Ensuring the Quasi-Termination of Needed Narrowing Computations [⋆]

J. Guadalupe Ramos [a] Josep Silva [b] Germán Vidal [b,∗]

[a]*I.T. La Piedad, Av. Tecnológico 2000, La Piedad, Mich., México*

[b]*DSIC, U. Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain*

Abstract

We present a characterization of first-order functional programs which are quasi-terminating w.r.t. the symbolic execution mechanism of needed narrowing, i.e., computations in these programs consist of a sequence of finitely many different function calls (up to variable renaming). Quasi-terminating programs are particularly useful for program analysis and transformation, since in this context quasi-termination often amounts to full termination.

Key words: Programming languages, functional programming, formal semantics

1 Introduction

The notion of *quasi-termination* can be traced back to term rewriting [2], where a system is called quasi-terminating if all its derivations contain only a finite number of distinct terms [3]. Later, Holst [4] introduced a similar notion in the context of (offline) partial evaluation of functional programs [5], where quasi-termination implies the full termination of the specialization process.

Many program analysis and transformations for functional programs deal with *incomplete* information, which is usually represented by means of terms containing free variables. Therefore, some form of *symbolic* computation is required. Here, the choice of *narrowing* [6] arises quite naturally, since it extends pure functional reduction with the ability to deal with *logic* variables by replacing pattern matching with unification. Currently, needed narrowing [7] is the strategy that presents better properties.

Our aim in this work is the characterization of a class of rewrite systems that ensures the quasi-termination

[⋆] Some preliminary material of this note appeared in ICFP 2005 [1].
This work has been partially supported by the EU (FEDER) and the Spanish MEC under grants TIN2004-00231 and TIN2005-09207-C03-02.
[∗] Corresponding author.

of needed narrowing computations. These systems can then be used for partial evaluation or other similar program analysis and transformations while ensuring the termination of the process. Unfortunately, ensuring the (quasi-)termination of needed narrowing computations is not an easy task. In particular, existing results from term rewriting do not transfer to narrowing. Consider, e.g., the following rewrite system which defines the addition on natural numbers (built from *zero* and *succ*):

$$\begin{aligned} add(zero, y) &\to y \\ add(succ(x), y) &\to succ(add(x, y)) \end{aligned}$$

In this system, any evaluation by rewriting will terminate. Needed narrowing, however, might not terminate; consider, e.g., the following computation where the second rule is always selected for unfolding the call to *add*:

$$\begin{array}{l} add(x, zero) \\ \quad \leadsto_{\{x \mapsto succ(x')\}} \; succ(add(x', zero)) \\ \quad \leadsto_{\{x' \mapsto succ(x'')\}} \; \cdots \end{array}$$

Here, each needed narrowing step is labeled with the computed binding for the free variable of the term.

In the following, we characterize a subclass of so-called *inductively sequential* systems that guarantees the quasi-termination of needed narrowing computations (Section 3). Our characterization is purely syntactical and, thus, straightforward to check. Then, in Section 4, we compare our characterization to some related works. Finally, in Section 5 we conclude and discuss some potential applications of our characterization.

2 Preliminaries

A set of rewrite rules $l \to r$ such that l is a nonvariable term and r is a term whose variables appear in l is called a *term rewriting system* (TRS for short); terms l and r are called the left-hand side and the right-hand side of the rule, respectively. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined* symbols $\mathcal{D}$ are the root symbols of the left-hand sides of the rules and the *constructors* are $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. We restrict ourselves to finite signatures and TRSs. We denote the domain of terms and *constructor terms* by $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{C}, \mathcal{V})$, resp., where $\mathcal{V}$ is a set of variables with $\mathcal{F} \cap \mathcal{V} = \varnothing$.

