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 ℓ 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 ℓ. 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.

Marisa Navarro (Ed.): Proceedings PROLE 2015
EPTCS 200, 2015, pp. 18–31, doi:10.4204/EPTCS.200.2

$$
\begin{array}{rcll}
\mathsf{evenNs} & \to & \mathsf{cons}(0,\mathsf{incr}(\mathsf{oddNs})) & (1)\\
\mathsf{oddNs} & \to & \mathsf{incr}(\mathsf{evenNs}) & (2)\\
\mathsf{incr}(\mathsf{cons}(x,xs)) & \to & \mathsf{cons}(\mathsf{s}(x),\mathsf{incr}(xs)) & (3)\\
\mathsf{take}(0,xs) & \to & \mathsf{nil} & (4)\\
\mathsf{take}(\mathsf{s}(n),\mathsf{cons}(x,xs)) & \to & \mathsf{consF}(x,\mathsf{take}(n,xs)) & (5)\\
\mathsf{zip}(\mathsf{nil},xs) & \to & \mathsf{nil} & (6)\\
\mathsf{zip}(xs,\mathsf{nil}) & \to & \mathsf{nil} & (7)\\
\mathsf{zip}(\mathsf{cons}(x,xs),\mathsf{cons}(y,ys)) & \to & \mathsf{cons}(\mathsf{frac}(x,y),\mathsf{zip}(xs,ys)) & (8)\\
\mathsf{tail}(\mathsf{cons}(x,xs)) & \to & xs & (9)\\
\mathsf{rep2}(\mathsf{nil}) & \to & \mathsf{nil} & (10)\\
\mathsf{rep2}(\mathsf{cons}(x,xs)) & \to & \mathsf{cons}(x,\mathsf{cons}(x,\mathsf{rep2}(xs))) & (11)\\
0+x & \to & x & (12)\\
s(x)+y & \to & \mathsf{s}(x+y) & (13)\\
0\times y & \to & 0 & (14)\\
s(x)\times y & \to & y+(x\times y) & (15)\\
\mathsf{prodFrac}(\mathsf{frac}(x,y),\mathsf{frac}(z,t)) & \to & \mathsf{frac}(x\times z,y\times t) & (16)\\
\mathsf{prodOfFracs}(\mathsf{nil}) & \to & \mathsf{frac}(\mathsf{s}(0),\mathsf{s}(0)) & (17)\\
\mathsf{prodOfFracs}(\mathsf{consF}(p,ps)) & \to & \mathsf{prodFrac}(p,\mathsf{prodOfFracs}(ps)) & (18)\\
\mathsf{halfPi}(n) & \to & \mathsf{prodOfFracs}(\mathsf{take}(n,\mathsf{zip}(\mathsf{rep2}(\mathsf{tail}(\mathsf{evenNs})),\mathsf{tail}(\mathsf{rep2}(\mathsf{oddNs})))))) & (19)
\end{array}
$$

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

Example 1 *The TRS $\mathscr{R}$ in Figure 1 [1, Example 1] can be used to compute approximations to $\frac{\pi}{2}$ as $\frac{\pi}{2} = lim_{n\to\infty}\frac{2}{1}\frac{2}{3}\frac{4}{3}\frac{4}{5}\cdots\frac{2n}{2n-1}\frac{2n}{2n+1}$ (Wallis' product). In $\mathscr{R}$, 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* $\mathsf{halfPi}(\mathsf{s}^n(0))$ *for some $n > 0$ returns the desired approximation whose computation is launched by rule (19).*

Note that $\mathscr{R}$ is nonterminating. *For instance we have the following infinite rewrite sequence:*

$$\underline{\mathsf{evenNs}} \to \mathsf{cons}(0,\mathsf{incr}(\underline{\mathsf{oddNs}})) \to \mathsf{cons}(0,\mathsf{incr}(\mathsf{incr}(\underline{\mathsf{evenNs}}))) \to \cdots \to \cdots \quad (20)$$

Context-sensitive rewriting (CSR [20, 21]) is a restriction of rewriting which imposes fixed, *syntactic* restrictions on reductions by means of a *replacement map* μ 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 $\mathscr{R}$ which (as in Example 1) are not terminating in the unrestricted case.

Example 2 *Let the replacement map μ be given by:*

$$\mu(\mathsf{cons}) = \varnothing \text{ and } \mu(f) = \{1,\ldots,ar(f)\} \text{ for all } f \in \mathscr{F} - \{\mathsf{cons}\}$$

That is, μ disallows rewriting on the arguments of the list constructor cons *(due to $\mu(\mathsf{cons}) = \varnothing$). 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* cons *and $2 \notin \mu(\mathsf{cons})$, i.e.,*

$$\mathsf{cons}(0,\mathsf{incr}(\underline{\mathsf{oddNs}})) \not\hookrightarrow_\mu \mathsf{cons}(0,\mathsf{incr}(\mathsf{incr}(\mathsf{evenNs})))$$

where we write $\hookrightarrow_\mu$ to emphasize that the rewriting step is issued using CSR *under the replacement map μ. This makes the infinite sequence impossible. Termination of* CSR *for the TRS $\mathscr{R}$ and μ 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 $\mathscr{R}$, if the replacement map μ is made *compatible* with the left-hand sides ℓ of the rules $\ell \to r$ of $\mathscr{R}$, then CSR has two properties which are specifically relevant for the purpose of this paper:

