RESEARCH NOTE

On the Existence of Nonterminating Queries for a Restricted Class of PROLOG-Clauses

Danny de Schreye*, Maurice Bruynooghe,
Kristof Verschaetse***

*Department of Computer Science, Katholieke Universiteit
Leuven, Celestijnenlaan 200A, 3030 Heverlee, Belgium*

ABSTRACT

This work studies the existence of nonterminating queries for a restricted class of recursive predicates. The class is restricted to predicates defined by a set of clauses containing only one (direct) recursive clause. Moreover, this clause must be of the form $P(\dots) \leftarrow P(\dots)$. For this restricted class of predicates, we prove a necessary and sufficient condition for the existence of a query which is nonterminating in the absence of the occur check. The condition is based upon concepts from rational trees.

1. Introduction

Our work was inspired by a conjecture of Smith, Genesereth and Ginsberg [11] on a necessary condition for the existence of an infinite computational path for a class of programs called *cyclic collections*. With the terminology of [9] a cyclic collection is a set of definite Horn clauses of the type:

$$P_1 :- P'_2, B_1,$$

$$P_2 :- P'_3, B_2,$$

$$\vdots$$

$$P_n :- P'_1, B_n,$$

* Supported by the Belgian IWONL-IRSIA under contract number 5203

** Supported by the Belgian National Fund for Scientific Research

such that P_i and P'_i unify, for $i = 1, \dots, n$. A cyclic collection is called *recursive*, if there exists a common set of bindings for all the unifications $P_i = P'_i$, $i = 1, \dots, n$. In [11, Theorem 2.1] it is shown that for any cyclic recursive collection, there exists a query q with an infinite computational path for the depth-first left-to-right computation rule. In the same paper, [11, Conjecture 2.3], it is conjectured that the inverse statement also holds. More precisely, their statement is that “if an infinite path exists in the search space, and the database is finite, then there must be a recursive collection of facts involved in the generation of that path.” This conjecture is clearly false, if it is considered within the context of SLD-refutations following a depth-first left-to-right computation rule, including the occur check. A simple counterexample is:

$$\text{append}([h|t], l, [h|s]) :- \text{append}(t, l, s)$$

This clause admits infinitely many queries causing a nonterminating computation (e.g. $^2\text{-append}([1, 2|1_1], [5, 7|1_2], [1|1_3])$). However, the unification $[h|t] = t$ fails, due to the occur check, so that it is not a recursive cyclic collection (in fact, it is not even a cyclic collection).

Nevertheless, the condition of the existence of a common set of bindings for all the unifications involved remains interesting, if it can be reformulated for unification without occur check—so that examples of the *append*-type will be covered. In this paper, we restrict our attention to simplified collections of the *append*-type, $P(\dots) :- P(\dots)$. Notice that any cyclic collection,

$$\begin{aligned} P_1 &:- P'_2, B_1, \\ P_2 &:- P'_3, B_2, \\ &\vdots \\ P_n &:- P'_1, B_n, \end{aligned}$$

can be reduced to $P''_1 :- P'''_1, B$ by unfolding, and that within this clause, using the depth-first left-to-right computation rule, the initial part $P''_1 :- P'''_1$ contains the essence of the termination problem.

Our contribution, presented in Section 3, is that we formulate and prove a modified version of the conjecture of Smith et al. [11] in the absence of the occur check and for clauses of the type $P(\dots) :- P(\dots)$ using a unification based on (infinite) rational trees. In this way, the problems caused by the occur check can be avoided, and a necessary and sufficient condition for the existence of queries causing an infinite computation can be formulated. Also, we show how a slightly weaker, but more practical result can be obtained from this for standard unification (including the occur check).

In the last section, we formulate some consequences of this result. We also show how it relates to the techniques described by Devienne [4] and Lebegue [8], where directed, weighted graphs are used to make termination (in certain cases) decidable. We also indicate how a combination of our results and the formalism of directed, weighted graphs seems very promising to obtain efficient termination tests for clauses of the type $P(\dots) :- P(\dots)$.