A TRS $\mathcal{R}$ is *constructor-based* if the left-hand sides of its rules have the form $f(s_1, \ldots, s_n)$ where s_i are constructor terms, i.e., $s_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$, for all $i = 1, \ldots, n$. The set of variables appearing in a term t is denoted by $\mathcal{V}ar(t)$. A term t is *linear* if every variable of $\mathcal{V}$ occurs at most once in t. $\mathcal{R}$ is left-linear (resp. right-linear) if l (resp. r) is linear for all rules $l \to r \in \mathcal{R}$. The *definition* of f in $\mathcal{R}$ is the set of rules in $\mathcal{R}$ whose root symbol in the left-hand side is f. A function $f \in \mathcal{D}$ is left-linear (resp. right-linear) if the rules in its definition are left-linear (resp. right-linear).

A term t is *operation-rooted* (resp. *constructor-rooted*) if it has the form $h(t_1, \ldots, t_n)$ with $h \in \mathcal{D}$ (resp. $h \in \mathcal{C}$). A *position* p in a term t is represented by a sequence of natural numbers, where ϵ denotes the root position (positions are used to address the nodes of a term viewed as a tree). $t|_p$ denotes the *subterm* of t at position p and $t[s]_p$ denotes the result of *replacing the subterm* $t|_p$ by the term s. A term t is *ground* if $\mathcal{V}ar(t) = \emptyset$. A term t is a *variant* of term t' if they are equal modulo variable renaming. A *substitution* σ is a mapping from variables to terms such that its domain $\mathcal{D}om(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is finite. The identity substitution is denoted by *id*. Term t' is an *instance* of term t if there is a substitution σ with $t' = \sigma(t)$. A *unifier* of two terms s and t is a substitution σ with $\sigma(s) = \sigma(t)$. In the following, we write $\overline{o_n}$ for the *sequence of objects* $o_1, \ldots, o_n$.

Inductively sequential TRSs [8] are a subclass of left-linear constructor-based TRSs. Essentially, a TRS is *inductively sequential* when all its operations are defined by rewrite rules that, recursively, make on their arguments a case distinction analogous to a data type (or structural) induction. Inductive sequentiality is not a limiting condition for programming. In fact, the first-order component of many functional programs written in, e.g., Haskell or ML, are inductively sequential.

Example 1 *Consider the following rules which define the less-or-equal function on natural numbers:*

$$\begin{array}{rcll} zero & \leqslant & y & \to & true \\ succ(x) & \leqslant & zero & \to & false \\ succ(x) & \leqslant & succ(y) & \to & x \leqslant y \end{array}$$

This function is inductively sequential since its left-hand sides can be hierarchically organized as in Fig. 1, where arguments in a box denote a case distinction (this is similar to the notion of definitional tree in [8]).

2.1 Semantics

The evaluation of ground terms w.r.t. a TRS is formalized with the notion of *rewriting*. A *rewrite step* is an application of a rewrite rule to a term, i.e., $t \to_{p,R} s$ if there exists a position p in t, a rewrite rule $R = (l \to r)$ and a substitution σ with $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$ (p and R will often be omitted in the notation of a reduction step). The instantiated left-hand side $\sigma(l)$ is called a *redex*. A term t is called *irreducible* or in *normal form* if there is no term s with $t \to s$.

Symbolic computations mainly differ from purely functional computations in that function calls may contain *free* variables. To evaluate such terms containing variables, we use the narrowing relation [6], which nondeterministically instantiates the variables such that a rewrite step is possible (see [9] for a survey). Formally, $t \leadsto_{p,R,\sigma} t'$ is a *narrowing step* iff p is a nonvariable position of t and $\sigma(t) \to_{p,R} t'$ (we sometimes omit p, R and/or σ when they are clear from the context). The computed substitution σ is often the *most general unifier* of $t|_p$ and the left-hand side of (a variant of) R, restricting its domain to $\mathcal{V}ar(t)$; nevertheless, some narrowing strategies (e.g., needed narrowing [7]) compute uni-

$$\boxed{n} \leqslant m \Longrightarrow \begin{cases} \mathit{zero} \leqslant m \\ \mathit{succ}(x) \leqslant \boxed{m} \Longrightarrow \begin{cases} \mathit{succ}(x) \leqslant \mathit{zero} \\ \mathit{succ}(x) \leqslant \mathit{succ}(y) \end{cases} \end{cases}$$

Fig. 1. Case distinction for function "$\leqslant$"

fiers which are not always the most general, see below.

