Term rewriting characterisation of LOGSPACE for finite and infinite data

Łukasz Czajka[1]
University of Copenhagen, Denmark
luta@di.ku.dk

Abstract

We show that LOGSPACE is characterised by finite orthogonal tail-recursive cons-free constructor term rewriting systems, contributing to a line of research initiated by Neil Jones. We describe a LOGSPACE algorithm which computes constructor normal forms. This algorithm is used in the proof of our main result: that simple stream term rewriting systems characterise LOGSPACE-computable stream functions as defined by Ramyaa and Leivant. This result concerns characterising logarithmic-space computation on infinite streams by means of infinitary rewriting.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic, Theory of computation → Equational logic and rewriting

Keywords and phrases LOGSPACE, implicit complexity, term rewriting, infinitary rewriting, streams

Digital Object Identifier 10.4230/LIPIcs.FSCD.2018.13

1 Introduction

The goal of the field of implicit computational complexity is to characterise computational complexity classes without reference to external measuring conditions. One of the first such implicit characterisations was that of LOGSPACE as the class of problems which can be decided by deterministic two-way multihead finite automata [6]. Inspired by this well-known characterisation, Neil Jones gave new characterisations of this class as "cons-free" tail-recursive programs in several formalisms [9, 7, 8]. In cons-free programs data constructors cannot occur in function bodies. Put differently, cons-free programs are read-only: recursive data can only be read from input, but not created or altered (except taking subterms). Cons-free programming was subseqently used to characterise a variety of complexity classes [9, 7, 8, 2, 3, 10, 11, 12].

In this paper we extend the cons-free approach to computation on infinite streams. In [14, 13] Ramyaa and Leivant define the class of LOGSPACE-computable stream functions and show that it is characterised by ramified corecurrence in two tiers. Our main contribution is a cons-free infinitary term-rewriting characterisation of this class. We show that a stream function is computable in LOGSPACE, in the sense of Ramyaa and Leivant, if and only if it is definable in a simple stream TRS. As an intermediate step, we also give infinitary rewriting characterisations of stream functions computable by (jumping) finite stream transducers.

In order to obtain our characterisation of LOGSPACE-computability on streams, we give an algorithm to compute the (finite) constructor normal form of a (finite) term of a certain

[1] Supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement number 704111.

3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018).
Editor: Hélène Kirchner; Article No. 13; pp. 13:1–13:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

form in a finite orthogonal tail-recursive cons-free constructor TRS. Using this algorithm we obtain a term rewriting characterisation of LOGSPACE (in the ordinary finite sense).

In previous work [9, 8, 2] LOGSPACE was characterised by tail-recursive cons-free programs. The idea to transpose characterisations obtained via cons-free programs into the formalism of TRSs has already been exploited to characterise other complexity classes in [3, 11, 10], but there orthogonality was not assumed. Our method of introducing $\bot$-reductions may be seen as a degenerate case of the method in [3] (see also [10]), but the algorithm used there to compute constructor normal forms in polynomial time is fundamentally different from ours and does not easily adapt to logarithmic space computation. In the first part of this paper, the main novelty is a trick to detect looping in logarithmic space, and using this to obtain a LOGSPACE algorithm for computing constructor normal forms.

2 Term rewriting systems

We assume familiarity with term rewriting [1]. In this short section we fix the notation and briefly recall some definitions.

▶ **Definition 2.1.** A *term rewriting system* (TRS) is a set of *rules* of the form $l \to r$ where l, r are terms and l is not a variable and $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$, where $\mathrm{Var}(t)$ denotes the variables occurring in t. Given a TRS R, the reduction relation $\to_R$ is the compatible closure of the contraction relation $\{(\sigma l, \sigma r) \mid l \to r \in R, \sigma \text{ a substitution}\}$. We use $\to^*$ for the transitive-reflexive closure of $\to$, and $\to^=$ for the reflexive closure, and $\rightrightarrows$ for the parallel closure. For precise definitions see [1]. In particular, $\rightrightarrows$ is reflexive.

A *defined symbol* in a TRS R is a function symbol which occurs at the root of a left-hand side of a rule in R. A *constructor symbol* in a TRS R is a function symbol which is not a defined symbol in R. A *constructor term* is a term which does not contain defined function symbols (it may contain variables). A *constructor normal form* is a constructor term which does not contain variables (so it contains only constructors). A *constructor head normal form* (chnf) is a term of the from $c(t_1, \ldots, t_n)$ with c a constructor. A *constructor TRS* is a TRS R such that for $l \to r \in R$ we have $l = f(l_1, \ldots, l_n)$ where $l_1, \ldots, l_n$ are constructor terms.

A redex is *innermost* if it does not contain other redexes. A reduction step is innermost if it contracts an innermost redex.

A *decision problem* is a set of binary words $A \subseteq \{0, 1\}^*$. Assuming the signature contains the constants $0, 1, \mathrm{nil}$ and a binary constructor symbol cons, every $w \in \{0, 1\}^*$ may be represented by a term $\bar{w}$ in an obvious way. A TRS R *accepts* a decision problem A if there is a function symbol f such that for every $w \in \{0, 1\}^*$ we have: $f(\bar{w}) \to_R^* 1$ iff $w \in A$.

3 LOGSPACE for finite data

In this section we show that finite orthogonal tail-recursive cons-free constructor TRSs characterise LOGSPACE, i.e., a decision problem is in LOGSPACE iff it is accepted by a finite orthogonal tail-recursive cons-free constructor TRS. As part of the proof we give an algorithm which computes the constructor normal form of a term of a certain form, if there exists one, or rejects otherwise. This algorithm will also be used in Section 6.

▶ **Definition 3.1.** A constructor TRS R is *cons-free* if for each $l \to r \in R$ every chnf subterm of r either occurs in l or is a constructor normal form. A constructor TRS R is *tail-recursive* if there is a preorder $\gtrsim$ on defined function symbols such that for every $f(u_1, \ldots, u_n) \to r \in R$ and every defined function symbol g the following hold:

- if $r = g(t_1, \ldots, t_k)$ then $f \gtrsim g$,
- if $g(t_1, \ldots, t_k)$ is a proper subterm of r then $f > g$.

A TRS is *strictly tail-recursive* if it is tail-recursive and each right-hand side of a rule contains at most one defined function symbol.

For terms $t_1, \ldots, t_n$ by $\mathcal{B}(t_1, \ldots, t_n)$ we denote the sets of all constructor normal forms occurring either in one of t_i or in a right-hand side of a rule of R. Note that $\mathcal{B}(t_1, \ldots, t_n)$ is finite if R is.

Our definition of tail-recursiveness is based on standard definitions in the literature [8, 2], adapted to the term rewriting framework.

▶ **Proposition 3.2.** *Any problem decidable in LOGSPACE is accepted by a finite orthogonal tail-recursive cons-free constructor TRS.*

Proof. This is a straightforward adaptation of previous work [7, 2]. One may e.g. easily encode any $\mathrm{CM}^{\setminus +}$ program from [7] by a finite orthogonal strictly tail-recursive cons-free constructor TRS. Because the obtained TRS is orthogonal and strictly tail-recursive, the reduction strategy does not play a significant role. We skip the routine details. ◀

It is more difficult to show the other direction of the characterisation result, i.e., that any decision problem accepted by a finite orthogonal tail-recursive cons-free constructor TRS is in LOGSPACE. Indeed, if the TRS is tail-recursive but not strictly tail-recursive, then terms which have a constructor normal form may also have arbitrarily large reducts. Consider e.g. the following TRS R:

$$f(x) \to_R f(g(x)) \qquad h(x) \to_R a$$

Then $h(f(a)) \to_R a$ but also $h(f(a)) \to_R^* h(f(g^n(a)))$ for any $n \in \mathbb{N}$. This example also shows that the innermost strategy may fail to give a normal form even if a term has one.

We will show that a constructor normal form may always be reached by an eager $R\bot$-reduction, denoted $\to_{R\bot e}^*$, which contracts only innermost R-redexes and eagerly (as soon as possible) replaces by $\bot$ an innermost subterm with no constructor normal form in R. For instance, in the example TRS R given above $h(f(a)) \to_\bot h(\bot) \to_R a$ is an eager $R\bot$-reduction, but $h(f(a)) \to_R h(f^2(a))$ is not. The term $f(a)$ does not have a constructor normal form in R, so it *cannot* be R-contracted in an eager $R\bot$-reduction – it *must* be contracted to $\bot$.

Whether a subterm has a constructor normal form in R may be decided using a constant number of logarithmic counters. An eager $R\bot$-reduction has the form

$$f_1(w_1^1, \ldots, w_{n_1}^1) \to_{R\bot e}^* f_1(t_1^1, \ldots, t_{n_1}^1) \to_R^\epsilon f_2(w_1^2, \ldots, w_{n_2}^2) \to_{R\bot e}^* f_2(t_1^2, \ldots, t_{n_2}^2) \to_R^\epsilon \ldots$$

where t_i^j is the constructor normal form w.r.t. eager $R\bot$-reduction of w_i^j ($\bot$ is considered to be a constructor) and $f_i \gtrsim f_j$ for $i \le j$. At some point either we reach a constructor normal form or a term $f_i(t_1^i, \ldots, t_{n_i}^i)$ repeats. Because of cons-freeness, there are only polynomially many such terms. Hence, a logarithmic counter may be used to detect looping. Because of tail-recursiveness, computing the constructor normal form (w.r.t. eager $R\bot$-reduction) t_i^j of w_i^j may be done by a recursive invocation, and the recursion depth will be constant. The rest of this section is devoted to making the above arguments precise.

