Termination of canonical context-sensitive rewriting and productivity of rewrite systems

Salvador Lucas
DSIC, Universitat Politècnica de València, Spain
http://users.dsic.upv.es/~slucas/

Termination of programs, i.e., the absence of infinite computations, ensures the existence of normal forms for all initial expressions, thus providing an essential ingredient for the definition of a normalization semantics for functional programs. In lazy functional languages, though, infinite data structures are often delivered as the outcome of computations. For instance, the list of all prime numbers can be returned as a neverending stream of numerical expressions or data structures. If such streams are allowed, requiring termination is hopeless. In this setting, the notion of productivity can be used to provide an account of computations with infinite data structures, as it "captures the idea of computability, of progress of infinite-list programs" (B.A. Sijtsma, On the Productivity of Recursive List Definitions, ACM Transactions on Programming Languages and Systems 11(4):633-649, 1989). However, in the realm of Term Rewriting Systems, which can be seen as (first-order, untyped, unconditional) functional programs, termination of Context-Sensitive Rewriting (CSR) has been showed equivalent to productivity of rewrite systems through appropriate transformations. In this way, tools for proving termination of CSR can be used to prove productivity. In term rewriting, CSR is the restriction of rewriting that arises when reductions are allowed on selected arguments of function symbols only. In this paper we show that well-known results about the computational power of CSR are useful to better understand the existing connections between productivity of rewrite systems and termination of CSR, and also to obtain more powerful techniques to prove productivity of rewrite systems.

Keywords: context-sensitive rewriting, functional programming, productivity, termination

1 Introduction

The computation of normal forms of initial expressions provides an appropriate computational principle for the semantic description of functional programs by means of a normalization semantics where initial expressions are given an associated normal form, i.e., an expression that do not issue any computation. However, lazy functional languages (like Haskell [14]) admit giving infinite values as the meaning of expressions. Infinite values are limits of converging infinite sequences of partially defined values which are more and more defined and only contain constructor symbols. An appropriate notion of progress in lazy functional computations is given by the notion of productivity [27] which concerns the progress in the computation of infinite values when normal forms cannot be obtained.

Term Rewriting Systems (TRSs [4, 25, 28]) provide suitable abstractions for functional programs which are often useful to investigate their computational properties. We can see a term rewriting system as a first-order functional program without any kind of type information associated to any expression, and where all rules in the program are unconditional rules $\ell \to r$ where $\ell$ is a term $f(\ell_1,\ldots,\ell_k)$ for some function symbol $f$ and terms $\ell_1,\ldots,\ell_k$, and $r$ is a term whose variables already occur in $\ell$. The following example illustrates the use of infinite data structures with term rewriting systems.

---

*Partially supported by the EU (FEDER), Spanish MINECO TIN 2013-45732-C4-1-P, and GV PROMETEOII/2015/013.
evenNs \rightarrow \text{cons}(0, \text{incr}(\text{oddNs})) \quad (1)
oddNs \rightarrow \text{incr}(\text{evenNs}) \quad (2)
incr(\text{cons}(x, xs)) \rightarrow \text{cons}(s(x), \text{incr}(xs)) \quad (3)
take(0, xs) \rightarrow \text{nil} \quad (4)
take(s(n), \text{cons}(x, xs)) \rightarrow \text{consF}(x, \text{take}(n, xs)) \quad (5)
zip(\text{nil}, xs) \rightarrow \text{nil} \quad (6)
zip(xs, \text{nil}) \rightarrow \text{nil} \quad (7)
zip(\text{cons}(x, xs), \text{cons}(y, ys)) \rightarrow \text{cons}(\text{frac}(x, y), \text{zip}(xs, ys)) \quad (8)
tail(\text{cons}(x, xs)) \rightarrow xs \quad (9)
rep2(\text{nil}) \rightarrow \text{nil} \quad (10)
rep2(\text{cons}(x, xs)) \rightarrow \text{cons}(x, \text{cons}(x, \text{rep2}(xs))) \quad (11)
0 + x \rightarrow x \quad (12)
s(x) + y \rightarrow s(x + y) \quad (13)
0 \times y \rightarrow 0 \quad (14)
s(x) \times y \rightarrow y + (x \times y) \quad (15)
\text{prodFrac}(\text{frac}(x, y), \text{frac}(z, t)) \rightarrow \text{frac}(x \times z, y \times t) \quad (16)
\text{prodOfFracs}(\text{nil}) \rightarrow \text{frac}(s(0), s(0)) \quad (17)
\text{prodOfFracs}(\text{consF}(p, ps)) \rightarrow \text{prodFrac}(p, \text{prodOfFracs}(ps)) \quad (18)
\text{halfPi}(n) \rightarrow \text{prodOfFracs}(\text{take}(n, \text{zip}(\text{rep2}(\text{tail}(\text{evenNs})), \text{tail}(\text{rep2}(\text{oddNs})))))) \quad (19)

Figure 1: Computing Wallis’ approximation to $\frac{\pi}{2}$.

**Example 1** The TRS $R$ in Figure 1 can be used to compute approximations to $\frac{\pi}{2}$ as $\frac{\pi}{2} = \lim_{n \to \infty} \frac{\prod_{k=1}^{2n} \frac{2n+1}{2n}}{2n\cdot \frac{2n-1}{2n+1}}$ (Wallis’ product). In $R$, symbols 0 and s implement Peano’s representation of natural numbers; we also have the usual arithmetic operations addition and product. Symbols cons and nil are list constructors to build (possibly infinite) lists of natural numbers like evenNs (the infinite list of even numbers) and oddNs (the infinite list of odd numbers), which are defined by mutual recursion with rules (1) and (2). Function incr increases the elements of a list in one unit through the application of s (rule (3)). Function zip merges a pair of lists into a list of fractions (rules (6) to (8)), and tail returns the elements of a list after removing the first one (rule (9)). Function take (defined by rules (4) and (5)) is used to obtain the components of a finite approximation to $\frac{\pi}{2}$ which we multiply with prodOfFracs, which calls the usual addition and product of natural numbers defined by rules (12) to (15). The explicit use of consF to build finite lists of fractions of natural numbers by means of take ensures that the product of their elements computed by prodOfFracs is well-defined. A call halfPi(s^n(0)) for some $n > 0$ returns the desired approximation whose computation is launched by rule (19).

Note that $R$ is nonterminating. For instance we have the following infinite rewrite sequence:

$$\text{evenNs} \rightarrow \text{cons}(0, \text{incr}(\text{oddNs})) \rightarrow \text{cons}(0, \text{incr}(\text{incr}(\text{evenNs}))) \rightarrow \cdots \rightarrow \cdots \quad (20)$$

Context-sensitive rewriting (CSR [20, 21]) is a restriction of rewriting which imposes fixed, syntactic restrictions on reductions by means of a replacement map $\mu$ that, for each $k$-ary symbol $f$, discriminates the argument positions $i \in \mu(f) \subseteq \{1, \ldots, k\}$ which can be rewritten and forbids them if $i \notin \mu(f)$. These
restrictions are raised to arbitrary subterms of terms in the obvious way. With CSR we can achieve a terminating behaviour for TRSs \( R \) which (as in Example 1) are not terminating in the unrestricted case.