As in proof procedures for logic programming, we assume that the rules of the TRS always contain fresh variables if they are used in a narrowing step. We denote by $t_0 \leadsto^*_\sigma t_n$ a sequence of narrowing steps $t_0 \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} t_n$ with $\sigma = \sigma_n \circ \cdots \circ \sigma_1$ (if $n = 0$ then $\sigma = id$).

Example 2 *Consider again the addition function on natural numbers:*

$$\mathit{add}(\mathit{zero}, y) \rightarrow y$$
$$\mathit{add}(\mathit{succ}(x), y) \rightarrow \mathit{succ}(\mathit{add}(x, y))$$

Given the term $\mathit{add}(x, \mathit{succ}(\mathit{zero}))$, narrowing nondeterministically performs the derivations shown in Fig. 2; here, only the computation of most general *unifiers is considered.*

In order to avoid unnecessary computations and to deal with infinite data structures, a demand-driven generation of the search space has been advocated by a number of *lazy* narrowing strategies [10–12]. Due to its optimality properties w.r.t. the length of derivations and the number of computed solutions, *needed narrowing* [7] is currently the best lazy narrowing strategy.

We say that $s \leadsto_{p,R,\sigma} t$ is a *needed narrowing step* iff $\sigma(s) \rightarrow_{p,R} t$ is a *needed rewrite* step in the sense of Huet and Lévy [13], i.e., in every computation from $\sigma(s)$ to a normal form, either $\sigma(s)|_p$ or one of its *descendants* must be reduced. Here, we are interested in a particular needed narrowing strategy, denoted by λ in [7, Def. 13], which is based on the notion of a *definitional tree* [8] (a hierarchical structure containing the rules of a function definition, which is used to guide the needed narrowing steps). This strategy is basically equivalent to *lazy narrowing* [12] where narrowing steps are applied to the outermost function, if possible, and inner functions are only narrowed if their evaluation is *demanded* by a constructor symbol in the left-hand side of some rule (i.e., a typical call-by-name evaluation strategy).

Example 3 *Consider again the rules defining function "$\leqslant$" of Example 1. In a term like $t_1 \leqslant t_2$, needed narrowing proceeds as follows: First, t_1 should be evaluated to some* head normal form *(i.e., a free variable or a constructor-rooted term) since all three rules defining "$\leqslant$" have a non-variable first argument. Then,*

(a) If t_1 evaluates to zero then the first rule is applied.

(b) If t_1 evaluates to $\mathit{succ}(t_1')$ then t_2 is evaluated to head normal form:

$$
\begin{array}{lll}
1)\ add(x, succ(zero)) & \leadsto_{\epsilon, R_1, \{x \mapsto zero\}} & succ(zero) \\[1ex]
2)\ add(x, succ(zero)) & \leadsto_{\epsilon, R_2, \{x \mapsto succ(y_1)\}} & succ(add(y_1, succ(zero))) \\
 & \leadsto_{1, R_1, \{y_1 \mapsto zero\}} & succ(succ(zero)) \\[1ex]
3)\ add(x, succ(zero)) & \leadsto_{\epsilon, R_2, \{x \mapsto succ(y_1)\}} & succ(add(y_1, succ(zero))) \\
 & \leadsto_{1, R_2, \{y_1 \mapsto succ(y_2)\}} & succ(succ(add(y_2, succ(zero)))) \\
 & \leadsto_{1.1, R_1, \{y_2 \mapsto zero\}} & succ(succ(succ(zero))) \\[1ex]
\quad \ldots & &
\end{array}
$$

Fig. 2. Narrowing derivations for the term $add(x, succ(zero))$

(1) If t_2 evaluates to zero then the second rule is applied.
(2) If t_2 evaluates to $succ(t'_2)$ then the third rule is applied.
(3) If t_2 evaluates to a free variable, then it is instantiated to a constructor-rooted term, here zero or $succ(x)$ and, depending on this instantiation, we proceed as in cases (1) or (2) above.

(c) Finally, if t_1 evaluates to a free variable, needed narrowing instantiates it to a constructor-rooted term (zero or $succ(x)$). Depending on this instantiation, we proceed as in cases (a) or (b) above.