▶ **Definition 3.3.** Let R be a constructor TRS and let $\bot$ be a fresh constant, i.e., not occurring in any of the rules of R. We define the $\bot$-*contraction* relation $\to_\bot^\epsilon$ by: $t \to_\bot^\epsilon \bot$ if t does not R-reduce to a constructor normal form. The $\bot$-*reduction* relation $\to_\bot$ is the compatible closure of $\to_\bot^\epsilon$. We set $\to_{R\bot} = \to_R \cup \to_\bot$. An $R\bot$-reduction is *eager* if only innermost $R\bot$-redexes are contracted and priority is given to $\bot$-reduction, i.e., an R-redex t such that $t \to_\bot \bot$ is not R-contracted in the reduction. We use $\to_{R\bot e}$ for an eager one-step $R\bot$-reduction.

Note that $\bot$ is a constructor. So a term of the form $c(t_1, \dots, t_n)$ with c a constructor never eagerly $R\bot$-reduces to $\bot$, because if it does not have a constructor normal form in R then there is a $R\bot$-redex in one of the t_i. Note that a term is in normal form w.r.t. $R\bot$-reduction iff it is a constructor normal form.

We first show that in a left-linear constructor TRS $\bot$-reduction may be postponed after R-reduction. This will imply that eager $R\bot$-reduction to a constructor normal form not containing $\bot$ may be replaced with R-reduction.

▶ **Lemma 3.4.** *In a left-linear constructor TRS, if $u \rightrightarrows_\bot t \to_R t'$ then there is u' with $u \to_R u' \rightrightarrows_\bot t'$.*

Proof. Without loss of generality we may assume that $t \to_R t'$ occurs at the root by a rule $l \to r$ with substitution σ. By the choice of $\bot$ the term l does not contain $\bot$. We have $t = \sigma(l)$. So $\bot$ in t may occur only below a variable position of l. Since $\bot$ are the contracta in $u \rightrightarrows_\bot t$, the expansions $u \rightrightarrows_\bot t$ in t occur below variable positions of l. Hence, there is σ' such that $\sigma'(x) \rightrightarrows_\bot \sigma(x)$ for all $x \in \mathrm{Var}(l)$ and $u = \sigma'(l)$. Then take $u' = \sigma'(r)$. ◀

▶ **Corollary 3.5.** *In a left-linear constructor TRS, if $t \to_{R\bot}^* t'$ then there is u with $t \to_R^* u \to_\bot^* t'$.*

▶ **Lemma 3.6.** *In a left-linear constructor TRS, if $t \to_{R\bot}^* s$ with s a constructor normal form not containing $\bot$, then $t \to_R^* s$.*

Proof. Induction on the number n of $\bot$-contractions in $t \to_{R\bot}^* s$. If $n > 0$ then consider the last $\bot$-contraction: $t \to_{R\bot}^* t' \to_\bot t'' \to_R^* s$. By Lemma 3.4 there is s' with $t' \to_R^* s' \rightrightarrows_\bot s$. Because s does not contain $\bot$, we have $s' = s$. So $t \to_{R\bot}^* s$ with $n-1$ $\bot$-contractions. Hence $t \to_R^* s$ by the inductive hypothesis. ◀

The following lemma shows that eager $R\bot$-reduction in $\sigma(t)$, with t a linear constructor term, occurs below variable positions.

▶ **Lemma 3.7.** *In a constructor TRS R, if t is a linear constructor term and $\sigma(t) \to_{R\bot e}^* t'$ then there is σ' such that $t' = \sigma'(t)$ and $\sigma(x) \to_{R\bot e}^* \sigma'(x)$ for all $x \in \mathrm{Var}(t)$.*

Proof. Induction on t. If $t = x$ then take $\sigma'(x) = t'$. Otherwise $t = c(t_1, \dots, t_n)$ and $t' = c(t_1', \dots, t_n')$ with $\sigma(t_i) \to_{R\bot e}^* t_i'$ and c a constructor. By the inductive hypothesis for $i = 1, \dots, n$ there is σ_i' with $\sigma_i'(t_i) = t_i'$ and $\sigma(x) \to_{R\bot e}^* \sigma_i'(x)$ for $x \in \mathrm{Var}(t_i)$. Because t is linear, $\mathrm{Var}(t_i) \cap \mathrm{Var}(t_j) = \emptyset$ for $i \neq j$. So the σ_i's may be combined into a single substitution σ' with the required properties. ◀

▶ **Corollary 3.8.** *In a left-linear constructor TRS R, if $f(t_1, \dots, t_n) \to_R^\epsilon t$ and $t_i \to_{R\bot e}^* t_i'$ for $i = 1, \dots, n$, then there is t' with $f(t_1', \dots, t_n') \to_R^\epsilon t' \;{}_{R\bot e}^*\!\!\leftarrow t$. Moreover, the contraction $f(t_1', \dots, t_n') \to_R^\epsilon t'$ is by the same rule as $f(t_1, \dots, t_n) \to_R^\epsilon t$.*

Proof. Assume $f(t_1, \dots, t_n) \to_R^\epsilon t$ by a rule $f(l_1, \dots, l_n) \to r$ with substitution σ. Because each l_i is a linear constructor term and $\mathrm{Var}(l_i) \cap \mathrm{Var}(l_j) = \emptyset$ for $i \neq j$, by Lemma 3.7 there is σ' such that for $i = 1, \dots, n$ we have $\sigma'(l_i) = t_i'$ and $\sigma(x) \to_{R\bot e}^* \sigma'(x)$. Thus $u = f(\sigma'(l_1), \dots, \sigma'(l_n)) \to_R \sigma'(r)$. Also $t' = \sigma(r) \to_{R\bot e}^* \sigma'(r)$, because $\mathrm{Var}(r) \subseteq \mathrm{Var}(l_1, \dots, l_n)$. So we may take $t' = \sigma'(r)$. ◀

The next lemma shows a strengthening of the diamond property for eager $R\bot$-reduction in orthogonal TRSs.

▶ **Lemma 3.9.** *In an orthogonal TRS R, if $t \to_{R\bot e} t_1$ and $t \to_{R\bot e} t_2$ then either $t_1 = t_2$ or there is t' with $t_1 \to_{R\bot e} t'$ and $t_2 \to_{R\bot e} t'$.*

Proof. If the redexes are parallel then the second part of the disjunction holds. Because both redexes are innermost, if they are not parallel we may assume without loss of generality that both of them are at the root. If both of them are R-redexes, then $t_1 = t_2$ by orthogonality. If both are $\bot$-redexes then $t_1 = t_2 = \bot$. It is not possible that one redex is a $\bot$-redex and the other an R-redex, because the reductions are eager. ◀

The following simple lemma is needed in the proof of Lemma 3.11.

▶ **Lemma 3.10.** *In a cons-free constructor TRS, if every subterm of t in chnf is in constructor normal form and $t \to_R^* t'$ and t' is in chnf, then t' is in constructor normal form.*

Proof. Because the TRS is cons-free, any chnf subterm of any R-reduct of t must be in $\mathcal{B}(t)$. More precisely, one shows that if $t \to_R u$ then still every subterm of u in chnf is in constructor normal form. ◀

In the rest of this section we assume that R is a finite orthogonal tail-recursive cons-free constructor TRS.

Note that because R is finite and tail-recursive the partial order on the equivalence classes determined by $\gtrsim$ may be extended to a well order $>_E$. We write $t_1 >_E t_2$ ($t_1 \geq_E t_2$) if the greatest equivalence class of a defined function symbol in t_1 is greater (greater or equal) than the greatest equivalence class of a defined function symbol in t_2. We write $f \leq_E t$ if the equivalence class of the defined function symbol f is less or equal to the greatest equivalence class of a defined function symbol in t. Note that if $t \to_{R\bot}^* t'$ then $t \geq_E t'$, because R is tail-recursive.

Our next goal is to show that every term has a constructor normal form (possibly containing $\bot$) reachable by eager $R\bot$-reduction. This will imply that eager $R\bot$-reduction commutes with R-reduction, and that eager $R\bot$-reduction is terminating.

▶ **Lemma 3.11.** *Assume that for all t' with $t' \leq_E t$ there is s in constructor normal form such that $t' \to_{R\bot e}^* s$. If $t' \;{}_R\!\leftarrow t \to_{R\bot e} u$ then there is u' with $t' \to_{R\bot e}^* u' \;{}^{=}_{R}\!\leftarrow u$.*

Proof. Note that because the redex contracted in $t \to_{R\bot e} u$ is innermost, it cannot happen that the redex contracted in $t \to_R t'$ occurs strictly inside this redex. So we may assume without loss of generality that the redex contracted in $t \to_R t'$ occurs at the root.

If $u = \bot$ then $t = f(s_1, \ldots, s_n)$ with $s_1, \ldots, s_n$ in constructor normal form, because the $R\bot$-reduction is innermost. Since $t' \leq_E t$, there is a constructor normal form s such that $t' \to_{R\bot e}^* s$. If $s = \bot$ then we may take $u' = \bot$. Otherwise, $s = c(s_1', \ldots, s_m')$ with $c \neq \bot$ a constructor. By Corollary 3.5 there is w with $t \to_R t' \to_R^* w \to_\bot^* s$. Then w is in chnf. But then by Lemma 3.10 it is in constructor normal form. This contradicts $t \to_\bot \bot$.