**Example 2** Let the replacement map \( \mu \) be given by:

\[
\mu(\text{cons}) = \emptyset \text{ and } \mu(f) = \{1, \ldots, ar(f)\} \text{ for all } f \in \mathcal{F} - \{\text{cons}\}
\]

That is, \( \mu \) disallows rewriting on the arguments of the list constructor \( \text{cons} \) (due to \( \mu(\text{cons}) = \emptyset \)). This makes a kind of lazy evaluation of lists possible. For instance, the rewrite sequence (20) above is not possible with CSR. The second step is disallowed because the replacement is issued on the second argument of \( \text{cons} \) and \( 2 \notin \mu(\text{cons}) \), i.e.,

\[
\text{cons}(0, \text{incr}(\text{oddNs})) \not\rightarrow_{\mu} \text{cons}(0, \text{incr}(\text{evenNs}))
\]

where we write \( \rightarrow_{\mu} \) to emphasize that the rewriting step is issued using CSR under the replacement map \( \mu \). This makes the infinite sequence impossible. Termination of CSR for the TRS \( R \) and \( \mu \) in Example 1 can be automatically proved with the termination tool MU-TERM [2].

A number of programming languages like CafeOBJ, [10], OBJ2, [9], OBJ3, [12], and Maude [5] admit the explicit specification of replacement restrictions under the so-called local strategies, which are sequences of argument indices associated to each symbol in the program.

Restrictions of rewriting may turn normal forms of some terms unreachable, leading to incomplete computations. Sufficient conditions ensuring that context-sensitive computations stop yielding head-normal forms, values or even normal forms have been investigated in [17, 18, 19, 20, 21].

The notion of *productivity* in term rewriting has to do with the ability of TRSs to compute possibly infinite values rather than arbitrary normal forms (as discussed in [6, 15], for instance). In CSR, early results showed that, for left-linear TRSs \( R \), if the replacement map \( \mu \) is made compatible with the left-hand sides \( \ell \) of the rules \( \ell \rightarrow r \) of \( R \), then CSR has two properties which are specifically relevant for the purpose of this paper:

1. every \( \mu \)-normal form (i.e., a term \( t \) where no further rewritings are allowed with CSR under \( \mu \)) is a head-normal form (i.e., a term that does not rewrite into a redex) [20, Theorem 8],

2. every term that rewrites into a constructor head-normal form can be rewritten with CSR into a constructor head-normal form with the same head symbol [20, Theorem 9].

The aforementioned compatibility of the replacement map \( \mu \) with the left-hand sides of the rules (which is then called a canonical replacement map) just ensures that the positions of nonvariable symbols in \( \ell \) are always reducible under \( \mu \). For instance, \( \mu \) in Example 1 is a canonical replacement map for \( R \) in the example. See also [22] where the role of the canonical replacement in connection with the algebraic semantics of computations with CSR, as defined in [13] and also [24], has been investigated.

In the following, we show that the facts (1) and (2) suffice to prove that termination of CSR is a sufficient condition for productivity (see Theorem 5 below). As mentioned before, the connection between termination of CSR and productivity is not new. In particular, Zantema and Raffelsieper proved that termination of CSR is a sufficient condition for productivity [30], and then Endrullis and Hendriks proved that, in fact, and provided that some appropriate transformations are used, it is also necessary, i.e., termination of CSR characterizes productivity [8].
Example 3 The following TRS $R$ can be used to define ordinal numbers [8 Example 6.8]:

\[
\begin{align*}
x + 0 &\rightarrow x & x \times 0 &\rightarrow 0 \\
x + S(y) &\rightarrow S(x + y) & x \times S(y) &\rightarrow (x \times y) + x \\
x + L(\sigma) &\rightarrow L(x + L \sigma) & x \times L(\sigma) &\rightarrow L(x \times L \sigma) \\
x + L(y : \sigma) &\rightarrow (x + y) : (x + L \sigma) & x \times L(y : \sigma) &\rightarrow (x \times y) : (x \times L \sigma) \\
\text{nats}(x) &\rightarrow x : \text{nats}(S(x)) & \omega &\rightarrow \text{nats}(0)
\end{align*}
\]

Here, 0 and $S$ are the usual constructors for natural numbers in Peano’s notation; a stream of ordinals can be obtained by means of the list constructor ‘:’ that combines an ordinal and a stream of ordinals to obtain a new stream of ordinals; finally, $L$ represents a limit ordinal defined by means of a stream of ordinals. For instance, $\omega$ is given as the limit $L(\text{nats}(0))$ of $\text{nats}(0)$, the stream that contains all natural numbers. Finally, $+$ and $\times$ are intended to, respectively, add and multiply ordinal numbers; symbol $+ \_L$ is an auxiliary operator that adds an ordinal number $x$ to a stream or ordinals by adding $x$ to each component of the stream using $\_$. Operation $\times \_L$ performs a similar task with $\_$. Endrullis and Hendriks use a transformation which introduces the replacement map $\mu(+) = \mu(\_L) = \mu(\times) = \mu(\_L \times) = \{2\}$, $\mu(S) = \{1\}$, and $\mu(L) = \mu(:) = \mu(\text{nats}) = \emptyset$, and also adds some rules to prove $R$ productive.

So, what is our contribution? First, we show that the ability of CSR to prove productivity is a consequence of essential properties of CSR, like (1) and (2) above. This theoretical clarification is valuable and useful for further developments in the field and, as far as we know, has not been addressed before. From a practical point of view, we are able to improve Zantema and Raffelsieper’s criterion that uses unnecessarily ‘permissive’ replacement maps which can fail to conclude productivity as termination of CSR in many cases. For instance, we can prove productivity of $R$ in Example 3 as termination of CSR for the replacement map $\mu$ in the example. Furthermore, we can do it automatically by using existing tools like AProVE [11] or MU-TERM. In contrast, with the replacement map $\mu'$ that would be obtained according to [30], $R$ is not terminating for CSR; thus, productivity cannot be proved by using Zantema and Raffelsieper’s technique. We are also able to improve the treatment in [8] because they need to apply a transformation to $R$ that we do not need to use. In fact, we were able to deal with all examples of productivity in those papers by using our main result together with the aforementioned termination tools to obtain automatic proofs. Our result, though, does not provide a characterization of productivity, as we show by means of an example.

However, our results apply to left-linear TRSs, whereas [8, 30] deal with orthogonal (constructor-based) TRSs only. Actually, we also supersede the main result of [26] which applies to non-orthogonal TRSs which are still left-linear. This is also interesting to understand the role of CSR in proofs of productivity. Actually, the results in the literature about completeness of CSR to obtain head-normal forms and values concern left-linear TRSs and canonical replacement maps only. The additional restrictions that are usually imposed on TRSs to achieve productivity as termination of CSR (in particular, exhaustive patterns in the left-hand sides) have to do with the notion of productivity rather than with CSR itself.