A precise definition of inductively sequential TRSs and needed narrowing is not necessary in this work (the interested reader can find detailed definitions in [7,8]). In the following, we use *needed narrowing* to refer to the particular strategy λ in [7, Def. 13].

3 Ensuring Quasi-Termination

In this section we introduce a sufficient condition for TRSs so that needed narrowing computations are always quasi-terminating. First, we need the following preparatory definitions:

Definition 4 *Given a TRS $\mathcal{R}$, its* graph of functional dependencies*, in symbols $\mathcal{G}(\mathcal{R})$, contains nodes labeled with the function symbols in $\mathcal{D}$ and there is an arrow from node f to node g iff there is a call to g from the right-hand side of some rule in the definition of f.*

Definition 5 *Let $\mathcal{R}$ be a TRS. A function $f \in \mathcal{D}$ is* cyclic *if node f belongs to a cycle in $\mathcal{G}(\mathcal{R})$ and it is* noncyclic *otherwise.*

Example 6 *Consider, for instance, the following TRS $\mathcal{R}$:*

$$
\begin{aligned}
f(s(x), y) &\to h(x, y) \\
h(0, y) &\to y \\
h(s(x), y) &\to c(i(x), h(x, y)) \\
i(x) &\to x
\end{aligned}
$$

where $f, h, i \in \mathcal{D}$ are defined functions and $0, s, c \in \mathcal{C}$ are constructors. The associated graph of functional de-

pendencies, $\mathcal{G}(\mathcal{R})$, is as follows:

$$\begin{tikzcd} \boxed{f} \arrow[r] & \boxed{h} \arrow[loop above] \arrow[r] & \boxed{i} \end{tikzcd}$$

Thus, functions f and i are noncyclic, while h is cyclic.

Clearly, noncyclic functions cannot introduce nonterminating (nor non-quasi-terminating) computations as long as the cyclic functions do not introduce them. Thus, we turn our attention to cyclic functions. Following [14], the *depth of a variable x in a constructor term t*, in symbols $dv(t, x)$, is defined in Fig. 3, where $c \in \mathcal{C}$ is a constructor term with arity $n \geqslant 0$.

Now, we introduce the notion of *nonincreasing* function, i.e., a function that always *consume* its parameters or leave them unchanged:

Definition 7 *Let $\mathcal{R}$ be a left-linear, constructor-based TRS. A function $f \in \mathcal{D}$ is* nonincreasing *iff each rule $f(\overline{s_n}) \to r$ in the definition of f fulfills the following conditions:*

(1) the right-hand side does not contain nested defined function symbols (i.e., defined function symbols that occur inside other defined function symbols), and
(2) $dv(s_i, x) \geqslant dv(t_j, x)$ for all operation-rooted subterms $g(\overline{t_m})$ in r, where $i \in \{1, \ldots, n\}$, $x \in \mathcal{V}ar(s_i)$, and $j \in \{1, \ldots, m\}$.

Example 8 *A function defined by the single rule*

$$f(x, y, s(z)) \to c(g(x), h(z))$$

with $s, c \in \mathcal{C}$ and $f, g, h \in \mathcal{D}$, is nonincreasing since the following relations hold:

$$\begin{aligned} dv(x,x) = 0 &\geqslant 0 = dv(x,x) \\ dv(x,x) = 0 &\geqslant -1 = dv(z,x) \\ dv(y,y) = 0 &\geqslant -1 = dv(x,y) \\ dv(y,y) = 0 &\geqslant -1 = dv(z,y) \\ dv(s(z),z) = 1 &\geqslant -1 = dv(x,z) \\ dv(s(z),z) = 1 &\geqslant 0 = dv(z,z) \end{aligned}$$

i.e., variable x is just copied, variable y vanishes, and (the depth of) variable z decreases.