If $t \to_{R\bot e} u$ contracts an R-redex at the root, then $u = t'$ because R is orthogonal, so take $u' = t'$. The remaining case, when the eager $R\bot$-contraction occurs strictly below the root, follows from Corollary 3.8. ◀

▶ **Lemma 3.12.** *Assume that for all t with $t <_E g$ there is s in constructor normal form such that $t \to_{R\bot e}^* s$. If $g(t_1, \ldots, t_n) \to_R^* s$ with g a defined function symbol and s in constructor normal form and $t_i <_E g$ and $t_i \to_{R\bot e}^* w_i$ for $i = 1, \ldots, n$, then $g(w_1, \ldots, w_n) \to_R^* s$.*

Proof. Induction on the number of root steps in $g(t_1,\ldots,t_n) \to_R^* s$. There is at least one root step, so $g(t_1,\ldots,t_n) \to_R^* g(t'_1,\ldots,t'_n) \to_R^\epsilon t \to_R^* s$ and, because R is cons-free and tail-recursive, either $t = s$ or $t = g'(u_1,\ldots,u_m)$ with $g' \leq_E g$ and $u_i <_E g$. By Lemma 3.11 there are $w'_1,\ldots,w'_n$ such that $w_i \to_R^* w'_i$ and $t'_i \to_{R\bot e}^* w'_i$ for $i = 1,\ldots,n$. By Corollary 3.8 there is t' with $g(w'_1,\ldots,w'_n) \to_R^\epsilon t'$ and $t \to_{R\bot e}^* t'$. If $t = s$ then $t' = s$ and we are done. Otherwise, by Corollary 3.8, $t' = g'(u'_1,\ldots,u'_m)$ and $u_i \to_{R\bot e}^* u'_i$. By the inductive hypothesis $t' \to_R^* s$. Hence $g(w_1,\ldots,w_n) \to_R^* g(w'_1,\ldots,w'_n) \to_R t' \to_R^* s$. ◀

▶ **Lemma 3.13.** *Assume that for all t with $t <_E g$ there is s in constructor normal form such that $t \to_{R\bot e}^* s$. If $g(t_1,\ldots,t_n) \to_R^* s$ with g a defined function symbol and s in constructor normal form and $t_i <_E g$ for $i = 1,\ldots,n$, then $g(t_1,\ldots,t_n) \to_{R\bot e}^* s$.*

Proof. The reduction $g(t_1,\ldots,t_n) \to_R^* s$ has the form:

$$g(t_1,\ldots,t_n) \to_R^* g(u_1,\ldots,u_n) \to_R^\epsilon g_1(t_1^1,\ldots,t_{n_1}^1) \to_R^* g_1(u_1^1,\ldots,u_{n_1}^1) \to_R^\epsilon \ldots \to_R^\epsilon s.$$

We proceed by induction on the number of root steps in this R-reduction. Since $t_i <_E g$ for $i = 1,\ldots,n$, there are $s_1,\ldots,s_n$ in constructor normal form such that $t_i \to_{R\bot e}^* s_i$. By Lemma 3.11 we also have $u_i \to_{R\bot e}^* s_i$. By Corollary 3.8 there is t' with $g(s_1,\ldots,s_n) \to_R^\epsilon t'$ and either $t' = s$, or $t' = g_1(w_1,\ldots,w_{n_1})$ and $t_i^1 \to_{R\bot e}^* w_i$. We have $g(s_1,\ldots,s_n) \to_{R\bot e} s$ because the R-reduction to t' is innermost and $g(s_1,\ldots,s_n) \to_R^* s$ by Lemma 3.12. Hence if $t' = s$ then $g(t_1,\ldots,t_n) \to_{R\bot e}^* s$. So assume $t' = g_1(w_1,\ldots,w_{n_1})$ with $t_i^1 \to_{R\bot e}^* w_i$. By the inductive hypothesis $g_1(t_1^1,\ldots,t_{n_1}^1) \to_{R\bot e}^* s$. By Lemma 3.9 we obtain $g_1(w_1,\ldots,w_{n_1}) \to_{R\bot e}^* s$. Thus $g(t_1,\ldots,t_n) \to_{R\bot e}^* g(s_1,\ldots,s_n) \to_{R\bot e} g_1(w_1,\ldots,w_{n_1}) \to_{R\bot e}^* s$. ◀

▶ **Lemma 3.14.** *For every term t there exists s in constructor normal form*[2] *such that $t \to_{R\bot e}^* s$.*

Proof. We proceed by induction on pairs $\langle e, n\rangle$ ordered lexicographically, where e is the greatest, w.r.t. $>_E$, equivalence class of a defined function symbol in t, and n is the size of t. This is obvious if t is a variable. So assume $t = f(t_1,\ldots,t_n)$. Since each t_k is smaller than t, by the inductive hypothesis for each $k = 1,\ldots,n$ there is a constructor normal form s_k with $t_k \to_{R\bot e}^* s_k$. If f is a constructor then we are done, so assume it is a defined function symbol. If $f(s_1,\ldots,s_n)$ does not R-reduce to a constructor normal form, then $f(s_1,\ldots,s_n) \to_{R\bot e} \bot$, so we may take $s = \bot$. Otherwise $f(s_1,\ldots,s_n) \to_R^* s$ for some s in constructor normal form. Of course, $f \leq_E t$, so the inductive hypothesis implies that for all t' with $t' <_E f$ there is s' in constructor normal form such that $t' \to_{R\bot e}^* s'$. Thus by Lemma 3.13: $t \to_{R\bot e}^* f(s_1,\ldots,s_n) \to_{R\bot e}^* s$. ◀

▶ **Corollary 3.15.** *If $t' \ {}_R^*\!\!\leftarrow t \to_{R\bot e}^* u$ then there is u' with $t' \to_{R\bot e}^* u' \ {}_R^*\!\!\leftarrow u$.*

Proof. Follows from Lemma 3.11 and Lemma 3.14. ◀

▶ Remark. Corollary 3.15 fails if the $R\bot$-reduction is not required to be eager (though innermost would suffice). Consider the TRS R:

$$f(x) \to_R f(x) \qquad g(c(x)) \to_R a$$

We have $g(c(f(x))) \to_R a$, but also $g(c(f(x))) \to_\bot g(\bot) \to_\bot \bot$, because $c(f(x))$ does not R-reduce to a constructor normal form.

[2] Recall that $\bot$ is considered to be a constructor.

The corollary also fails if R is not required to be cons-free. Consider the TRS R:

$$f(x) \to_R f(x) \qquad g(x) \to_R c(f(x))$$

Then $g(x) \to^*_{R\bot e} \bot$. On the other hand $g(x) \to_R c(f(x))$ and $c(f(x)) \not\to^*_{R\bot e} \bot$.

If R is not required to be tail-recursive then this also fails. Consider the TRS R:

$$h(x) \to_R h(f(x)) \qquad f(x) \to_R g(x, f(x)) \qquad g(x, y) \to_R x$$

Then $h(a) \to_{R\bot e} \bot$, because $h(t)$ does not have a constructor normal form for any t. Also $h(a) \to_R h(f(a))$. The term $h(f(a))$ has no constructor normal form, but $h(f(a)) \not\to_{R\bot e} \bot$ because the $\bot$-redex is not innermost. And there is no constructor normal form s with $f(a) \to^*_{R\bot e} s$ (note that $f(a) \to_R g(a, f(a)) \to_R a$ but the reduction is not innermost). Hence, there is no eager $R\bot$-reduction from $h(f(a))$ to $\bot$.

The proof of the next lemma is an adaptation of the standard argument that in an orthogonal TRS if a term is weakly innermost normalising then it is innermost terminating.

▶ **Lemma 3.16.** *Eager $R\bot$-reduction is terminating.*

Proof. Follows from Lemma 3.14 and Lemma 3.9. Assume there is an infinite eager $R\bot$-reduction $t_0 \to_{R\bot e} t_1 \to_{R\bot e} t_2 \to_{R\bot e} \ldots$. By Lemma 3.14 there is u in constructor normal form with $t_0 \to^*_{R\bot e} u$. Using Lemma 3.9 one shows by induction on the length of $t_0 \to^*_{R\bot e} u$ that there is an infinite eager $R\bot$-reduction starting at u. This contradicts that u is a constructor normal form. ◀

Termination of eager $R\bot$-reduction is crucial in justifying the correctness of the algorithm described in the proof of the following theorem.

▶ **Proposition 3.17.** *Let R be a finite orthogonal tail-recursive cons-free constructor TRS. There is a LOGSPACE algorithm which given a term $t = f(t_1, \ldots, t_n)$, with $t_1, \ldots, t_n$ in constructor normal form (possibly containing $\bot$), computes the constructor normal form $s \in \mathcal{B}(t, \bot)$ such that $t \to^*_{R\bot e} s$.*

Proof. Note that because R is cons-free, if $t \to^*_{R\bot} t'$ then any subterm of t' with a constructor symbol at the root is in $\mathcal{B}(t, \bot)$. Because the size of $\mathcal{B}(t, \bot)$ is polynomial (there is only a constant number of constructor normal forms occurring in right-hand sides of rules in R), constructor normal forms occurring in $R\bot$-reducts of t may be represented using a logarithmic number of bits.

Because R is a tail-recursive constructor TRS, $f(t_1, \ldots, t_n)$ either is R-irreducible, in which case it may be contracted to $\bot$, or it R-contracts (eagerly) to a constructor normal form, or it R-contracts (not necessarily eagerly) to a term $f'(t'_1, \ldots, t'_m)$ where f' is a defined function symbol and $f \gtrsim f'$ and for each defined function symbol g in one of $t'_1, \ldots, t'_m$ we have $f > g$. Apply the procedure recursively, in depth-first order, to subterms of $t'_1, \ldots, t'_m$ of the form $g(u_1, \ldots, u_k)$ with g a defined function symbol and $u_1, \ldots, u_k$ in constructor normal form. This results in $s_1, \ldots, s_m$ in constructor normal form such that $t'_k \to^*_{R\bot e} s_k$. Note that the number of defined function symbols in $t'_1, \ldots, t'_m$ is constant and depends only on the rule of R applied to t. Hence only logarithmic space is needed to store (representations of) intermediate results. Note also that $f > g$ for g a defined symbol in $t'_1, \ldots, t'_m$, which guarantees termination of the recursion.

