Term rewriting characterisation of LOGSPACE for finite and infinite data

Łukasz Czajka, University of Copenhagen

July 11, 2018

Cons-free programs

Neil Jones:

- introduced a minimal functional programming language with lists,

Cons-free programs

Neil Jones:

- introduced a minimal functional programming language with lists,
- defined cons-free programs: cannot generate new data,

Cons-free programs

Neil Jones:

- introduced a minimal functional programming language with lists,
- defined cons-free programs: cannot generate new data,
- showed a characterisation of LOGSPACE by tail-recursive cons-free programs.

Cons-free rewriting

Here we extend the cons-free approach to:

- term rewriting systems (TRSs), showing that finite orthogonal tail-recursive cons-tree constructor TRSs characterise LOGSPACE,

Cons-free rewriting

Here we extend the cons-free approach to:

- term rewriting systems (TRSs), showing that finite orthogonal tail-recursive cons-tree constructor TRSs characterise LOGSPACE, (extension to TRSs for other complexity classes studied earlier by e.g. Kop, Simonsen, Carvalho)

Cons-free rewriting

Here we extend the cons-free approach to:

- term rewriting systems (TRSs), showing that finite orthogonal tail-recursive cons-tree constructor TRSs characterise LOGSPACE, (extension to TRSs for other complexity classes studied earlier by e.g. Kop, Simonsen, Carvalho)
- infinitary reductions, showing that simple stream TRSs characterise logarithmic space computation on streams, as defined by Ramyaa and Leivant.

Term Rewriting Systems

- 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 $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$.

Term Rewriting Systems

- 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 $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$.
- 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.

Term Rewriting Systems

- 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 $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$.
- 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 not a defined symbol in R.

Term Rewriting Systems

- 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 $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$.
- 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 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).

Term Rewriting Systems

- 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 $\mathrm{Var}(r) \subseteq \mathrm{Var}(l)$.
- 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 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 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.

Term Rewriting Systems – Example

$$\begin{array}{rcl} f(c(x)) & \rightarrow & g(x) \\ g(x, c(y)) & \rightarrow & c(f(g(x, y))) \end{array}$$

Term Rewriting Systems

Cons-free:

- right-hand sides may contain constructor normal forms as subterms,

Term Rewriting Systems

Cons-free:

- right-hand sides may contain constructor normal forms as subterms,
- right-hand sides may contain subterms of their corresponding left-hand side,

Term Rewriting Systems

Cons-free:

- right-hand sides may contain constructor normal forms as subterms,
- right-hand sides may contain subterms of their corresponding left-hand side,
- right-hand sides cannot contain any other occurrences of constructors.

Term Rewriting Systems

Cons-free:

- right-hand sides may contain constructor normal forms as subterms,
- right-hand sides may contain subterms of their corresponding left-hand side,
- right-hand sides cannot contain any other occurrences of constructors.

Example:

$$\begin{array}{rcl} f(c(x)) & \to & g(c(x)) \\ g(x, c(y)) & \to & f(g(y, c(0))) \end{array}$$

Term Rewriting Systems

Cons-free:

- right-hand sides may contain constructor normal forms as subterms,
- right-hand sides may contain subterms of their corresponding left-hand side,
- right-hand sides cannot contain any other occurrences of constructors.

Example:

$$\begin{array}{rcl} f(c(x)) & \to & g(c(x)) \\ g(x, c(y)) & \to & f(g(y, c(0))) \end{array}$$

Non-example:

$$\begin{array}{rcl} f(c(x)) & \to & c(g(x)) \\ g(x, c(y)) & \to & f(g(y, c(x))) \end{array}$$

Term Rewriting Systems

Tail-recursive: 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 $f > g$.

Term Rewriting Systems

Tail-recursive: 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 $f > g$.

Example:

$$\begin{array}{rcl} f(c(x)) & \to & g(c(x)) \\ g(x, c(y)) & \to & f(h(x, y)) \\ h(x, 0) & \to & x \\ h(x, c(y)) & \to & h(y, x) \end{array}$$

Term Rewriting Systems

Tail-recursive: 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 $f > g$.