Analogously to [3], we say that a TRS is *quasi-terminating for a set of terms T w.r.t. needed narrowing* iff all needed narrowing derivations issuing from the terms in T are quasi-terminating. Now, we give a sufficient condition for quasi-termination:

Definition 9 *Let $\mathcal{R}$ be an inductively sequential TRS. $\mathcal{R}$ is* nonincreasing *iff all functions $f \in \mathcal{D}$ are right-linear and either noncyclic or nonincreasing.*

The restriction to inductively sequential TRSs is not really necessary (i.e., left-linear, constructor TRSs would suffice) but we impose this condition because needed narrowing is only defined for this class of TRSs.

On the other hand, right-linearity is not only necessary to guarantee quasi-termination but also for ensuring that no repeated computations are introduced by function unfolding. Consider, e.g., the following

$$\begin{aligned}
dv(c(\overline{t_n}), x) &= \;\; 1 + max(\overline{dv(t_n, x)}) && \text{if } x \in \mathcal{V}ar(c(\overline{t_n})) \\
dv(c(\overline{t_n}), x) &= -1 && \text{if } x \notin \mathcal{V}ar(c(\overline{t_n})) \\
dv(y, x) &= \;\; 0 && \text{if } x = y \text{ and } y \in \mathcal{V} \\
dv(y, x) &= -1 && \text{if } x \neq y \text{ and } y \in \mathcal{V}
\end{aligned}$$

Fig. 3. Depth of a variable x in a constructor term t: $dv(t, x)$

nonincreasing functions:

$$\begin{aligned}
f(0, y) &\to y \\
f(s(x), y) &\to f(x, y) \\
g(x) &\to f(x, x)
\end{aligned}$$

where $0, s \in \mathcal{C}$ and $f, g \in \mathcal{D}$. This is not a nonincreasing TRS since function g is not right-linear. Thus, quasi-termination w.r.t. needed narrowing is not ensured:

$$\begin{aligned}
g(x) &\leadsto_{id} && f(x, x) \\
&\leadsto_{\{x \mapsto s(x')\}} && f(x', s(x')) \\
&\leadsto_{\{x' \mapsto s(x'')\}} && f(x'', s(s(x''))) \\
&\leadsto && \ldots
\end{aligned}$$

The correctness of our characterization is stated as follows:

Theorem 10 *If $\mathcal{R}$ is a nonincreasing TRS, then $\mathcal{R}$ is quasi-terminating for any linear term w.r.t. needed narrowing.*

In order to prove this theorem, we need some preliminary notations. Given a nonincreasing TRS $\mathcal{R}$ and a noncyclic function $f \in \mathcal{D}$, we denote by $path(f)$ the length of the longest path from f to either a sink node (i.e., a node with no output arrows) or to a cycle in $\mathcal{G}(\mathcal{R})$; for a cyclic function f, we let $path(f) = 0$. Now, we define the *complexity*, $comp(t)$, of a term t as the finite multiset $\langle path(f_1), \ldots, path(f_n) \rangle$, where $f_1, \ldots, f_n$ are the defined function symbols of t. Roughly speaking, the complexity of a term gives an indication of the *distance* from this term to a point where the computation either stops or enters a cycle.

We consider the well founded total ordering $<_{mul}$ over multiset complexities by extending the well founded ordering $<$ on $I\!N$ to the set $M(I\!N)$ of finite multisets over $I\!N$. The set $M(I\!N)$ is well founded under the ordering $<_{mul}$, since $I\!N$ is well founded under $<$. Let C, C' be multiset complexities, then $C <_{mul} C' \;\Leftrightarrow\; \exists X \subseteq C, X' \subseteq C'$ such that $C = (C' - X') \cup X$ and $\forall n \in X,\; \exists n' \in X'.\; n < n'$. We let $C \leqslant_{mul} C'$ if either $C = C'$ or $C <_{mul} C'$.

Given a term t, its *depth*, $depth(t)$, is defined in Fig. 4. This notion will be used in the proof below to establish a *bound* on the depth of the computed terms.

PROOF. In order to prove the claim, we prove that there is a bound on the *depth* of the computed terms, so that only finitely many different

$$\begin{array}{lll} depth(x) = 0 & & \text{if } x \in \mathcal{V} \\ depth(h(s_1,\ldots,s_m)) = 1 + max(depth(s_1),\ldots,depth(s_m)) & & \text{if } h \in \mathcal{F},\ m \geqslant 0 \end{array}$$

Fig. 4. Depth of a term t: *depth*(t)

terms (modulo variable renaming) can be obtained by needed narrowing.

