Productivity of Non-Orthogonal Term Rewrite Systems

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

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

1 Introduction

Productivity is the property that a given set of computation rules computes a desired infinite object. This has been studied mostly in the setting of *streams*, the simplest infinite objects. However, as already observed in [14], productivity is also of interest for other infinite structures, for example infinite trees, or mixtures of finite and infinite structures. A prominent example of the latter are lists in the programming language Haskell [10], which can be finite (by ending with a sentinel "[]") or which can go on forever.

Existing approaches for automatically checking productivity, e.g., [2, 3, 14], are restricted to *orthogonal* systems. The main reason for this restriction is that it disallows non-determinism. A complete computer program (i.e., a program and all possible input sequences, neglecting sources of true randomness) always behaves deterministically, as the steps of computation are precisely determined. However, often a complete program is not available, too large to be studied, or its inputs are provided by the user or they are not specified completely. In this case, non-determinism can be used to abstract from certain parts by describing a number of possible behaviors. In such a setting, the restriction to orthogonal systems, which is even far stronger than only disallowing non-determinism, should be removed. An example of such a setting are hardware components, describing streams of output values which are depending on the streams of input values. To analyze such components in isolation, all possible input streams have to be considered.

This paper presents an extension of the techniques in [14] to analyze productivity of specifications that may contain non-determinism. As in that work, the main technique to prove productivity is by analyzing termination of a corresponding context-sensitive term rewrite system [7]. Here however, overlapping rules are allowed and the data TRS is only required to be terminating, but it need not be confluent nor left-linear. This technique can be used to prove stabilization of hardware circuits, which have external inputs whose exact sequence of values is unknown. Thus, stabilization should be proven for all possible input sequences, which are therefore abstracted to be random Boolean streams, i.e., arbitrary streams containing the data values 0 and 1.

Structure of the Paper. In Section 2 we introduce *proper specifications*, which are the forms of rewrite systems studied in this paper. After that, in Section 3, the different notions of productivity are discussed.

S. Escobar (Ed.): Workshop on Reduction Strategies
in Rewriting and Programming (WRS2011)
EPTCS 82, 2012, pp. 53–67, doi:10.4204/EPTCS.82.4

For non-orthogonal specifications as studied in this paper, there exist both *weak* and *strong* productivity. We will motivate that strong productivity is the notion that we are interested in, as it does guarantee a constructor term to be reached by any outermost-fair reduction. The theoretical basis is laid in Section 4, proving our desired result that termination of a corresponding context-sensitive TRS implies strong productivity of a proper specification. Section 5 then applies this theory to an example hardware circuit, checking that for a given circuit the output values always stabilize, regardless of the sequence of input values. Finally, Section 6 concludes the paper.

2 Specifications

A *specification* gives the symbols and rules that shall be used to compute an intended infinite object. This section gives a brief introduction to term rewriting, mainly aimed at fixing notation. For an in-depth description of term rewriting, see for example [1, 11]. All symbols are assumed to have one of two possible *sorts*. The first sort d is for *data*. Terms of this sort represent the elements in an infinite structure, but which are not infinite terms by themselves. An example for data are the Booleans false and true (which are also written $\mathsf{0}$ and $\mathsf{1}$), or the natural numbers represented in Peano form by the two constructors $\mathsf{0}$ and succ. The set of all terms of sort d is denoted $\mathcal{T}_d(\Sigma_d, \mathcal{V}_d)$, where Σ_d is a set of function symbols all having types of the form $d^m \to d$ and where $\mathcal{V}_d$ is a set of *variables* all having sort d. The second sort is the sort s for *structure*. Terms of this sort are to represent the intended structure containing the data and therefore are allowed to be infinite. The set of all well-typed structure terms is denoted $\mathcal{T}_s(\Sigma_d \cup \Sigma_s, \mathcal{V})$, where Σ_s is disjoint from Σ_d and contains function symbols having types of the form $d^m \times s^n \to s$ and where $\mathcal{V} = \mathcal{V}_d \cup \mathcal{V}_s$ for a set $\mathcal{V}_s$ of variables all having sort s, which is disjoint from $\mathcal{V}_d$. We define the set of all well-typed terms as $\mathcal{T}(\Sigma_d \cup \Sigma_s, \mathcal{V}) = \mathcal{T}_d(\Sigma_d, \mathcal{V}_d) \cup \mathcal{T}_s(\Sigma_d \cup \Sigma_s, \mathcal{V})$ and denote the set of all *ground terms*, i.e., terms not containing any variables, by $\mathcal{T}(\Sigma_d \cup \Sigma_s) = \mathcal{T}(\Sigma_d \cup \Sigma_s, \emptyset)$. A term $t \in \mathcal{T}(\Sigma_d \cup \Sigma_s, \mathcal{V})$ of sort $\varsigma \in \{d, s\}$ is either a variable, i.e., $t \in \mathcal{V}_\varsigma$, or $t = f(u_1, \ldots, u_m, t_1, \ldots, t_n)$ with $f \in \Sigma_\varsigma$ of type $d^m \times s^n \to \varsigma$ (where $n = 0$ if $\varsigma = d$), $u_1, \ldots, u_m \in \mathcal{T}_d(\Sigma_d, \mathcal{V}_d)$, and $t_1, \ldots, t_n \in \mathcal{T}_s(\Sigma_d \cup \Sigma_s, \mathcal{V})$. In the latter case, i.e., when $t = f(u_1, \ldots, u_m, t_1, \ldots, t_n)$, we define the *root* of the term t as $\mathrm{root}(t) = f$.

A *Term Rewrite System (TRS)* over a signature Σ is a collection of rules $(\ell, r) \in \mathcal{T}(\Sigma, \mathcal{V})^2$ such that $\ell \notin \mathcal{V}$ and every variable contained in r is also contained in ℓ. As usual, we write $\ell \to r$ instead of (ℓ, r). A term $t \in \mathcal{T}(\Sigma, \mathcal{V})$ *rewrites* to a term $t' \in \mathcal{T}(\Sigma, \mathcal{V})$ with the rule $\ell \to r \in \mathcal{R}$, denoted $t \to_{\ell \to r, p} t'$ at *position* $p \in \mathrm{Pos}(t)$, if a substitution σ exists such that $t|_p = \ell\sigma$ and $t' = t[r\sigma]_p$. A position is as usual a sequence of natural numbers that identifies a number of argument positions taken to reach a certain subterm. The notation $t[r\sigma]_p$ represents the term t in which the subterm at position p, that is denoted by $t|_p$, has been replaced by the term $r\sigma$. This is the term r in which all variables have been replaced according to the substitution σ, which is a map from variables to terms. It is allowed to only indicate the term rewrite system $\mathcal{R}$ instead of the specific rule $\ell \to r$ or to leave out the subscripts in case they are irrelevant or clear from the context. The set of all *normal forms* of a TRS $\mathcal{R}$ over a signature Σ is denoted $\mathrm{NF}(\mathcal{R})$ and is defined as $\mathrm{NF}(\mathcal{R}) = \{t \in \mathcal{T}(\Sigma, \mathcal{V}) \mid \forall t' \in \mathcal{T}(\Sigma, \mathcal{V}) : t \not\to_{\mathcal{R}} t'\}$. The set of *ground normal forms* $\mathrm{NF}_{\mathrm{gnd}}(\mathcal{R})$ additionally requires that all contained terms are ground terms, i.e., $\mathrm{NF}_{\mathrm{gnd}}(\mathcal{R}) = \mathrm{NF}(\mathcal{R}) \cap \mathcal{T}(\Sigma)$.

We still have to impose some restrictions on specifications to make our approach work. These restrictions are given below in the definition of *proper* specifications, which are similar to those of [14].

Definition 1. A *proper specification* is a tuple $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$, where Σ_d is the signature of data symbols, each of type $d^m \to d$ (then the data arity of such a symbol g is defined to be $\mathsf{ar}_d(g) = m$), Σ_s is the signature of structure symbols f, which have types of the shape $d^m \times s^n \to s$ (and data arity $\mathsf{ar}_d(f) = m$, structure arity $\mathsf{ar}_s(f) = n$), $\mathcal{C} \subseteq \Sigma_s$ is a set of *constructors*, $\mathcal{R}_d$ is a terminating TRS over the signature Σ_d, and $\mathcal{R}_s$ is a TRS over the signature $\Sigma_d \cup \Sigma_s$, containing rules $f(u_1,\dots,u_m,t_1,\dots,t_n) \to t$ that satisfy the following properties:

- $f \in \Sigma_s \setminus \mathcal{C}$ with $\mathsf{ar}_d(f) = m$, $\mathsf{ar}_s(f) = n$,
- $f(u_1,\dots,u_m,t_1,\dots,t_n)$ is a well-sorted linear term,
- t is a well-sorted term of sort s, and
- for all $1 \le i \le n$ and for all $p \in \mathrm{Pos}(t_i)$ such that $t_i|_p$ is not a variable and $\mathrm{root}(t_i|_p) \in \Sigma_s$, it holds that $\mathrm{root}(t_i|_{p'}) \notin \mathcal{C}$ for all $p' < p$ (i.e., no structure symbol is below a constructor).

