Term rewriting characterisation of LOGSPACE for finite and infinite data*

Łukasz Czajka[1]

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. This result is non-trivial, because in contrast to previous work on characterising LOGSPACE by tail-recursive cons-free programs we do not impose any fixed evaluation strategy. We provide a LOGSPACE algorithm which computes constructor normal forms. We then use this algorithm 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.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

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

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

In cons-free programs [4] 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 has been used to characterise a variety of complexity classes [1, 2, 3, 4, 5, 6].

In this paper we extend the cons-free approach to computation on infinite streams. In [8, 7] Ramyaa and Leivant define the class of LOGSPACE-computable stream functions and show that it is characterised by ramified corecurrence in two tiers. We provide a characterisation of LOGSPACE-computable stream functions by means of infinitary rewriting. 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.

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 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 works [1, 4] LOGSPACE was characterised by tail-recursive cons-free programs, but in contrast to the present paper only a fixed call-by-value evaluation strategy was considered. We generalise this to the setting of term rewriting where no fixed reduction strategy is required. In [2, 5] it has been shown that first-order semi-linear cons-free TRSs characterise PTIME. This result is similar to ours in that it requires no fixed reduction strategy. In particular, our method of introducing $\bot$-reductions is similar to [5].

* This work was supported by ....

This paper is an extended abstract only stating some of the obtained results. A version of this work with the proofs is available at `www.mimuw.edu.pl/~lukaszcz/logspace.pdf`.

2 Term rewriting systems

▶ **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 occuring 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.

A *defined symbol* in a TRS R is a 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 symbol which is neither a defined symbol in R nor a variable. 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 constructor normal form $\bar{w}$ 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

We show that 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.

▶ **Definition 3.1.** A constructor TRS R is *cons-free* if for each $l \to r \in R$ every chnf subterm of r occurs in l. 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.

▶ **Lemma 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 [3, 1]. ◀

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. The non-triviality comes from the fact that we do not impose any fixed

evaluation strategy. 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 $n \in \mathbb{N}$.

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 $\bot$ (a fresh constant) 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

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

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 $f_i(t_1^i,\dots,t_{n_i}^i)$ 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. Rigorous proofs may be found in the long version of this paper. Here we only state the definitions and the result.

▶ **Definition 3.3.** Let R be a constructor TRS and let $\bot$ be a fresh constant, i.e., not occuring 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.

▶ **Theorem 3.4.** *Let R be a finite orthogonal tail-recursive cons-free constructor TRS. There is a LOGSPACE algorithm which given a term $t = f(t_1,\dots,t_n)$, with $t_1,\dots,t_n$ in constructor normal form not containing $\bot$, computes the constructor normal form of t in R if it has one, or rejects if it doesn't.*

▶ **Corollary 3.5.** *A decision problem is in LOGSPACE iff it is accepted by a finite orthogonal tail-recursive cons-free constructor TRS.*

The main difference between our results and those of e.g. [1] is that here *no evaluation strategy is assumed*. We do use a specific strategy for $R\bot$-reduction (which includes both R-reduction *and* $\bot$-reduction) in the proof of Theorem 3.4 as a technical device, but the statement of the theorem refers to reduction in R. The proof essentially shows that the eager $R\bot$-reduction strategy is computable and normalizing, and that it gives the same constructor normal forms for terms which have a constructor normal form in R.

4 Stream Term Rewriting Systems

▶ **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 $\mathrm{cons}'\, t_1\, t_2$.

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. $\tau_1 \times \ldots \times \tau_n \to d$), where $\tau_i \in \{s, d\}$.

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, occurs in $u_1, \ldots, u_n$.

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.

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

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

Here are some non-examples:

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

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

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

▶ **Theorem 4.4.** *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 word in $\Sigma^\omega \cup \Sigma^*$ may be represented as a possibly infinite stream term. For a term t by $|t|$ we denote the corresponding finite or infinite word in $\Sigma^\omega \cup \Sigma^*$.

▶ **Definition 4.5.** A *stream function* $F : (\Sigma^\omega)^n \to \Sigma^\omega \cup \Sigma^*$ is *defined* by an n-ary stream function symbol f if for any $w_1, \ldots, w_n \in \Sigma^\omega$ and $s_1, \ldots, s_n$ with $|s_i| = w_i$ we have $f(s_1, \ldots, s_n) \to_R^\infty s$ with $|s| = F(w_1, \ldots, w_n)$. 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* finite tail-recursive cons-free constructor TRS. A *simple* stream TRS is a finite orthogonal data tail-recursive stream TRS with simple stream rules and 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.

5 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 [8, 7].

▶ **Definition 5.1.** An n-ary *jumping Turing transducer* (JTT) is a finite jumping state transducer with additional read-write work tapes. It has a finite number of states, a read-only input tape, a write-only output tape, a finite number of read-write work tapes, a finite number of cursors on the input tape and the work tapes, and a single cursor on the output tape. The cursors on the input tape may be moved in both directions or set to the position of another input cursor, but they cannot be compared. The cursor on the output tape may be moved only to the right. The cursors on the work tapes may be moved in both directions.

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 $O(\log n)$.

▶ **Theorem 5.2.** *A stream function is definable in a simple stream TRS iff it is computable in LOGSPACE, as defined by Ramyaa and Leivant.*

Proof. In one direction, we encode any JFT with a local counter [8, Proposition 2.4] in a simple stream TRS, encoding the states as stream function symbols, and the counter as a data term. In the other direction, we construct a JTT operating in LOGSPACE which computes the function defined by the stream TRS, using the memory to store a representation of the data terms and the algorithm from Theorem 3.4 to compute constructor normal forms of data terms. ◀

References

1 G. Bonfante. Some programming languages for LOGSPACE and PTIME. In *AMAST 2006*, pages 66–80, 2006.

2 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.

3 N. D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theor. Comput. Sci.*, 228(1-2):151–174, 1999.

4 N. D. Jones. The expressive power of higher-order types or, life without CONS. *J. Funct. Program.*, 11(1):5–94, 2001.

5 C. Kop. On first-order cons-free term rewriting and PTIME. In *DICE 2016*, 2016.

6 C. Kop and J. G. Simonsen. Complexity hierarchies and higher-order cons-free rewriting. In *FSCD 2016*, pages 23:1–23:18, 2016.

7 D. Leivant and R. Ramyaa. The computational contents of ramified corecurrence. In *FoSSaCS 2015*, pages 422–435, 2015.

8 R. Ramyaa and D. Leivant. Ramified corecurrence and logspace. *Electr. Notes Theor. Comput. Sci.*, 276:247–261, 2011.