Productivity of non-orthogonal term rewrite systems

Raffelsieper, M.

Published in: Reduction Strategies in Rewriting and Programming (10th International Workshop, WRS 2011, Novi Sad, Serbia, May 29, 2011. Informal proceedings)

Published: 01/01/2011


General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Productivity of Non-Orthogonal Term Rewrite Systems

Matthias Raffelsieper
Department of Computer Science, TU Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
eMail: M.Raffelsieper@tue.nl

Abstract

Productivity is the property that finite prefixes of an infinite constructor term can be computed using a given term rewrite system. Hitherto, productivity has only been considered for orthogonal systems, where non-determinism is not allowed. This paper presents techniques to also prove productivity of non-orthogonal term rewrite systems. For such systems, it is desired that one does not have to guess the reduction steps to perform, instead any outermost-fair reduction should compute an infinite constructor term in the limit. As a main result, it is shown that for possibly non-orthogonal term rewrite systems this kind of productivity can be concluded from context-sensitive termination.

1 Introduction

Productivity is the property that a given set of computation rules computes a desired infinite object. This has been studied mostly in the setting of streams, the simplest infinite objects. However, as already observed in [9], productivity is also of interest for other infinite structures, for example infinite trees, or mixtures of finite and infinite structures. A prominent example of the latter are lists in the programming language Haskell [7], which can be finite (by ending with a sentinel “[]”) or which can go on forever.

Existing approaches for automatically checking productivity, e.g., [2, 3, 9], are restricted to orthogonal systems. The main reason for this restriction is that it disallows non-determinism. A complete computer program (i.e., a program and all possible input sequences, neglecting sources of true randomness) always behaves deterministically, as the steps of computation are precisely determined. However, often a complete program is not available, too large to be studied, or its inputs are provided by the user or they are not specified completely. In this case, non-determinism can be used to abstract from certain parts by describing a number of possible behaviors. In such a setting, the restriction to orthogonal systems, which is even far stronger than only disallowing non-determinism, should be removed. An example of such a setting are hardware components, describing streams of output values which are depending on the streams of input values. To analyze such a component in isolation, all possible input streams have to be considered.

This paper presents an extension of the techniques in [9] to analyze productivity of specifications that may contain non-determinism. These specifications are assumed to be given as term rewrite systems (TRS) [1], where the terms are considered to have two possible sorts. The first sort \( d \) is for data, and the second sort \( s \) for structure. Terms of this sort represent the intended structure containing the data and therefore are allowed to be infinite.

This paper presents an extension of the techniques in [9] to analyze productivity of specifications that may contain non-determinism. These specifications are assumed to be given as term rewrite systems (TRS) [1], where the terms are considered to have two possible sorts. The first sort \( d \) is for data, and the second sort \( s \) for structure. Terms of this sort represent the intended structure containing the data and therefore are allowed to be infinite.

We still have to impose some restrictions on specifications to make our approach work. These restrictions are given below in the definition of proper specifications.

Definition 1. A proper specification is a tuple \( S = (\Sigma_d, \Sigma_s, C, R_d, R_s) \), where \( \Sigma_d \) is the signature of data symbols, each of type \( d^m \rightarrow d \) (then the data arity of such a symbol \( g \) is defined to be \( \text{ar}_d(g) = m \)), \( \Sigma_s \) is the signature of structure symbols \( f \), which have types of the shape \( d^m \times s^n \rightarrow s \) (and data arity \( \text{ar}_d(f) = m \), structure arity \( \text{ar}_s(f) = n \)), \( C \subseteq \Sigma_s \) is a set of constructors, \( R_d \) is a terminating TRS over the signature \( \Sigma_d \), and \( R_s \) is a TRS containing rules \( f(u_1, \ldots, u_m, t_1, \ldots, t_n) \rightarrow t \) satisfying the following properties:
Productivity of Non-Orthogonal Term Rewrite Systems

Matthias Raffelsieper

- $f \in \Sigma \setminus \mathcal{C}$ with $\text{ar}_d(f) = m$, $\text{ar}_s(f) = n$.
- $f(t_1, \ldots, t_m, t_1, \ldots, t_n)$ is a well-sorted linear term, $t$ is a well-sorted term of sort $s$, and
- for all $1 \leq i \leq n$ and for all $p \in \text{Pos}(t_i)$ such that $t_i[p]$ is not a variable and $\text{root}(t_i[p]) \in \Sigma$, it holds that $\text{root}(t_i[p]) \notin \mathcal{C}$ for all $p' < p$ (i.e., no structure symbol is below a constructor).