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

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

The above definition coincides with the definition of proper specifications given in [14] for orthogonal proper specifications.[1] We will illustrate the restrictions in the above definition later in Section 4. In the following, all examples except for Example 18 will be using the domain of Boolean streams, where $\mathcal{C} = \{:\}$ and $\Sigma_d \supseteq \{0,1\}$ with $\mathsf{ar}_d(0) = \mathsf{ar}_d(1) = 0$ and $\mathsf{ar}_d(:) = \mathsf{ar}_s(:) = 1$. In these examples, only a data TRS $\mathcal{R}_d$ and a structure TRS $\mathcal{R}_s$ are given from which the remaining symbols in Σ_d and Σ_s and their arities can be derived. If the data TRS $\mathcal{R}_d$ is not provided it is assumed to be empty.

3 Productivity

For orthogonal proper specifications, productivity is the property that every ground term t of sort s can, in the limit, be rewritten to a possibly infinite term consisting only of constructors. This is equivalent to stating that for every prefix depth $k \in \mathbb{N}$, the term t can be rewritten to another term t' having only constructor symbols on positions of depth k or less.

Definition 2. An orthogonal proper specification $\mathcal{S} = (\Sigma_d,\Sigma_s,\mathcal{C},\mathcal{R}_d,\mathcal{R}_s)$ is *productive*, iff for every ground term t of sort s and every $k \in \mathbb{N}$, there is a reduction $t \to^*_{\mathcal{R}_d \cup \mathcal{R}_s} t'$ such that every symbol of sort s in t' on depth less or equal to k is a constructor.

Productivity of an orthogonal proper specification is equivalent to the following property, as was shown in [14].

Proposition 3. *An orthogonal proper specification $\mathcal{S} = (\Sigma_d,\Sigma_s,\mathcal{C},\mathcal{R}_d,\mathcal{R}_s)$ is* productive*, iff for every ground term t of sort s there is a reduction $t \to^*_{\mathcal{R}_d \cup \mathcal{R}_s} t'$ such that* $\mathrm{root}(t') \in \mathcal{C}$.

It was already observed in [4, 2] that productivity of orthogonal specifications is equivalent to the existence of an *outermost-fair* reduction computing a constructor prefix for any given depth. Below, we give a general definition of outermost-fair reductions, as they will also be used in the non-orthogonal setting.

[1] To see this, one should observe that a defined symbol cannot occur on a non-root position of a left-hand side. This holds since otherwise the innermost such symbol would have variables and constructors as structure arguments and data arguments that do not unify with any of the data rules (due to orthogonality), which therefore are normal forms and can be instantiated to ground normal forms. Thus, exhaustiveness would require a left-hand side to match this term when instantiating all structure variables with some terms having a constructor root, which would give a contradiction to non-overlappingness.

Definition 4.

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

For non-orthogonal proper specifications, requiring just the existence of a reduction to a normal form (or to a constructor prefix of arbitrary depth) does not guarantee the computation to reach it, due to the possible non-deterministic choices. This can be observed for the term maybe in the following example.

Example 5. Consider a proper specification with the TRS $\mathcal{R}_s$ consisting of the following rules:

$$\begin{array}{lclclcl} \mathsf{maybe} & \to & 0 : \mathsf{maybe} & \quad & \mathsf{random} & \to & 0 : \mathsf{random} \\ \mathsf{maybe} & \to & \mathsf{maybe} & & \mathsf{random} & \to & 1 : \mathsf{random} \end{array}$$

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

Requiring just the existence of a constructor normal form is called *weak productivity* in [4, 2]. We already stated above that this is not the notion of productivity we are interested in. The one we are interested in is *strong productivity*, which is also defined in [4, 2], since it requires all reductions that make progress on outermost positions to reach constructor normal forms.

Definition 6. A proper specification $\mathcal{S}$ is called *strongly productive* iff for every ground term t of sort s all maximal outermost-fair rewrite sequences starting in t end in (i.e., have as limit for infinite sequences) a constructor normal form.

It was observed in [4, 2] that weak and strong productivity coincide for orthogonal (proper) specifications. However, for non-orthogonal (proper) specifications this is not the case anymore. The rules for maybe in Example 5 are not strongly productive, since they allow the infinite outermost-fair reduction $\mathsf{maybe} \to \mathsf{maybe} \to \dots$. However, these rules are weakly productive, since any ground term can be rewritten to an infinite stream containing only 0 elements after some finite prefix. For example, the ground term $1 : \mathsf{maybe}$ can be rewritten to the infinite stream $1 : 0 : 0 : \dots$.

An example of a non-orthogonal proper specification that is both strongly and weakly productive are the rules for random in Example 5, which always produce an infinite stream. In this case, the restriction to outermost-fair reductions is not needed. However, if we add the rule $\mathsf{id}(xs) \to xs$ and replace the rule $\mathsf{random} \to 1 : \mathsf{random}$ by the rule $\mathsf{random} \to \mathsf{id}(1 : \mathsf{random})$, then the infinite reduction $\mathsf{random} \to \mathsf{id}(1 : \mathsf{random}) \to \mathsf{id}(1 : \mathsf{id}(1 : \mathsf{random})) \to \dots$ exists. This reduction is not outermost-fair since the outermost redex $\mathsf{id}(\dots)$ survives infinitely often. When restricting to outermost-fair reductions, then indeed an infinite stream of Boolean values is obtained for every such reduction, so this is a strongly productive proper specification, too. Note that strong productivity implies weak productivity, so the example is also weakly productive.

4 Criteria for Strong Productivity

For orthogonal proper specifications, it is sufficient to just consider reductions that create a constructor at the top, as stated in Proposition 3. We will show next that this is also the case for non-orthogonal proper specifications. However, in contrast to [14], here we have to consider all maximal outermost-fair reductions, instead of just requiring the existence of such a reduction.

Proposition 7. *A proper specification* $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ *is* strongly productive *iff for every maximal outermost-fair reduction* $t_0 \to_{\mathcal{R}_d \cup \mathcal{R}_s} t_1 \to_{\mathcal{R}_d \cup \mathcal{R}_s} \dots$ *with* t_0 *being of sort* s *there exists* $k \in \mathbb{N}$ *such that* $\mathrm{root}(t_k) \in \mathcal{C}$.

Proof. The "only if"-direction is trivial. For the "if"-direction, we show inductively that for every depth $z \in \mathbb{N}$ and every maximal outermost-fair reduction $\rho \equiv t_0 \to_{p_0} t_1 \to_{p_1} \dots$ there exists an index $j \in \mathbb{N}$ such that for all positions $p \in \mathrm{Pos}(t_j)$ of sort s with $|p| < z$, $\mathrm{root}(t_j|_p) \in \mathcal{C}$.

For $z = 0$, the index j can be set to 0, thus here the claim trivially holds. Otherwise, we get that an index $k \in \mathbb{N}$ exists such that $\mathrm{root}(t_k) \in \mathcal{C}$. Let $t_k = c(u'_1, \dots, u'_m, t'_1, \dots, t'_n)$ with $c \in \mathcal{C}$. Because c is a constructor, we know that $p_l > \varepsilon$ for all $l \geq k$. Define $P_r = \{p'_i \mid p_i = (m+r).p'_i\}$ for $1 \leq r \leq n$ (i.e., the positions in the maximal outermost-fair reduction that are occurring in structure argument r). Then, for $1 \leq r \leq n$ and $P_r = \{p^r_0, p^r_1, \dots\}$ the reduction $t'_r = t_{r,0} \to_{p^r_0} t_{r,1} \to_{p^r_1} \dots$ is also a maximal outermost-fair reduction, otherwise an infinitely long surviving outermost redex would also be an infinitely long surviving outermost redex of the reduction ρ. By the induction hypothesis for $z-1$ we get that indices j_r for $1 \leq r \leq n$ exist such that $\mathrm{root}(t_{r,j_r}|_p) \in \mathcal{C}$ for all positions $p \in \mathrm{Pos}(t_{r,j_r})$ with $|p| < z-1$. Since all these reductions were taken from the original reduction, we define $j = k + \#\text{d-red} + \sum_{i=1}^{n} j_i$, where #d-red denotes the number of reductions performed in the data arguments of the constructor c such that $p_j = p^r_{j_r}$ for the last r. This shows that the initial reduction ρ has the form $t_0 \to^* t_k = c(u'_1, \dots, u'_m, t'_1, \dots, t'_n) \to^* c(u''_1, \dots, u''_m, t''_1, \dots, t''_n) = t_{j+1}$, where $t_{r,j_r} \to^* t''_r$ for every $1 \leq r \leq n$. Since there are only constructors in t_{r,j_r} for depths $0, \dots, z-2$, these constructors are still present in t''_r. This proves the proposition, since $c \in \mathcal{C}$ and thus for all positions $p \in \mathrm{Pos}(t_j)$ of sort s with $|p| < z$ we have $\mathrm{root}(t_j|_p) \in \mathcal{C}$. □

This characterization of strong productivity will be used in the remainder of the paper. Note that it is similar to the requirements for infinitary strong normalization SN^∞ observed in [12], where it is found that for left-linear and finite term rewrite systems, SN^∞ holds if and only if every infinite reduction only contains a finite number of root steps. Thus, it could seem possible to define strong productivity of proper specifications by requiring that every reduction starting in a finite ground term is infinitary strongly normalizing, i.e., SN^∞ holds for the relation $\to_{\mathcal{R}_d \cup \mathcal{R}_s} \cap\, \mathcal{T}(\Sigma_d \cup \Sigma_s)^2$. However, this is not the case, as the following example shows.