2. Preliminaries on Rational Trees

Rational (or regular) trees have quite often been used in computer science to describe infinite computational processes. In [3] Courcelle gives a good overview of their properties. In the field of logic programming, the notion of the rational tree was introduced by Colmerauer in [2]. In this section, we recall some definitions and results from [2].

Definition 2.1. A *rational* or *regular* tree is a tree containing only a finite number of distinct subtrees.

Any finite tree is rational. Figure 1(a) contains an example of an infinite rational tree. It can also be represented using a finite, directed graph, as shown in Fig. 1(b). Yet another representation can be given using infinite data-structures, e.g. $f(x, g(a, f(x, g(\dots))))$, or as a finite set of equations, such as

$$\{y = f(x, g(a, y))\} \quad \text{or} \quad \{u = f(x, v), v = g(a, u)\}$$

and many others. From the example, it should be clear that many different systems of equations, $\{s_1 = t_1, \dots, s_n = t_n\}$, where s_i and t_i are PROLOG-terms, describe the same rational tree or set of rational trees. Also, observe that—generalizing the examples above—any system of equations of the form $\{x_1 = t_1, \dots, x_k = t_k\}$, where $x_1, \dots, x_k$ are distinct variables and $t_1, \dots, t_k$ are terms, describes a set of rational trees. In [2] a technique is given to determine the set of rational trees which is represented by a given set of equations.

Fig 1

Definition 2.2. A system of equations $\{s_1 = t_1, \dots, s_n = t_n\}$ is in *simplified form*, if:

- (1) It is of type $\{x_1 = t_1, \dots, x_n = t_n\}$, where the x_i are distinct variables and the t_i are terms. We say that the system is in *solvable simplified form*. Or:
- (2) It contains an equation of type $f(p_1, \dots, p_k) = g(q_1, \dots, q_m)$, where f and g are distinct functors and q_i, p_j are terms. We say that it is in *unsolvable simplified form*.

Definition 2.3. Given a system of equations S and a set of rational trees T , then T is a *rational tree solution* for S , if the trees of T satisfy all the equations of S .

A system in solvable simplified form has at least one rational tree solution, whereas a system in unsolvable simplified form has none.

There exist five basic transformations that allow to reduce any system of equations to a simplified form, without losing any of the system's solutions:

- *compaction*: eliminate any equation of the form $x = x$, where x is a variable,
- *variable elimination*: if x, y are distinct variables, $x = y$ is in the system and x has other occurrences in that system, then replace the other occurrences by y ,
- *variable anteposition*: if x is a variable and t is not a variable, then replace $t = x$ by $x = t$,
- *confrontation*: if x is a variable, t_1, t_2 are not variables and $|t_1| \leq |t_2|$, then replace $\{x = t_1, x = t_2\}$ by $\{x = t_1, t_1 = t_2\}$; by $|t|$ we denote the number of occurrences of elements from the underlying alphabet in t ,
- *splitting*: replace $\{f(p_1, \dots, p_k) = f(q_1, \dots, q_k)\}$ by $\{p_1 = q_1, \dots, p_k = q_k\}$.

The following basic property regarding these transformations holds. every repeated application of these transformations terminates after a finite number of steps. The result is an equivalent (same rational tree solutions) simplified form.

Finally, we introduce the following notation: let $\nu = \{x_1 = t_1, \dots, x_k = t_k\}$, where $x_1, \dots, x_k$ are distinct variables, and let s be any term, then $\nu(s)$ is a notation for $s\theta$, where θ is the substitution $\{x_1/t_1, \dots, x_k/t_k\}$.

3. Rational Trees and Termination

In this paper, we focus on Horn clauses of type $P(t_1, \dots, t_n) \rightarrow P(s_1, \dots, s_n)$, where s_i and t_i are PROLOG terms. To each such clause, we can associate a system of equations $\{s_1 = t_1, \dots, s_n = t_n\}$. We denote this system by S . Variable renaming will be expressed by adding superscript on the variable (e.g. x'_j is a renaming of x_j). Using s'_j as a notation for the term obtained by

renaming all the variables $x_1, \dots, x_k$ occurring in s_j to $x'_1, \dots, x'_k$, the substitutions resulting from performing m resolution steps with the given clause and for the query $?-P(s_1^1, \dots, s_n^1)$ are identical to the finite tree solutions of the system of equations:

$$\begin{aligned} \{s_1^1 &= t_1^2, \dots, s_n^1 = t_n^2, \\ s_1^2 &= t_1^3, \dots, s_n^2 = t_n^3, \\ &\vdots \\ s_1^m &= t_1^{m+1}, \dots, s_n^m = t_n^{m+1}\}. \end{aligned}$$