After some preliminaries in Section 2 Section 3 introduces the notions about CSR that we need for the development of our results on productivity via termination of CSR in Section 4. Section 5 compares with related work and Section 6 concludes.

2 Preliminaries

This section collects a number of definitions and notations about term rewriting [4, 28]. Throughout the paper, $X$ denotes a countable set of variables and $\mathcal{R}$ denotes a signature, i.e., a set of function
Termination of canonical context-sensitive rewriting and productivity of rewrite systems

A mapping \( \sigma : \mathcal{F} \to \mathcal{N} \) is said to be head-normalizing if it rewrites into a head-normal form. A constructor system is terminating. A term \( t \) is a linear term. Given \( t = (V,P) \leq \bot \), a rewrite rule is an ordered pair \( (l,F) \). The set of positions of a term \( t \) is denoted as \( \mathcal{P}os(t) \). Positions of non-variable symbols in \( t \) are denoted as \( \mathcal{P}os(\mathcal{F}) \), and \( \mathcal{P}os(\mathcal{X}) \) are the positions of variables. The subterm at position \( p \) of \( t \) is denoted as \( t \mid _p \) and \( t \mid_s \) is the term \( t \) with the subterm at position \( p \) replaced by \( s \). The symbol labelling the root of \( t \) is denoted as \( \text{root}(t) \). Given terms \( t \) and \( s \), \( \mathcal{P}os_s(t) \) denotes the set of positions of \( s \) in \( t \), i.e., \( \mathcal{P}os_s(t) = \{ p \in \mathcal{P}os(t) \mid t \mid _p = s \} \). A substitution is a mapping \( \sigma : \mathcal{X} \to \mathcal{I}((\mathcal{F},\mathcal{X})) \) which is homomorphically extended to a mapping \( \sigma : \mathcal{I}((\mathcal{F},\mathcal{X})) \to \mathcal{I}((\mathcal{F},\mathcal{X})) \) by abuse, we denote using the same symbol \( \sigma \).

A rewrite rule is an ordered pair \( (l,r) \), written \( l \to r \), with \( l,r \in \mathcal{I}((\mathcal{F},\mathcal{X})) \), \( l \notin \mathcal{X} \) and \( \forall ar(l) \subseteq \forall ar(r) \subseteq \forall ar(l) \). The left-hand side (lhs) of the rule is \( l \) and \( r \) is the right-hand side (rhs). A TRS is a pair \( \mathcal{R} = (\mathcal{F},R) \) where \( R \) is a set of rewrite rules. \( L(\mathcal{R}) \) denotes the set of lhs’s of \( \mathcal{R} \). An instance \( \sigma(l) \) of a lhs \( l \) of a rule is a redex. The set of redex positions in \( t \) is denoted as \( \mathcal{P}os_{\mathcal{R}}(t) \). A TRS \( \mathcal{R} \) is left-linear if for all \( l \in L(\mathcal{R}) \), \( l \) is a linear term. Given \( \mathcal{R} = (\mathcal{F},R) \), we consider \( \mathcal{F} \) as the disjoint union \( \mathcal{F} = \mathcal{C} \uplus \mathcal{P} \) of symbols \( c \in \mathcal{C} \), called constructors and symbols \( f \in \mathcal{P} \), called defined functions, where \( \mathcal{P} = \{ \text{root}(l) \mid l \to r \in R \} \) and \( \mathcal{C} = \mathcal{F} - \mathcal{P} \). Then, \( \mathcal{I}(\mathcal{C},\mathcal{X}) \) (resp. \( \mathcal{I}(\mathcal{C}) \)) is the set of constructor (resp. ground constructor) terms.

A TRS \( \mathcal{R} = (\mathcal{C} \uplus \mathcal{P},R) \) is a constructor system (CS) if for all \( f(\ell_1,\ldots,\ell_k) \to r \in R, \ell_i \in \mathcal{I}(\mathcal{C},\mathcal{X}) \), for \( 1 \leq i \leq k \).

A term \( t \in \mathcal{I}((\mathcal{F},\mathcal{X})) \) rewrites to \( s \) (at position \( p \)), written \( t \xrightarrow{\mathcal{R}}_p s \) (or just \( t \to s \)), if \( t \mid_p = \sigma(l) \) and \( s = t|_{\sigma(r)} \), for some rule \( \rho : l \to r \in R, p \in \mathcal{P}os(t) \) and substitution \( \sigma \). A TRS is terminating if \( \to \) is terminating. A term \( s \) is root-stable (or a head-normal form) if \( \forall t, \) if \( s \to^* t \), then \( t \) is not a redex. A term is said to be head-normalizing if it rewrites into a head-normal form.

3 Context-sensitive rewriting

A mapping \( \mu : \mathcal{F} \to \mathcal{N} \) is a replacement map (\( \mathcal{F} \)-map) if for all \( f \in \mathcal{F} \), \( \mu(f) \subseteq \{ 1,\ldots,ar(f) \} \). \( M_\mathcal{F} \) is the set of \( \mathcal{F} \)-maps. Replacement maps can be compared according to their ‘restriction power’: \( \mu \sqsubseteq \mu' \) if for all \( f \in \mathcal{F} \), \( \mu(f) \subseteq \mu'(f) \). If \( \mu \subseteq \mu' \), we say that \( \mu \) is more restrictive than \( \mu' \). Then, \( (\mathcal{F},\sqsubseteq,\sqcup,\mathcal{N},\cup) \) induces a complete lattice \( (M_\mathcal{F},\sqsubseteq,\sqcup,\mu_\bot,\mu_\top) \): the minimum (maximum) element is \( \mu_\bot \) (\( \mu_\top \)), given by \( \mu_\bot(f) = \emptyset \) (\( \mu_\top(f) = \{ 1,\ldots,ar(f) \} \)) for all \( f \in \mathcal{F} \). The lub \( \sqcup \) is given by \( (\mu \sqcup \mu')(f) = \mu(f) \cup \mu'(f) \) for all \( f \in \mathcal{F} \).

