corollary 2.2 of [5].* □

8 Discussion

The concept of bounded-acceptability proposed here is quite similar to that of rigid-acceptability defined by [13, 16]. This latter notion forms the basis of a practical, demand-driven termination analysis. The analysis is essentially top-down, attempting to prove termination for a set of queries S. An important step in the analysis is the calculation of the call set $Call(P, S)$, the set of all calls which may occur during the derivation of an atom in S. The analysis focuses on the recursive components to derive a level mapping $|.|$, enforcing boundedness of sub-computations by imposing a rigidity constraint on the call set. That is, during the derivation of $|.|$, every atom in $Call(P, S)$ is required to be rigid wrt $|.|$.

For program specialisation, and partial deduction in particular, it is more useful to derive sufficient termination conditions for individual predicates rather than proving that a given top-level goal will terminate [10]. The reason is that the overall computation is unlikely to be left-terminating but some sub-computations probably will be. The required conditions can be derived in a bottom-up manner on the strongly connected components of the predicate dependency graph. The notion of bounded-acceptability lends itself naturally to this process.

In [14], the analysis of [13] is adapted to obtain the above mentioned conditions. It attempts to derive for each predicate a maximal set S of left-terminating queries. Essentially, this amounts to deriving a level mapping $|.|$ which defines S, in that an atom A is in S if and only if A is bounded wrt $|.|$. However, an important step is omitted from the paper, and the set S may contain queries which are not left-terminating. The level mapping $|.|$ is derived by only considering the recursive components of the program and thus corresponds to the level mapping $|.|_1$ in the definition of bounded-acceptability. Sub-computations are no longer guaranteed to start from bounded goals since no rigidity constraint is placed on the level mapping during its derivation as in [13]: specifically, this is because the set $Call(P, S)$ is unknown since S is unknown (the idea after all being to derive S), and as a result no rigidity constraint can be imposed on $Call(P, S)$. Hence, in relation to the current work, the missing step is the derivation of the second level mapping $|.|_2$. The maximal set $S' \subseteq S$ of left-terminating queries then, contains only those atoms which are bounded wrt $|.|_1$ and $|.|_2$. Note that $|.|_2$ can be derived entirely independently of $|.|_1$, in the sense that there is never any need to alter the definition of $|.|_1$ in order to obtain a definition of $|.|_2$ which can be used to prove bounded-acceptability. Thus the notion of bounded-acceptability allows the set S' to be easily constructed from S without requiring any change to the method of [14].

Recently, Bossi *et al* [6] have developed an entirely modular approach to termination in which acceptibiliy is proven on a module-by-module basis by choosing a natural level mapping that focuses solely on the predicates defined within the module. The key concept is strong boundedness. A query to a program that is defined over n modules $R_1, \ldots, R_n$ is said to be strongly bounded if each call to a predicate that is defined in module R_i is bounded with respect to the level mapping for that module $|.|_i$. A sufficient condition for left-termination of a strongly bounded query is for each module R_i to be acceptable with respect to its own level mapping $|.|_i$ and a model of the whole program.

Observe, however, that strong boundedness is a property of derivations rather than a model. The authors, however, argue that the approach is still attractive because strong boundedness can be verified by approximating call-patterns by goal-dependent abstract interpretation [8]. Moreover, well-moded [17] and well-typed logic programs [7] are in some sense well-behaved with respect to strong boundedness and thereby provide another route for asserting strong boundedness. This work is applicable to general logic programs and therefore generalises the modular termination proofs for well-moded definite programs originally proposed in [18].

In summary, the notions of bounded-recurrency and bounded-acceptability provide practical criteria for constructing modular termination proofs based purely on the recursive structure of a program.

Acknowledgements

This work was supported, in part, by the EPSRC studentship 93315269; in fact much of this paper is adapted from chapters 3 and 6 of [22]. The work has benefited for useful discussions with Danny De Schreye and Fred Mesnard. The authors gratefully acknowledge Nuffield grant SCI/180/94/417/G and EPSRC grant GR/MO8769 for funding their collaboration.

References

[1] K. R. Apt and M. Bezem. Acyclic Programs. *New Generation Computing*, 9(3/4):335–364, 1991.

[2] K. R. Apt and D. Pedreschi. Reasoning about Termination of Pure Prolog Programs. *Information and Computation*, 106(1):109–157, 1993.

[3] K. R. Apt and D. Pedreschi. Modular Termination Proofs for Logic and Pure Prolog programs. In G. Levi, editor, *Advances in Logic Programming Theory*, pages 183–229. Oxford University Press, 1994. Also available as technical report CS-R9316 from Centrum voor Wiskunde en Informatica, CWI, Amesterdam.

[4] M. Bezem. Characterizing Termination of Logic Programs with Level Mappings. In E. L. Lusk and R. A. Overbeek, editors, *North American Conference on Logic Programming*, pages 69–80. MIT Press, 1989.