We denote this system of equations by S_m . The main result of the paper can now be formulated as:

Theorem 3.1. *The system of equations S has a rational tree solution, if and only if for every $m > 0$, the system S_m has a rational tree solution.*

Proof. We start off with the *only if* part. Assume that S has a solution:

$$\nu = \{x_1 = r_1, \dots, x_k = r_k\}.$$

We may assume that $x_1, \dots, x_k$ are *all* the distinct variables occurring in S . If this is not the case, then we can add equations of the type $x = x$ to ν , for every x occurring in S but not in $\{x_1, \dots, x_k\}$. Then, for each $i = 1, \dots, m$, the system $\{s_1^i = t_1^{i+1}, \dots, s_n^i = t_n^{i+1}\}$ has a rational tree solution:

$$\nu_i = \{x_1^i = r_1, \dots, x_k^i = r_k, x_1^{i+1} = r_1, \dots, x_k^{i+1} = r_k\},$$

because: (1) all the variables at the left-hand side of the equations are distinct, (2) the rational tree is a solution of the given equations, since $\nu_i(s_l^i) = \nu(s_l) = \nu(t_l) = \nu_i(t_l^{i+1})$ for all $l = 1, \dots, n$.

Finally, the system S_m , $m > 0$, has the rational tree solution:

$$\nu^m = \{x_l^i = r_l : l = 1, \dots, k \text{ and } i = 1, \dots, m + 1\}.$$

The proof for the *if* part of the result is more complicated. We assume that S_m is solvable for any $m > 0$ and that S is unsolvable. From these, we derive a contradiction.

First a definition: The unindexed equivalent of a system of equations is obtained by removing all superscripts of all variables (x'_j becomes x_j). As an example, S is the unindexed equivalent of S_1 . Notice the one-to-one correspondence between the equations in a system and in the unindexed equivalent.

Next, besides the five basic transformations of Section 2, we introduce two auxiliary transformations. The first one, called *removal* consists of removing one equation from a given system of equations ($\{s_1 = t_1, s_2 = t_2, \dots\}$ is transformed to $\{s_2 = t_2, \dots\}$). Obviously, removal does not lead to an equivalent system; the transformed system does not yield the same set of rational tree solutions. However, the transformation does preserve solvability. The second transformation, called *duplication*, consists of duplicating one equation of the system ($\{s_1 = t_1, \dots\}$ is transformed to $\{s_1 = t_1, s_1 = t_1, \dots\}$). Again, duplication preserves solvability.

A final concept which we introduce before proceeding with the proof is that of *augmentation*. Given a system of equations R , such that all the variables in R are adorned with superscripts, we derive a new system, $Aug(R, k)$, by augmenting all superscripts of variables in R by the constant k . This operation can be performed both for positive and negative k -values. But, for $k < 0$, we must take care that all superscripts of variables occurring in R are larger than $-k$. Observe that if R can be obtained from a system S_p , through the application of a sequence $(T_1, T_2, \dots, T_l)$ of transformations (possibly including auxiliary ones), then $Aug(R, k)$ can be obtained by a corresponding sequence of transformations $(T'_1, T'_2, \dots, T'_l)$ from the system S_{p+k} . Also, augmentation preserves solvability.

We now prove, by induction, that the following statement holds for every $q \geq 0$

- (a) there exists a sequence of basic transformations (compaction, ..., splitting), $(T_1, T_2, \dots, T_q)$, which are consecutively applicable to S , and
- (b) let $R = T_q(T_{q-1}(\dots(T_1(S))\dots))$, then there exists a system of equations R_1 , such that
 - R is the unindexed equivalent of R_1 , and
 - R_1 is derivable from some S_p , through a sequence of transformations (possibly including auxiliary ones)

If this statement holds, then there exists an infinite sequence of basic transformations, consecutively applicable to S . But, since the transformation process must terminate, this is a contradiction.