The replacement restrictions introduced by a replacement map \( \mu \) on the arguments of function symbols are raised to positions of terms \( t \in \mathcal{I}((\mathcal{F},\mathcal{X})) \): the set \( \mathcal{P}os_\mu(t) \) of \( \mu \)-replacing positions of \( t \) is:

\[
\mathcal{P}os_\mu(t) = \begin{cases} \{ \lambda \} & \text{if } t \in \mathcal{X} \\ \{ \lambda \} \cup \bigcup_{i \in \mu(\text{root}(t))} \mathcal{P}os_\mu(t) |_i & \text{if } t \notin \mathcal{X} \end{cases}
\]

Given terms \( s,t \in \mathcal{I}((\mathcal{F},\mathcal{X})) \), \( \mathcal{P}os_\mu(t) \) is the set of positions corresponding to \( \mu \)-replacing occurrences of \( s \) in \( t \): \( \mathcal{P}os^\mu(t) = \mathcal{P}os_\mu(t) \cap \mathcal{P}os_s(t) \). The set of \( \mu \)-replacing variables occurring in \( t \in \mathcal{I}((\mathcal{F},\mathcal{X})) \) is \( \forall ar_\mu(t) = \{ x \in \mathcal{X} \mid \mathcal{P}os_\mu(t) \neq \emptyset \} \).
3.1 Canonical replacement map

Given $t \in \mathcal{F}(\mathcal{F}, \mathcal{X})$, a replacement map $\mu \in M_{\mathcal{F}}$, is called compatible with $t$ (and vice versa) if $\mathcal{P}os_{\mathcal{F}}(t) \subseteq \mathcal{P}os_{\mu}(t)$. Furthermore, $\mu$ is called strongly compatible with $t$ if $\mathcal{P}os_{\mathcal{F}}(t) = \mathcal{P}os_{\mu}(t)$. And $\mu$ is (strongly) compatible with $T \subseteq \mathcal{F}(\mathcal{F}, \mathcal{X})$ if for all $t \in T$, $\mu$ is (strongly) compatible with $t$.

Definition 1 [20] Let $\mathcal{R}$ be a TRS. The canonical replacement map of $\mathcal{R}$ is $\mu_{can}^{\mathcal{R}} = \bigcup_{t \in L(\mathcal{R})} \mu_t$.

Note that $\mu_{can}^{\mathcal{R}}$ can be automatically associated to $\mathcal{R}$ by means of a very simple calculus: for each symbol $f \in \mathcal{F}$ and $i \in \{1, \ldots, \text{ar}(f)\}$, $i \in \mu_{can}(f)$ iff $\forall \mu \in L(\mathcal{R}), p \in \mathcal{P}os_{\mathcal{R}}(l), (\mu(l)_{p}) = f \land p, i \in \mathcal{P}os_{\mathcal{R}}(l)$.

Given a TRS $\mathcal{R}$, $CM_{\mathcal{R}} = \{ \mu \in M_{\mathcal{F}} \mid \mu_{can}^{\mathcal{R}} \subseteq \mu \}$ is the set of replacement maps that are equal to or less restrictive than the canonical replacement map. If $\mu \in CM_{\mathcal{R}}$, we also say that $\mu$ is a canonical replacement map for $\mathcal{R}$.

Example 4 For $\mathcal{R}$ in Example 3 we have:

$$\begin{align*}
\mu_{can}^{\mathcal{R}}(S) &= \mu_{can}^{\mathcal{R}}(L) = \mu_{can}^{\mathcal{R}}(\text{nats}) = \mu_{can}^{\mathcal{R}}(;) = \emptyset \\
\mu_{can}^{\mathcal{R}}(+ &= \mu_{can}^{\mathcal{R}}(+L) = \mu_{can}^{\mathcal{R}}(\times) = \mu_{can}^{\mathcal{R}}(\times L) = \{2\}
\end{align*}$$

For instance, $\mu_{can}^{\mathcal{R}}(S) = \emptyset$ because for all subterms $S(t)$ in the left-hand sides $\ell$ of the rules $\ell \to r$ of $\mathcal{R}$, $t$ is always a variable. However, $\mu_{can}^{\mathcal{R}}(+) = \{2\}$ because the second argument of $+$ in the left-hand side $x + 0$ of the first rule in $\mathcal{R}$ is not a variable.

Note that, $\mu$ in Example 3 prescribes $\mu(S) = \{1\}$. Thus, $\mu_{can}^{\mathcal{R}} \not\subseteq \mu$ and $\mu \in CM_{\mathcal{R}}$ but $\mu \neq \mu_{can}^{\mathcal{R}}$.

3.2 Strongly compatible TRSs

Given $t \in \mathcal{X}(\mathcal{F}, \mathcal{X})$, the only $\mathcal{F}(t)$-map $\mu$ (if any) which is strongly compatible with $t$ is $\mu_t$ [18 Proposition 3.6]. We call $t \in \mathcal{F}(\mathcal{F}, \mathcal{X})$ strongly compatible if $\mu_t$ is strongly compatible with $t$. Similarly, the only $\mathcal{F}(T)$-map $\mu$ which can be strongly compatible with $T$ is $\mu_T = \bigcup_{t \in T} \mu_t$. We call $T$ strongly compatible if $\mu_T$ is strongly compatible with $T$; we call $T$ weakly compatible if $t$ is strongly compatible for all $t \in T$.

Definition 2 [18,19] A TRS $\mathcal{R}$ is strongly (weakly) compatible, if $L(\mathcal{R})$ is a strongly (weakly) compatible set of terms.

The only replacement map (if any) which makes $\mathcal{R}$ strongly compatible is $\mu_{can}^{\mathcal{R}}$. For instance, $\mathcal{R}$ in Example 3 is strongly compatible, but $\mu$ is not strongly compatible with $L(\mathcal{R})$ (variable $y$ in the left-hand side of the second rule is $\mu$-replacing).

1The specification for constant symbols $a$ is omitted, as it is always the empty set $\mu(a) = \emptyset$. 
3.3 Context-sensitive rewriting

Given a TRS $\mathcal{R} = (\mathcal{F}, R)$, $\mu \in M_\mathcal{R}$, and $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $s \mu$-rewrites to $t$ at position $p$, written $s \xrightarrow{\mu}[p] t$ (or $s \xrightarrow{\mu} t$, $s \xrightarrow{\mu} t$, or even $s \xrightarrow{\mu}$ $t$), if $s \xrightarrow{\mu} t$ and $p \in \text{Pos}_\mu(s)$ [16][20]. A TRS $\mathcal{R}$ is $\mu$-terminating if $\xrightarrow{\mu}$ is terminating. Several tools can be used to prove termination of CSR; for instance, AProVE and MU-TERM, among others.

**Remark 1** In the following, when considering a TRS $\mathcal{R}$ together with a canonical replacement map $\mu \in CM_\mathcal{R}$, we often say that $\xrightarrow{\mu}$ performs canonical context-sensitive rewriting steps [27].

The $\xrightarrow{\mu}$-normal forms are called $\mu$-normal forms, and $\text{NF}_\mu^{\mathcal{R}}$ is the set of $\mu$-normal forms for a given TRS $\mathcal{R}$. As for unrestricted rewriting, $t \in \text{NF}_\mu^{\mathcal{R}}$ if and only if $\text{Pos}_\mu^{\mathcal{R}}(t) = \emptyset$ (i.e., $t$ contains no $\mu$-replacing redex). Rewriting with canonical replacement maps $\mu$ has important computational properties that we enumerate here and use below.