Consider an arbitrary (possibly infinite) needed narrowing derivation for a linear term t in $\mathcal{R}$. We associate a pair (C_s, D_s) to each term s in this derivation, where $C_s = comp(s)$ is the complexity of s and $D_s = depth(s)$ is its depth. Trivially, for every needed narrowing step, $s \rightsquigarrow s'$, in this derivation, its associated pairs, (C_s, D_s) and $(C_{s'}, D_{s'})$, fulfill the condition $C_{s'} \leqslant_{mul} C_s$ (since $C_s = C_{s'}$ when a cyclic function is unfolded and $C_{s'} <_{mul} C_s$ when a noncyclic function is unfolded). Therefore, in order to prove that the considered needed narrowing derivation is quasi-terminating, it suffices to prove that every (possibly infinite) subderivation in which the first component remains unchanged only contains a finite number of distinct terms (modulo variable renaming).

Let us consider one of such subderivations: $t_0 \rightsquigarrow t_1 \rightsquigarrow t_2 \rightsquigarrow \ldots$ Now, we consider a *reordering* of this derivation in which redexes are exploited in leftmost innermost order. [1] Moreover, we also consider that in the reordered derivation only most general unifiers, restricted to the variables of $\mathcal{V}ar(s_{i-1})$, are computed (rather than simple unifiers, i.e., we do not anticipate some bindings as in λ [7, Def. 13]). This *innermost* derivation is denoted by $s_0 \rightsquigarrow_{p_1,R_1,\sigma_1} s_1 \rightsquigarrow_{p_2,R_2,\sigma_2} s_2 \rightsquigarrow_{p_3,R_3,\sigma_3} \ldots$ Clearly, the original subderivation is quasi-terminating iff the reordered subderivation is. These conditions on the considered subderivation greatly simplify the rest of the proof.

Consider the first narrowing step, $s_0 \rightsquigarrow_{p_1,R_1,\sigma_1} s_1$, in the subderivation. Since the rules of $\mathcal{R}$ are right-linear, we have $s_1 = s_0[\theta_1(r_1)]_{p_1}$, where $\delta_1 = \theta_1 \cup \sigma_1$ is the most general unifier of $s_0|_{p_1}$ and l_1, with $R_1 = (l_1 \to r_1)$ and $\sigma_1(s_0|_{p_1}) = \theta_1(l_1)$, i.e., the bindings in σ_1 need not be applied to $s_0[\theta_1(r_1)]_{p_1}$. Since the unfolded function is nonincreasing, by Def. 7, we have $dv(s'_i, x) \geqslant dv(t'_j, x)$ for all operation-rooted subterms $g(\overline{t'_m})$ in r_1, where $l_1 = f(\overline{s'_n})$, $i \in \{1,\ldots,n\}$, $x \in \mathcal{V}ar(s_i)$, and $j \in \{1,\ldots,m\}$. Since θ_1 only contains bindings for the variables in $\mathcal{V}ar(l_1)$, the depth of $\theta_1(r_1)$ is bounded by $max(depth(s_0|_{p_1}), depth(r_1|_{p_{11}}), \ldots, depth(r_1|_{p_{1n_1}})) + k$, where $p_{11}, \ldots, p_{1n_1}$ are the positions of the operation-rooted subterms in r_1 and k is a finite number to take into account the depth of the constructors demanded by the outer functions in s_0 (e.g., $k = 0$ if $p_1 = \epsilon$).

[1] We note that this reordering is always possible since $\mathcal{R}$ is right-linear.