1. every μ-normal form (i.e., a term t where no further rewritings are allowed with CSR under μ) 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 μ 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 ℓ are always *reducible* under μ. For instance, μ in Example 1 is a canonical replacement map for $\mathscr{R}$ 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 $\mathscr{R}$ can be used to define* ordinal numbers *[8, Example 6.8]:*

$$\begin{array}{rcl@{\qquad}rcl}
x+0 & \to & x & x\times 0 & \to & 0\\
x+\mathsf{S}(y) & \to & \mathsf{S}(x+y) & x\times \mathsf{S}(y) & \to & (x\times y)+x\\
x+\mathsf{L}(\sigma)) & \to & \mathsf{L}(x+_L \sigma) & x\times \mathsf{L}(\sigma) & \to & \mathsf{L}(x\times_L \sigma)\\
x+_L (y:\sigma) & \to & (x+y):(x+_L \sigma) & x\times_L (y:\sigma) & \to & (x\times y):(x\times_L \sigma)\\
\mathsf{nats}(x) & \to & x:\mathsf{nats}(\mathsf{S}(x)) & \omega & \to & \mathsf{L}(\mathsf{nats}(0))
\end{array}$$

Here, $\mathsf{0}$ *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* ω *is given as the limit* $\mathsf{L}(\mathsf{nats}(0))$ *of* $\mathsf{nats}(0)$*, 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* $\times$*. Endrullis and Hendriks use a transformation which introduces the replacement map* $\mu(+)=\mu(+_L)=\mu(\times)=\mu(\times_L)=\{2\}$*,* $\mu(\mathsf{S})=\{1\}$*, and* $\mu(\mathsf{L})=\mu(:)=\mu(\mathsf{nats})=\varnothing$*, and also adds some rules to prove* $\mathscr{R}$ 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 'permisive' replacement maps which can fail to conclude productivity as termination of CSR in many cases. For instance, we can prove productivity of $\mathscr{R}$ in Example 3 as termination of CSR for the replacement map μ 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 μ' that would be obtained according to [30], $\mathscr{R}$ 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 $\mathscr{R}$ 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, $\mathscr{X}$ denotes a countable set of variables and $\mathscr{F}$ denotes a signature, i.e., a set of function

symbols $\{\mathtt{f},\mathtt{g},\ldots\}$, each having a fixed arity given by a mapping $ar : \mathscr{F} \to \mathbb{N}$. The set of terms built from $\mathscr{F}$ and $\mathscr{X}$ is $\mathscr{T}(\mathscr{F},\mathscr{X})$. Given a (set of) term(s) $t \in \mathscr{T}(\mathscr{F},\mathscr{X})$ (resp. $T \subseteq \mathscr{T}(\mathscr{F},\mathscr{X})$), we write $\mathscr{F}(t)$ (resp. $\mathscr{F}(T)$) to denote the subset of symbols in $\mathscr{F}$ occurring in t (resp. T). A term is said to be linear if it has no multiple occurrences of a single variable. Terms are viewed as labelled trees in the usual way. Positions $p,q,\ldots$ are represented by chains of positive natural numbers used to address subterms of t. Given positions p,q, we denote its concatenation as $p.q$. Positions are ordered by the standard prefix ordering $\leq$. Given a set of positions P, $minimal_{\leq}(P)$ is the set of minimal positions of P w.r.t. $\leq$. If p is a position, and Q is a set of positions, $p.Q = \{p.q \mid q \in Q\}$. We denote the empty chain by Λ. The set of positions of a term t is $\mathscr{P}os(t)$. Positions of non-variable symbols in t are denoted as $\mathscr{P}os_{\mathscr{F}}(t)$, and $\mathscr{P}os_{\mathscr{X}}(t)$ are the positions of variables. The subterm at position p of t is denoted as $t|_p$ and $t[s]_p$ is the term t with the subterm at position p replaced by s. The symbol labelling the root of t is denoted as $root(t)$. Given terms t and s, $\mathscr{P}os_s(t)$ denotes the set of positions of s in t, i.e., $\mathscr{P}os_s(t) = \{p \in \mathscr{P}os(t) \mid t|_p = s\}$. A substitution is a mapping $\sigma : \mathscr{X} \to \mathscr{T}(\mathscr{F},\mathscr{X})$ which is homomorphically extended to a mapping $\sigma : \mathscr{T}(\mathscr{F},\mathscr{X}) \to \mathscr{T}(\mathscr{F},\mathscr{X})$ which, by abuse, we denote using the same symbol σ.

A rewrite rule is an ordered pair (l,r), written $l \to r$, with $l,r \in \mathscr{T}(\mathscr{F},\mathscr{X})$, $l \notin \mathscr{X}$ and $\mathscr{V}ar(r) \subseteq \mathscr{V}ar(l)$. The left-hand side (*lhs*) of the rule is l and r is the right-hand side (*rhs*). A TRS is a pair $\mathscr{R} = (\mathscr{F},R)$ where R is a set of rewrite rules. $L(\mathscr{R})$ denotes the set of *lhs*'s of $\mathscr{R}$. An instance $\sigma(l)$ of a *lhs* l of a rule is a redex. The set of redex positions in t is $\mathscr{P}os_{\mathscr{R}}(t)$. A TRS $\mathscr{R}$ is left-linear if for all $l \in L(\mathscr{R})$, l is a linear term. Given $\mathscr{R} = (\mathscr{F},R)$, we consider $\mathscr{F}$ as the disjoint union $\mathscr{F} = \mathscr{C} \uplus \mathscr{D}$ of symbols $c \in \mathscr{C}$, called *constructors* and symbols $f \in \mathscr{D}$, called *defined functions*, where $\mathscr{D} = \{root(l) \mid l \to r \in R\}$ and $\mathscr{C} = \mathscr{F} - \mathscr{D}$. Then, $\mathscr{T}(\mathscr{C},\mathscr{X})$ (resp. $\mathscr{T}(\mathscr{C})$) is the set of constructor (resp. ground constructor) terms. A TRS $\mathscr{R} = (\mathscr{C} \uplus \mathscr{D},R)$ is a *constructor system* (CS) if for all $f(\ell_1,\ldots,\ell_k) \to r \in R$, $\ell_i \in \mathscr{T}(\mathscr{C},\mathscr{X})$, for $1 \leq i \leq k$.