Furthermore, $\mathcal{R}_s$ is required to be exhaustive, meaning that for every $f \in \Sigma \setminus \mathcal{C}$ with $\text{ar}_d(f) = m$, $\text{ar}_s(f) = n$, ground normal forms $u_1, \ldots, u_m \in \text{NF}_{\text{gnd}}(\mathcal{R}_d)$, and terms $t_1, \ldots, t_n \in T(\Sigma_d \cup \Sigma_s)$ such that for every $1 \leq i \leq n$, $t_i = c_i(u'_1, \ldots, u'_k, t'_1, \ldots, t'_l)$ with $u'_j \in \text{NF}_{\text{gnd}}(\mathcal{R}_d)$ for $1 \leq j \leq k = \text{ar}_d(c_i)$ and $c_i \in \mathcal{C}$, there exists at least one rule $\ell \rightarrow r \in \mathcal{R}_s$ such that $\ell$ matches the term $f(u_1, \ldots, u_m, t_1, \ldots, t_n)$.

A proper specification $\mathcal{S}$ is called orthogonal, if $\mathcal{R}_d \cup \mathcal{R}_s$ is orthogonal, otherwise it is called non-orthogonal.

Nesting of structure symbols inside constructors on left-hand sides of $\mathcal{R}_s$ is disallowed for technical reasons, however it is not a severe restriction in practice. Often, it can be achieved by unfolding the specification, as was presented in [4, 8].

The above definition coincides with the definition of proper specifications given in [9] for orthogonal proper specifications. For such orthogonal proper specifications, productivity is the property that every ground term $t$ of sort $s$ can, in the limit, be rewritten to a possibly infinite term consisting only of constructors. This is equivalent to stating that for every prefix depth $d \in \mathbb{N}$, the term $t$ can be rewritten to another term $t'$ having only constructor symbols on positions of depth $d$ or less. It was already observed in [2] that productivity of orthogonal specifications is equivalent to the existence of an outermost-fair reduction computing a constructor prefix for any given depth. Below, we give a general definition of outermost-fair reductions, as they will also be used in the non-orthogonal setting.

Definition 2.

- A redex is a subterm $t[p]$ of a term $t$ at position $p \in \text{Pos}(t)$ such that a rule $\ell \rightarrow r$ and a substitution $\sigma$ exist with $t[p] = \ell \sigma$. The redex $t[p]$ is said to be matched by the rule $\ell \rightarrow r$.
- A redex is called outermost iff it is not a strict subterm of another redex.
- A redex $t[p] = \ell \sigma$ is said to survive a reduction step $t \rightarrow t' \rightarrow r', q$ if $p \parallel q$, or if $p < q$ and $t' = t[\ell \sigma'][p]$ for some substitution $\sigma'$ (i.e., the same rule can still be applied at $p$).
- A rewrite sequence (reduction) is called outermost-fair, iff there is no outermost redex that survives as an outermost redex infinitely long.
- A rewrite sequence (reduction) is called maximal, iff it is infinite or ends in a normal form (a term that cannot be rewritten further).

For non-orthogonal proper specifications, requiring just the existence of a reduction to a constructor prefix of arbitrary depth does not guarantee an outermost-fair rewrite sequence to reach it, due to the possible non-deterministic choices.