Example 8. Consider the proper specification containing the following TRS $\mathcal{R}_s$:

$$\mathsf{a} \;\to\; \mathsf{f}(\mathsf{a}) \qquad\qquad \mathsf{f}(x : xs) \;\to\; x : xs$$

This TRS has the property SN^∞, intuitively because either the symbol f remains at the root position and can never be rewritten again (in case the first rule is applied), or the constructor : is created at the root. Formally, this can for example be proven by the technique presented in [12]: Let $\Sigma_\# = \Sigma \uplus \{g_\# \mid g \in \Sigma\}$, where $\Sigma = \{0, 1, :, \mathsf{a}, \mathsf{f}\}$ is the signature of the specification. Then we choose the finite weakly monotone $\Sigma_\#$ algebra $(\{0,1,2\}, [\cdot], \bot, >, \geq)$, where $\bot = 0$, $[0] = 0$, $[1] = 0$, $[\mathsf{a}] = 1$, $[\mathsf{f}](n) = n$, $[:](m,n) = \min\{m+n, 2\}$, $[a_\#] = 2$, $[f_\#](n) = 1$, and $[:_\#](m,n) = 0$ for $m,n \in \{0,1,2\}$ and $>$ and $\geq$ are the natural comparison operators on the numbers $\{0,1,2\}$. It is easy to check that this algebra is indeed weakly monotone (i.e., that $>$ is well-founded, $> \cdot \geq \subseteq > \subseteq \geq$, and for every $g \in \Sigma_\#$, the operation $[g]$ is monotone with respect to $\geq$). Additionally, the requirements of the combination of [12, Theorem 5 and Theorem 6] are satisfied, i.e., $\{0,1,2\}$ is finite, $\geq$ is transitive, $a \geq b$ implies $a > b$ or $a = b$, $a \geq \bot = 0$ for all $a,b \in \{0,1,2\}$, and $[\ell\sigma] \geq [r\sigma]$ and $[\ell_\#\sigma] > [r_\#\sigma]$ for all $\ell \to r \in \mathcal{R}_s$ and all substitutions σ, where $g(t_1,\ldots,t_k)_\# = g_\#(t_1,\ldots,t_k)$. This proves SN^∞ of $\to_{\mathcal{R}_s}$, which especially entails SN^∞ of the relation $\to_{\mathcal{R}_s} \cap \mathcal{T}(\Sigma_d \cup \Sigma_s)^2$.

However, the above proper specification is not strongly productive, since the infinite outermost-fair reduction $\mathsf{a} \to_{\mathcal{R}_s} \mathsf{f}(\mathsf{a}) \to_{\mathcal{R}_s} \mathsf{f}(\mathsf{f}(\mathsf{a})) \to_{\mathcal{R}_s} \ldots$, continued by repeatedly reducing the symbol a, never produces any constructors.

The above example shows that even though we require exhaustiveness of proper specifications, this exhaustiveness only refers to constructor terms, i.e., the objects we are interested in, and not to arbitrary terms. A similar observation, namely that top termination is not equivalent to productivity, was already made in [15].

A first technique to prove strong productivity of proper specifications is given next. It is a simple syntactic check that determines whether every right-hand side of sort s starts with a constructor. For orthogonal proper specifications, this was already observed in [14]. It has to be proven again since here we consider strong productivity, which requires all possible outermost-fair reductions to reach a constructor normal form, instead of weak productivity as in [14], for which only a single reduction to a constructor normal form needs to be constructed.

Theorem 9. *Let $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ be a proper specification. If for all rules $\ell \to r \in \mathcal{R}_s$ we have $\mathrm{root}(r) \in \mathcal{C}$, then $\mathcal{S}$ is strongly productive.*

Proof. Let $\rho \equiv t_0 \to_{p_0} t_1 \to_{p_1} \ldots$ be a maximal outermost-fair reduction and let $t_0 = f(u'_1,\ldots,u'_m,t'_1,\ldots,t'_n)$. If $f \in \mathcal{C}$ we are done, so we assume $f \in \Sigma_s \setminus \mathcal{C}$ and perform structural induction on t_0 to prove that $\mathrm{root}(t_k) \in \mathcal{C}$ for some $k \in \mathbb{N}$.

From the induction hypothesis we get that for every $1 \leq i \leq n$ and every maximal outermost-fair reduction $t'_i = t_{i,0} \to t_{i,1} \to \ldots$ there exists an index $k_i \in \mathbb{N}$ such that $\mathrm{root}(t_{i,k_i}) \in \mathcal{C}$.

Assume that for all $j \in \mathbb{N}$, $p_j \neq \varepsilon$. As in the proof of Proposition 7, we therefore again obtain maximal outermost-fair reductions $t'_i \to \ldots$, thus we get indices $k_i \in \mathbb{N}$ such that $\mathrm{root}(t_{i,k_i}) \in \mathcal{C}$, as explained above. This makes our reduction ρ have the shape $t_0 = f(u'_1,\ldots,u'_m,t'_1,\ldots,t'_n) \to^* f(u''_1,\ldots,u''_m,t''_1,\ldots,t''_n) = t_j$ for some $j \in \mathbb{N}$, where $u''_1,\ldots,u''_m \in \mathrm{NF}_{\mathrm{gnd}}(\mathcal{R}_d)$ (since the reduction ρ is maximal outermost-fair and $\mathcal{R}_d$ is terminating) and $t_{i,k_i} \to^* t''_i$, thus also $\mathrm{root}(t''_i) \in \mathcal{C}$. Because $\mathcal{R}_s$ is exhaustive, we get that t_j contains a redex at the root position ε, which of course is outermost. This gives rise to a contradiction to ρ being outermost fair, as this outermost redex survives infinitely often, because $p_j \neq \varepsilon$ for all $j \in \mathbb{N}$. Therefore, $p_j = \varepsilon$ for some $j \in \mathbb{N}$ and the reduction has the shape $t_0 \to^* t_j \to_\varepsilon r\sigma$, where the last step is with respect to some rule $\ell \to r \in \mathcal{R}_s$. By the assumption on the shape of the rules in $\mathcal{R}_s$, we have $\mathrm{root}(r) \in \mathcal{C}$, hence also $\mathrm{root}(r\sigma) \in \mathcal{C}$, which proves productivity according to Proposition 7. □

This technique is sufficient to prove strong productivity of the proper specification consisting of the two rules for random in Example 5, since both have right-hand sides with the constructor : at the root. However, it is easy to create examples which are strongly productive, but do not satisfy the syntactic requirements of Theorem 9.

Example 10. Consider the proper specification with the following TRS $\mathcal{R}_s$:

$$\begin{array}{rclrcl} \mathsf{ones} & \to & \mathsf{1} : \mathsf{ones} & \mathsf{finZeroes} & \to & \mathsf{0} : \mathsf{ones} \\ \mathsf{finZeroes} & \to & \mathsf{0} : \mathsf{0} : \mathsf{ones} & \mathsf{finZeroes} & \to & \mathsf{0} : \mathsf{0} : \mathsf{0} : \mathsf{ones} \\ \mathsf{f}(\mathsf{0} : xs) & \to & \mathsf{f}(xs) & \mathsf{f}(\mathsf{1} : xs) & \to & \mathsf{1} : \mathsf{f}(xs) \end{array}$$

The constant $\mathsf{finZeroes}$ produces non-deterministically a stream that starts with one, two, or three zeroes followed by an infinite stream of ones. Function f takes a binary stream as argument and filters out all occurrences of zeroes. Thus, productivity of this example proves that only a finite number of zeroes can be produced. This however cannot be proven with the technique of Theorem 9, since the right-hand side of the rule $\mathsf{f}(\mathsf{0} : xs) \to \mathsf{f}(xs)$ does not start with the constructor :.