**Theorem 1** [20, Theorem 8] Let $\mathcal{R}$ be a left-linear TRS and $\mu \in CM_\mathcal{R}$. Every $\mu$-normal form is a head-normal form.

**Theorem 2** [20, Theorem 9] Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \cup \mathcal{F}, R)$ be a left-linear TRS and $\mu \in CM_\mathcal{R}$. Let $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $t = c(t_1, \ldots, t_k)$ for some $c \in \mathcal{C}$. If $s \xrightarrow{s} t$, then there is $u = c(u_1, \ldots, u_k)$ such that $s \xrightarrow{s} u$ and, for all $i$, $1 \leq i \leq k$, $u_i \xrightarrow{s} t_i$.

4 Productivity and termination of CSR

The operational semantics of rewriting-based programming languages can be abstracted, for each program (i.e., TRS) $\mathcal{R}$, as a mapping from terms $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ into (possibly empty) sets of (possibly infinite) terms $T_s \subseteq \mathcal{T}(\mathcal{F}, \mathcal{X})$, which are (possibly infinite) reducts of $s$. The intended shape of terms in $T_s$ depends on the application:

1. In functional programming, (ground) values $t \in \mathcal{T}(\mathcal{C})$ are the meaningful reducts of (ground) initial expressions $s$ (evaluation semantics) and $T_s \subseteq \mathcal{T}(\mathcal{C})$.
2. In lazy functional programming infinite values are also accepted in the semantic description, i.e., $T_s \subseteq \mathcal{T}(\mathcal{C})$, but the infinite terms are not actually obtained but only approximated as sequences of appropriate finite terms which are prefixes of the infinite values.
3. In equational programming and rewriting-based theorem provers, computing normal forms is envisaged (normalization semantics), i.e., $T_s \subseteq \text{NF}_\mathcal{R}$.

In functional programming (both in the eager and lazy case), computations can be understood as decomposed into the computation of a head-normal form $t'\text{'}$ (i.e., $s \xrightarrow{s} t'$) which is then rewritten (below the root!) into $t$. When a head-normal form $t'$ is obtained, the root symbol $f = \text{root}(t')$ is checked. If $f$ is a constructor symbol, then the evaluation continues on an argument of $t'$. Otherwise, the evaluation fails and an error is reported (this corresponds to $T_s$ empty). Thus, a head-normalization process is involved in the computation of the semantic sets $T_s$.

The notion of productivity in term rewriting has to do with the ability of TRSs to compute possibly infinite values. Most presentations of productivity analysis use sorted signatures and terms [8][30].

---

2Such finite approximations to infinite terms are described as partial values using a special symbol $\bot$ to denote undefinedness. An infinite value $\delta \in \mathcal{T}(\mathcal{C})$ is the limit of an infinite sequence $\delta_1, \delta_2, \ldots$ of such partial values where, for all $i \geq 1$, $\delta_{i+1} \in \mathcal{T}(\mathcal{C} \cup \{\bot\})$ is obtained from $\delta_i \in \mathcal{T}(\mathcal{C} \cup \{\bot\})$ by replacing occurrences of $\bot$ in $\delta_i$ by partial values different from $\bot$.
set of sorts $\mathcal{S}$ is partitioned into $\mathcal{S} = \Delta \cup \Gamma$, where $\Delta$ is the set of data sorts, intended to model inductive data types (booleans, natural numbers, finite lists, etc.). On the other hand, $\Gamma$ is the set of codata sorts, intended to model coinductive datatypes such as streams and infinite trees. Terms of sort $\Delta$ are called data terms and terms of sort $\Gamma$ are called codata terms. Given a symbol $f : \tau_1 \times \cdots \times \tau_n \to \tau$, $ar_\Delta(f)$ (resp. $ar_\Gamma(f)$) is the number of arguments of $f$ of sort $\Delta$ (resp. $\Gamma$). Endrullis et al. (and also [30]) assume all data arguments to be in the first argument positions of the symbols.

**Definition 3** [8, Definition 3.1] A tree specification is a $(\Delta \cup \Gamma)$-sorted, orthogonal, exhaustive constructor TRS $\mathcal{R}$ where $\Delta \cap \Gamma = \emptyset$.

Here, $\mathcal{R}$ is called exhaustive if for all $f \in \mathcal{R}$, every term $f(t_1, \ldots, t_k)$ is a redex whenever $t_i \in \mathcal{T}^\alpha(\mathcal{C})$ are (possibly infinite) closed constructor terms for all $i$, $1 \leq i \leq k$ [8, Definition 2.9]. As in [8, Definition 2.4], we assume here a generalized notion of substitution as an $\mathcal{I}$-sorted mapping $\sigma : \mathcal{X} \to \mathcal{T}^\alpha(\mathcal{R}, \mathcal{R})$ which is also extended to a mapping $\tau : \mathcal{T}^\alpha(\mathcal{R}, \mathcal{R}) \to \mathcal{T}^\alpha(\mathcal{R}, \mathcal{R})$.

**Example 5** Consider the tree specification $\mathcal{R}$ in Example 3 where, according to [8, Example 6.8], $\Delta = \{\text{Ord}\}$ with $\text{Ord}$ a data sort for ordinals and $\Gamma = \{\text{Str}\}$ with $\text{Str}$ a codata sort for streams of ordinals. The types for the constructor symbols are: $0 :: \text{Ord}$, $S :: \text{Ord} \to \text{Ord}$, $L :: \text{Str} \to \text{Ord}$ and $(:) :: \text{Ord} \times \text{Str} \to \text{Str}$. Thus, $\mathcal{C}_\Delta = \{0, S, L\}$, $\mathcal{C}_\Gamma = \{::\}$, $\mathcal{D}_\Delta = \{+, \times, \omega\}$, and $\mathcal{D}_\Gamma = \{+L, \times L, \text{nats}\}$.

**Definition 4** [8, Definition 3.5] A tree specification $\mathcal{R}$ is constructor normalizing if all finite ground terms $t \in \mathcal{T}(\mathcal{R})$ rewrite to a possibly infinite constructor normal form $\delta \in \mathcal{T}^\alpha(\mathcal{C})$.

Being exhaustive is a necessary condition for productivity.

**Theorem 3** If $\mathcal{R}$ is constructor normalizing, then it is exhaustive.

**Proof.** If not, then there is a finite ground normal form $t$ containing a defined symbol. This contradicts $\mathcal{R}$ being constructor normalizing. \hfill $\Box$

**Theorem 4** Let $\mathcal{R}$ be an exhaustive, left-linear TRS and $\mu \in \text{CM}_\mathcal{R}$. If $\mathcal{R}$ is $\mu$-terminating, then $\mathcal{R}$ is constructor normalizing.