If $\theta_1(r_1)$ still contains needed narrowing redexes, then we have another narrowing step, $s_1 \leadsto_{p_2,R_2,\sigma_2} s_2$, where $s_2 = s_1[\theta_2(r_2)]_{p_2}$ and $\delta_2 = \theta_2 \cup \sigma_2$ is the most general unifier of $s_1|_{p_2}$ and l_2, with $R_2 = (l_2 \to r_2)$ and $\sigma_2(s_1|_{p_2}) = \theta_2(l_2)$. Again, since the unfolded function is nonincreasing, the depth of $\theta_2(r_2)$ is bounded by $\mathit{max}(\mathit{depth}(s_1|_{p_2}), \mathit{depth}(r_2|_{p_{21}}), \ldots, \mathit{depth}(r_2|_{p_{2n_2}})) + k$, where $p_{21}, \ldots, p_{2n_2}$ are the positions of the operation-rooted subterms in r_2.

Furthermore, since $s_1|_{p_2}$ is a subterm of $\theta_1(r_1)$, i.e., $s_2 = s_0[\theta_1(r_1)[\theta_2(r_2)]_{p_2}]_{p_1}$, then we have that $\theta_2(r_2)$ is also bounded by $\mathit{max}(\mathit{depth}(s_0|_{p_1}), \mathit{depth}(r_1|_{p_{11}}), \ldots, \mathit{depth}(r_1|_{p_{1n_1}}), \mathit{depth}(r_2|_{p_{21}}), \ldots, \mathit{depth}(r_2|_{p_{2n_2}})) + k$. Extending this result to the (possibly infinite) subderivation where $s_0|_{p_1}$ is completely narrowed, we have that $s_i|_{p_1}$ is bounded through the derivation by

$$\begin{array}{l}\mathit{max}(\mathit{depth}(s_0|_{p_1}),\\ \mathit{depth}(r_1|_{p_{11}}), \ldots, \mathit{depth}(r_1|_{p_{1n_1}}),\\ \ldots,\\ \mathit{depth}(r_m|_{p_{m1}}), \ldots, \mathit{depth}(r_m|_{p_{mn_m}}))\\ +\, k\end{array}$$

where $r_1, \ldots, r_m$ are the right-hand sides of the rules in the definitions of all nonincreasing functions and $p_{j1}, \ldots, p_{jn_j}$, $1 \leqslant j \leqslant m$, are the positions of the operation-rooted subterms in these right-hand sides. Since there is a bound for the depth of the terms found during the reduction of the innermost redex $s_0|_{p_1}$, only finitely many different terms (modulo variable renaming) can be obtained.

Once $s_0|_{p_1}$ is completely narrowed, we distinguish two cases. If a noncyclic function is unfolded, the proof concludes since this will strictly reduce the first component of the associated pair. If a cyclic function is unfolded, a similar reasoning can be made. Hence, the depth of the terms computed in each subderivation where the first component remains unchanged is bounded by

$$\begin{array}{l}l \times \mathit{max}(\mathit{depth}(s_0|_{p_1}),\\ \mathit{depth}(r_1|_{p_{11}}), \ldots, \mathit{depth}(r_1|_{p_{1n_1}}),\\ \ldots,\\ \mathit{depth}(r_m|_{p_{m1}}), \ldots, \mathit{depth}(r_m|_{p_{mn_m}}))\\ +\, k'\end{array}$$

for a sufficiently large (but finite) k', where l is the number of noncyclic functions in s_0, which concludes the proof. $\square$

4 Related Work

The closest characterizations to ours have been presented by Wadler [15] and Chin and Khoo [14]. Wadler introduced the notion of *treeless* functions in order to ensure the termination of *deforestation* [15]. Treeless functions are a subclass of our nonincreasing functions where, additionally, all function calls in the right-hand sides of the rules can only have variable arguments.