A term $t \in \mathscr{T}(\mathscr{F},\mathscr{X})$ rewrites to s (at position p), written $t \xrightarrow{p}_{\mathscr{R}} s$ (or just $t \to s$), if $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$, for some rule $\rho : l \to r \in R$, $p \in \mathscr{P}os(t)$ and substitution σ. 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 $\mu : \mathscr{F} \to \wp(\mathbb{N})$ is a *replacement map* ($\mathscr{F}$-map) if for all $f \in \mathscr{F}$, $\mu(f) \subseteq \{1,\ldots,ar(f)\}$ [16, 20]. $M_{\mathscr{F}}$ is the set of $\mathscr{F}$-maps. Replacement maps can be compared according to their 'restriction power': $\mu \sqsubseteq \mu'$ if for all $f \in \mathscr{F}$, $\mu(f) \subseteq \mu'(f)$. If $\mu \sqsubseteq \mu'$, we say that μ is *more restrictive* than μ'. Then, $(\wp(\mathbb{N}),\subseteq,\varnothing,\mathbb{N},\cup)$ induces a complete lattice $(M_{\mathscr{F}},\sqsubseteq,\mu_\bot,\mu_\top,\sqcup)$: the minimum (maximum) element is $\mu_\bot$ ($\mu_\top$), given by $\mu_\bot(f) = \varnothing$ ($\mu_\top(f) = \{1,\ldots,ar(f)\}$) for all $f \in \mathscr{F}$. The *lub* $\sqcup$ is given by $(\mu \sqcup \mu')(f) = \mu(f) \cup \mu'(f)$ for all $f \in \mathscr{F}$.

The replacement restrictions introduced by a replacement map μ on the *arguments* of function *symbols* are raised to *positions* of *terms* $t \in \mathscr{T}(\mathscr{F},\mathscr{X})$: the set $\mathscr{P}os^{\mu}(t)$ of *μ-replacing positions* of t is:

$$\mathscr{P}os^{\mu}(t) = \begin{cases} \{\Lambda\} & \text{if } t \in \mathscr{X} \\ \{\Lambda\} \cup \bigcup_{i \in \mu(root(t))} i.\mathscr{P}os^{\mu}(t|_i) & \text{if } t \notin \mathscr{X} \end{cases}$$

Given terms $s,t \in \mathscr{T}(\mathscr{F},\mathscr{X})$, $\mathscr{P}os^{\mu}_s(t)$ is the set of positions corresponding to μ-replacing occurrences of s in t: $\mathscr{P}os^{\mu}_s(t) = \mathscr{P}os^{\mu}(t) \cap \mathscr{P}os_s(t)$. The set of *$\mu$-replacing variables* occurring in $t \in \mathscr{T}(\mathscr{F},\mathscr{X})$ is $\mathscr{V}ar^{\mu}(t) = \{x \in \mathscr{X} \mid \mathscr{P}os^{\mu}_x(t) \neq \varnothing\}$.

3.1 Canonical replacement map

Given $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, 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, μ is called *strongly compatible* with t if $\mathcal{P}os_{\mathcal{F}}(t) = \mathcal{P}os^{\mu}(t)$. And μ is (strongly) compatible with $T \subseteq \mathcal{T}(\mathcal{F},\mathcal{X})$ if for all $t \in T$, μ is (strongly) compatible with t [18, 20]. The *minimum* replacement map which is compatible with $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is [20]:

$$\mu_t = \begin{cases} \mu_\perp & \text{if } t \in \mathcal{X} \\ \mu_t^\Lambda \sqcup \mu_{t|_1} \sqcup \cdots \sqcup \mu_{t|_{ar(root(t))}} & \text{if } t \notin \mathcal{X} \end{cases}$$

with $\mu_t^\Lambda(root(t)) = \{i \in \{1,\ldots,ar(root(t))\} \mid t|_i \notin \mathcal{X}\}$ and $\mu_t^\Lambda(f) = \varnothing$ if $f \neq root(t)$.

For a TRS $\mathcal{R} = (\mathcal{F},R)$, we use $M_{\mathcal{R}}$ instead of $M_{\mathcal{F}}$. The canonical replacement map $\mu_{\mathcal{R}}^{can}$ of $\mathcal{R}$ is *the most restrictive replacement map ensuring that the non-variable subterms of the left-hand sides of the rules of $\mathcal{R}$ are active*.

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

Note that $\mu_{\mathcal{R}}^{can}$ can be automatically associated to $\mathcal{R}$ by means of a very simple calculus: for each symbol $f \in \mathcal{F}$ and $i \in \{1,\ldots,ar(f)\}$, $i \in \mu_{\mathcal{R}}^{can}(f)$ iff $\exists l \in L(\mathcal{R}), p \in \mathcal{P}os_{\mathcal{F}}(l), (root(l|_p) = f \wedge p.i \in \mathcal{P}os_{\mathcal{F}}(l))$. Given a TRS $\mathcal{R}$, $CM_{\mathcal{R}} = \{\mu \in M_{\mathcal{R}} \mid \mu_{\mathcal{R}}^{can} \sqsubseteq \mu\}$ 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 μ is *a* canonical replacement map for $\mathcal{R}$.