**Proof.** Since $\mathcal{R}$ is $\mu$-terminating, every ground term $s$ has a (finite) $\mu$-normal form $t$. By Theorem 1, $t$ is a head-normal form. We prove by induction on $t$ that $t$ rewrites into a (possibly infinite) constructor term $\delta \in \mathcal{T}^\alpha(\mathcal{C})$. If $t$ is a constant, then since $t$ is a $\mu$-normal form, it must be a normal form. Since $\mathcal{R}$ is exhaustive, $t = \delta \in \mathcal{T}(\mathcal{C})$. If $t = f(t_1, \ldots, t_k)$ for ground terms $t_1, \ldots, t_k$, then by the induction hypothesis, for all $i$, $1 \leq i \leq k$, $t_i$ has a (possibly infinite) constructor normal form $\delta_i \in \mathcal{T}^\alpha(\mathcal{C})$. We have two cases:

1. If $f \in \mathcal{C}$, then $t$ has a (possibly infinite) constructor normal form $f(\delta_1, \ldots, \delta_k)$.
2. If $f \notin \mathcal{C}$, then, since $t$ is a head-normal form, $f(\delta_1, \ldots, \delta_k)$ is a ground (possibly infinite) normal form which contradicts that $\mathcal{R}$ is exhaustive.

Thus, $s$ has a (possibly infinite) constructor normal form as well and $\mathcal{R}$ is constructor normalizing. \hfill $\Box$

Since tree specifications are left-linear and exhaustive, Theorem 4 holds for tree specifications.

**Example 6** The following tree specification $\mathcal{R}$ (cf. [30, Example 4.6])

\[
\begin{align*}
p & \to \text{zip}(\text{alt}, p) \\
\text{alt} & \to 0 : 1 : \text{alt} \\
\text{zip}(x : \sigma, \tau) & \to x : \text{zip}(\tau, \sigma)
\end{align*}
\]
(where no constant for empty lists is included!) is easily proved \( \mu^\text{can}_\mathcal{R} \)-terminating (use MU-TERM). By Theorem \( 4 \) it is constructor normalizing. Note that \( \mathcal{R} \) is exhaustive due to the sort discipline (for instance, \( \text{zip}(0,0) \) is not allowed) and to the fact that no constructor for lists is provided (i.e., there is no finite list and all lists are of the form \( \text{cons}(s,t) \) for terms \( s, t \) where \( t \) is always infinite).

As remarked in [8 Section 3.2], several authors define \( \mathcal{R} \) to be productive if it is constructor normalizing (e.g., [7, 29, 30]). Endrullis and Hendriks give a more elaborated (and restrictive) definition of productivity. Given \( t \in \mathcal{F}_\omega(\mathcal{F}, \mathcal{R}) \) and \( \mathcal{F}' \subseteq \mathcal{F} \), a \( \mathcal{F}' \)-path in \( t \) is a (finite or infinite) sequence \( \langle p_1, c_1 \rangle, \langle p_2, c_2 \rangle, \ldots \) such that \( c_i = \text{root}(t|_{p_i}) \in \mathcal{F}' \) and \( p_{i+1} = p_i \cdot j \) with \( 1 \leq j \leq \text{ar}(c_i) \) [8 Definition 3.7].

**Definition 5** [8 Definition 3.8] A tree specification is said data-finite if for all finite ground terms \( s \in \mathcal{F}(\mathcal{F}) \) and (possibly infinite) constructor normal forms \( t \) of \( s \), every \( \mathcal{E}_\Delta \)-path in \( t \) (containing data constructors only) is finite.

**Definition 6** [8 Definition 3.11] A tree specification \( \mathcal{R} \) is productive if \( \mathcal{R} \) is constructor normalizing and data-finite.

In the following result, \( \mu_\Delta \) is given by \( \mu_\Delta(c) = \{1, \ldots, \text{ar}_\Delta(c)\} \) for all \( c \in \mathcal{C}_\Delta \), and \( \mu_\Delta(f) = \emptyset \) for all other symbols \( f \).

**Theorem 5** Let \( \mathcal{R} \) be a left-linear, exhaustive TRS and \( \mu \in M_\mathcal{R} \) be such that \( \mu^\text{can}_\mathcal{R} \cup \mu_\Delta \subseteq \mu \). If \( \mathcal{R} \) is \( \mu \)-terminating, then \( \mathcal{R} \) is productive.

**Proof.** Since \( \mu^\text{can}_\mathcal{R} \subseteq \mu \), constructor normalization of \( \mathcal{R} \) follows by Theorem 4. Thus, if \( \mathcal{R} \) is not productive, there must be a ground normal form \( t \) of a term \( s \) with an infinite \( \mathcal{E}_\Delta \)-path. Without loss of generality, we can assume that \( s \rightarrow^* s_1 = c_1(s^1_1, \ldots, s^1_{k_1}) \) for some \( c_1 \in \mathcal{C}_\Delta \), and then \( s^1_{i_1} \rightarrow^* c_2(s^2_1, \ldots, s^2_{k_2}) \) for some \( i_1 \), \( 1 \leq i_1 \leq \text{ar}_\Delta(c_1) \) and \( c_2 \in \mathcal{C}_\Delta \), etc., in such a way that this reduction sequences follow the computation of \( t \) and produce the \( \mathcal{E}_\Delta \)-path \( \langle \lambda, c_1 \rangle, \langle i_1, c_2 \rangle, \langle i_1, \ldots, \rangle \)...

By Theorem 2, \( s \rightarrow^* \overrightarrow{s_1} = c_1(\overrightarrow{s^1_1}, \ldots, \overrightarrow{s^1_{k_1}}) \) for some terms \( \overrightarrow{s^1_1}, \ldots, \overrightarrow{s^1_{k_1}} \) such that \( \overrightarrow{s_j} \rightarrow^* s_j^j \) for all \( j \), \( 1 \leq j \leq k_1 \). Thus, by Theorem 2 we also have \( \overrightarrow{s^1_i} \rightarrow^* c_2(\overrightarrow{s^2_1}, \ldots, \overrightarrow{s^2_{k_2}}) \) and \( \overrightarrow{s_j} \rightarrow^* s^j_2 \) for all \( j \), \( 1 \leq j \leq k_2 \). Since \( i_1 \in \mu_\Delta(c_1) \), we have \( s \rightarrow^* \overrightarrow{s_2} = c_1(\overrightarrow{s^1_1}, \ldots, \overrightarrow{s^1_{k_1-1}}, c_2(\overrightarrow{s^2_1}, \ldots, \overrightarrow{s^2_{k_2}}), \ldots, \overrightarrow{s^1_{k_1}}) \) with \( \overrightarrow{s^2_j} \rightarrow^* s^j_2 \) again. Since \( i_1, i_2 \in \mathcal{R}_\alpha \mu^\text{can}(\overrightarrow{s_2}) \), we can continue with this construction to obtain an infinite \( \mu \)-rewriting sequence which contradicts \( \mu \)-termination of \( \mathcal{R} \).