Another technique presented in [14] to show productivity of orthogonal proper specifications is based on context-sensitive termination [7]. The idea is to disallow rewriting in structure arguments of constructors, thus context-sensitive termination implies that for every ground term of sort s, a term starting with a constructor can be reached (due to the exhaustiveness requirement). As was observed by Endrullis and Hendriks recently in [5], this set of blocked positions can be enlarged, making the approach even stronger.

Below, the technique for proving productivity by showing termination of a corresponding context-sensitive TRS is extended to also be applicable in the case of our more general proper specifications. This version already includes an adaption of the improvement mentioned above.

Definition 11. Let $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ be a proper specification. The replacement map $\mu_{\mathcal{S}} : \Sigma_d \cup \Sigma_s \to 2^{\mathbb{N}}$ is defined as follows: [2]

- $\mu_{\mathcal{S}}(f) = \{1, \dots, \mathsf{ar}_d(f)\}$, if $f \in \Sigma_d \cup \mathcal{C}$
- $\mu_{\mathcal{S}}(f) = \{1, \dots, \mathsf{ar}_d(f) + \mathsf{ar}_s(f)\} \setminus \{1 \le i \le \mathsf{ar}_d(f) + \mathsf{ar}_s(f) \mid t|_i$ is a variable for all $\ell \to r \in \mathcal{R}_s$ and all non-variable subterms t of ℓ with $\mathrm{root}(t) = f\}$,[3] otherwise

In the remainder, we leave out the subscript $\mathcal{S}$ if the specification is clear from the context. The replacement map μ is used to define the set of *allowed* positions of a non-variable term t as $\mathrm{Pos}_\mu(t) = \{\varepsilon\} \cup \{i.p \mid i \in \mu(\mathrm{root}(t)), p \in \mathrm{Pos}_\mu(t|_i)\}$ and the set of *blocked* positions of t as $\mathrm{blocked}_\mu(t) = \mathrm{Pos}(t) \setminus \mathrm{Pos}_\mu(t)$. Context-sensitive rewriting [7] then is the restriction of the rewrite relation to those redexes on positions from Pos_μ. Formally, we have $t \xrightarrow{\mu}_{\ell \to r,p} t'$ iff $t \to_{\ell \to r,p} t'$ and $p \in \mathrm{Pos}_\mu(t)$ and we say a TRS $\mathcal{R}$ is *μ-terminating* iff no infinite $\xrightarrow{\mu}_{\mathcal{R}}$-chain exists.

The replacement map $\mu_{\mathcal{S}}$ is *canonical* [8] for the left-linear TRS $\mathcal{R}_s$, guaranteeing through the second condition of the above Definition 11 that non-variable positions of left-hand sides are allowed. In that definition, the replacement map $\mu_{\mathcal{S}}$ is extended to the possibly non-left-linear TRS $\mathcal{R}_d \cup \mathcal{R}_s$ by allowing all arguments of symbols from Σ_d.

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

Theorem 12. *A proper specification $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ is strongly productive, if $\mathcal{R}_d \cup \mathcal{R}_s$ is $\mu_{\mathcal{S}}$-terminating.*

Before proving the above theorem, we will show first that it subsumes Theorem 9. Intuitively, this holds because structure arguments of constructors are blocked, and if every right-hand side of $\mathcal{R}_s$ starts with a constructor then the number of allowed redexes of sort s in a term steadily decreases.

---

[2] Note that in [5], Endrullis and Hendriks consider orthogonal TRSs and also block arguments of symbols in Σ_d which only contain variables. This however is problematic when allowing data rules that are not left-linear. Example:

$$\begin{array}{llllllllll} \mathcal{R}_s: & \mathsf{f}(1) & \to & \mathsf{f}(\mathsf{d}(0,\mathsf{d}(1,0))) & & \mathsf{f}(0) & \to & 0 : \mathsf{f}(0) & & \\ \mathcal{R}_d: & \mathsf{d}(x,x) & \to & 1 & & \mathsf{d}(0,x) & \to & 0 & \mathsf{d}(1,x) & \to \; 0 \end{array}$$

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

[3] The requirement of t not being a variable ensures that $\mathrm{root}(t)$ is defined.

Proposition 13. *Let $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ be a proper specification. If for all rules $\ell \to r \in \mathcal{R}_s$ we have* $\text{root}(r) \in \mathcal{C}$*, then $\mathcal{R}_d \cup \mathcal{R}_s$ is $\mu_{\mathcal{S}}$-terminating.*

Proof. Let $t \in \mathcal{T}(\Sigma_d \cup \Sigma_s, \mathcal{V})$ be well-typed. If t has sort d, then all subterms must also be of sort d, as symbols from Σ_d only have arguments of that sort. Hence, rewriting can only be done with rules from $\mathcal{R}_d$, which is assumed to be terminating.

Otherwise, let t be of sort s and assume that t starts an infinite μ-reduction $t = t_0 \xrightarrow{\mu}_{\ell_0 \to r_0, p_0} t_1 \xrightarrow{\mu}_{\ell_1 \to r_1, p_1} t_2 \xrightarrow{\mu}_{\ell_2 \to r_2, p_2} \ldots$. We define $\text{Pos}_\mu^{\text{red}_s}(t') = \{p \in \text{Pos}_\mu(t') \mid t'|_p \text{ is a redex of sort } s\}$ for any term $t' \in \mathcal{T}(\Sigma_d \cup \Sigma_s, \mathcal{V})$. It will be proven that in every step $t_i \xrightarrow{\mu}_{\ell_i \to r_i, p_i} t_{i+1}$ of the infinite reduction, $|\text{Pos}_\mu^{\text{red}_s}(t_{i+1})| \le |\text{Pos}_\mu^{\text{red}_s}(t_i)|$ and that for steps with $\ell_i \to r_i \in \mathcal{R}_s$, we even have $|\text{Pos}_\mu^{\text{red}_s}(t_{i+1})| < |\text{Pos}_\mu^{\text{red}_s}(t_i)|$. To this end, case analysis of the rule $\ell_i \to r_i$ is performed. If $\ell_i \to r_i \in \mathcal{R}_d$, then $t_i = t_i[\ell_i\sigma_i]_{p_i}$ and $t_{i+1} = t_i[r_i\sigma_i]_{p_i}$ for some substitution σ_i. Because $\ell_i, r_i \in \mathcal{T}(\Sigma_d, \mathcal{V})$, $|\text{Pos}_\mu^{\text{red}_s}(\ell_i\sigma_i)| = |\text{Pos}_\mu^{\text{red}_s}(r_i\sigma_i)| = 0$ since all symbols in Σ_d have arguments of sort d. Thus, $\text{Pos}_\mu^{\text{red}_s}(t_{i+1}) = \text{Pos}_\mu^{\text{red}_s}(t_i)$. In the second case, $\ell_i \to r_i \in \mathcal{R}_s$. Let $t_i = t_i[\ell_i\sigma_i]_{p_i}$ and $t_{i+1} = t_i[r_i\sigma_i]_{p_i}$ for some substitution σ_i. Then, $\text{Pos}_\mu^{\text{red}_s}(t_i) = \text{Pos}_\mu^{\text{red}_s}(t_i[z]_{p_i}) \uplus \{p_i.p \mid p \in \text{Pos}_\mu^{\text{red}_s}(t_i|_{p_i})\}$ for any variable $z \in \mathcal{V}$ of sort s. For t_{i+1} we observe that $\text{Pos}_\mu^{\text{red}_s}(t_{i+1}) = \text{Pos}_\mu^{\text{red}_s}(t_i[r_i\sigma_i]_{p_i}) = \text{Pos}_\mu^{\text{red}_s}(t_i[z]_{p_i}) \uplus \{p_i.p \mid p \in \text{Pos}_\mu^{\text{red}_s}(t_i[r_i\sigma_i]_{p_i}|_{p_i})\}$ for any variable $z \in \mathcal{V}$ of sort s. Here, it holds that $\text{Pos}_\mu^{\text{red}_s}(t_i|_{p_i}) = \text{Pos}_\mu^{\text{red}_s}(\ell_i\sigma_i) \ni \varepsilon$, therefore $p_i \in \text{Pos}_\mu^{\text{red}_s}(t_i)$. Furthermore, $\text{Pos}_\mu^{\text{red}_s}(t_i[r_i\sigma_i]_{p_i}|_{p_i}) = \text{Pos}_\mu^{\text{red}_s}(r_i\sigma_i) = \emptyset$, since $\text{root}(r_i) \in \mathcal{C}$ by assumption, hence $\mu(\text{root}(r_i)) = \{1, \ldots, \mathsf{ar}_d(\text{root}(r_i))\}$ and because symbols from Σ_d only have arguments of sort d. Thus, $\text{Pos}_\mu^{\text{red}_s}(t_{i+1}) \subsetneq \text{Pos}_\mu^{\text{red}_s}(t_i)$.