Example 3. Consider a proper specification with the TRS $\mathcal{R}_s$ consisting of the following rules:

\[
\begin{align*}
\text{maybe} & \rightarrow 0 : \text{maybe} & \text{random} & \rightarrow 0 : \text{random} \\
\text{maybe} & \rightarrow \text{maybe} & \text{random} & \rightarrow 1 : \text{random}
\end{align*}
\]

This specification is not orthogonal, since the rules for maybe and those for random overlap. We do not want to call this specification productive, since it admits the infinite outermost-fair reduction $\text{maybe} \rightarrow \text{maybe} \rightarrow \ldots$ that never produces any constructors. However, there exists an infinite reduction producing infinitely many constructors starting in the term $\text{maybe}$, namely $\text{maybe} \rightarrow 0 : \text{maybe} \rightarrow 0 : 0 : \text{maybe} \rightarrow \ldots$. When only considering the rules for random then we want to call the resulting specification productive, since no matter what rule of random we choose, an element of the stream is created.

\footnote{To see this, observe that a defined symbol cannot occur on a non-root position of a left-hand side. Otherwise, such a subterm would unify with some left-hand side due to exhaustiveness, which would contradict orthogonality.}
Requiring just the existence of a constructor normal form is called \textit{weak productivity} in [2]. We already stated above that this is not the notion of productivity we are interested in, since it requires to “guess” the reduction steps leading to a constructor term. The notion we are interested in is \textit{strong productivity}, which requires all outermost-fair reductions to reach a constructor term. This kind of productivity was also defined in [2].

\textbf{Definition 4.} A proper specification $S$ is called \textit{strongly productive} iff for every ground term $t$ of sort $s$ all maximal outermost-fair rewrite sequences starting in $t$ end in (i.e., have as limit for infinite sequences) a constructor normal form.

Thus, the term \texttt{maybe} in Example 3 is not strongly productive, whereas the term \texttt{random} in that example is strongly productive. It was shown in [2] that weak and strong productivity coincide for orthogonal proper specifications.

2 \hspace{1em} \textbf{Criteria for Strong Productivity}

An orthogonal proper specification is productive if and only if a reduction exists that creates a constructor at the top, as was shown in [9]. This is also the case for non-orthogonal proper specifications. However, here we have to consider all maximal outermost-fair reductions, instead of just requiring the existence of such a reduction. Hence, the characterization for strong productivity used in this paper is the following.

\textbf{Proposition 5.} Let $S$ be a proper specification. Then $S$ is strongly productive iff for every maximal outermost-fair reduction $t_0 \rightarrow_{R_d \cup R_s} t_1 \rightarrow_{R_d \cup R_s} \ldots$ with $t_0$ being of sort $s$ there exists $k \in \mathbb{N}$ such that $\text{root}(t_k) \in \mathcal{C}$.

This characterization of productivity leads to a first technique to prove strong productivity of proper specifications. It is a simple syntactic check that determines whether every right-hand side of sort $s$ starts with a constructor. For orthogonal proper specifications, this was already observed in [9].

\textbf{Theorem 6.} Let $S$ be a proper specification. If for all rules $\ell \rightarrow r \in R_s$ we have $\text{root}(r) \in \mathcal{C}$, then $S$ is strongly productive.

The above criterion is sufficient to prove strong productivity of the proper specification consisting of the two rules for \texttt{random} in Example 3, since both have right-hand sides with the constructor : at the root. However, it is easy to create examples which are strongly productive, but do not satisfy the syntactic requirements of Theorem 6.

\textbf{Example 7.} Consider the proper specification with the following TRS $R_s$:

\begin{align*}
\text{ones} & \rightarrow 1 : \text{ones} & \text{finZeroes} & \rightarrow 0 : 0 : \text{ones} & f(0 : xs) & \rightarrow f(xs) \\
\text{finZeroes} & \rightarrow 0 : \text{ones} & \text{finZeroes} & \rightarrow 0 : 0 : 0 : \text{ones} & f(1 : xs) & \rightarrow 1 : f(xs)
\end{align*}

The constant \texttt{finZeroes} produces non-deterministically a stream that starts with one, two, or three zeroes followed by an infinite stream of ones. Function $f$ takes a binary stream as argument and filters out all occurrences of zeroes. Thus, productivity of this example proves that only a finite number of zeroes can be produced. This however cannot be proven with the technique of Theorem 6, since the right-hand side of the rule $f(0 : xs) \rightarrow f(xs)$ does not start with the constructor :.