So $f(t_1, \ldots, t_n) \to^\epsilon_R f'(t'_1, \ldots, t'_m) \to^*_{R\bot e} f'(s_1, \ldots, s_m)$ with $s_1, \ldots, s_m$ again in constructor normal form. We keep repeating the steps described in the previous paragraph, starting with $f'(s_1,\ldots,s_m)$ now, until we reach a constructor normal form or we detect looping in which case $\perp$ is returned. Looping detection may be realised using a single counter with a logarithmic number of bits. Indeed, by repeating the steps described in the previous paragraph we obtain a reduction of the form

$$t \to_R^\epsilon f_1(w_1^1,\ldots,w_{n_1}^1) \to_{R\perp e}^* f_1(t_1^1,\ldots,t_{n_1}^1) \to_R^\epsilon f_2(w_1^2,\ldots,w_{n_2}^2) \to_{R\perp e}^* f_2(t_1^2,\ldots,t_{n_2}^2) \to_R^\epsilon ..$$

where the $R\perp e$-reductions occur strictly below the root. Let M be the maximum arity of a defined function symbol in R, and K the number of defined function symbols in R, and N the size of $\mathcal{B}(t)$ (note that N is bounded by the size of t plus a constant). There are at most N different constructor normal forms occurring in the $R\perp$-reducts of t, so if the above reduction contains more than KN^M root steps, then one of the root R-redexes $f_i(t_1^i,\ldots,t_{n_i}^i)$ must repeat. So we keep a counter and return $\perp$ after performing KN^M root steps if we do not stop with a constructor normal form earlier. To see that this is correct, note that if a root redex repeats then an infinite reduction of the above form may be constructed. Assume $t \to_R^* s$ for a constructor normal form s. Then the initial R-contraction $t \to_R^\epsilon f_1(w_1^1,\ldots,w_{n_1}^1)$ is eager, so $t \to_{R\perp e}^* f_1(t_1^1,\ldots,t_{n_1}^1)$, and thus $f_1(t_1^1,\ldots,t_{n_1}^1) \to_R^* s$ by Corollary 3.15. By induction on k we show that $f_k(t_1^k,\ldots,t_{n_k}^k) \to_R^* s$ and each of the root R-contractions $f_k(t_1^k,\ldots,t_{n_k}^k) \to_R^\epsilon f_{k+1}(w_1^{k+1},\ldots,w_{n_{k+1}}^{k+1})$ is eager, i.e.

$$t \to_{R\perp e}^+ f_1(t_1^1,\ldots,t_{n_1}^1) \to_{R\perp e}^+ f_2(t_1^2,\ldots,t_{n_2}^2) \to_{R\perp e}^+ f_3(t_1^3,\ldots,t_{n_3}^3) \to_{R\perp e}^+ \ldots$$

Hence, there exists an infinite eager $R\perp$-reduction from t, which contradicts Lemma 3.16. Thus, if a root redex repeats then $t \to_\perp \perp$. So returning $\perp$ is correct in this case.

The above algorithm terminates and the recursion depth (the maximum nesting of recursive calls) is constant, because in the recursive calls for subterms of $t_1',\ldots,t_m'$ the defined function symbol at the root is strictly smaller in the preorder $\gtrsim$. Also note that in each recursive call on a subterm $g(u_1,\ldots,u_n)$ of one of $t_1',\ldots,t_m'$ the constructor normal forms $u_1,\ldots,u_n$ are in $\mathcal{B}(t,\perp)$, because then $g(u_1,\ldots,u_n)$ is a subterm of an $R\perp$-reduct of t. So $u_1,\ldots,u_n$ may still be represented in logarithmic space. Hence, at each recursive invocation the algorithm uses logarithmic space to store the representations of the function symbol arguments, a constant number of logarithmic-space variables to store the intermediate results of recursive calls, and a logarithmic counter to detect looping. Since the recursion depth is constant, the algorithm altogether uses logarithmic space. ◀

▶ **Theorem 3.18.** *A decision problem is in LOGSPACE iff it is accepted by a finite orthogonal tail-recursive cons-free constructor TRS.*

Proof. The direction from left to right follows from Proposition 3.2. For the other direction it suffices to show an algorithm which given a finite orthogonal tail-recursive cons-free constructor TRS R and a term $t = f(t_1,\ldots,t_n)$ with $t_1,\ldots,t_n$ in constructor normal form not containing $\perp$, computes in LOGSPACE the constructor normal for of t, if it has one, or rejects otherwise. The algorithm is to run the procedure from Proposition 3.17 to find a constructor normal form s with $t \to_{R\perp e}^* s$. If s does not contain $\perp$ then $t \to_R^* s$ by Lemma 3.6. Otherwise, t does not have a constructor normal form in R and we reject. Indeed, if $t \to_R^* s'$ with s' in constructor normal form then s' does not contain $\perp$ because t does not. But $s = s'$ by Corollary 3.15. ◀

4 Stream Term Rewriting Systems

In this section we define stream TRSs which allow possibly infinite stream terms. We define infinitary reduction in a stream TRS which captures the notion of a "limit" of an infinite reduction sequence.

▶ **Definition 4.1.** A *stream TRS* is a two-sorted constructor TRS with sorts s (the sort of streams) and d (the sort of finite data), finitely many defined function symbols, finitely many data constructors $c_i : d^n \to d$, and one binary stream constructor $\mathrm{cons} : d \times s \to s$. Terms of sort s are *stream terms*. Terms of sort d are *data terms*. For stream TRSs we allow terms to be infinite. We write $t_1 :: t_2$ instead of $\mathrm{cons}\, t_1\, t_2$. If $l \to r \in R$ is a rule, then we require that l and r have the same sort.

Stream rules are the rules $l \to r$ such that l is a stream term. *Data rules* are the rules $l \to r$ such that l is a data term. A *stream (resp. data) function symbol* is a defined function symbol of type $\tau_1 \times \ldots \times \tau_n \to s$ (resp. $\tau_1 \times \ldots \times \tau_n \to d$).

A *simple stream rule* has the form:

$$f(u_1, \ldots, u_n) \to t_1 :: \ldots :: t_k :: g(w_1, \ldots, w_m)$$

where $k \geq 0$ and we require:

1. $u_1, \ldots, u_n$ are constructor terms,
2. every stream subterm of one of $t_1, \ldots, t_k, w_1, \ldots, w_m$ occurs (as a subterm) in $u_1, \ldots, u_n$,
3. if $k = 0$ then every data subterm $c(v_1, \ldots, v_j)$ of each of $w_1, \ldots, w_m$, with $c : d^j \to d$ a data constructor, either occurs in $u_1, \ldots, u_n$ or is a constructor normal form.

The intuitive interpretation of the restrictions of a simple stream rule is that it is cons-free with respect to stream subterms, and if the rule does not produce a new stream element then it is also cons-free with respect to data subterms.

Note that by requiring $u_1, \ldots, u_n$ to be constructor terms and every stream subterm of each of $t_1, \ldots, t_k, w_1, \ldots, w_m$ to occur in $u_1, \ldots, u_n$, we ensure that stream function symbols cannot occur in $t_1, \ldots, t_k, w_1, \ldots, w_m$, i.e., g is the only stream function symbol in the right-hand side. Hence, the only function symbols present in $t_1, \ldots, t_k, w_1, \ldots, w_m$ are of data sort.

▶ **Example 4.2.** Here are some examples of simple stream rules, where x, x' are stream variables, and y is a data variable, and c is a data constructor, and h is a defined data function symbol:

$$\begin{array}{rcl} f(a :: x, y) & \to & a :: f(x, c(y)) \\ f(a :: x, b :: x') & \to & a :: b :: f(b :: x', a :: x) \\ f(a :: x) & \to & a :: g(x, c(a)) \\ f(a :: x, y) & \to & f(x, h(y)) \end{array}$$

Here are some non-examples:

$$\begin{array}{rcl} f(a :: x, y) & \to & f(x, c(y)) \\ f(a :: x, b :: x') & \to & a :: b :: f(g(x'), a :: x) \\ f(a :: x) & \to & a :: g(b :: x, c(a)) \\ f(a :: x, h(y)) & \to & f(x, h(y)) \end{array}$$

▶ **Definition 4.3.** Given a stream TRS R, *infinitary R-reduction* is defined coinductively.

$$\frac{t \to_R^* t'}{t \to_R^\infty t'} \qquad \frac{t \to_R^* u :: w \quad w \to_R^\infty w'}{t \to_R^\infty u :: w'}$$

Coinductive definitions of infinitary rewriting originate from [4, 5]. Intuitively, the definition means that $t \to_R^\infty t'$ holds if this may be derived using the above rules in a possibly infinite derivation. For example, if $f(x) \to x :: f(S(x))$ is a stream rule in R, then $f(0) \to_R^\infty 0 :: S(0) :: S(S(0)) :: \ldots$, i.e., $f(0)$ infinitarily reduces to an infinite stream of consecutive natural numbers.

The above definition differs from the standard definition of infinitary reduction via strongly convergent reduction sequences. The difference is mainly because we effectively disallow an infinitary reduction to produce an infinite nesting of defined function symbols. This eliminates the problems with confluence in infinitary rewriting. Infinitary R-reduction, defined as above, is confluent if R is finite and orthogonal. First of all, confluence holds also for finitary R-reduction.