Combining these observations, we therefore only have finitely many reductions with rules from $\mathcal{R}_s$ in the infinite reduction. Thus, an infinite tail of steps with rules from $\mathcal{R}_d$ exists. This however contradicts the assumption that $\mathcal{R}_d$ is terminating, hence no infinite μ-reduction can exist which proves μ-termination of $\mathcal{R}_d \cup \mathcal{R}_s$. □

Hence, we could restrict ourselves to analyzing context-sensitive termination only. However, the syntactic check of Theorem 9 can be done very fast and should therefore be the first method to try.

In order to prove Theorem 12 we will show that a maximal outermost-fair reduction that never reaches a constructor entails an infinite μ-reduction. For this purpose we need the following lemma, which shows that in every ground term not starting with a constructor there exists a redex that is not blocked by the replacement map μ.

Lemma 14. *Let $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ be a proper specification. For all ground terms t of sort s with* $\text{root}(t) \notin \mathcal{C}$ *there exists a position $p \in \text{Pos}_\mu(t)$ such that $t \to_p$.*

Proof. Let $t = f(u_1, \ldots, u_m, t_1, \ldots, t_n)$. We perform structural induction on t. If $u_i \to_{p'}$ for some $1 \le i \le m$ with $i \in \mu(f)$, then $t \to_{i.p'}$ and $i.p' \in \text{Pos}_\mu(t)$ since arguments of data symbols are never blocked. Thus, we assume in the remainder that $u_i \in \text{NF}_{\text{gnd}}(\mathcal{R}_d)$ for all $1 \le i \le m$ with $i \in \mu(f)$. If $\text{root}(t_i) \in \mathcal{C}$ for all $1 \le i \le n$, $i \in \mu(f)$, then $t \to_\varepsilon$ by the exhaustiveness requirement (and because all arguments u_j, t_j with $j \notin \mu(f)$ are being matched by pairwise different variables, due to left-linearity). Otherwise, there exists $1 \le i \le n$, $i \in \mu(f)$ such that $\text{root}(t_i) \notin \mathcal{C}$. By the induction hypothesis we get that $t_i \to_{p'}$ for some $p' \in \text{Pos}_\mu(t_i)$. Therefore, we also have $i.p' \in \text{Pos}_\mu(t)$ and $t \to_{i.p'}$. □

A second lemma that is required for the proof of Theorem 12 states that a specialized version of the Parallel Moves Lemma [1, Lemma 6.4.4] holds for our restricted format of term rewrite systems. It allows us to swap the order of reductions blocked by μ with reductions not blocked by μ. To formulate the lemma, we need the notion of a parallel reduction step $t \overset{\shortparallel}{\to}_P t'$, which is defined for a set $P = \{p_1, \ldots, p_n\} \subseteq \mathrm{Pos}(t)$ such that for every pair $1 \le i < j \le n$ we have $p_i \parallel p_j$ and a term $t = t[\ell_1\sigma_1]_{p_1} \ldots [\ell_n\sigma_n]_{p_n}$ as $t' = t[r_1\sigma_1]_{p_1} \ldots [r_n\sigma_n]_{p_n}$ for rules $\ell_i \to r_i \in \mathcal{R}_d \cup \mathcal{R}_s$ and substitutions σ_i, $1 \le i \le n$.

Lemma 15. *Let $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ be a proper specification. For all ground terms t, t', t'' and positions $p \in \mathrm{Pos}_\mu(t')$, $P \subseteq \mathrm{blocked}_\mu(t)$ with $t \overset{\shortparallel}{\to}_P t' \to_{\ell\to r,p} t''$, a term $\hat{t}$ and a set $P' \subseteq \mathrm{Pos}(\hat{t})$ exist such that $t \to_{\ell\to r,p} \hat{t} \overset{\shortparallel}{\to}_{P'} t''$.*

Proof. Let $P = \{p_1, \ldots, p_k\} \subseteq \mathrm{blocked}_\mu(t)$. Then $t = t[\ell_1\sigma_1]_{p_1} \ldots [\ell_k\sigma_k]_{p_k} \overset{\shortparallel}{\to}_P t[r_1\sigma_1]_{p_1} \ldots [r_k\sigma_k]_{p_k} = t' = t'[\ell\sigma]_p$ for some rules $\ell_1 \to r_1, \ldots, \ell_k \to r_k, \ell \to r \in \mathcal{R}_d \cup \mathcal{R}_s$ and substitutions $\sigma_1, \ldots, \sigma_k, \sigma$. W.l.o.g., let $0 \le j \le k$ be such that $p_i \not\parallel p$ for all $1 \le i \le j$ and $p_i \parallel p$ for all $j < i \le k$. Since $p \in \mathrm{Pos}_\mu(t')$ and $p_i \in \mathrm{blocked}_\mu(t')$, it must hold that $p < p_i$ for all $1 \le i \le j$. Therefore, the term t' must have the shape $t' = t\left[\ell\sigma[r_1\sigma_1]_{p_1-p} \ldots [r_j\sigma_j]_{p_j-p}\right]_p [r_{j+1}\sigma_{j+1}]_{p_{j+1}} \ldots [r_k\sigma_k]_{p_k}$.

If $\ell \to r \in \mathcal{R}_d$, then it must hold that $j = 0$, since arguments of data symbols are never blocked. Hence, the lemma trivially holds in this case, as all reductions are on independent positions.

Otherwise, $\ell \to r \in \mathcal{R}_s$. Because the positions p_i for $1 \le i \le j$ are blocked, it must be the case that they are either below a variable in all rules containing a certain symbol f (hence, they are also below a variable in ℓ), or they are below a structure argument of a constructor $c \in \mathcal{C}$. By requirement of specifications, if a constructor is present on a left-hand side of a rule, all its structure arguments must be variables. Thus, we conclude that all positions p_i, and thereby all terms $r_i\sigma_i$, are below some variable of ℓ in t'. Additionally, the left-hand side ℓ is required to be linear, therefore there exist pairwise different variables $x_1, \ldots, x_j$, contexts $C_1, \ldots, C_j$, and a substitution σ' being like σ except that $\sigma'(x_i) = x_i$ for $1 \le i \le j$ such that:

$$\begin{aligned} t' \ =\ & t\left[\ell\sigma'\{x_1{:=}C_1[r_1\sigma_1], \ldots, x_j{:=}C_j[r_j\sigma_j]\}\right]_p [r_{j+1}\sigma_{j+1}]_{p_{j+1}} \ldots [r_k\sigma_k]_{p_k} \\ \to_p\ & t\left[r\sigma'\{x_1{:=}C_1[r_1\sigma_1], \ldots, x_j{:=}C_j[r_j\sigma_j]\}\right]_p [r_{j+1}\sigma_{j+1}]_{p_{j+1}} \ldots [r_k\sigma_k]_{p_k} \ =\ t'' \end{aligned}$$

We conclude that $p \in \mathrm{Pos}_\mu(t)$, as all reduction steps in $t \overset{\shortparallel}{\to}_P t'$ are either below or independent of p. Thus:

$$\begin{aligned} t \ =\ & t\left[\ell\sigma'\{x_1{:=}C_1[\ell_1\sigma_1], \ldots, x_j{:=}C_j[\ell_j\sigma_j]\}\right]_p [\ell_{j+1}\sigma_{j+1}]_{p_{j+1}} \ldots [\ell_k\sigma_k]_{p_k} \\ \to_p\ & t\left[r\sigma'\{x_1{:=}C_1[\ell_1\sigma_1], \ldots, x_j{:=}C_j[\ell_j\sigma_j]\}\right]_p [\ell_{j+1}\sigma_{j+1}]_{p_{j+1}} \ldots [\ell_k\sigma_k]_{p_k} \ =\ \hat{t} \\ \overset{\shortparallel}{\to}_{P'}\ & t\left[r\sigma'\{x_1{:=}C_1[r_1\sigma_1], \ldots, x_j{:=}C_j[r_j\sigma_j]\}\right]_p [r_{j+1}\sigma_{j+1}]_{p_{j+1}} \ldots [r_k\sigma_k]_{p_k} \ =\ t'' \end{aligned}$$

In the second reduction step, the positions of the terms $\ell_i\sigma_i$ in $\hat{t}$ constitute the set $P' \subseteq \mathrm{Pos}(\hat{t})$. □

We are now able to prove our main theorem, showing that context-sensitive termination implies productivity of the considered proper specification.

Proof of Theorem 12. Assume $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ is not strongly productive. Then, a maximal outermost-fair reduction sequence $\rho \equiv t_0 \to t_1 \to \ldots$ exists where for all $k \in \mathbb{N}$, $\mathrm{root}(t_k) \notin \mathcal{C}$.

This reduction sequence is infinite, since otherwise it would end in a term t_m for some $m \in \mathbb{N}$ with $\mathrm{root}(t_m) \notin \mathcal{C}$. Then however, according to Lemma 14, the term t_m would contain a redex, giving a contradiction to the sequence being maximal.