Example:

$$\begin{array}{rcl} f(c(x)) & \to & g(c(x)) \\ g(x, c(y)) & \to & f(h(x, y)) \\ h(x, 0) & \to & x \\ h(x, c(y)) & \to & h(y, x) \end{array}$$

Non-example:

$$\begin{array}{rcl} f(c(x)) & \to & g(c(x)) \\ g(x, c(y)) & \to & f(g(y, c(0))) \end{array}$$

Term Rewriting Systems – Acceptance

Assuming the signature contains the constants $0, 1, \mathtt{nil}$ and a binary constructor symbol `cons`, every $w \in \{0, 1\}^*$ may be represented by a term $\bar{w}$ in an obvious way.

Term Rewriting Systems – Acceptance

Assuming the signature contains the constants $0, 1, \texttt{nil}$ 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 $A \subseteq \{0,1\}^*$ if there is a function symbol f such that for every $w \in \{0,1\}^*$ we have: $f(\bar{w}) \to_R^* 1$ iff $w \in A$.

Characterisation of LOGSPACE

Theorem

A problem is decidable in LOGSPACE iff it is accepted by a finite orthogonal tail-recursive cons-free constructor TRS.

Characterisation of LOGSPACE

Theorem

A problem is decidable in LOGSPACE iff it is accepted by a finite orthogonal tail-recursive cons-free constructor TRS.

($\Rightarrow$) follows directly from previous work,

Characterisation of LOGSPACE

Theorem

A problem is decidable in LOGSPACE iff it is accepted by a finite orthogonal tail-recursive cons-free constructor TRS.

($\Rightarrow$) follows directly from previous work,

($\Leftarrow$) more difficult, because the TRS may be not terminating and arbitrarily large reducts are possible:

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

Then $h(f(a)) \to_R a$ but also $h(f(a)) \to_R^* h(f(g^n(a)))$ for any $n \in \mathbb{N}$.

Characterisation of LOGSPACE

Idea: show that a constructor normal form may always be reached by an eager $R\bot$-reduction, denoted $\rightarrow^*_{R\bot e}$, 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. Then show that eager $R\bot$-reduction is computable in LOGSPACE.

Eager $R\bot$-reduction – Example

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

- Eager $R\bot$-reduction: $h(f(a)) \to_\bot h(\bot) \to_R a$.

Eager $R\bot$-reduction – Example

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

- Eager $R\bot$-reduction: $h(f(a)) \to_\bot h(\bot) \to_R a$.
- *Not* an eager $R\bot$-reduction: $h(f(a)) \to_R h(f^2(a))$.

Eager $R\bot$-reduction – Example

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

- Eager $R\bot$-reduction: $h(f(a)) \to_\bot h(\bot) \to_R a$.
- *Not* an eager $R\bot$-reduction: $h(f(a)) \to_R h(f^2(a))$.

$f(a)$ 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 $\bot$.

Computing eager $R\perp$-reduction

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

- t_i^j is the constructor normal form w.r.t. eager $R\perp$-reduction of w_i^j ($\perp$ is considered to be a constructor),
- $f_i \gtrsim f_j$ for $i \leq j$.

Computing eager $R\bot$-reduction

$$f_1(w^1_1,\ldots,w^1_{n_1}) \to^*_{R\bot e} f_1(t^1_1,\ldots,t^1_{n_1}) \to^{\epsilon}_R$$
$$f_2(w^2_1,\ldots,w^2_{n_2}) \to^*_{R\bot e} f_2(t^2_1,\ldots,t^2_{n_2}) \to^{\epsilon}_R \ldots$$

- t^j_i is the constructor normal form w.r.t. eager $R\bot$-reduction of w^j_i ($\bot$ is considered to be a constructor),
- $f_i \gtrsim f_j$ for $i \leq j$.

At some point either we reach a constructor normal form or a term $f_i(t^i_1,\ldots,t^i_{n_i})$ repeats.

Computing eager $R\bot$-reduction

$$
\begin{array}{l}
f_1(w^1_1, \ldots, w^1_{n_1}) \rightarrow^*_{R\bot e} f_1(t^1_1, \ldots, t^1_{n_1}) \rightarrow^\epsilon_R \\
f_2(w^2_1, \ldots, w^2_{n_2}) \rightarrow^*_{R\bot e} f_2(t^2_1, \ldots, t^2_{n_2}) \rightarrow^\epsilon_R \ldots
\end{array}
$$

- t^j_i is the constructor normal form w.r.t. eager $R\bot$-reduction of w^j_i ($\bot$ is considered to be a constructor),
- $f_i \gtrsim f_j$ for $i \leq j$.