Example 4 *For $\mathcal{R}$ in Example 3, we have*[1]*:*

$$\begin{array}{rcl} \mu_{\mathcal{R}}^{can}(\mathsf{S}) = \mu_{\mathcal{R}}^{can}(\mathsf{L}) = \mu_{\mathcal{R}}^{can}(\mathsf{nats}) = \mu_{\mathcal{R}}^{can}(:) & = & \varnothing \\ \mu_{\mathcal{R}}^{can}(+) = \mu_{\mathcal{R}}^{can}(+_L) = \mu_{\mathcal{R}}^{can}(\times) = \mu_{\mathcal{R}}^{can}(\times_L) & = & \{2\} \end{array}$$

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

Note that, μ in Example 3 prescribes $\mu(\mathsf{S}) = \{1\}$. Thus, $\mu_{\mathcal{R}}^{can} \sqsubset \mu$ and $\mu \in CM_{\mathcal{R}}$ but $\mu \neq \mu_{\mathcal{R}}^{can}$.

3.2 Strongly compatible TRSs

Given $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$, the only $\mathcal{F}(t)$-map μ (if any) which is strongly compatible with t is μ_t [18, Proposition 3.6]. We call $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ strongly compatible if μ_t is strongly compatible with t. Similarly, the only $\mathcal{F}(T)$-map μ which can be strongly compatible with T is $\mu_T = \sqcup_{t \in T} \mu_t$. We call T *strongly compatible* if μ_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 $\mu_{\mathcal{R}}^{can}$. For instance, $\mathcal{R}$ in Example 3 is strongly compatible, but μ is *not* strongly compatible with $L(\mathcal{R})$ (variable y in the left-hand side of the second rule is μ-replacing).

[1] The specification for constant symbols a is omitted, as it is always the empty set $\mu(a) = \varnothing$.

3.3 Context-sensitive rewriting

Given a TRS $\mathscr{R} = (\mathscr{F}, R)$, $\mu \in M_{\mathscr{R}}$, and $s, t \in \mathscr{T}(\mathscr{F}, \mathscr{X})$, s μ-rewrites to t at position p, written $s \overset{p}{\hookrightarrow}_{\mathscr{R},\mu} t$ (or $s \hookrightarrow_{\mathscr{R},\mu} t$, $s \hookrightarrow_{\mu} t$, or even $s \hookrightarrow t$), if $s \overset{p}{\to}_{\mathscr{R}} t$ and $p \in \mathscr{P}os^{\mu}(s)$ [16, 20]. A TRS $\mathscr{R}$ is μ-terminating if $\hookrightarrow_{\mu}$ 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 $\mathscr{R}$ together with a canonical replacement map $\mu \in CM_{\mathscr{R}}$, we often say that $\hookrightarrow_{\mu}$ performs* canonical *context-sensitive rewriting steps [21].*

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

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

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

4 Productivity and termination of CSR

The operational semantics of rewriting-based programming languages can be abstracted, for each program (i.e., TRS) $\mathscr{R}$, as a mapping from terms $s \in \mathscr{T}(\mathscr{F}, \mathscr{X})$ into (possibly empty) sets of (possibly infinite) terms $T_s \subseteq \mathscr{T}^{\omega}(\mathscr{F}, \mathscr{X})$, 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* $t \in \mathscr{T}(\mathscr{C})$ are the meaningful reducts of (ground) initial expressions s (*evaluation* semantics) and $T_s \subseteq \mathscr{T}(\mathscr{C})$.
2. In *lazy* functional programming *infinite values* are also accepted in the semantic description, i.e., $T_s \subseteq \mathscr{T}^{\omega}(\mathscr{C})$, but the infinite terms are not actually obtained but only *approximated* as sequences of appropriate finite terms which are *prefixes* of the infinite values[2].
3. In *equational programming* and rewriting-based theorem provers, computing *normal forms* is envisaged (*normalization* semantics), i.e., $T_s \subseteq \mathsf{NF}_{\mathscr{R}}$.

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' (i.e., $s \to^* t'$) which is then rewritten (below the root!) into t. When a head-normal form t' is obtained, the root symbol $f = root(t')$ 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]. The

[2] Such finite approximations to infinite terms are described as *partial values* using a special symbol $\bot$ to denote undefinedness. An infinite value $\delta \in \mathscr{T}^{\omega}(\mathscr{C})$ is the limit of an infinite sequence $\delta_1, \ldots, \delta_n, \ldots$ of such partial values where, for all $i \ge 1$, $\delta_{i+1} \in \mathscr{T}(\mathscr{C} \cup \{\bot\})$ is obtained from $\delta_i \in \mathscr{T}(\mathscr{C} \cup \{\bot\})$ by replacing occurrences of $\bot$ in δ_i by partial values different from $\bot$.

set of sorts $\mathscr{S}$ is partitioned into $\mathscr{S} = \Delta \cup \Gamma$, where Δ is the set of data sorts, intended to model inductive data types (booleans, natural numbers, finite lists, etc.). On the other hand, Γ is the set of *codata* sorts, intended to model coinductive datatypes such as streams and infinite trees. Terms of sort Δ are called *data terms* and terms of sorts Γ 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 Δ (resp. Γ). 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* $\mathscr{R}$ *where* $\Delta \cap \Gamma = \varnothing$.

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

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

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

Being exhaustive is a necessary condition for productivity.