The sequence might however perform reductions that are below a variable argument of a constructor or below a variable in all left-hand sides of a defined symbol. These reduction steps are not allowed when considering context-sensitive rewriting with respect to μ. Such reductions however can be reordered. First, we observe that there is always a redex which is not blocked, due to Lemma 14, thus there is also an outermost such one. Because the reduction is outermost-fair, and because reductions below a variable cannot change the matching of a rule, as shown in Lemma 15, such redexes must be contracted an infinite number of times in the infinite reduction sequence ρ. Thus, we can reorder the reduction steps in ρ: If there is a (parallel) reduction below a variable before performing a step that is allowed by μ, then we swap these two steps using Lemma 15. Repeating this, we get an infinite reduction sequence ρ' consisting of steps which are not blocked by μ. Thus, this is an infinite μ-reduction sequence, showing that $\mathcal{R}_d \cup \mathcal{R}_s$ is not μ-terminating, which proves the theorem. □

The technique of Theorem 12, i.e., proving μ-termination of the corresponding context-sensitive TRS, is able to prove strong productivity of Example 10. By Definition 11, the corresponding replacement map μ is defined as $\mu(\mathsf{0}) = \mu(\mathsf{1}) = \mu(\mathsf{ones}) = \mu(\mathsf{finZeroes}) = \emptyset$ and $\mu(\mathsf{f}) = \mu(:) = \{1\}$, i.e., rewriting is allowed on all positions except those that are inside a second argument of the constructor :. Context-sensitive termination of the TRS together with the above replacement map μ can for example be shown by the tool AProVE [6]. Thus, productivity of that example has been shown according to Theorem 12. Also, strong productivity of the proper specification consisting of the rules $\mathsf{random} \to \mathsf{0} : \mathsf{random}$, $\mathsf{random} \to \mathsf{id}(\mathsf{1} : \mathsf{random})$, and $\mathsf{id}(xs) \to xs$ can be proven using Theorem 12 and the tool AProVE [6], where $\mu(\mathsf{0}) = \mu(\mathsf{1}) = \mu(\mathsf{random}) = \mu(\mathsf{id}) = \emptyset$ and $\mu(:) = \{1\}$ according to Definition 11. Note that for this example, one could also have used $\mu(\mathsf{id}) = \{1\}$, i.e., here the removal of argument positions in the second item of Definition 11 is irrelevant.

This is not the case in the next example, showing that this improvement, which was inspired by [5] and blocks more argument positions, allows to prove productivity of specifications where this would otherwise not be possible.

Example 16. Consider the following proper specification, given by the TRS $\mathcal{R}_s$:

$$\mathsf{a} \to \mathsf{f}(\mathsf{1} : \mathsf{a}, \mathsf{a}) \qquad \begin{aligned} \mathsf{f}(x : xs, ys) &\to x : ys \\ \mathsf{f}(\mathsf{f}(xs, ys), zs) &\to \mathsf{f}(xs, \mathsf{f}(ys, zs)) \end{aligned}$$

When defining $\mu(\mathsf{1}) = \mu(\mathsf{a}) = \emptyset$ and $\mu(:) = \{1\}$ by the first case of Definition 11, and defining $\mu(\mathsf{f}) = \{1,2\}$ (i.e., not removing any argument positions, as was done in the orthogonal case in [14]), then an infinite μ-reduction exists: $\mathsf{a} \overset{\mu}{\to} \mathsf{f}(\mathsf{1} : \mathsf{a}, \mathsf{a}) \overset{\mu}{\to} \mathsf{f}(\mathsf{1} : \mathsf{a}, \mathsf{f}(\mathsf{1} : \mathsf{a}, \underline{\mathsf{a}})) \overset{\mu}{\to} \ldots$
This reduction can be continued in the above style by reducing the underlined redex further, which will always create the term a on an allowed position of the form 2^n. However, such positions are not required for any of the f-rules to be applicable; for both rules it holds that all subterms of left-hand sides that start with the symbol f, which are the terms $\mathsf{f}(x : xs, ys)$, $\mathsf{f}(\mathsf{f}(xs, ys), zs)$, and $\mathsf{f}(xs, ys)$, have a variable as second argument. Thus, according to Definition 11, the replacement map μ' can be defined to be like μ, except that $\mu'(\mathsf{f}) = \{1\}$. With this improved replacement map, μ'-termination of the above TRS can for example be proven by the tool AProVE [6], which implies productivity by Theorem 12.

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

Example 17. Consider the proper specification with the following rules in $\mathcal{R}_s$:

$$\mathsf{a} \to \mathsf{f}(\mathsf{a}) \qquad \mathsf{f}(x : xs) \to x : \mathsf{f}(xs) \qquad \mathsf{f}(\mathsf{f}(xs)) \to \mathsf{1} : xs$$

Starting in the term a, we observe that an infinite μ-reduction starting with $\mathsf{a} \to \mathsf{f}(\underline{\mathsf{a}})$ exists, which can be continued by reducing the underlined redex repeatedly, since $\mu(\mathsf{f}) = \{1\}$. Thus, the example is not μ-terminating. However, the specification is productive, as can be shown by case analysis based on the root symbol of some arbitrary ground term t. In case $\mathrm{root}(t) = :$, then nothing has to be done, according to Proposition 7. Otherwise, if $\mathrm{root}(t) = \mathsf{a}$, then any maximal outermost-fair reduction must start with

$t = \mathsf{a} \to \mathsf{f}(\mathsf{a})$, thus we can reduce our analysis to the final case, where $\mathrm{root}(t) = \mathsf{f}$. In this last case, $t = \mathsf{f}(t')$. Due to the rules for the symbol f, we have to perform a further case analysis based on the root symbol of t'. If $\mathrm{root}(t') = :$, i.e., $t' = u : t''$ for some terms u and t'', then this constructor cannot be reduced further. Also, $t = \mathsf{f}(u : t'')$ is a redex, due to the second rule. Hence, in any maximal outermost-fair reduction sequence this redex must eventually be reduced using the second rule, which results in a term with the constructor : at the root. For $\mathrm{root}(t') = \mathsf{a}$ we again must reduce $t = \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{f}(\mathsf{a}))$. Finally, in case $\mathrm{root}(t') = \mathsf{f}$, we have two possibilities. The first one occurs when the term t' is eventually reduced at the root. Since $\mathrm{root}(t') = \mathsf{f}$, this has to happen with either of the f-rules, creating a constructor : which, as we already observed, must eventually result in the term t also being reduced to a term with the constructor : at the root. Otherwise, in the second possible scenario, the term t' is never reduced at the root. Then however, an outermost redex of the shape $\mathsf{f}(\mathsf{f}(t''))$ exists in all terms that t can be rewritten to in this way, thus it has to be reduced eventually with the third rule. This again creates a term with constructor : at the root. Combining all these observations, we see that in every maximal outermost-fair reduction there exists a term with the constructor : as root symbol, which proves productivity due to Proposition 7.

In the remainder of this section we want to illustrate the requirements of proper specifications in Definition 1, namely that the TRS $\mathcal{R}_s$ should be left-linear and that structure arguments of constructors in left-hand sides must not be structure symbols, i.e., they must be variables. We begin with an example specification that is not left-linear and not productive, but μ-terminating.

Example 18. We consider the non-proper specification $\mathcal{S} = (\Sigma_d, \Sigma_s, \mathcal{C}, \mathcal{R}_d, \mathcal{R}_s)$ with $\Sigma_d = \mathcal{R}_d = \emptyset$, $\mathcal{C} = \{\mathsf{a}, \mathsf{c}\} \subseteq \Sigma_s = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{f}\}$, and the following rules in $\mathcal{R}_s$ which also imply the arities of the symbols:

$$\mathsf{b} \to \mathsf{a} \qquad \mathsf{f}(\mathsf{a}) \to \mathsf{a} \qquad \mathsf{f}(\mathsf{c}(x,x)) \to \mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b})) \qquad \mathsf{f}(\mathsf{c}(x,y)) \to \mathsf{c}(x,y)$$

The example specification is not productive, as it admits the infinite outermost-fair reduction sequence $\mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{a})) \to \mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b})) \to \mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{a})) \to \dots$. However, the TRS is μ-terminating, as shown by the tool AProVE [6], where $\mu(\mathsf{f}) = \{1\}$ and $\mu(\mathsf{a}) = \mu(\mathsf{b}) = \mu(\mathsf{c}) = \emptyset$. This is the case because rewriting below the constructor c is not allowed, thus the second step of the above reduction sequence is blocked. The reason why Theorem 12 fails is the reordering of reductions, since in this example a reduction of the form $t \overset{\shortparallel}{\to}_P t' \to_{\ell\to r,p} t''$ (here: $\mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b})) \overset{\shortparallel}{\to}_{\{1.1\}} \mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{a})) \to_{\mathsf{f}(\mathsf{c}(x,x))\to\mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b})),\varepsilon} \mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b}))$) does not imply that $t \to_{\ell\to r,p}$ (in the example, $\mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b})) \not\to_{\mathsf{f}(\mathsf{c}(x,x))\to\mathsf{f}(\mathsf{c}(\mathsf{a},\mathsf{b})),\varepsilon}$), i.e., Lemma 15 does not hold.