Another technique presented in [9] to show productivity of orthogonal proper specifications is based on context-sensitive termination. The idea is to disallow rewriting in structure arguments of constructors, thus context-sensitive termination implies that for every ground term of sort $s$, a term starting with a constructor can be reached (due to the exhaustiveness requirement). As was observed by Endrullis and Hendriks recently in [4], this set of blocked positions can be enlarged, making the approach even stronger.
Below, the technique for proving productivity by showing termination of a corresponding context-sensitive TRS is extended to also be applicable in the case of our more general proper specifications. This version already includes an adaption of the improvement mentioned above.

**Definition 8.** Let \( S \) be a proper specification. The replacement map \( \mu_S : \Sigma_d \cup \Sigma_r \rightarrow 2^N \) is defined as follows:

\[
\begin{align*}
\mu_S(f) &= \{1, \ldots, ar_d(f)\}, \text{ if } f \in \Sigma_d \cup C \\
\mu_S(f) &= \{1, \ldots, ar_d(f) + ar_s(f)\} \setminus \{1 \leq i \leq ar_d(f) + ar_s(f) \mid t_i \text{ is a variable for all } \ell \rightarrow r \in \mathcal{R}_s \text{ and all non-variable subterms } t \text{ of } \ell \text{ with root}(t) = f\},^3 \text{ otherwise }
\end{align*}
\]

In the remainder, we leave out the subscript \( S \) if the specification is clear from the context. The replacement map \( \mu \) is used to define the set of *allowed* positions of a non-variable term \( t \) as \( \text{Pos}_\mu(t) = \{1\} \cup \{i, p \mid i \in \text{root}(t), p \in \text{Pos}_\mu(t)\} \). This replacement map is canonical [6] for the left-linear TRS \( \mathcal{R}_s \), guaranteeing that non-variable positions of left-hand sides are allowed. We extend it to the non-left-linear TRS \( \mathcal{R}_d \cup \mathcal{R}_s \) by allowing all arguments of symbols from \( \Sigma_d \). Context-sensitive rewriting is the restriction of the rewrite relation to those redexes on positions from \( \text{Pos}_\mu \). Formally, \( t \rightarrow_{t \rightarrow r, p} t' \) iff \( t \rightarrow_{\mu} t' \) and \( p \in \text{Pos}_\mu(t) \) and we say a TRS \( \mathcal{R} \) is \( \mu \)-terminating iff no infinite \( \mu \)-chain exists.

Our main result of this paper is that also for possibly non-orthogonal proper specifications, \( \mu \)-termination implies productivity.

**Theorem 9.** A proper specification \( S = (\Sigma_d, \Sigma_r, C, \mathcal{R}_d, \mathcal{R}_s) \) is strongly productive, if \( \mathcal{R}_d \cup \mathcal{R}_s \) is \( \mu \)-terminating.

It can be shown that Theorem 9 subsumes Theorem 6, since for a TRS \( \mathcal{R}_s \) with root\( (r) \in C \) for all \( \ell \rightarrow r \in \mathcal{R}_s \), the only allowed positions on right-hand sides are of sort \( d \) and \( \mathcal{R}_d \) is terminating. The technique of Theorem 9, i.e., proving \( \mu \)-termination of the corresponding context-sensitive TRS, is able to prove strong productivity of Example 7. This can for example be seen by typing the corresponding context-sensitive TRS into a modern termination tool such as AProVE [5].

Checking productivity in this way, i.e., by checking context-sensitive termination, can only prove productivity but not disprove it. This is illustrated in the next example.

**Example 10.** Consider the proper specification with the following rules in \( \mathcal{R}_s \):

\[
\begin{align*}
a & \rightarrow f(a) \\
f(x \cdot xs) & \rightarrow x : f(xs) \\
f(f(xs)) & \rightarrow 1 : xs
\end{align*}
\]

Starting in the term \( a \), we observe that an infinite \( \mu \)-reduction starting with \( a \rightarrow f(a) \) exists, which can be continued by reducing the underlined redex repeatedly, since \( \mu(f) = \{1\} \). Thus, the example is not \( \mu \)-terminating. However, the specification is productive, as can be shown by a case analysis based on the root symbol of some arbitrary ground term \( t \). In case \( \text{root}(t) = \cdot \), then nothing has to be done, according to Proposition 5. Otherwise, if \( \text{root}(t) = a \), then any maximal outermost-fair reduction must start with \( t = a \rightarrow f(a) \), thus we can reduce our analysis to the final case, where \( \text{root}(t) = f \). In this last case, \( t = f(t') \).