[5] M. Bezem. Strong Termination of Logic Programs. *The Journal of Logic Programming*, 15(1&2):79–97, 1993.

[6] A. Bossi, N. Cocco, S. Etalle, and S. Rossi. On Modular Termination Proofs of General Logic Programs. *Theory and Practice of Logic Programming*, 2(3):263–291, 2002.

[7] F. Bronsard, T. K. Lakshman, and U. S. Reddy. A Framework of Directionality for Proving Termination of Logic Programs. In K. R. Apt, editor, *Joint International Conference and Symposium on Logic Programming*, pages 321–335. MIT Press, 1992.

[8] M. Bruynooghe. A Practical Framework for the Abstract Interpretation of Logic Programs. *The Journal of Logic Programming*, 10(1/2/3&4):91–124, 1991.

[9] M. Bruynooghe, M. Codish, S. Genaim, and W. Vanhoof. Reuse of Results in Termination Analysis of Typed Logic Programs. In M. V. Hermenegildo and G. Puebla, editors, *Static*

Analysis Symposium, number 2477 in Lecture Notes in Computer Science, pages 477–492. Springer-Verlag, 2002.

[10] M. Bruynooghe, M. Leuchel, and K. F. Sagonas. A polyvariant binding-time analysis for off-line partial deduction. In C. Hankin, editor, *European Symposium on Programming*, volume 1381 of *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 1998.

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

[12] D. De Schreye, K. Verschaetse, and M. Bruynooghe. A Framework for Analysing the Termination of Definite Logic Programs with Respect to Call Patterns. In *International Conference on Fifth Generation Computer Systems*, pages 481–488. IOS Press, 1992.

[13] S. Decorte and D. De Schreye. Demand-driven and constraint-based automatic left-termination analysis of logic programs. In L. Naish, editor, *International Conference on Logic Programming*, pages 78–92. MIT Press, 1997.

[14] S. Decorte and D. De Schreye. Termination analysis: Some practical properties of the Norm and Level Mapping Space. In J. Jaffar, editor, *Joint International Conference and Symposium on Logic Programming*, pages 235–249. MIT Press, 1998.

[15] S. Decorte, D. De Schreye, and M. Fabris. Automatic Inference of Norms: A Missing Link in Automatic Termination Analysis. In D. Miller, editor, *International Logic Programming Symposium*, pages 420–436. MIT Press, 1993.

[16] S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based Termination Analysis of Logic Programs. *ACM Transactions on Programming Languages and Systems*, 21(6):1137–1195, 1999.

[17] P. Dembiński and J. Maluszyński. And-Parallelism with Intelligent Backtracking for Annotated Logic Programs. In *Symposium on Logic Programming*, pages 29–38. IEEE Press, 1985.

[18] S. Etalle, A. Bossi, and N. Cocco. Termination of Well-Moded Programs. *The Journal of Logic Programming*, 38(2):243–257, 1999.

[19] S. Genaim, M. Codish, J. P. Gallagher, and V. Lagoon. Combining Norms to Prove Termination. In A. Cortesi, editor, *Verification, Model Checking and Abstract Interpretation*, number 2294 in Lecture Notes in Computer Science, pages 126–138. Springer-Verlag, 2002.

[20] V. Lagoon, F. Mesnard, and P. Stuckey. Termination Analysis with Types is More Accurate. In C. Palamidessi, editor, *International Conference on Logic Programming*, Lecture Notes in Computer Science. Springer-Verlag, 2003.

[21] J. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[22] J. C. Martin. *Judgement Day: Terminating Logic Programs.* PhD thesis, Department of Electronics and Computer Science, University of Southampton, 2000.

[23] J. C. Martin, A. King, and P. Soper. Typed Norms for Typed Logic Programs. In J. Gallagher, editor, *Logic Program Synthesis and Transformation (Selected Papers)*, volume 1207 of *Lecture Notes in Computer Science*, pages 224–238. Springer-Verlag, 1997.

[24] D. Pedreschi and S. Ruggieri. Verification of Logic Programs. *The Journal of Logic Programming*, 39(1-3):125–176, 1999.

[25] J. Van Leeuwen, editor. *Handbook of Theoretical Computer Science: Volume B.* Elsevier, 1990.

[26] W. Vanhoof and M. Bruynooghe. When Size Does Matter. In A. Pettorossi, editor, *Logic Based Program Synthesis and Transformation (Selected Papers)*, volume 2372 of *Lecture Notes in Computer Science*, pages 129–147, 2001.

[27] S. Verbaeten, D. De Schreye, and K. F. Sagonas. Termination Proofs for Logic Programs with Tabling. *ACM Transactions on Computational Logic*, 2(1):57–92, 2001.