The next example illustrates why non-variable structure arguments of constructors are not allowed in left-hand sides.

Example 19. Let $\mathcal{R}_s$ contain the following rules:

$$\mathsf{ones} \to 1 : \mathsf{ones} \qquad \mathsf{f}(x : y : xs) \to \mathsf{f}(y : xs) \qquad \mathsf{f}(x : xs) \to x : xs$$

Here, we have non-productivity of the corresponding non-proper specification due to the infinite outermost-fair reduction sequence $\mathsf{f}(\mathsf{ones}) \to_1 \mathsf{f}(1 : \mathsf{ones}) \to_{1.2} \mathsf{f}(1 : 1 : \mathsf{ones}) \to_\varepsilon \mathsf{f}(1 : \mathsf{ones}) \to \dots$, however the second step is not allowed when performing context-sensitive rewriting, since $\mu(:) = \{1\}$. Using the tool AProVE [6], context-sensitive termination of the above TRS together with the replacement map μ can be shown.

We can however unfold this example (cf. [5, 13]), which makes the resulting specification proper, by introducing a fresh symbol g and replacing the two rules for f with the following three rules:

$$\mathsf{f}(x : xs) \to \mathsf{g}(x, xs) \qquad \mathsf{g}(x, y : xs) \to \mathsf{f}(y : xs) \qquad \mathsf{g}(x, xs) \to x : xs$$

Then, in the corresponding context-sensitive TRS, we have $\mu(\mathsf{f}) = \mu(:) = \{1\}$, $\mu(\mathsf{g}) = \{2\}$, and $\mu(\mathsf{ones}) = \mu(0) = \mu(1) = \emptyset$. This context-sensitive TRS is not μ-terminating, since it admits the infinite reduction $\mathsf{f}(\mathsf{ones}) \overset{\mu}{\to}_1 \mathsf{f}(1 : \mathsf{ones}) \overset{\mu}{\to}_\varepsilon \mathsf{g}(1, \mathsf{ones}) \overset{\mu}{\to}_2 \mathsf{g}(1, 1 : \mathsf{ones}) \overset{\mu}{\to}_\varepsilon \mathsf{f}(1 : \mathsf{ones}) \overset{\mu}{\to} \dots$.

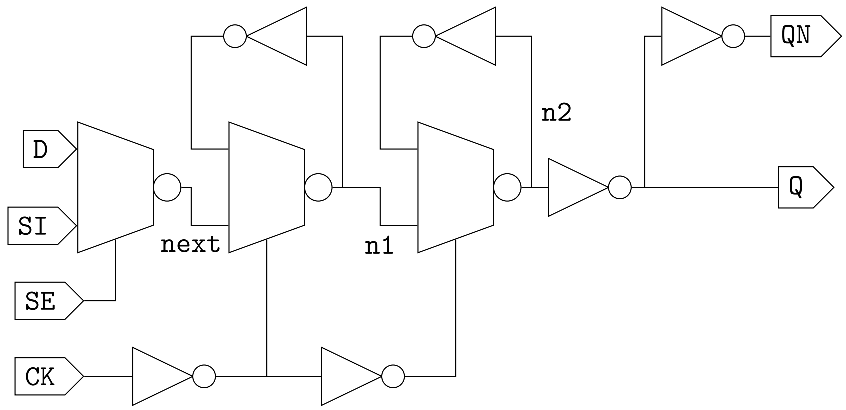

Figure 1: Example hardware circuit

It should be noted that the restriction for left-hand sides to only contain variables in constructor arguments was already made in [14]. This is the case because matching constructors nested within constructors would otherwise invalidate the approach of disallowing rewriting inside structure arguments of constructors.

5 Application to Hardware Circuits

Proving productivity can be used to verify stabilization of hardware circuits. In such a circuit, the inputs can be seen as an infinite stream of zeroes and ones, which in general can occur in any arbitrary sequence. Furthermore, a circuit contains a number of internal signals, which also carry different Boolean values over time.

To store a value over time, feedback loops are used. In such a loop, a value that is computed from some logic function is also used as an input to that function. Thus, it is desired that such values stabilize, instead of oscillating infinitely.

To check this, productivity analysis can be used. We will illustrate this by means of an example, that will be considered throughout the rest of this section.

Consider the circuit shown in Figure 1, which was constructed from the transistor netlist of the cell SDFF_X1 in the Nangate Open Cell Library [9] and which implements a scanable D flip-flop. This circuit first selects, based on the value of the input SE (scan enable), either the negation of the data input D (in case SE=0) or the negation of the scan data input SI (in case SE=1). This value, called next in Figure 1, is then fed into another multiplexer (mux), for which a feedback loop exists. This mux is controlled by the negation of the clock input CK. If the clock is 0 then the negated value of next is forwarded to the output n1, otherwise the stored value of n1 is kept. Similarly, n2 implements such a latch structure, however this time the latch forwards the negation of the n1 input in case CK is 1, and it keeps its value when CK is 0. The outputs Q and QN are computed from this stored value n2.

Note that a lot of the negations are only contained to refresh the signals, otherwise a high voltage value might decay and not be detected properly anymore.

From the example circuit, we create a proper specification, where the data symbols consist of the two Boolean values 0 and 1 and the symbol not used for negating:

$$\mathsf{not}(0) \rightarrow 1 \qquad \mathsf{not}(1) \rightarrow 0$$

$$\begin{aligned}
\mathsf{rand} &\to \mathsf{0} : \mathsf{rand}\\
\mathsf{rand} &\to \mathsf{1} : \mathsf{rand}\\
\mathsf{next}(\mathsf{0} : ses, d : ds, si : sis) &\to \mathsf{not}(d) : \mathsf{next}(ses, ds, sis)\\
\mathsf{next}(\mathsf{1} : ses, d : ds, si : sis) &\to \mathsf{not}(si) : \mathsf{next}(ses, ds, sis)\\
\\
\mathsf{n1}(\mathsf{0} : cks, nextv : nexts, n1l) &\to \mathsf{not}(nextv) : \mathsf{n1}(cks, nexts, \mathsf{not}(nextv))\\
\mathsf{n1}(\mathsf{1} : cks, nextv : nexts, n1l) &\to \mathsf{n1}'(cks, nexts, n1l, \mathsf{not}(\mathsf{not}(n1l)))\\
\mathsf{n1}'(cks, nexts, \mathsf{0}, \mathsf{0}) &\to \mathsf{0} : \mathsf{n1}(cks, nexts, \mathsf{0})\\
\mathsf{n1}'(cks, nexts, \mathsf{1}, \mathsf{1}) &\to \mathsf{1} : \mathsf{n1}(cks, nexts, \mathsf{1})\\
\mathsf{n1}'(cks, nexts, \mathsf{0}, \mathsf{1}) &\to \mathsf{n1}'(cks, nexts, \mathsf{1}, \mathsf{not}(\mathsf{not}(\mathsf{1})))\\
\mathsf{n1}'(cks, nexts, \mathsf{1}, \mathsf{0}) &\to \mathsf{n1}'(cks, nexts, \mathsf{0}, \mathsf{not}(\mathsf{not}(\mathsf{0})))\\
\\
\mathsf{n2}(\mathsf{0} : cks, n1v : n1s, n2l) &\to \mathsf{n2}'(cks, n1s, n2l, \mathsf{not}(\mathsf{not}(n2l)))\\
\mathsf{n2}(\mathsf{1} : cks, n1v : n1s, n2l) &\to \mathsf{not}(n1v) : \mathsf{n2}(cks, n1s, \mathsf{not}(n1v))\\
\mathsf{n2}'(cks, n1s, \mathsf{0}, \mathsf{0}) &\to \mathsf{0} : \mathsf{n2}(cks, n1s, \mathsf{0})\\
\mathsf{n2}'(cks, n1s, \mathsf{1}, \mathsf{1}) &\to \mathsf{1} : \mathsf{n2}(cks, n1s, \mathsf{1})\\
\mathsf{n2}'(cks, n1s, \mathsf{0}, \mathsf{1}) &\to \mathsf{n2}'(cks, n1s, \mathsf{1}, \mathsf{not}(\mathsf{not}(\mathsf{1})))\\
\mathsf{n2}'(cks, n1s, \mathsf{1}, \mathsf{0}) &\to \mathsf{n2}'(cks, n1s, \mathsf{0}, \mathsf{not}(\mathsf{not}(\mathsf{0})))\\
\\
\mathsf{q}(n2v : n2s) &\to \mathsf{not}(n2v) : \mathsf{q}(n2s)\\
\\
\mathsf{qn}(qv : qs) &\to \mathsf{not}(qv) : \mathsf{qn}(qs)
\end{aligned}$$

Figure 2: Structure TRS $\mathcal{R}_s$ for the circuit shown in Figure 1

The structures we are interested in are infinite streams containing Boolean values, thus the set of constructors is $\mathcal{C} = \{:\}$. The structure TRS $\mathcal{R}_s$ is shown in Figure 2.