▶ **Lemma 4.4.** *If R is finite and orthogonal then the finitary reduction relation $\to_R$ is confluent.*

Proof. Note that the terms may be infinite. But because both the left- and right-hand sides of all rules are finite, we may use virtually the same proof as in the case of ordinary orthogonal term rewriting systems, *mutatis mutandis.* ◀

Because of space limits we delegate the proof of confluence of infinitary reduction to Appendix A. Here we only state the result.

▶ **Theorem 4.5.** *If R is finite and orthogonal then $\to_R^\infty$ is confluent, i.e., if $t \to_R^\infty t_1$ and $t \to_R^\infty t_2$ then there exists t' such that $t_1 \to_R^\infty t'$ and $t_2 \to_R^\infty t'$.*

Let Σ be an alphabet. Assuming all elements of Σ are data constants in the rewriting system, each Σ-stream (infinite word in Σ^ω) may be treated as an infinite stream term. Also, finite words over Σ may be represented as stream terms in the TRS, where after the symbols representing the word there is a term with no constructor head normal form, e.g., $a :: b :: c :: \Omega$ represents the word abc, where Ω has no chnf. Note that a stream term in chnf (Definition 2.1) has the form $u :: w$. We denote the set of terms representing finite and infinite words over Σ by $\mathcal{S}^+(\Sigma)$, and the set of terms representing infinite words by $\mathcal{S}(\Sigma)$. More precisely, the set $\mathcal{S}^+(\Sigma)$ is defined coinductively as follows.

$$\frac{t \text{ has no chnf}}{t \in \mathcal{S}^+(\Sigma)} \qquad \frac{c \in \Sigma \quad t \in \mathcal{S}^+(\Sigma)}{(c :: t) \in \mathcal{S}^+(\Sigma)}$$

For each term t in $\mathcal{S}^+(\Sigma)$ there is exactly one corresponding finite or infinite word $|t|$ in $\Sigma^{\leq\omega} = \Sigma^\omega \cup \Sigma^*$ which this term represents.

▶ **Lemma 4.6.** *Assume $t \to_R^\infty t'$. Then t has a chnf iff t' has a chnf.*

Proof. Follows from definitions and Lemma 4.4. ◀

▶ **Corollary 4.7.** *Let R be a finite orthogonal stream TRS. If $t \to_R^\infty s$ and $t \to_R^\infty s'$ and $s, s' \in \mathcal{S}^+(\Sigma)$ then $|s| = |s'|$.*

▶ **Definition 4.8.** A *stream function* $F : (\Sigma^\omega)^n \to \Sigma^{\leq\omega}$ is *defined* by an n-ary stream function symbol f if for any $w_1, \ldots, w_n \in \Sigma^\omega$ and $s_1, \ldots, s_n \in \mathcal{S}(\Sigma)$ with $|s_i| = w_i$ we have $f(s_1, \ldots, s_n) \to_R^\infty s$ where $|s| = F(w_1, \ldots, w_n)$. A stream function is *definable* in a stream TRS if it is defined by one of its stream function symbols.

A stream TRS R is *data tail-recursive* if the data rules of R form a *single-sorted* (i.e. neither left- nor right-hand sides of data rules of R contain stream subterms) finite tail-recursive cons-free constructor TRS.

Note that if R is data tail-recursive then data terms do not contain stream subterms, because then neither data constructors nor data function symbols can have stream arguments. In particular, if $l \to t :: r$ is a rule in R, then t does not contain stream subterms.

▶ **Definition 4.9.** A *pure* stream TRS is a finite orthogonal stream TRS with simple stream rules, no data rules and no data constructors of arity > 0.

A stream TRS has *simple data* if there exists a unary data constructor $S : d \to d$ such that for every stream rule $l \to r \in R$, if t is a data subterm of r such that $\mathrm{Var}(t) \neq \emptyset$ then $t = S(t')$ or t is a variable.

A *simple* stream TRS is a finite orthogonal data tail-recursive stream TRS with simple stream rules and simple data.

▶ **Example 4.10.** Here is an example of a simple stream TRS, where x, x' are stream variables and y, y' are data variables.

$$\begin{array}{rcl} f(x) & \to & g(x, x, 0, 0) \\ g(y :: x, x', 0, y') & \to & y :: g(x', x', S(y'), S(y')) \\ g(0 :: x, x', S(y), y') & \to & g(x, x', y, y') \\ g(1 :: x, x', S(y), y') & \to & g(x, x', y', y') \end{array}$$

In this stream TRS the stream function symbol f defines a function $F : \Sigma^\omega \to \Sigma^{\leq\omega}$ such that $F(s)$ has in position n the first element of s following a block of n consecutive 0's.

The following simple stream TRS defines the Thue-Morse sequence T:

$$\begin{array}{rclrcl} T & \to & f(0) & f(x) & \to & h(x, x) :: f(S(x)) \\ h(0, 0) & \to & 0 & \tilde{h}(0, 0) & \to & 1 \\ h(0, x) & \to & h(x, x) & \tilde{h}(0, x) & \to & \tilde{h}(x, x) \\ h(S(0), S(x)) & \to & \tilde{h}(x, x) & \tilde{h}(S(0), S(x)) & \to & h(x, x) \\ h(S(S(x)), S(y)) & \to & h(x, y) & \tilde{h}(S(S(x)), S(y)) & \to & \tilde{h}(x, y) \end{array}$$

The n-th element T_n of T is defined by the recurrence:

$$T_0 = 0 \qquad T_{2n} = T_n \qquad T_{2n+1} = 1 - T_n$$

Identifying natural numbers with their representations in the TRS, it may be shown by induction on $\langle 2m - n, n\rangle$ ordered lexicographically that the data term $h(n, m)$ reduces to T_{2m-n} and $\tilde{h}(n, m)$ to $1 - T_{2m-n}$.

5 Finite Stream Transducers

In this section we characterise the classes of stream functions computable by (jumping) finite stream transducers. In short, pure stream TRSs characterise the class of stream functions computable by jumping finite transducers, and right-linear pure stream TRSs characterise the class of stream functions computable by finite transducers. We first recall the definitions of (jumping) finite transducers from [14, 13].

▶ **Definition 5.1.** An n-ary *jumping finite transducer* (JFT) over Σ-streams with m cursors is a tuple $\langle Q, q_0, C, \gamma, \delta\rangle$ where Q is a finite set of states, q_0 is the start state, $C = \{c_1, \ldots, c_m\}$ is the set of cursors, $\gamma : C \to \{1, \ldots, n\}$ is the initial cursor configuration, and

$$\delta : Q \times \Sigma^m \to Q \times (C \to C \cup \{+\}) \times (\Sigma \cup \{\epsilon\})$$

is the transition function. Intuitively, $\delta(q, \sigma_1, \ldots, \sigma_m)$ consists of the next state, an indication of cursor movement, and an optional output symbol. A cursor may either move forward or jump to the position of another cursor. In other words, an n-ary JFT is a finite automaton with n read-only input tapes and one write-only output tape, and m cursors which can move forward on the input tapes and jump to positions of other cursors, but cannot be compared.

A *finite transducer* (FT) is a JFT such that no cursor ever jumps to the position of another (except to itself, which is equivalent to not moving). A *configuration* of a JFT consists of a state and a function $\pi : C \to \{1, \ldots, n\} \times \mathbb{N}$ which assigns to each cursor c a stream index $i \in \{1, \ldots, n\}$ and a position in the stream. The *successor configuration* K' of a configuration K is determined in the obvious way by the transition function δ. The *initial configuration* is $\langle q_0, \pi_0 \rangle$ where $\pi_0(c) = \langle \gamma(c), 0 \rangle$ for $c \in C$. A *run* of a JFT $\langle Q, q_0, C, \gamma, \delta \rangle$ is an infinite sequence of configurations $K_0, K_1, K_2, \ldots$ such that K_0 is the initial configuration and K_{n+1} is the successor configuration of K_n for each $n \in \mathbb{N}$. The function $F : (\Sigma^\omega)^n \to \Sigma^{\leq \omega}$ *computed* by a given n-ary FT (JFT) is defined in an obvious way, with $F(w_1, \ldots, w_n)$ being the output of the transducer on inputs $w_1, \ldots, w_n$. The output may be finite, because the transducer may loop.

▶ **Theorem 5.2.** *An n-ary stream function is definable in a pure stream TRS with maximum function symbol arity m iff it is computable by an n-ary JFT with m cursors.*

Proof. Let $\langle Q, q_0, C, \gamma, \delta \rangle$ be an n-ary JFT with m cursors. Without loss of generality $C = \{1, \ldots, m\}$. In the TRS we have a stream function symbol $f_q : s^m \to s$ for each state $q \in Q$. There is also the "start" stream function symbol $g : s^n \to s$. We have the rules e.g.

$$f_q(\sigma_1 :: x_1, \ldots, \sigma_m :: x_m) \to \sigma :: f_{q'}(\sigma_{\rho(1)} :: x_{\rho(1)}, x_2, \sigma_{\rho(3)} :: x_{\rho(3)}, \ldots)$$

when $\delta(q, \sigma_1, \ldots, \sigma_m) = \langle q', \rho, \sigma \rangle$ and $\rho(1), \rho(3), \ldots \in C$ and $\rho(2) = +$. Intuitively, the arguments of f_q encode the m cursors. We also have the "start" rule:

$$g(x_1, \ldots, x_n) \to f_{q_0}(x_{\gamma(1)}, \ldots, x_{\gamma(n)}).$$

Note that all of the above rules are simple stream rules and the TRS is orthogonal, so it is a pure stream TRS. It is easy to see that for each $s_1, \ldots, s_n \in \mathcal{S}(\Sigma)$ there is a bijective correspondence between the infinite runs of the JFT on $|s_1|, \ldots, |s_n|$ and infinite reductions starting at $g(s_1, \ldots, s_n)$. This implies that the function defined by g is the same as the function computed by the JFT.