**Example 7** For the tree specification \( \mathcal{R} \) in Example 3 (see also Example 5), we have \( \text{ar}_\Delta(S) = 1 \) and \( \text{ar}_\Delta(L) = 0 \). Then, \( \mu_\Delta(S) = \{1\} \) and \( \mu_\Delta(L) = \emptyset \). Now \( \mu = \mu^\text{can}_\mathcal{R} \cup \mu_\Delta \) is as given in Example 3. The \( \mu \)-termination of \( \mathcal{R} \) can be proved with MU-TERM. By Theorem 5 productivity of \( \mathcal{R} \) follows.

**Example 8** We also prove productivity of \( \mathcal{R} \) in Example 6 Here, \( \Delta = \{d\} \) and \( \Gamma = \{s\} \) with \( \mathcal{C}_\Delta = \{0,1\} \) and \( \mathcal{C}_\Gamma = \{\text{cons}\} \) where \( \text{ar}_\Delta(\text{cons}) = 1 \). Thus, \( \mu = \mu^\text{can}_\mathcal{R} \cup \mu_\Delta \) yields \( \mu(\text{zip}) = \mu(\text{cons}) = \{1\} \). The \( \mu \)-termination of \( \mathcal{R} \) can be proved with MU-TERM and by Theorem 5 productivity of \( \mathcal{R} \) follows.

In general, Theorem 5 does not hold in the opposite direction, i.e., productivity of \( \mathcal{R} \) does not imply its \( \mu \)-termination.

**Example 9** Let \( \mathcal{R} \) be (cf. [8 Example 5.3]):

\[
\begin{align*}
s & \rightarrow b : s \\
f(a, \sigma) & \rightarrow \sigma \\
f(b, x : y : \sigma) & \rightarrow b : f(b, y : \sigma)
\end{align*}
\]

Note that \( \mu^\text{can}_\mathcal{R}(\_\_\_) = \{2\} \) due to the third rule. This makes \( \mathcal{R} \) non-\( \mu^\text{can}_\mathcal{R} \)-terminating due to the first rule. We cannot use Theorem 5 to prove \( \mathcal{R} \) productive, but it is (see Example 10 below).
Regarding constructor normalization, we have:

**Theorem 6** Let $\mathcal{R}$ be a orthogonal strongly compatible TRS such that either

1. $\mu_{\text{can}}^\mathcal{R}(c) = \emptyset$ for all $c \in \mathcal{C}$, or
2. $\mathcal{R}$ contains no collapsing rule and $\mu_{\text{can}}^\mathcal{R}(c) = \emptyset$ for all constructor symbols $c \in \mathcal{C}_\mathcal{R}$ such that $c = \text{root}(r)$ for some $\ell \rightarrow r \in \mathcal{R}$.

If $\mathcal{R}$ is constructor normalizing, then it is $\mu_{\text{can}}^\mathcal{R}$-terminating.

**Proof.** Since $\mathcal{R}$ is constructor normalizing, $\mathcal{R}$ is head-normalizing, i.e., every term $s$ has a (constructor) head-normal form $t$, i.e., $\text{root}(t) \in \mathcal{C}$. By [18, Theorem 4.6], every $\mu_{\text{can}}^\mathcal{R}$-replacing redex in a term $s$ which is not a head-normal form is root-needed (see [23]). Thus, every $\mu_{\text{can}}^\mathcal{R}$-reduction sequence with $\mathcal{R}$ is head-normalizing.

Furthermore, since every term $s$ is head-normalizing, every $\mu_{\text{can}}^\mathcal{R}$-rewrite sequence starting from $s$ yields a head-normal form $t$ which, by confluence of $\mathcal{R}$, is a constructor head-normal form, i.e., $t = c(t_1, \ldots, t_k)$ for some $c \in \mathcal{C}$. We have two cases:

1. If $\mu_{\text{can}}^\mathcal{R}(c) = \emptyset$ for all constructor symbols $c$, then $t$ is a $\mu$-normal form.
2. Otherwise, we can assume that $s$ is not a head-normal form and then, since there is no collapsing rule, the root symbol $c$ of $t$ must be introduced by the last rule applied to the root in the head-normalizing sequence. Hence, by our assumption, $\mu(c) = \emptyset$ as well.

Thus, every $\mu_{\text{can}}^\mathcal{R}$-rewrite sequence starting from any term $s$ is finite and $\mathcal{R}$ is $\mu_{\text{can}}^\mathcal{R}$-terminating.

\[\square\]

5 Related work

In [30], Zantema and Raffelsieper develop a general technique to prove productivity of specifications of infinite objects based on proving context-sensitive termination. In the following result, we use the terminology in Section 4 borrowed from [8]. Consistently, since the notion of ‘productivity’ in [30], corresponds to constructor normalization (see Section 4), we have the following.

**Theorem 7** [30, Theorem 4.1] Let $\mathcal{R}$ be a proper tree specification and $\mu \in M_\mathcal{R}$ given by $\mu(f) = \{1,\ldots, ar(f)\}$ if $f \in \mathcal{D}$ and $\mu(c) = \{1,\ldots, ar_\Delta(c)\}$ if $c \in \mathcal{C}$. If $\mathcal{R}$ is $\mu$-terminating, then $\mathcal{R}$ is constructor normalizing.

**Remark 2** Theorem 7 is a particular case of Theorem 4 proper tree specifications are TRSs with rules $\ell \rightarrow r$ whose left-hand sides $\ell$ contain no nested constructor symbols, i.e., they are of the form $\ell = f(\delta_1, \ldots, \delta_k)$, where $\delta_i$ is either a variable or a flat constructor term $c_i(x_1, \ldots, x_m)$ for some constructor symbol $c_i$ and variables $x_1, \ldots, x_m$. In this case, the replacement map $\mu$ required in Theorem 7 is canonical, i.e., $\mu \in CM_\mathcal{R}$.

Example 6 is given in [30, Example 4.6] to illustrate a tree specification $\mathcal{R}$ where Theorem 7 can not be used to prove constructor normalization. Indeed, $\mathcal{R}$ is not $\mu$-terminating if $\mu$ is defined as required in Theorem 7. In contrast, Theorem 4 was used in Example 6 to prove constructor normalization of $\mathcal{R}$ and Theorem 5 was used in Example 8 to prove productivity of $\mathcal{R}$.

In [8], Endrullis and Hendriks have devised a sound and complete transformation of productivity to context-sensitive termination. The transformation proceeds in two steps. First, an inductively sequential (see [3]) tree specification $\mathcal{R}$ is transformed into a shallow tree specification $\mathcal{R}'$ by a productivity preserving transformation [8, Definition 5.1] and [8, Theorem 5.5]. Here, $\mathcal{R}$ is shallow if for each $k$-ary defined symbol $f \in \mathcal{D}$ there is a set $I_f \subseteq \{1,\ldots, k\}$ such that for each rule $f(p_1, \ldots, p_k) \rightarrow r$, every $p_i$ satisfies [8, Definition 3.14]:
1. If \( i \in I_f \), then \( p_i = c_i(x_1, \ldots, x_m) \) for some \( c \in \mathcal{C} \) and variables \( x_1, \ldots, x_m \in \mathcal{X} \); and
2. If \( i \notin I_f \), then \( p_i \in \mathcal{X} \).