Chin and Khoo [14] introduced the class of *nonincreasing consumers* and proved that any set of mutually recursive functions that are nonincreasing consumers can be transformed into an equivalent set of treeless functions, so that deforestation can be applied. This characterization differs from ours mainly in two points. Firstly, Chin and Khoo only require linear function calls in the right-hand sides of the rules (rather than being linear the entire right-hand sides, as we impose). This relaxed definition, however, is not safe in our context. Consider, e.g., the following nonincreasing consumers according to Chin and Khoo [14]:

$$\begin{aligned} f(x) &\to c(g(x), x) \\ g(s(x)) &\to g(x) \\ h(c(s(x), y)) &\to x \end{aligned}$$

where $c, s \in \mathcal{C}$ and $f, g, h \in \mathcal{D}$. Here, given the initial term $h(f(x))$, needed narrowing has an infinite derivation which is not quasi-terminating:

$$\begin{array}{ll} h(f(x)) & \\ \leadsto_{id} & h(c(g(x), x)) \\ \leadsto_{\{x \mapsto s(x')\}} & h(c(g(x'), s(x'))) \\ \leadsto_{\{x' \mapsto s(x'')\}} & h(c(g(x''), s(s(x'')))) \\ \leadsto & \ldots \end{array}$$

And, secondly, Chin and Khoo do not accept nested function calls in the right-hand side of any rule. In contrast, we accept arbitrary (linear) terms in the right-hand sides of noncyclic functions, which allows us to cope with a wider range of functions.

5 Discussion

We have presented a novel characterization for rewrite systems that guarantees the quasi-termination of needed narrowing computations. This is a difficult problem that has not been tackled before. Our characterization may be useful for several program analysis and transformations where terminating symbolic computations—by needed narrowing—are required.

One might argue that the class of nonincreasing TRSs is too restrictive. However, our characterization can also be used over more general rewrite systems (e.g., inductively sequential systems) in order to identify those terms that are (potentially) dangerous for quasi-termination, so that they can be *marked* and, then, properly generalized during the subsequent analysis or transformation. This is, e.g., the approach followed in [1] to offline partial evaluation of inductively sequential systems.

References

[1] J. Ramos, J. Silva, G. Vidal, Fast Narrowing-Driven
Partial Evaluation for Inductively Sequential Systems, in: Proc. of the 10th ACM SIGPLAN International Conference on Functional Programming (ICFP 2005), ACM Press, 2005, pp. 228–239.

[2] F. Baader, T. Nipkow, Term Rewriting and All That, Cambridge University Press,

1998.

[3] N. Dershowitz, Termination of Rewriting, Journal of Symbolic Computation 3 (1&2) (1987) 69–115.

[4] C. Holst, Finiteness Analysis, in: Proc. of Functional Programming Languages and Computer Architecture, Springer LNCS 523, 1991, pp. 473–495.

[5] N. Jones, C. Gomard, P. Sestoft, Partial Evaluation and Automatic Program Generation, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[6] J. Slagle, Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity, Journal of the ACM 21 (4) (1974) 622–642.

[7] S. Antoy, R. Echahed, M. Hanus, A Needed Narrowing Strategy, Journal of the ACM 47 (4) (2000) 776–822.

[8] S. Antoy, Definitional trees, in: Proc. of the 3rd Int'l Conference on Algebraic and Logic Programming (ALP'92), Springer LNCS 632, 1992, pp. 143–157.

[9] M. Hanus, The Integration of Functions into Logic Programming: From Theory to Practice, Journal of Logic Programming 19&20 (1994) 583–628.

[10] E. Giovannetti, G. Levi, C. Moiso, C. Palamidessi, Kernel Leaf: A Logic plus Functional Language, Journal of Computer and System Sciences 42 (1991) 363–377.

[11] R. Loogen, F. López-Fraguas, M. Rodríguez-Artalejo, A Demand Driven Computation Strategy for Lazy Narrowing, in: Proc. of PLILP'93, Springer LNCS 714, 1993, pp. 184–200.

[12] J. Moreno-Navarro, M. Rodríguez-Artalejo, Logic Programming with Functions and Predicates: The language Babel, J. Logic Programming 12 (3) (1992) 191–224.

[13] G. Huet, J. Lévy, Computations in orthogonal rewriting systems, Part I + II, in: J. Lassez, G. Plotkin (Eds.), Computational Logic – Essays in Honor of Alan Robinson, 1992, pp. 395–443.

[14] W. Chin, S. Khoo, Better Consumers for Program Specializations, Journal of Functional and Logic Programming 1996 (4).

[15] P. Wadler, Deforestation: Transforming programs to eliminate trees, Theoretical Computer Science 73 (1990) 231–248.