For the other direction, let R be a pure stream TRS with maximum function symbol arity m, and let the n-ary symbol g define a function $F : (\Sigma^\omega)^n \to \Sigma^{\leq \omega}$ where Σ is the set of data constants in R. We construct an n-ary JFT with m cursors.

Because there are no data rules or data constructors of arity > 0, each rule is a simple stream rule of the form e.g.

$$f(a :: u :: b :: x, a :: y, c) \to d :: g(u :: b :: x, e)$$

where $a, b, c, d, e \in \Sigma$, and u is a data variable. We will encode stream function symbols by (possibly many) states. Stream arguments will correspond to cursor positions.

Let N be the maximum size of the left-hand side l of a rule $l \to r \in R$. For a function symbol f with k stream and j data arguments, and words $w_1, \ldots, w_k \in \Sigma^N$, and constants $c_1, \ldots, c_j \in \Sigma$, we add a state $q_f^{w_1, \ldots, w_k, c_1, \ldots, c_j}$. The words $w_1, \ldots, w_k$ buffer the last N symbols read from each of the cursors. Let s_i be a stream term representing the word w_i, with a variable x_i at the tail, e.g., if $w_i = abc$ then $s_i = a :: b :: c :: x_i$. Without loss of generality assume the stream arguments of f occur before the data arguments. Because R is orthogonal, there is at most one rule $l \to r \in R$ such that l matches $f(s_1, \ldots, s_k, c_1, \ldots, c_j)$ with some substitution σ, i.e., $\sigma l = f(s_1, \ldots, s_k, c_1, \ldots, c_j)$. Note that because of the choice of N, if there is no rule $l \to r \in R$ with l matching $f(s_1, \ldots, s_k, c_1, \ldots, c_j)$, then no left-hand side of a rule unifies with $f(s_1, \ldots, s_k, c_1, \ldots, c_j)$. Assume e.g.

$$\sigma l = f(a :: b :: x, c :: d :: y, c_1)$$

and

$$\sigma r = d :: g(c :: d :: y, b :: x, d :: y, c).$$

Then in the state q_f^{ab,cd,c_1} the JFT outputs d and simultaneously sets the first cursor to the second one, the second to the first, and the third to the second. Then it reads one symbol from the second cursor and one from the third, moving them forward. Let the read symbols be a_1, a_2 respectively. The JFT then enters the state $q_g^{cd,ba_1,da_2,c}$. This behaviour may always be encoded using a finite number of states.

The JFT starts in a state q_0 with the i-th cursor initialised to the beginning of the i-th input tape, for $i = 1, \ldots, n$, and other cursors initialised arbitrarily. Then the JFT reads N symbols from each of the n input tapes, and reaches the state $q_g^{w_1,\ldots,w_n}$ where $w_i \in \Sigma^N$ is the word consisting of the symbols read from the i-th input tape.

We also add a "trash" state q_T and add appropriate transitions to q_T from other states to make δ a total function.

For any $s_1, \ldots, s_n \in \mathcal{S}(\Sigma)$ there is a bijective correspondence between the runs of the JFT on $|s_1|, \ldots, |s_n|$ and the infinite reductions starting at $g(s_1, \ldots, s_n)$, and the function computed by the JFT is the same as the function defined by g. ◀

▶ **Theorem 5.3.** *An n-ary stream function is definable in a right-linear pure stream TRS with maximum function symbol arity m iff it is computable by an n-ary FT with m cursors.*

Proof. An adaptation of the proof of Theorem 5.2. More details are in Appendix B. ◀

6 LOGSPACE for streams

In this section we show that stream functions definable in simple stream TRSs are exactly the stream functions computable in LOGSPACE as defined by Ramyaa and Leivant [14, 13]. First, we recall the definition of jumping Turing transducers from [14].

▶ **Definition 6.1.** A *jumping Turing transducer* (JTT) is defined analogously to a JFT, except that it has additional read-write work tapes with two-way cursors on them. The function computed by a JTT is defined in an obvious way. A JTT *operates in space* $f(n)$ if the computation for the first n output symbols does not involve work-tapes of length $> f(n)$. A stream function is computable in LOGSPACE if there is a JTT computing this function which operates in space $O(\log n)$.

Note that the space used by a JTT is defined in terms of the output. Time restrictions defined in terms of the output do not make much sense for JTTs, because even for FTs no restriction is placed on how long it takes to output the next symbol (e.g. consider an FT over binary streams skipping all zeros and copying all ones).

We will show that JTTs operating in LOGSPACE compute exactly the stream functions definable in simple stream TRSs. First, we generalise eager $R\bot$-reduction from Section 3 to stream TRSs.

▶ **Definition 6.2.** Let $\bot$ be a fresh nullary data constructor. We define the relation $\to_\bot$ by: $t \to_\bot \bot$ if t is a data term and it does not R-reduce to a constructor normal form. We set $\to_{R\bot} = \to_R \cup \to_\bot$. A finitary $R\bot$-reduction is *eager* if only innermost $R\bot$-redexes are contracted and priority is given to $\bot$-reduction. We denote one-step eager $R\bot$-reduction by $\to_{R\bot e}$. The relation $\to^\infty_{R\bot e}$ of *infinitary eager $R\bot$-reduction* is defined coinductively.

$$\frac{t \to^*_{R\bot e} t'}{t \to^\infty_{R\bot e} t'} \qquad \frac{t \to^*_{R\bot e} u :: w \quad w \to^\infty_{R\bot e} w'}{t \to^\infty_{R\bot e} u :: w'}$$

Because of space limits, the proofs of lemmas concerning infinitary eager $R\bot$-reduction are delegated to Appendix C.

In the rest of this section we assume R to be a simple stream TRS.

▶ **Definition 6.3.** A term is *proper* if all its data subterms are finite.

If t is proper and $t \to_R t'$ then t' is also proper, because R is finite.

▶ **Lemma 6.4.** *If t is proper and $t \to^\infty_R t_1$ and $t \to^\infty_{R\bot e} t_2$ then there is t' with $t_2 \to^\infty_R t'$ and $t_1 \to^\infty_{R\bot e} t_2$.*

▶ **Lemma 6.5.** *If $s \in \mathcal{S}^+(\Sigma)$ and $s \to^\infty_{R\bot e} s'$ (resp. $s \to^\infty_R s'$), then $s \sim s'$ and $s' \in \mathcal{S}^+(\Sigma)$.*

▶ **Theorem 6.6.** *If a stream function is definable in a simple stream TRS then it is computable in LOGSPACE.*

Proof. Let $F : (\Sigma^\omega)^n \to \Sigma^{\leq\omega}$ be a function defined by an n-ary stream function symbol f_0 in a simple stream TRS R, i.e., a finite orthogonal data tail-recursive stream TRS with simple stream rules and simple data. We describe how to construct a JTT operating in LOGSPACE which computes F.

For $s_1, \ldots, s_n \in \mathcal{S}(\Sigma)$ we have $f_0(s_1, \ldots, s_n) \to^\infty_R s \in \mathcal{S}^+(\Sigma)$ where $F(|s_1|, \ldots, |s_n|) = |s|$. The constructed JTT will essentially compute an $s' \in \mathcal{S}^+(\Sigma)$ such that $f_0(s_1, \ldots, s_n) \to^\infty_{R\bot e} s'$, for a certain fixed infinitary eager $R\bot$-reduction. By Lemma 6.4 and Lemma 6.5 we then have $|s| = |s'|$.

Note that because the TRS is finite and has simple data, all constructor normal form data terms occurring in any reduction $f_0(s_1, \ldots, s_n) \to^\infty_{R\bot e} s$ have the form $S^m(t)$ where either $t \in \Sigma$ or it is one of the finitely many constructor normal form data terms occurring in the right-hand sides of the stream or data rules. Because S cannot occur in the right-hand side of a simple stream rule if no stream element is produced, and data rules are cons-free, m is at most proportional to the number of output stream elements already produced. Hence $S^m(t)$ may be represented in logspace, using a logarithmic counter for m and a constant number of bits to represent t. Because the reduction is eager and the size of right-hand sides of stream rules is bounded by a constant, using an analogon of Proposition 3.17 we obtain a JTT which computes in logarithmic space the constructor normal form of a given data term occurring in the reduction, if it has one. This JTT computes the constructor normal forms "inside-out". For a term $f(t_1, \ldots, t_k)$ first the constructor normal forms $t'_1, \ldots, t'_k$ are computed. Each t'_i has the form $S^{m_i}(u'_i)$ where u'_i is either $\bot$ or one of the finitely many constructor normal forms occurring in the right-hand sides of the rules. Then using (an analogon of) Proposition 3.17 we compute the constructor normal form of $f(t'_1, \ldots, t'_k)$. For $S(t)$ first the constructor normal form $S^m(t')$ of t is computed using Proposition 3.17, and then $S^{m+1}(t')$ is returned as the constructor normal form of t. Note that the only property the constructor normal forms needed in Proposition 3.17 is that they can be represented using a logarithmic number of bits, and given a representation of $S(t)$ the representation of t may be computed in logarithmic space.