Due to the rules for the symbol \( f \), another case analysis is performed for \( t' \). If \( \text{root}(t') = \cdot \), then this

---

^2 Note that in [4], Endrullis and Hendriks consider orthogonal TRSs and also block arguments of symbols in \( \Sigma_d \) that only contain variables on left-hand sides of \( \mathcal{R}_d \). This however is problematic when allowing non-left-linear data rules. Example:

\[
\begin{align*}
\mathcal{R}_s : & \ f(1) \rightarrow f(d(0, d(1, 0))) & f(0) \rightarrow c \in C \\
\mathcal{R}_d : & \ d(x, x) \rightarrow 1 & d(0, x) \rightarrow 0 & d(1, x) \rightarrow 0
\end{align*}
\]

Here, the term \( f(d(0, d(1, 0))) \) can only be \( \mu \)-rewritten to the term \( f(0) \) (which then in turn has to be rewritten to \( c \)) if defining \( \mu(d) = \{1\} \), since the subterm \( d(1, 0) \) can never be rewritten to \( 0 \). However, the example is not strongly productive, as reducing in this way gives rise to an infinite outermost-fair reduction \( f(d(0, d(1, 0))) \rightarrow f(d(0, 0)) \rightarrow f(1) \rightarrow \ldots \). Blocking arguments of data symbols can only be done when \( \mathcal{R}_d \) is left-linear.

^3 The requirement of \( t \) not being a variable ensures that \( \text{root}(t) \) is defined.
constructor cannot be reduced further. Hence, in any maximal outermost-fair reduction sequence a redex of the form \( f(\bar{u} : t) \) exists at the root, until it is eventually reduced using the second rule which results in a term with the constructor \( : \) at the root. For root\((t') = a\) we again must reduce \( t = f(a) \rightarrow f(f(a)) \). Finally, in case \( t = f(t') = f(f(t'')) \), we have two possibilities. The first one occurs when the term \( t' \) is eventually reduced at the root. Since root\((t') = f\), this has to happen with either of the f-rules, creating a constructor \( : \) which, as we already observed, must eventually result in the term \( t \) also being reduced to a term with the constructor \( : \) at the root. Otherwise, in the second possible scenario, the term \( t' \) is never reduced at the root. Then however, an outermost redex of the shape \( f(f(\bar{t})) \) exists in all terms that \( t \) can be rewritten to in this way, thus it has to be reduced eventually with the third rule. This again creates a term with constructor \( : \) at the root. Combining all these observations, we see that in every maximal outermost-fair reduction there exists a term with the constructor \( : \) as root symbol, which proves productivity due to Proposition 5.

3 Conclusions and Future Work

We have presented a generalization of the productivity checking techniques in [9] (including an improvement observed in [4]) to non-orthogonal specifications, which are able to represent non-deterministic systems. These naturally arise when abstracting away certain details of an implementation, such as for example concrete sequences of input values. A main difference to the orthogonal setting is that in the non-orthogonal case, a single reduction to a constructor term is not sufficient, instead all outermost-fair reductions must be considered.

Our non-orthogonal setting still imposes certain restrictions on the specifications that can be treated. The most severe restriction is the requirement of left-linear rules in \( \mathcal{R} \). Dropping this requirement however would make Theorem 9 unsound. Similarly, also the requirement that structure arguments of constructors must be variables cannot be dropped without losing soundness of Theorem 9. This requirement however is not that severe in practice, since many specifications can be unfolded by introducing fresh symbols [4, 8].

In the future, it would be interesting to investigate whether transformations of non-orthogonal proper specifications, similar to those in [9], can be defined. It is clear that rewriting of right-hand sides for example is not productivity-preserving for non-orthogonal specifications, since it only considers one possible reduction. However, it would be interesting to investigate whether for example narrowing of right-hand sides is productivity preserving, as it considers all possible reductions.

References