To prove the statement, first consider the base case, $q = 0$. The sequence of transformations is empty, so $R = S$. With $R_1 = S_1$ (R_1 is derived from S_1 by the empty sequence of transformations), obviously R is an unindexed equivalent of R_1 .

For the induction step, assume that the sequence of basic transformations $(T_1, T_2, \dots, T_q)$ has been applied to S , resulting in a system R . Also, assume that R is the unindexed equivalent of R_1 and that R_1 is obtained from S_p through a sequence of (general) transformations.

If there are no transformations applicable to R , then R must be in its simplified form. But, since we assumed that S is unsolvable, R contains an

equation of the type $f(s_1, \dots, s_{n_1}) = g(t_1, \dots, t_{n_2})$, where $f \neq g$. However, this implies that R_1 also contains an equation of this type, which contradicts the solvability of R_1 .

Let T_{q+1} be the additional transformation and denote $R' = T_{q+1}(R)$. We prove that there exists a system R'_1 derivable from some $S_{p'}$ and such that R' is the unindexed equivalent of R'_1 . This is done using the following case study on the type of T_{q+1} :

(1) T_{q+1} is *compaction*. If $x = x$ is the equation eliminated from R , we can apply removal on the corresponding equation $x' = x'$ from R_1 , obtaining the system R'_1 . Clearly, R' and R'_1 satisfy the conditions.

(2) T_{q+1} is *anteposition*. By performing anteposition on $t = x$ in R , we obtain $x = t$ and R' . Performing anteposition on the corresponding equation $t' = x'$ of R_1 results in $x' = t'$ and R'_1 .

(3) T_{q+1} is *splitting*. This is completely analogous to the cases (1) and (2).

(4) T_{q+1} is *confrontation*. In R we have the equations $x = t_1$ and $x = t_2$, with t_1, t_2 nonvariables and $|t_1| \leq |t_2|$. In R_1 we have the corresponding equations $x' = t'_1$ and $x' = t'_2$. If $i = j$, then the corresponding confrontation can be applied to R_1 and the new systems R' and R'_1 are obtained. If $i \neq j$, then without loss of generality we may assume that $k = i - j > 0$. The system $Aug(R_1, k)$ contains the equations $x'^{i+k} = t''_1$ and $x'^{j+k} = t''_2$ (where $j + k = i$). Moreover, $Aug(R_1, k)$ is derivable from S_{p+k} . We then define $R_{1,k}$ to be the system of equations obtained from R_1 by replacing the equation $x' = t'_2$ by $x' = t''_2$.

In order to show that $R_{1,k}$ can still be derived through a sequence of (general) transformations from S_{p+k} , we need the *duplication* and *removal* transformations. First, we apply a number of successive duplications on S_{p+k} to obtain a new system Q containing both S_{p+k} and (duplicated) copies of all the equations in S_p ($S_p \subset S_{p+k}$). From the S_{p+k} -part of Q , we can derive $Aug(R_1, k)$, leaving the S_p -part of Q untouched. Then, from the S_p -part of Q we derive R_1 . Finally, $R_{1,k}$ is obtained by removing the equation $x' = t'_2$ from the R_1 -part of the transformation of Q and by removing all the equations from the $Aug(R_1, k)$ -part of the transformed Q , except for $x' = t''_2$.

Clearly, R is the unindexed equivalent of $R_{1,k}$, so, by applying confrontation to both R and $R_{1,k}$, we obtain the systems R' and R'_1 .

(5) T_{q+1} is *variable elimination*. This is the most difficult case. Assume that $x = y$ is contained in R and that in addition R contains at least one occurrence of x . In R_1 , we have the equation $x' = y'$ and at least one additional occurrence of a variable x' . Let $x^{k_1}, x^{k_2}, \dots, x^{k_r}$ be all the distinct renamings of x occurring in R_1 (not counting the occurrence of x' in $x' = y'$, but counting possible other occurrences of x'). Without loss of generality we assume that $k_1 \leq k_2 \leq \dots \leq i \leq \dots \leq k_r$.