We construct the JTT like in Theorem 5.2, except that now the data arguments are stored in memory instead of the state. We compute constructor normal forms of data terms using Proposition 3.17. This is done eagerly, before transitioning to the state associated with the stream function symbol in the right-hand side, which ensures that the size of the "prefix" containing all defined function symbols of each data term occurring in the reduction is constant – it is bounded by the size of the right-hand side of a rule in R. More details are in Appendix C. ◀

▶ **Theorem 6.7.** *If a stream function is computable in LOGSPACE then it is definable in a simple stream TRS.*

Proof. Let $F : (\Sigma^\omega)^n \to \Sigma^{\leq\omega}$ be a function computed by a JTT operating in LOGSPACE. As shown in [14, Proposition 2.4], the function F is also computed by a JFT with a local counter, i.e., a JFT with an additional input tape which contains 1^n when computing the n-th output symbol. In other words, a 1 is appended to the local counter whenever a symbol is output by the JFT. Initially, the local counter contains the empty word. The JFT has a fixed number of cursors on the local counter, which are reset to the beginning of the local counter tape whenever a symbol is output. As with the cursors on the input, the cursors on the local counter may move to the right or jump to other cursors. Hence, they may be encoded in an analogous way as the cursors on the input stream.

A simple stream TRS defining a function computed by a JFT with a local counter may be constructed in a way analogous to the construction of a pure stream TRS in the proof of Theorem 5.2. The difference is that now every function symbol f_q corresponding to a state q has an additional data argument representing the local counter, and data arguments encoding the cursors on the local counter. The local counter contents 1^n is represented by the data term $S^n(0)$, where $S : d \to d$ and $0 : d$. If a rule associated with f_q produces a new output symbol, then in the right-hand side of the rule the local counter is "increased" by prepending S, and the data arguments encoding cursors on the local counter are set to the local counter. This may be encoded in a simple stream rule. The resulting stream TRS has simple data.

Note that the constructed simple stream TRS actually has no data rules. It is not a pure stream TRS because it has a unary data constructor S. ◀

▶ **Corollary 6.8.** *A stream function is computable in LOGSPACE iff it is definable in a simple stream TRS.*

7 Conclusions

We have shown an infinitary rewriting characterisation of LOGSPACE-computable stream functions as defined by Ramyaa and Leivant. In the realm of finite data, we proved that finite orthogonal tail-recursive cons-free constructor TRSs characterise LOGSPACE.

Our proof could probably be adapted to show that finite semi-linear [10] tail-recursive cons-free constructor TRSs characterise NLOGSPACE. In the nondeterministic case the trick with logarithmic counters is not necessary as the procedure may simply guess when to contract a subterm to $\bot$. Semi-linearity ensures that subterms containing redexes cannot get duplicated, which is crucial to show that a constructor normal form may always be reached via an eager $R\bot$-reduction.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1999.
2. G. Bonfante. Some programming languages for LOGSPACE and PTIME. In *AMAST 2006*, pages 66–80, 2006.
3. D. de Carvalho and J. G. Simonsen. An implicit characterization of the polynomial-time decidable sets by cons-free rewriting. In *RTA-TLCA 2014*, pages 179–193, 2014.
4. Jörg Endrullis, Helle Hvid Hansen, Dimitri Hendriks, Andrew Polonsky, and Alexandra Silva. A coinductive framework for infinitary rewriting and equational reasoning. In *RTA 2015*, 2015.
5. Jörg Endrullis and Andrew Polonsky. Infinitary rewriting coinductively. In *TYPES 2011*, pages 16–27, 2011.
6. Juris Hartmanis. On non-determinancy in simple computing devices. *Acta Informatica*, 1(4):336–344, Dec 1972.
7. N. D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theor. Comput. Sci.*, 228(1-2):151–174, 1999.
8. N. D. Jones. The expressive power of higher-order types or, life without CONS. *J. Funct. Program.*, 11(1):5–94, 2001.
9. Neil D. Jones. *Computability and complexity - from a programming perspective.* Foundations of computing series. MIT Press, 1997.
10. C. Kop. On first-order cons-free term rewriting and PTIME. In *DICE 2016*, 2016.
11. C. Kop and J. G. Simonsen. Complexity hierarchies and higher-order cons-free rewriting. In *FSCD 2016*, pages 23:1–23:18, 2016.
12. C. Kop and J. G. Simonsen. The power of non-determinism in higher-order implicit complexity. In *ESOP 2017*, pages 668–695, 2017.
13. D. Leivant and R. Ramyaa. The computational contents of ramified corecurrence. In *FoSSaCS 2015*, pages 422–435, 2015.
14. R. Ramyaa and D. Leivant. Ramified corecurrence and logspace. *Electr. Notes Theor. Comput. Sci.*, 276:247–261, 2011.

A Confluence of infinitary reduction

▶ **Lemma A.1.** *If* $t \to_R^\infty t' \to_R^* t''$ *then* $t \to_R^\infty t''$.

Proof. By coinduction. If $t \to_R^* t'$ then this is obvious. Otherwise $t' = u :: w'$ and $t \to_R^* u :: w$ and $w \to_R^\infty w'$ and $t'' = u'' :: w''$ and $u \to_R^* u''$ and $w' \to_R^* w''$. Then $t \to_R^* u'' :: w$. By coinduction also $w \to_R^\infty w''$. Hence $t \to_R^\infty u'' :: w'' = t''$. ◀

▶ **Lemma A.2.** *If* $t \to_R^\infty t' \to_R^\infty t''$ *then* $t \to_R^\infty t''$.

Proof. By coinduction, using Lemma A.1. ◀

▶ **Lemma A.3.** *Let R be finite and orthogonal. If* $t \to_R^\infty t'$ *and* $t \to_R^* s$ *then there is* s' *with* $s \to_R^\infty s'$ *and* $t' \to_R^\infty s'$.

Proof. By coinduction, analysing $t \to_R^\infty t'$. If $t \to_R^* t'$ then this follows from Lemma 4.4. Otherwise $t' = u :: w'$, and $t \to_R^* u :: w$ and $w \to_R^\infty w'$. By Lemma 4.4 there are u_1, w_1 such that $s \to_R^* u_1 :: w_1$ and $u \to_R^* u_1$ and $w \to_R^* w_1$. By coinduction we obtain w_2 with $w_1 \to_R^\infty w_2$ and $w' \to_R^\infty w_2$. Hence $s \to_R^\infty u_1 :: w_2$, because $s \to_R^* u_1 :: w_1$ and $w_1 \to_R^\infty w_2$; and $t' \to_R^\infty u_1 :: w_2$, because $t' = u :: w' \to_R^* u_1 :: w'$ and $w' \to_R^\infty w_2$. ◀

Note that $t' \to_R^* s'$ would not suffice in the conclusion of the above lemma, because the infinitary reduction $t \to_R^\infty t'$ may create in t' infinitely many descendants of a redex in t.

The relation $\to_R^{2\infty}$ is defined coinductively.

$$\frac{t \to_R^\infty t'}{t \to_R^{2\infty} t'} \qquad \frac{t \to_R^\infty u :: w \quad w \to_R^{2\infty} w'}{t \to_R^{2\infty} u :: w'}$$

▶ **Lemma A.4.** *If* $t \to_R^\infty t' \to_R^{2\infty} t''$ *then* $t \to_R^{2\infty} t''$.

Proof. Follows directly from Lemma A.2 ◀

▶ **Lemma A.5.** *If* $t \to_R^{2\infty} t'$ *then* $t \to_R^\infty t'$.

Proof. By coinduction, using Lemma A.4. ◀

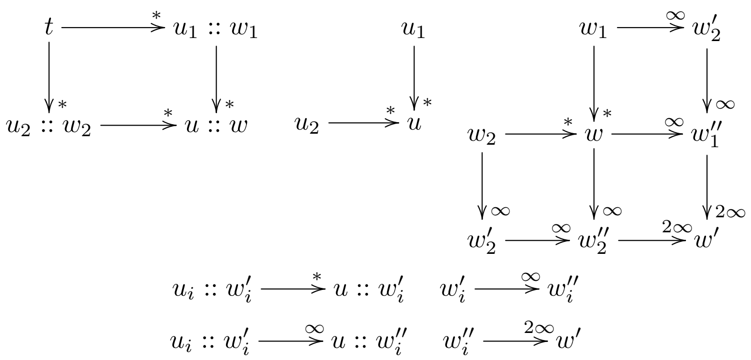

Figure 1 Proof of confluence of infinitary reduction

▶ **Theorem 4.5.** *If R is finite and orthogonal then* $\to_R^\infty$ *is confluent, i.e., if* $t \to_R^\infty t_1$ *and* $t \to_R^\infty t_2$ *then there exists* t' *such that* $t_1 \to_R^\infty t'$ *and* $t_2 \to_R^\infty t'$.