Theorem 3 *If* $\mathscr{R}$ *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 $\mathscr{R}$ being constructor normalizing. □

Theorem 4 *Let* $\mathscr{R}$ *be an exhaustive, left-linear TRS and* $\mu \in CM_{\mathscr{R}}$. *If* $\mathscr{R}$ *is* μ*-terminating, then* $\mathscr{R}$ *is constructor normalizing.*

PROOF. Since $\mathscr{R}$ is μ-terminating, every ground term s has a (finite) μ-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 $\delta \in \mathscr{T}^\omega(\mathscr{C})$. If t is a constant, then since t is a μ-normal form, it must be a normal form. Since $\mathscr{R}$ is exhaustive, $t = \delta \in \mathscr{T}(\mathscr{C})$. If $t = f(t_1,\ldots,t_k)$ for ground terms $t_1,\ldots,t_k$, then by the induction hypothesis, for all i, $1 \le i \le k$, t_i has a (possibly infinite) constructor normal form $\delta_i \in \mathscr{T}^\omega(\mathscr{C})$. We have two cases:

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

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

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

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

$$\begin{aligned} \mathsf{p} &\to \mathsf{zip}(\mathsf{alt},\mathsf{p}) \\ \mathsf{alt} &\to 0 : 1 : \mathsf{alt} \\ \mathsf{zip}(x:\sigma,\tau) &\to x : \mathsf{zip}(\tau,\sigma) \end{aligned}$$

(where no constant for empty lists is included!) is easily proved $\mu_{\mathscr{R}}^{can}$-terminating (use MU-TERM*). By Theorem 4, it is constructor normalizing. Note that $\mathscr{R}$ is* exhaustive *due to the sort discipline (for instance, $zip(0,0)$ 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 $cons(s,t)$ for terms s, t where t is always infinite).*

As remarked in [8, Section 3.2], several authors define $\mathscr{R}$ 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 $t \in \mathscr{T}^{\omega}(\mathscr{F},\mathscr{X})$ and $\mathscr{F}' \subseteq \mathscr{F}$, a $\mathscr{F}'$-path in t is a (finite or infinite) sequence $\langle p_1,c_1\rangle$, $\langle p_2,c_2\rangle,\ldots$ such that $c_i = root(t|_{p_i}) \in \mathscr{F}'$ and $p_{i+1} = p_i.j$ with $1 \le j \le ar(c_i)$ [8, Definition 3.7].

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

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

In the following result, μ_Δ is given by $\mu_\Delta(c) = \{1,\ldots,ar_\Delta(c)\}$ for all $c \in \mathscr{C}_\Delta$, and $\mu_\Delta(f) = \varnothing$ for all other symbols f.

Theorem 5 *Let $\mathscr{R}$ be a left-linear, exhaustive TRS and $\mu \in M_{\mathscr{R}}$ be such that $\mu_{\mathscr{R}}^{can} \sqcup \mu_\Delta \sqsubseteq \mu$. If $\mathscr{R}$ is μ-terminating, then $\mathscr{R}$ is productive.*

PROOF. Since $\mu_{\mathscr{R}}^{can} \sqsubseteq \mu$, constructor normalization of $\mathscr{R}$ follows by Theorem 4. Thus, if $\mathscr{R}$ is not productive, there must be a ground normal form t of a term s with an infinite $\mathscr{C}_\Delta$-path. Without loss of generality, we can assume that $s \to^* s_1 = c_1(s_1^1,\ldots,s_{k_1}^1)$ for some $c_1 \in \mathscr{C}_\Delta$, and then $s_{i_1}^1 \to^* c_2(s_1^2,\ldots,s_{k_2}^2)$ for some i_1, $1 \le i_1 \le ar_\Delta(c_1)$ and $c_2 \in \mathscr{C}_\Delta$, etc., in such a way that this reduction sequences follow the computation of t and produce the $\mathscr{C}_\Delta$-path $\langle\Lambda,c_1\rangle$, $\langle i_1,c_2\rangle$, $\langle i_1.i_2,c_3\rangle,\ldots$

By Theorem 2, $s \hookrightarrow^* \overline{s}_1 = c_1(\overline{s}_1^1,\ldots,\overline{s}_{k_1}^1)$ for some terms $\overline{s}_1^1,\ldots,\overline{s}_{k_1}^1$ such that $\overline{s}_j^1 \to^* s_j^1$ for all j, $1 \le j \le k_1$. Thus, by Theorem 2 we also have $\overline{s}_{i_1}^1 \hookrightarrow^* c_2(\overline{s}_1^2,\ldots,\overline{s}_{k_2}^2)$ and $\overline{s}_j^2 \to^* s_j^2$ for all j, $1 \le j \le k_2$. Since $i_1 \in \mu_\Delta(c_1)$, we have $s \hookrightarrow^* \overline{s}_2 = c_1(\overline{s}_1^1,\ldots,\overline{s}_{i_1-1}^1,c_2(\overline{s}_1^2,\ldots,\overline{s}_{i_2}^2,\ldots,\overline{s}_{k_2}^2),\ldots,\overline{s}_{k_1}^1)$ with $\overline{s}_{i_2}^2 \to^* s_{i_2}^2$ again. Since $i_1.i_2 \in \mathscr{P}os^{\mu_\Delta}(\overline{s}_2)$, we can continue with this construction to obtain an infinite μ-rewriting sequence which contradicts μ-termination of $\mathscr{R}$. □

Example 7 *For the tree specification $\mathscr{R}$ in Example 3 (see also Example 5), we have $ar_\Delta(\mathsf{S}) = 1$ and $ar_\Delta(\mathsf{L}) = 0$. Then, $\mu_\Delta(\mathsf{S}) = \{1\}$ and $\mu_\Delta(\mathsf{L}) = \varnothing$. Now $\mu = \mu_{\mathscr{R}}^{can} \sqcup \mu_\Delta$ is as given in Example 3. The μ-termination of $\mathscr{R}$ can be proved with* MU-TERM*. By Theorem 5, productivity of $\mathscr{R}$ follows.*

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

In general, Theorem 5 does *not* hold in the opposite direction, i.e., productivity of $\mathscr{R}$ does not imply its μ-termination.