It should be remarked that in the shown rules, some simplifications regarding the clock input `CK` have been made. The inverters for the clock have been removed, and the two muxes that output the signals `n1` and `n2` are provided with decoupled clock values.

The defined function symbols next, n1, n2, q, and qn reflect the wires and output signals with the corresponding name in Figure 1. The constant rand is added to abstract the values of the inputs. It provides a random stream of Boolean values, thus it is able to represent any sequence of input values provided to the circuit. The rules of the symbol next implement the mux selecting either the next data input value d in case the next scan enable input value se is 0, or the next scan input value si in case se is 1.

The output of `n1` is also computed by a mux, however, here the previous output value has to be considered due to the feedback loop. We break the cycle by introducing a new parameter $n1l$ that stores the previously output value. Then, the next value of the stream at `n1` is computed from the next value of the clock ck, the input stream $nextv : nexts$ coming from the previously described multiplexer, and from the previous output value *n1l*. If the clock *ck* is 0, then the latch simply outputs the negated value of *nextv* and continues on the remaining streams, setting the parameter *n1l* to this value to remember it. Otherwise, if *ck* is 1, then the feedback loop is active and has to be evaluated until it stabilizes. This is done by the function $\mathsf{n1}'$. It has as arguments the remaining input stream of the clock, the remaining input stream of the scan multiplexer, and the previous output value and the newly computed output value. If both of these values are the same, then the value of the wire n1 has stabilized and hence can be output. The tail of the output stream is computed by again calling the function n1 with the remaining streams for the clock and the scan multiplexer. Otherwise, the new output value (the last argument of $\mathsf{n1}'$) differs from the old output value (the penultimate argument of $\mathsf{n1}'$). In that case, the new output value becomes the old output value and the new output is recomputed. This is repeated until eventually the output value stabilizes, or it will oscillate and never produce a stable output.

Similar to the function n1, the function n2 computes stable values for the corresponding wire in Figure 1. Again, the parameter *n2l* is added to store a previously output value, and the auxiliary function $\mathsf{n2}'$ is used to compute a stable value for the feedback loop. The only difference to the function n1 is that the cases of the clock are inverted, due to the additional inverter in Figure 1 that feeds the select input of the multiplexer that computes n2. Finally, the functions q and qn implement the two inverters that feed the corresponding output signals in Figure 1.

The above specification is productive, since the TRS $\mathcal{R}_d \cup \mathcal{R}_s$ can be proven context-sensitive terminating, for example by the tool AProVE [6]. Hence, according to Theorem 12, the specification is productive, meaning that every ground term of sort *s* rewrites to a constructor term. This especially holds for the ground terms $t_{\mathsf{q}} = \mathsf{q}(t_{\mathsf{n2}})$ and $t_{\mathsf{qn}} = \mathsf{qn}(t_{\mathsf{q}})$, where $t_{\mathsf{n2}} = \mathsf{n2}(\mathsf{rand}, \mathsf{n1}(\mathsf{rand}, \mathsf{nexts}(\mathsf{rand}, \mathsf{rand}, \mathsf{rand}), n1l), n2l)$ and the variables *n1l* and *n2l* are instantiated with all possible combinations of 0 and 1. Thus, the circuit produces an infinite stream of stable output values, regardless of its initial state and input streams, and does not oscillate infinitely long. This illustrates that productivity analysis can be used to prove stabilization of digital circuits with arbitrary input sequences, when encoding them as non-orthogonal proper specifications.

6 Conclusions and Future Work

We have presented a generalization of the productivity checking techniques in [14] (including the improvements of [5]) to non-orthogonal specifications, which are able to represent non-deterministic systems. These naturally arise for example when abstracting away certain details of an implementation, such as the concrete sequence of input values. This was used to verify stabilization of hardware descriptions whose environment is left unspecified, as was demonstrated in Section 5.

Our setting still imposes certain restrictions on the specifications that can be treated. The most severe restriction is the requirement of left-linear rules in the structure TRS $\mathcal{R}_s$. Dropping this requirement however would make Theorem 12 unsound. Similarly, also the requirement that structure arguments of constructors must be variables cannot be dropped without losing soundness of Theorem 12. This requirement however is not that severe in practice, since many specifications can be unfolded by introducing fresh symbols, as was presented in [5, 13].

In the future, it would be interesting to investigate whether transformations of non-orthogonal proper specifications, similar to those in [14], can be defined. It is clear that rewriting of right-hand sides for example is not productivity-preserving for non-orthogonal specifications, since it only considers one possible reduction. However, it would be interesting to investigate whether for example narrowing of right-hand sides is productivity preserving, as it considers all possible reductions.

Acknowledgment. The author would like to thank the anonymous reviewers for their valuable comments and suggestions that helped to improve the paper.

References

[1] F. Baader & T. Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press. Available at http://dx.doi.org/10.2277/0521779200.

[2] J. Endrullis (2010): *Termination and Productivity*. Ph.D. thesis, Vrije Universiteit Amsterdam. Available at http://hdl.handle.net/1871/15879.

[3] J. Endrullis, C. Grabmayer & D. Hendriks (2008): *Data-oblivious Stream Productivity*. In: *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'08)*, *Lecture Notes in Computer Science* 5330, Springer-Verlag, pp. 79–96. Available at http://dx.doi.org/10.1007/978-3-540-89439-1_6. Web interface tool: http://infinity.few.vu.nl/productivity/.

[4] J. Endrullis, C. Grabmayer & D. Hendriks (2009): *Complexity of Fractran and Productivity*. In: *Proceedings of the 22nd Conference on Automated Deduction (CADE'09)*, *Lecture Notes in Computer Science* 5663, Springer-Verlag, pp. 371–387. Available at http://dx.doi.org/10.1007/978-3-642-02959-2_28.

[5] J. Endrullis & D. Hendriks (2011): *Lazy Productivity via Termination*. *Theoretical Computer Science* 412(28), pp. 3203–3225. Available at http://dx.doi.org/10.1016/j.tcs.2011.03.024.

[6] J. Giesl, P. Schneider-Kamp & R. Thiemann (2006): *AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework*. In: *Proceedings of the 3rd International Joint Conference on Automatic Reasoning (IJCAR'06)*, *Lecture Notes in Computer Science* 4130, Springer-Verlag, pp. 281–286. Available at http://dx.doi.org/10.1007/11814771_24. Web interface tool: http://aprove.informatik.rwth-aachen.de.

[7] S. Lucas (1998): *Context-sensitive Computations in Functional and Functional Logic Programs*. *Journal of Functional and Logic Programming* 1998(1). Available at http://mitpress.mit.edu/e-journals/JFLP/articles/1998/A98-01/A98-01.html.

[8] S. Lucas (2002): *Context-Sensitive Rewrite Strategies*. *Information and Computation* 178(1), pp. 294–343. Available at http://dx.doi.org/10.1006/inco.2002.3176.

[9] Nangate Inc. (2008): *Open Cell Library v2008_10 SP1*. Available at http://www.nangate.com/openlibrary/.

[10] S. Peyton Jones (2003): Haskell 98 *Language and Libraries: The Revised Report*. Cambridge University Press. Available at http://www.haskell.org/definition.

[11] Terese (2003): *Term Rewriting Systems*. *Cambridge Tracts in Theoretical Computer Science* 55, Cambridge University Press, Cambridge, UK. Available at http://dx.doi.org/10.2277/0521391156.

[12] H. Zantema (2008): *Normalization of Infinite Terms*. In: *Proceedings of the 19th International Conference on Rewriting Techniques and Applications (RTA'08)*, *Lecture Notes in Computer Science* 5117, Springer-Verlag, pp. 441–455. Available at http://dx.doi.org/10.1007/978-3-540-70590-1_30.

[13] H. Zantema (2009): *Well-definedness of Streams by Termination*. In: *Proceedings of the 20th International Conference on Rewriting Techniques and Applications (RTA'09)*, *Lecture Notes in Computer Science* 5595, Springer-Verlag, pp. 164–178. Available at http://dx.doi.org/10.1007/978-3-642-02348-4_12.

[14] H. Zantema & M. Raffelsieper (2010): *Proving Productivity in Infinite Data Structures*. In: *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA'10)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 6, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 401–416. Available at http://dx.doi.org/10.4230/LIPIcs.RTA.2010.401.

[15] H. Zantema & M. Raffelsieper (2010): *Stream Productivity by Outermost Termination*. In: *Proceedings of the 9th International Workshop in Reduction Strategies in Rewriting and Programming (WRS'09)*, *Electronic Proceedings in Theoretical Computer Science* 15. Available at http://dx.doi.org/10.4204/EPTCS.15.7.