The idea is that we want to construct a new system, R_{1+} , including all the equations of R_1 and, in addition, including one equation $x^{k_n} = y^{l_n}$ for each

$n = 1, \dots, r$. Here, the values of the superscripts l_n will be of no importance (we only need that they exist). If such a system can be derived through a sequence of (general) transformations from some $S_{p'}$, then we can apply variable elimination on each equation $x^{k_n} = y^{l_n}$ in R_{1+} and by consecutively removing all the equations $x^{k_n} = y^{l_n}$ again, we obtain a system R'_1 , having the desired properties.

Generating the equations $x^{k_n} = y^{l_n}$ can be done by augmentation on R_1 , using the equation $x^i = y^j$ (similar to what was described in the previous case). Proving that the system R_{1+} can be derived from some $S_{p'}$ is also similar to what was done in the previous case. We start from a system S_{p+k} , such that $p+k$ is the largest index involved in any of the augmentations. Then, we duplicate equations, to obtain copies of each of the smaller $S_{p+k'}$ involved in the different augmentations, we apply sequences of transformations to each of the subsets of equations independently and, finally, we remove all undesired equations.

There is only one problem, if R_1 contains variables with superscripts smaller than $i - k_1$, then the augmentation $Aug(R_1, k_1 - i)$ cannot be constructed.

To solve this, we start by applying augmentation with factor $i - k_1$ to R_1 . In this way we obtain a system $R_{1+(i-k_1)}$ ($= Aug(R_1, i - k_1)$) which can take the role of R_1 without having the above problem (R is the unindexed equivalent of $R_{1+(i-k_1)}$ and $R_{1+(i-k_1)}$ can be derived from $S_{p+(i-k_1)}$ through a finite number of transformations).

Also, $R_{1+(i-k_1)}$ contains the equation $x^{2i-k_1} = y^{i+i-k_1}$ and occurrences of the variables $x^i, x^{k_2+i-k_1}, \dots, x^{k_r+i-k_1}$. Then, apply the steps described above for R_1 on $R_{1+(i-k_1)}$ instead. This produces R'_1 . $\square$

Informally stated, this theorem says that the solvability of the set of equations S , in terms of rational trees, is a necessary and sufficient condition for the existence of a query $?-Q$, such that $?-Q$ gives rise to a nonterminating computation applying the SLD-refutation, depth-first left-to-right computation rule and ignoring the occur check. In the presence of the occur check, Theorem 3.1 can be split up into two corollaries:

Corollary 3.2. *If the system of equations S is solvable in terms of rational trees, then there exists a query $?-Q$ such that either*

- $?-Q$ causes a nonterminating computation, or
- $?-Q$ finitely fails due to the occur check

Corollary 3.3. *If $P(\dots) :- P(\dots)$ admits a query $?-Q$ causing a nonterminating computation, then the system S is solvable in terms of rational trees*

The idea of the proof is that in each case we may assume that Q is of the form $P(t_1^1, \dots, t_n^1)$. For this Q to cause a nonterminating computation in the

presence of the occur check, is equivalent to the condition that S_m admits a *finite* rational tree solution (meaning that every tree in the rational tree solution is finite), for every $m > 0$. From this observation and Theorem 3.1, the results are obvious.

It is important to notice that the solvability condition on S is decidable, since the simplification process terminates. Also, observe that we cannot simply decide between the two possible conclusions of Corollary 3.2 (nontermination or failure due to the occur check) on the basis of the existence of a *finite* rational tree solution for S itself. The *append* example illustrates that we can have systems S without finite rational tree solutions, but such that the query $?-P(t_1^1, \dots, t_n^1)$ does not terminate (i.e. every system S_m has a finite rational tree solution).