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

$$\begin{aligned} \mathsf{s} &\to \mathsf{b} : \mathsf{s} \\ \mathsf{f}(\mathsf{a},\sigma) &\to \sigma \\ \mathsf{f}(\mathsf{b},x:y:\sigma) &\to \mathsf{b} : \mathsf{f}(\mathsf{b},y:\sigma) \end{aligned}$$

Note that $\mu_{\mathscr{R}}^{can}(:) = \{2\}$ due to the third rule. This makes $\mathscr{R}$ non-$\mu_{\mathscr{R}}^{can}$-terminating due to the first rule. We cannot use Theorem 5 to prove $\mathscr{R}$ productive, but it is (see Example 10 below).

Regarding constructor normalization, we have:

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

1. *$\mu^{can}_{\mathscr{R}}(c) = \varnothing$ for all $c \in \mathscr{C}$, or*
2. *$\mathscr{R}$ contains no collapsing rule and $\mu^{can}_{\mathscr{R}}(c) = \varnothing$ for all constructor symbols $c \in \mathscr{C}_{\mathscr{R}}$ such that $c = root(r)$ for some $\ell \to r \in \mathscr{R}$.*

If $\mathscr{R}$ is constructor normalizing, then it is $\mu^{can}_{\mathscr{R}}$-terminating.

PROOF. Since $\mathscr{R}$ is constructor normalizing, $\mathscr{R}$ is head-normalizing, i.e., every term s has a (constructor) head-normal form t, i.e., $root(t) \in \mathscr{C}$. By [18, Theorem 4.6], every $\mu^{can}_{\mathscr{R}}$-replacing redex in a term s which is not a head-normal form is root-needed (see [23]). Thus, every $\mu^{can}_{\mathscr{R}}$-reduction sequence with $\mathscr{R}$ is head-normalizing. Furthermore, since every term s is head-normalizing, every $\mu^{can}_{\mathscr{R}}$-rewrite sequence starting from s yields a head-normal form t which, by confluence of $\mathscr{R}$, is a constructor head-normal form, i.e., $t = c(t_1,\ldots,t_k)$ for some $c \in \mathscr{C}$. We have two cases:

1. If $\mu^{can}_{\mathscr{R}}(c) = \varnothing$ for all constructor symbols c, then t is a μ-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, $\mu(c) = \varnothing$ as well.

Thus, every $\mu^{can}_{\mathscr{R}}$-rewrite sequence starting from any term s is finite and $\mathscr{R}$ is $\mu^{can}_{\mathscr{R}}$-terminating. □

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 $\mathscr{R}$ be a* proper *tree specification and $\mu \in M_{\mathscr{R}}$ given by $\mu(f) = \{1,\ldots,ar(f)\}$ if $f \in \mathscr{D}$ and $\mu(c) = \{1,\ldots,ar_{\Delta}(c)\}$ if $c \in \mathscr{C}$. If $\mathscr{R}$ is μ-terminating, then $\mathscr{R}$ is constructor normalizing.*

Remark 2 *Theorem 7 is a particular case of Theorem 4: proper tree specifications are TRSs with rules $\ell \to r$ whose left-hand sides ℓ contain no nested constructor symbols, i.e., they are of the form $\ell = f(\delta_1,\ldots,\delta_k)$, where δ_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 μ required in Theorem 7 is* canonical, *i.e., $\mu \in CM_{\mathscr{R}}$.*

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

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 $\mathscr{R}$ is transformed into a *shallow* tree specification $\mathscr{R}'$ by a *productivity preserving* transformation [8, Definition 5.1] and [8, Theorem 5.5]. Here, $\mathscr{R}$ is *shallow* if for each k-ary defined symbol $f \in \mathscr{D}$ there is a set $I_f \subseteq \{1,\ldots,k\}$ such that for each rule $f(p_1,\ldots,p_k) \to r$, 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 $c \in \mathscr{C}$ and variables $x_1,\ldots,x_m \in \mathscr{X}$; and
2. If $i \notin I_f$, then $p_i \in \mathscr{X}$.

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

$$\begin{array}{rcl} \mathsf{s} & \to & \mathsf{b} : \mathsf{s} \\ \mathsf{f}(\mathsf{a},\sigma) & \to & \mathsf{f}_a(\sigma) \\ \mathsf{f}_a(\sigma) & \to & \sigma \\ \mathsf{f}(\mathsf{b},\sigma) & \to & \mathsf{f}_b(\sigma) \\ \mathsf{f}_b(x:\sigma) & \to & \mathsf{f}_{b:}(x,\sigma) \\ \mathsf{f}_{b:}(x,y:\sigma) & \to & \mathsf{b} : \mathsf{f}(\mathsf{b},y:\sigma) \end{array}$$

Since $\mathscr{R}'$ is productive if and only if $\mathscr{R}$ is, we use now Theorem 5 (with $\mu = \mu^{can}_{\mathscr{R}}$, since $\mu_\Delta = \mu_\perp$) to prove $\mathscr{R}$ 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 $\mathscr{R}$ are strongly compatible constructor TRSs where $\mu^{can}_{\mathscr{R}}(c) = \varnothing$ for all $c \in \mathscr{C}$.*