**Example 10** The (inductively sequential) TRS \( \mathcal{R} \) in Example 9 is not shallow, but it is transformed by the first transformation into the following TRS \( \mathcal{R}' \) (adapted from [8, Example 5.3]):

\[
\begin{align*}
s & \to b : s \\
f(a, \sigma) & \to f_a(\sigma) \\
f_a(\sigma) & \to \sigma \\
f(b, \sigma) & \to f_b(\sigma) \\
f_b(x : \sigma) & \to f_b(x, \sigma) \\
f_b(x, y : \sigma) & \to b : f(b, y : \sigma)
\end{align*}
\]

Since \( \mathcal{R}' \) is productive if and only if \( \mathcal{R} \) is, we use now Theorem 5 (with \( \mu = \mu_{\text{can}}^{\mathcal{R}} \), since \( \mu_{\Delta} = \mu_{\perp} \)) to prove \( \mathcal{R} \) productive. This shows (see Example 9) that Theorem 5 does not extend to a characterization of productivity as termination of CSR.

**Proposition 1** Shallow tree specifications \( \mathcal{R} \) are strongly compatible constructor TRSs where \( \mu_{\text{can}}^{\mathcal{R}}(c) = \emptyset \) for all \( c \in \mathcal{C} \).

**Proof.** Let \( \mu(f) = I_f \) for all \( f \in \mathcal{D} \) and \( \mu(f) = \emptyset \) for all \( f \in \mathcal{C} \). For all \( \ell \in L(\mathcal{R}) \), \( \text{Pos}^{\mu}(\ell) = \text{Pos}_{\mathcal{R}}(\ell) \), i.e., \( \mathcal{R} \) is strongly compatible. Since \( \mu_{\text{can}}^{\mathcal{R}} \) is the only replacement map that makes \( \mathcal{R} \) strongly compatible, \( \mu = \mu_{\text{can}}^{\mathcal{R}} \) and \( \mu_{\text{can}}^{\mathcal{R}}(c) = \emptyset \) for all \( c \in \mathcal{C} \). \( \square \)

In Endrullis and Hendriks’ approach, a second transformation obtains a CS-TRS \( (\mathcal{R}'', \mu) \) from \( \mathcal{R}' \) (see [8, Definition 6.1]) in such a way that \( \mu \)-termination of \( \mathcal{R}'' \) is equivalent to productivity of \( \mathcal{R}' \) [8, Theorem 6.6].

**Remark 3** First Endrullis and Hendriks’ transformation preserves productivity. Thus, we can use \( \mathcal{R}' \) together with Theorem 5 to prove productivity of \( \mathcal{R} \) without using the second transformation. We proceed in this way in Example 10 where we conclude productivity of \( \mathcal{R}' \) without using the second transformation described in [8, Definition 6.1].

By Theorem 6 and Proposition 1 we have:

**Corollary 1** Constructor normalizing shallow tree specifications \( \mathcal{R} \) are \( \mu_{\text{can}}^{\mathcal{R}} \)-terminating.

With Theorem 4 we have the following characterization of shallow tree specifications (see also [8, Theorem 6.5]).

**Corollary 2** A shallow tree specification \( \mathcal{R} \) is constructor normalizing if and only if it is \( \mu_{\text{can}}^{\mathcal{R}} \)-terminating.

However, we also have

**Corollary 3** A strongly compatible tree specification \( \mathcal{R} \) without collapsing rules and such that \( \mu_{\text{can}}^{\mathcal{R}}(c) = \emptyset \) for all constructor symbols \( c \in \mathcal{C}_{\mathcal{R}} \) such that \( c = \text{root}(r) \) for some \( \ell \to r \in \mathcal{R} \) is constructor normalizing if and only if it is \( \mu_{\text{can}}^{\mathcal{R}} \)-terminating.

Since productive tree specifications are constructor normalizing, we have the following.

**Corollary 4** Productive shallow tree specifications \( \mathcal{R} \) are \( \mu_{\text{can}}^{\mathcal{R}} \)-terminating.
In [26], Raffelsieper investigates productivity of non-orthogonal TRSs. However, he still requires left-linearity and exhaustiveness of $\mathcal{R}$. Thus, our results in Section 4 also apply to his framework. Raffelsieper also introduces the notion of *strong productivity* meaning that every maximal outermost-fair $\mathcal{R}$-sequence starting from a term of sort $\Delta$ is constructor head-normalizing [26, Definition 6 and Proposition 7]. He also uses termination of CSR to prove strong productivity of his *proper specifications*. He defines a replacement map $\mu_\not\Delta$ (see [26, Definition 11]) which is, however, less restrictive than our replacement map $\mu_\Delta$ in Theorem 5. Thus, his main result in this respect [26, Theorem 12] is a particular case of our Theorem 5.

### 6 Conclusions and future work

We have identified Theorems 1 and 2 (originally in [20]) as bearing the essentials of the use of termination of canonical CSR to prove productivity of rewrite systems (see the proofs of Theorems 4 and 5). Although termination of CSR had been used before to prove (and even characterize) productivity, we believe that our presentation sheds new light on this connection and also shows that the use of such well-known results about CSR also simplifies the proofs of the results that connect termination of CSR and productivity. Furthermore, the use of the canonical replacement map as one of the (bounding) components of the replacement map at stake is new in the literature and improves on previous approaches that systematically use less restrictive replacement maps, thus losing opportunities to prove termination of CSR and hence productivity. We improved Endrullis and Hendriks’ approach because we avoid the use of transformations, being able to directly prove productivity of a non-shallow TRS $\mathcal{R}$ as termination of CSR for $\mathcal{R}$ itself. For instance, we directly prove productivity of $\mathcal{R}$ in Example 3 without any transformation, whereas Endrullis and Hendriks require the addition of new rules due to their second transformation (see [8, Example 6.8]). In Example 10 we conclude productivity of $\mathcal{R}'$ without using their second transformation. As a matter of fact, we were able to find automatic proofs of productivity for all the examples in [8, 26, 30] by using Theorem 5 together with AProVE or MU-TERM to obtain the automatic proofs of termination of CSR. Our results, though, do not provide a characterization of productivity, as witnessed by Examples 9 and 10. In contrast to [8, 30], which deal with orthogonal (constructor-based) TRSs only, our results apply to left-linear TRSs and supersede [26] which applies to non-orthogonal TRSs which are still left-linear.

In the future, we plan to apply other powerful results about completeness of CSR in (infinitary) normalization and computation of (possibly infinite) values to develop more general notions of productivity and apply them to broader classes of programs.

**Acknowledgments.** I thank the anonymous referees for their comments and suggestions.

**References**


Termination of canonical context-sensitive rewriting and productivity of rewrite systems