4. Discussion

An interesting consequence of Theorem 3.1 is that the solvability condition is invariant under the application of any of the transformations of Section 2. This means that if you have a clause of type $P(t_1, \dots, t_n) :- P(s_1, \dots, s_n)$, for which there exists a query with a nonterminating computation, then applying any sequence of the transformations of Section 2 to the system $\{s_1 = t_1, \dots, s_n = t_n\}$ will give rise to a new clause $P(t'_1, \dots, t'_m) :- P(s'_1, \dots, s'_m)$, which also admits a query causing either an infinite computation or which finitely fails due to the occur check. This is trivial for the transformations *compaction* and *splitting*, because the computed substitutions in each resolution step are identical for the original and the transformed clause. But, for the three remaining transformations, the set of equations S_m obtained for the original clause is very different from the set S'_m for the transformed clause.

However, Theorem 3.1 is mainly of theoretical (existence) value. The most we can obtain from Theorem 3.1 in order to determine the behaviour for a concrete query, is a sufficient condition under which the given query will cause a nonterminating computation or fails due to the occur check

Corollary 4.1. *Let $?-P(r_1, \dots, r_n)$ be a query to $P(s_1, \dots, s_n) :- P(t_1, \dots, t_n)$. If the system:*

$$S_r = \{r_1 = s_1, \dots, r_n = s_n\}, \quad s_1 = t_1, \dots, s_n = t_n\}$$

has a rational tree solution, then the query causes a nonterminating computation or fails due to the occur check.

Proof. The proof is a slightly adapted version of the proof for the *only if* part of the proof of Theorem 3.1. Let

$$\nu = \{y_1 = p_1, \dots, y_{k_1} = p_{k_1}, x_1 = q_1, \dots, x_{k_2} = q_{k_2}\}$$

describe a rational tree solution for S_i , where we assume that $y_1, \dots, y_{k_1}$ are *all* the distinct variables occurring in $r_1, \dots, r_n$ and $x_1, \dots, x_{k_2}$ are *all* the distinct variables occurring in $s_1, \dots, s_n, t_1, \dots, t_n$ (see Theorem 3.1). Then,

$$\nu_0 = \{y_1 = p_1, \dots, y_{k_1} = p_{k_1}, x_1^1 = q_1, \dots, x_{k_2}^1 = q_{k_2}\}$$

is a rational tree solution for $\{r_1 = s_1^1, \dots, r_n = s_n^1\}$ and, for all $i = 1, \dots, m$,

$$\nu_i = \{x_1' = q_1, \dots, x_{k_2}' = q_{k_2}, x_1^{i+1} = q_1, \dots, x_{k_2}^{i+1} = q_{k_2}\}$$

is a rational tree solution for $\{s_1' = t_1^{i+1}, \dots, s_n' = t_n^{i+1}\}$ (see Theorem 3.1). Clearly, these solutions can be combined to obtain a rational tree solution for $S_m \cup \{r_1 = s_1, \dots, r_n = s_n\}$ for any $m > 0$. $\square$

Unfortunately, this is a sufficient condition which is much too strong. Reconsidering the clause.

$$\text{append}([h|t], l, [h|s]) . - \text{append}(t, l, s)$$

Corollary 4.1 allows us to determine that the query $?-\text{append}([1, 1|x_1], [3, 5|x_2], x_3)$ will cause a nonterminating computation (or fail due to the occur check), but it does not allow us to draw any conclusions for the query $?-\text{append}(1, 2|x_1], [3, 5|x_2], x_3)$.

The reason is that the rational tree solution for S is $\{t = [h|t], s = [h|s]\}$ (i.e. s and t are structures of the type $[h, h, h, \dots]$).

In order to establish more practical results, one needs nonrational trees. In fact, the structure one needs to start an analysis from, would be the infinite system of equations $S_\infty = \bigcup \{S_m, m > 0\}$.

For the *append* clause, the infinite system of equations S_∞ is:

$$\begin{aligned} \{t_1 = [h_2|t_2], s_1 = [h_2|s_2], \dots, t_n = [h_{n+1}|t_{n+1}], \\ s_n = [h_{n+1}|s_{n+1}], \dots\}. \end{aligned}$$

It can also be represented as the pair of infinite trees in Fig. 2(a).