PROOF. Let $\mu(f) = I_f$ for all $f \in \mathscr{D}$ and $\mu(f) = \varnothing$ for all $f \in \mathscr{C}$. For all $\ell \in L(\mathscr{R})$, $\mathscr{P}os^{\mu}(\ell) = \mathscr{P}os_{\mathscr{F}}(\ell)$, i.e., $\mathscr{R}$ is strongly compatible. Since $\mu^{can}_{\mathscr{R}}$ is the only replacement map that makes $\mathscr{R}$ strongly compatible, $\mu = \mu^{can}_{\mathscr{R}}$ and $\mu^{can}_{\mathscr{R}}(c) = \varnothing$ for all $c \in \mathscr{C}$. □

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

Remark 3 *First Endrullis and Hendriks' transformation preserves productivity. Thus, we can use $\mathscr{R}'$ together with Theorem 5 to prove productivity of $\mathscr{R}$* without using *the second transformation. We proceed in this way in Example 10, where we conclude productivity of $\mathscr{R}'$ 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 $\mathscr{R}$ are $\mu^{can}_{\mathscr{R}}$-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 $\mathscr{R}$ is constructor normalizing if and only if it is $\mu^{can}_{\mathscr{R}}$-terminating.*

However, we also have

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

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

Corollary 4 *Productive shallow tree specifications $\mathscr{R}$ are $\mu^{can}_{\mathscr{R}}$-terminating.*

In [26], Raffelsieper investigates productivity of non-orthogonal TRSs. However, he still requires left-linearity and exhaustiveness of $\mathscr{R}$. 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 $\mathscr{R}$-sequence starting from a term of sort Δ 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 $\mu_{\mathscr{S}}$ (see [26, Definition 11]) which is, however, *less* restrictive than our replacement map μ_Δ 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 $\mathscr{R}$ as termination of CSR for $\mathscr{R}$ itself. For instance, we directly prove productivity of $\mathscr{R}$ 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 $\mathscr{R}'$ 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

[1] Beatriz Alarcón, Raúl Gutiérrez & Salvador Lucas (2010): *Context-sensitive dependency pairs*. *Inf. Comput.* 208(8), pp. 922–968, doi:10.1016/j.ic.2010.03.003.

[2] Beatriz Alarcón, Raúl Gutiérrez, Salvador Lucas & Rafael Navarro-Marset (2010): *Proving Termination Properties with mu-term*. In Michael Johnson & Dusko Pavlovic, editors: *Algebraic Methodology and Software Technology - 13th International Conference, AMAST 2010, Lac-Beauport, QC, Canada, June 23-25, 2010. Revised Selected Papers, Lecture Notes in Computer Science* 6486, Springer, pp. 201–208, doi:10.1007/978-3-642-17796-5_12.

[3] Sergio Antoy (1992): *Definitional Trees*. In Hélène Kirchner & Giorgio Levi, editors: *Algebraic and Logic Programming, Third International Conference, Volterra, Italy, September 2-4, 1992, Proceedings*, *Lecture Notes in Computer Science* 632, Springer, pp. 143–157, doi:10.1007/BFb0013825.

[4] Franz Baader & Tobias Nipkow (1998): *Term rewriting and all that*. Cambridge University Press.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer & Carolyn L. Talcott, editors (2007): *All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*. *Lecture Notes in Computer Science* 4350, Springer, doi:10.1007/978-3-540-71999-1.

[6] Nachum Dershowitz, Stéphane Kaplan & David A. Plaisted (1991): *Rewrite, Rewrite, Rewrite, Rewrite, Rewrite, . . Theor. Comput. Sci.* 83(1), pp. 71–96, doi:10.1016/0304-3975(91)90040-9.

[7] Jörg Endrullis, Clemens Grabmayer, Dimitri Hendriks, Ariya Isihara & Jan Willem Klop (2010): *Productivity of stream definitions*. *Theor. Comput. Sci.* 411(4-5), pp. 765–782, doi:10.1016/j.tcs.2009.10.014.

[8] Jörg Endrullis & Dimitri Hendriks (2011): *Lazy productivity via termination*. *Theor. Comput. Sci.* 412(28), pp. 3203–3225, doi:10.1016/j.tcs.2011.03.024.

[9] Kokichi Futatsugi, Joseph A. Goguen, Jean-Pierre Jouannaud & José Meseguer (1985): *Principles of OBJ2*. In Mary S. Van Deusen, Zvi Galil & Brian K. Reid, editors: *Conference Record of the Twelfth Annual ACM Symposium on Principles of Programming Languages, New Orleans, Louisiana, USA, January 1985*, ACM Press, pp. 52–66, doi:10.1145/318593.318610.

[10] Kokichi Futatsugi & Ataru T. Nakagawa (1997): *An Overview of CAFE Specification Environment - An Algebraic Approach for Creating, Verifying, and Maintaining Formal Specifications over Networks*. In: *ICFEM*, p. 170, doi:10.1109/ICFEM.1997.630424.

[11] Jürgen Giesl, Peter Schneider-Kamp & René Thiemann (2006): *Automatic Termination Proofs in the Dependency Pair Framework*. In Ulrich Furbach & Natarajan Shankar, editors: *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, *Lecture Notes in Computer Science* 4130, Springer, pp. 281–286, doi:10.1007/11814771_24.