Proof. By coinduction we construct t' such that $t_1 \to_R^{2\infty} t'$ and $t_2 \to_R^{2\infty} t'$. This suffices by Lemma A.5. If $t \to_R^* t_1$ or $t \to_R^* t_2$ then the claim follows from Lemma A.3. Otherwise, $t_i = u_i :: w_i'$ and $t \to_R^* u_i :: w_i$ and $w_i \to_R^\infty w_i'$ for $i = 1, 2$. By Lemma 4.4 there are u, w such that $u_i \to_R^* u$ and $w_i \to_R^* w$. By Lemma A.3 there are w_1'', w_2'' such that $w_i' \to_R^\infty w_i''$ and $w \to_R^\infty w_i''$. Hence $t_i = u_i :: w_i' \to_R^\infty u :: w_i''$. By coinduction we obtain w' with $w_i'' \to_R^{2\infty} w'$. Thus $t_i \to_R^{2\infty} u :: w'$, so we may take $t' = u :: w'$. See Figure 1. ◀

B Characterisation of Finite Stream Transducers

▶ **Theorem 5.3.** *An n-ary stream function is definable in a right-linear pure stream TRS with maximum function symbol arity m iff it is computable by an n-ary FT with m cursors.*

Proof. First note that for an FT the construction of a stream TRS in the proof of Theorem 5.2 gives a right-linear system. Conversely, if the TRS is right-linear, then we may modify the construction of a JFT in the proof of Theorem 5.2 to obtain an FT, by keeping in the state the information which cursor a given function argument corresponds to. So a state corresponding to a function symbol f is now $q_f^{w_1,\ldots,w_k,c_1,\ldots,c_j,i_1,\ldots,i_k}$ where $i_1, \ldots, i_k$ indicate the cursors corresponding to the stream arguments of f. For instance, if

$$\sigma l = f(a :: b :: x, c :: d :: y, c_1)$$

and

$$\sigma r = d :: h(c :: d :: y, b :: x, c)$$

then the transition from the state q_f^{ab,cd,c_1,i_1,i_2} is constructed as follows. First, output d and read one symbol e from the i_1-th cursor moving it forward. Then change the state to q_h^{cd,be,c,i_2,i_1}. ◀

C Proofs for Section 6

In this section we assume that R is a simple stream TRS.

▶ **Lemma C.1.** *If t is proper and $t \to_R^\infty t_1$ and $t \to_{R\perp e}^* t_2$ then there is t' with $t_2 \to_R^\infty t'$ and $t_1 \to_{R\perp e}^\infty t_2$.*

Proof. By coinduction, analysing $t \to_R^\infty t_1$. If $t \to_R^* t_1$ then this follows from Corollary 3.15. Otherwise $t \to_R^* u :: w$ and $w \to_R^\infty w'$ and $t_1 = u :: w'$. By Corollary 3.15 there are u_2, w_2 with $t_2 \to_R^* u_2 :: w_2$ and $u \to_{R\perp e}^* u_2$ and $w \to_{R\perp e}^* w_2$. Note that w is proper. By coinduction we obtain w_2' with $w_2 \to_R^\infty w_2'$ and $w' \to_{R\perp e}^\infty w_2'$. Take $t' = u_2 :: w_2'$. We have $t_2 \to_R^* u_2 :: w_2$ and $w_2 \to_R^\infty w_2'$, so $t_2 \to_R^\infty t'$. Also $t_1 = u :: w' \to_{R\perp e}^* u_2 :: w'$ and $w' \to_{R\perp e}^\infty w_2'$, so $t_1 \to_{R\perp e}^\infty t'$. ◀

▶ **Lemma 6.4.** *If t is proper and $t \to_R^\infty t_1$ and $t \to_{R\perp e}^\infty t_2$ then there is t' with $t_2 \to_R^\infty t'$ and $t_1 \to_{R\perp e}^\infty t_2$.*

Proof. By coinduction, analysing $t \to_{R\perp e}^\infty t_2$. If $t \to_{R\perp e}^* t_2$ then this is a consequence of Lemma C.1. Otherwise $t \to_{R\perp e}^* u :: w$ and $w \to_{R\perp e}^\infty w'$ and $t_2 = u :: w'$. By Lemma C.1 there are u_1, w_1 such that $t_1 \to_{R\perp e}^* u_1 :: w_1$ and $u \to_R^* u_1$ and $w \to_R^\infty w_1$. Note that w is proper. By coinduction we obtain w_2 such that $w' \to_R^\infty w_2$ and $w_1 \to_{R\perp e}^\infty w_2$. Take $t' = u_1 :: w_2$. We have $t_1 \to_{R\perp e}^* u_1 :: w_1$ and $w_1 \to_{R\perp e}^\infty w_2$, so $t_1 \to_{R\perp e}^\infty t'$. Also $t_2 = u :: w' \to_R^* u_1 :: w'$ and $w' \to_R^\infty w_2$, so $t_2 \to_R^\infty t'$. ◀

▶ **Lemma C.2.** *If $t \to^*_{R\bot} u :: w$ then t has a chnf (in R).*

Proof. Induction on the number of $\bot$-reduction steps in $t \to^*_{R\bot} u :: w$. If there are none then $t \to^*_R u :: w$. Otherwise by the inductive hypothesis $t \to^*_R t' \to_\bot t'' \to^*_R u' :: w'$. Because R is finite, by the same argument as in the proof of Lemma 3.4 we conclude that $t \to^*_R t' \to^*_R u'' :: w'' \to^*_\bot u' :: w'$. ◀

▶ **Lemma 6.5.** *If $s \in \mathcal{S}^+(\Sigma)$ and $s \to^\infty_{R\bot e} s'$ (resp. $s \to^\infty_R s'$), then $s \sim s'$ and $s' \in \mathcal{S}^+(\Sigma)$.*

Proof. It suffices to notice that if t is a stream term without a chnf and $t \to^\infty_{R\bot e} t'$ (resp. $t \to^\infty_R t'$) then t' does not have a chnf either. This follows from Lemma C.2 (resp. Lemma 4.6). ◀

▶ **Theorem 6.6.** *If a stream function is definable in a simple stream TRS then it is computable in LOGSPACE.*

Proof. We describe in more detail the construction of a JTT already sketched in Section 6.

The constructed JTT computes the stream $c_1 :: c_2 :: c_3 :: \ldots$ where e.g.

$$\begin{aligned}&f_0(s_1,\ldots,s_n) \to^\epsilon_R t_1 :: f_1(w^1_1,\ldots,w^1_{k_1}) \to^*_{R\bot e} c_1 :: f_1(u^1_1,\ldots,u^1_{k_1}) \to^\epsilon_R\\ &c_1 :: t_2 :: t_3 :: f_2(w^2_1,\ldots,w^2_{k_2}) \to^*_{R\bot e} c_1 :: c_2 :: c_3 :: f_2(u^1_1,\ldots,u^1_{k_2}) \to^\epsilon_R \ldots\end{aligned}$$

and none of the u^j_i contain $R\bot$-redexes. So all of the root R-reduction steps are in fact eager $R\bot$-reductions. Note that all terms appearing in this reduction are proper.

Let N be the maximum size of the left-hand side l of a rule $l \to r \in R$. For a stream function symbol f with k stream arguments, and words $w_1,\ldots,w_k \in \Sigma^N$ we add a state $q_f^{w_1,\ldots,w_k}$. Let s_i be a stream term representing the word w_i, with a variable x_i at the tail, like in the proof of Theorem 5.2. Assume without loss of generality that the stream arguments of f occur before the data arguments, and let $y_1,\ldots,y_j$ be data variables. Let $l_1 \to r_1,\ldots,l_n \to r_n \in R$ be all rules such that $f(s_1,\ldots,s_k,y_1,\ldots,y_j)$ unifies with l_i with substitution σ_i. Let M be the maximum number of data arguments of any defined stream function symbol in R. We keep the representations of data arguments in constructor normal form on M separate work tapes: we call them argument work tapes.

Assume e.g. $k = 2$ and $w_1 = ab$ and $w_2 = cd$ and $j = 2$. In the state $q_f^{ab,cd}$ the JTT first checks which of the left-hand sides $l_1,\ldots,l_n$ matches $f(a :: b :: x_1, c :: d :: x_2, u_1, u_2)$ where u is the first data argument – the data term whose representation is stored on the first argument work tape. There is at most one matching l_i because R is orthogonal, and this can be checked using only logarithmic space (it suffices to check whether the two data arguments in l_i match u_1, u_2, respectively). If none of the l_i matches then the JTT loops. Assume e.g. l_i matches with substitution σ and

$$\sigma l_i = f(a :: b :: x_1, c :: d :: x_2, S(y), z)$$

and

$$\sigma r_i = h_1(a,b,y,z) :: g(c :: d :: x_2, b :: x_1, d :: x_2, h_2(y), y).$$

Then the JTT outputs the constructor normal form of $h_1(a,b,t_1,t_2)$, computed using Proposition 3.17, where $S(t_1)$ is the constructor normal form of the first data argument, stored on the first argument work tape, and t_2 is the constructor normal form of the second data argument, sored on the second argument work tape. If the constructor normal form of $h_1(a,b,t_1,t_2)$ is not in Σ, then the JTT loops. Next, the JTT simultaneously sets the first cursor to the second one, the second to the first, and the third to the second. Then it computes the constructor normal form of $h_2(t)$, using Proposition 3.17, and writes it to the first argument tape, and also copies t to the second argument tape. Next, the JTT reads one symbol from the second cursor and one from the third, moving them forward. Let these symbols be a_1, a_2 respectively. The JTT then enters the state q_g^{cd,ba_1,da_2}. This behaviour may always be encoded using a finite number of states.

The rest of the construction is like in the proof of Theorem 5.2.

It is clear that the constructed JTT computes a stream $|s'| \in \Sigma^{\leq\omega}$ for an $s' \in \mathcal{S}^+(\Sigma)$ such that $f_0(s_1, \ldots, s_n) \to^{\infty}_{R\bot e} s'$. As mentioned before, Lemma 6.4 and Lemma 6.5 imply that this is correct. Indeed, we have $f_0(s_1, \ldots, s_n) \to^{\infty}_{R} s$ where $F(|s_1|, \ldots, |s_n|) = |s|$. By Lemma 6.4 there is w with $s \to^{\infty}_{R\bot e} w$ and $s' \to^{\infty}_{R} w$. By Lemma 6.5 we have $w \in \mathcal{S}^+(\Sigma)$ and $s \sim w$ and $s' \sim w$. Thus $|s| = |w| = |s'|$. So the JTT computes the stream $|s|$, as required. ◀