At each consecutive level of depth within these trees, we obtain a subtree which is a renaming of the tree at the previous level. Therefore, systems of the type S_∞ can be represented using a set of *weighted, directed graphs* instead (see Fig. 2(b)). Here, each node (e.g. h) may represent an infinite number of different terms (e.g. $\{h_i : i > 0\}$). In order to obtain the indices of the variables that are really represented, one starts at the root with index 1 and augments the index by the values denoted as labels on the arcs. It is clear that a result, similar

Fig 2

to Corollary 4.1, but for weighted directed graphs, will be able to detect a much larger class of problem queries. In fact, such a result would detect all problem queries in the *append* example.

Devienne and Lebegue have introduced and thoroughly discussed weighted, directed graphs in [4–6, 8] including their use for the determination of looping queries. They also formulate a condition similar to Theorem 3.1, using weighted, directed graphs instead of rational trees. With their result, the distinction between the existence of a nonterminating computation and one which fails due to the occur check is decidable. This is not the case when rational trees are used. However, our result has the advantage of relating nontermination to the more simple and elegant notion of the rational tree. Also, the result is less trivial for rational trees, since the existence of a weighted, directed graph is in fact obviously equivalent to the solvability of S_∞ and therefore to the existence of an infinite computation (without occur check).

Alternatively, one can apply techniques for loop detection at run time (e.g. [1, 13]). The syntactic approach had the advantage of avoiding overhead, but the class of programs for which termination is decidable on a syntactic basis is very limited. Also, there are techniques that detect termination at compile time based on a combination of abstract interpretation and methods for solving systems of inequations to prove that the length of certain data-structures decreases during the computation (e.g. [7, 12]). Again, these techniques (although certainly more powerful) are computationally more expensive. Another alternative is the control generation described in [10]. This approach is of a much more practical nature than the ones mentioned above. There is some disagreement in the literature on the complexity of the algorithms involved. Still, in that paper no necessary or sufficient conditions for termination are formulated and proved. It describes a practical (more heuristic) approach to loop avoidance.

ACKNOWLEDGEMENT

We wish to thank the referees for several very useful comments on this work.

REFERENCES

- 1 Brough, D. R. and Walker, A., Some practical properties of logic programming interpreters, in *Proceedings Fifth Generation Computer Systems*, Tokyo (1984) 149–156
- 2 Colmerauer, A., Prolog and infinite trees, in L. Clark and S. A. Tarnlund eds., *Logic Programming* (Academic Press, New York, 1982) 173–185
- 3 Courcelle, B., Fundamental properties of infinite trees, *Theor. Comput. Sci.* 17 (1983) 95–169
- 4 Devienne, P., Les graphes orientés pondérés. Un outil pour la programmation logique, Ph.D. Thesis, Université de Lille I (1987)
- 5 Devienne, P., Weighted graphs: a tool for expressing the behaviour of recursive rules in Logic Programming, in *Proceedings Fifth Generation Computer Systems*, Tokyo (1988) 397–404
- 6 Devienne, P. and Lebegue, P., Weighted graph: A tool for logic programming, in *CAAP '86*, Lecture Notes in Computer Science 214 (Springer, Berlin, 1986) 100–111
- 7 Kanamori, T., Kawamura, T. and Horiuchi, K., Detecting termination of logic programs based on abstract hybrid interpretation, Research Report, Mitsubishi Electric Corporation, Amagasaki Hyogo, Japan (1988)
- 8 Lebegue, P., Contribution à l'étude de la programmation logique par les graphes orientés pondérés, Ph.D. Thesis, Université de Lille I (1987)
- 9 Minker, J. and Nicolas, J., On recursive axioms in deductive databases, *Inf. Syst.* 8 (1) (1983) 1–13
- 10 Naish, L., Automating control for logic programs, *J. Logic Programming* 2 (1985) 167–183
- 11 Smith, D. E., Genesereth, M. R. and Ginsberg, M. L., Controlling recursive inference, *Artificial Intelligence* 30 (1986) 343–389
- 12 Ullman, J. D. and van Gelder, A., Efficient tests for top-down termination of logical rules, *J. ACM* 35 (1988) 345–373
- 13 van Gelder, A., Efficient loop-detection in Prolog using the tortoise and hare technique, *J. Logic Programming* 4 (1987) 23–31

Received February 1989; revised version received April 1989