[12] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi & Jean-Pierre Jouannaud (2000): *Introducing OBJ*. In: *Software Engineering with OBJ: Algebraic Specification in Action*, Kluwer, pp. 3–167, doi:10.1007/978-1-4757-6541-0.

[13] Joe Hendrix & José Meseguer (2007): *On the Completeness of Context-Sensitive Order-Sorted Specifications*. In Franz Baader, editor: *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, *Lecture Notes in Computer Science* 4533, Springer, pp. 229–245, doi:10.1007/978-3-540-73449-9_18.

[14] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph H. Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Richard B. Kieburtz, Rishiyur S. Nikhil, Will Partain & John Peterson (1992): *Report on the Programming Language Haskell, A Non-strict, Purely Functional Language*. *SIGPLAN Notices* 27(5), p. 1, doi:10.1145/130697.130699.

[15] Richard Kennaway, Jan Willem Klop, M. Ronan Sleep & Fer-Jan de Vries (1995): *Transfinite Reductions in Orthogonal Term Rewriting Systems*. *Inf. Comput.* 119(1), pp. 18–38, doi:10.1006/inco.1995.1075.

[16] Salvador Lucas (1995): *Fundamentals of Context=Sensitive Rewriting*. In Miroslav Bartosek, Jan Staudek & Jirí Wiedermann, editors: *SOFSEM '95, 22nd Seminar on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic, November 23 - December 1, 1995, Proceedings*, *Lecture Notes in Computer Science* 1012, Springer, pp. 405–412, doi:10.1007/3-540-60609-2_25.

[17] Salvador Lucas (1996): *Context-Sensitive Computations in Confluent Programs*. In Herbert Kuchen & S. Doaitse Swierstra, editors: *Programming Languages: Implementations, Logics, and Programs, 8th International Symposium, PLILP'96, Aachen, Germany, September 24-27, 1996, Proceedings*, *Lecture Notes in Computer Science* 1140, Springer, pp. 408–422, doi:10.1007/3-540-61756-6_100.

[18] Salvador Lucas (1997): *Needed Reductions with Context-Sensitive Rewriting*. In Michael Hanus, Jan Heering & Karl Meinke, editors: *Algebraic and Logic Programming, 6th International Joint Conference, ALP '97 -*

HOA '97, Southampton, U.K., Spetember 3-5, 1997, Proceedings, *Lecture Notes in Computer Science* 1298, Springer, pp. 129–143, doi:10.1007/BFb0027007.

[19] Salvador Lucas (1997): *Transformations for Efficient Evaluations in Functional Programming*. In Hugh Glaser, Pieter H. Hartel & Herbert Kuchen, editors: *Programming Languages: Implementations, Logics, and Programs, 9th International Symposium, PLILP'97, Including a Special Trach on Declarative Programming Languages in Education, Southampton, UK, September 3-5, 1997, Proceedings*, *Lecture Notes in Computer Science* 1292, Springer, pp. 127–141, doi:10.1007/BFb0033841.

[20] Salvador Lucas (1998): *Context-sensitive Computations in Functional and Functional Logic Programs*. *Journal of Functional and Logic Programming* 1998(1).

[21] Salvador Lucas (2002): *Context-Sensitive Rewriting Strategies*. *Inf. Comput.* 178(1), pp. 294–343, doi:10.1006/inco.2002.3176.

[22] Salvador Lucas (2015): *Completeness of context-sensitive rewriting*. *Inf. Process. Lett.* 115(2), pp. 87–92, doi:10.1016/j.ipl.2014.07.004.

[23] Aart Middeldorp (1997): *Call by Need Computations to Root-Stable Form*. In Peter Lee, Fritz Henglein & Neil D. Jones, editors: *Conference Record of POPL'97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, Paris, France, 15-17 January 1997*, ACM Press, pp. 94–105, doi:10.1145/263699.263711.

[24] Masaki Nakamura, Kazuhiro Ogata & Kokichi Futatsugi (2010): *Reducibility of operation symbols in term rewriting systems and its application to behavioral specifications*. *J. Symb. Comput.* 45(5), pp. 551–573, doi:10.1016/j.jsc.2010.01.008.

[25] Enno Ohlebusch (2002): *Advanced topics in term rewriting*. Springer, doi:10.1007/978-1-4757-3661-8.

[26] Matthias Raffelsieper (2011): *Productivity of Non-Orthogonal Term Rewrite Systems*. In Santiago Escobar, editor: *Proceedings 10th International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2011, Novi Sad, Serbia, 29 May 2011.*, *EPTCS* 82, pp. 53–67, doi:10.4204/EPTCS.82.4.

[27] Ben A. Sijtsma (1989): *On the Productivity of Recursive List Definitions*. *ACM Trans. Program. Lang. Syst.* 11(4), pp. 633–649, doi:10.1145/69558.69563.

[28] editor TeReSe (2003): *Term Rewriting Systems*. Cambridge University Press.

[29] Hans Zantema & Matthias Raffelsieper (2009): *Stream Productivity by Outermost Termination*. In Maribel Fernández, editor: *Proceedings Ninth International Workshop on Reduction Strategies in Rewriting and Programming, WRS 2009, Brasilia, Brazil, 28th June 2009.*, *EPTCS* 15, pp. 83–95, doi:10.4204/EPTCS.15.7.

[30] Hans Zantema & Matthias Raffelsieper (2010): *Proving Productivity in Infinite Data Structures*. In Christopher Lynch, editor: *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010, July 11-13, 2010, Edinburgh, Scottland, UK*, *LIPIcs* 6, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 401–416, doi:10.4230/LIPIcs.RTA.2010.401.