At some point either we reach a constructor normal form or a term $f_i(t^i_1, \ldots, t^i_{n_i})$ repeats.

- "cons-free" implies: there are only polynomially many such terms,

Computing eager $R\bot$-reduction

$$f_1(w^1_1,\dots,w^1_{n_1}) \to^*_{R\bot e} f_1(t^1_1,\dots,t^1_{n_1}) \to^\epsilon_R$$
$$f_2(w^2_1,\dots,w^2_{n_2}) \to^*_{R\bot e} f_2(t^2_1,\dots,t^2_{n_2}) \to^\epsilon_R \dots$$

- t^j_i is the constructor normal form w.r.t. eager $R\bot$-reduction of w^j_i ($\bot$ is considered to be a constructor),
- $f_i \gtrsim f_j$ for $i \leq j$.

At some point either we reach a constructor normal form or a term $f_i(t^i_1,\dots,t^i_{n_i})$ repeats.

- "cons-free" implies: there are only polynomially many such terms, so a logarithmic counter may be used to detect looping,

Computing eager $R\bot$-reduction

$$
\begin{array}{l}
f_1(w_1^1, \ldots, w_{n_1}^1) \to^*_{R\bot e} f_1(t_1^1, \ldots, t_{n_1}^1) \to^\epsilon_R \\
f_2(w_1^2, \ldots, w_{n_2}^2) \to^*_{R\bot e} f_2(t_1^2, \ldots, t_{n_2}^2) \to^\epsilon_R \ldots
\end{array}
$$

- 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),
- $f_i \gtrsim f_j$ for $i \leq j$.

At some point either we reach a constructor normal form or a term $f_i(t_1^i, \ldots, t_{n_i}^i)$ repeats.

- "cons-free" implies: there are only polynomially many such terms, so a logarithmic counter may be used to detect looping,
- "tail-recursive" implies: t_i^j may be computed recursively, and the recursion depth will be constant.

Stream TRSs

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 $(::) : d \times s \to s$. Terms of sort s are *stream terms*. Terms of sort d are *data terms*.

Simple stream TRSs

Simple stream rule:

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

- $u_1, \ldots, u_n$ are constructor terms,
- $w_1, \ldots, w_m$ cons-free with respect to stream subterms,
- if $k = 0$ then cons-free with respect to data subterms.

Simple stream TRSs

Simple stream rule:

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

- $u_1, \ldots, u_n$ are constructor terms,
- $w_1, \ldots, w_m$ cons-free with respect to stream subterms,
- if $k = 0$ then cons-free with respect to data subterms.

Simple stream TRS:

- finite and orthogonal,

Simple stream TRSs

Simple stream rule:

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

- $u_1, \ldots, u_n$ are constructor terms,
- $w_1, \ldots, w_m$ cons-free with respect to stream subterms,
- if $k = 0$ then cons-free with respect to data subterms.

Simple stream TRS:

- finite and orthogonal,
- simple stream rules,

Simple stream TRSs

Simple stream rule:

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

- $u_1,\ldots,u_n$ are constructor terms,
- $w_1,\ldots,w_m$ cons-free with respect to stream subterms,
- if $k = 0$ then cons-free with respect to data subterms.

Simple stream TRS:

- finite and orthogonal,
- simple stream rules,
- data rules form a finite tail-recursive cons-free constructor TRS,

Simple stream TRSs

Simple stream rule:

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

- $u_1, \ldots, u_n$ are constructor terms,
- $w_1, \ldots, w_m$ cons-free with respect to stream subterms,
- if $k = 0$ then cons-free with respect to data subterms.

Simple stream TRS:

- finite and orthogonal,
- simple stream rules,
- data rules form a finite tail-recursive cons-free constructor TRS,
- 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.

Simple stream TRS – example

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

Simple stream TRS – example

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

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

Simple stream TRS – example

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

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

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 $g(n,m)$ reduces to T_{2m-n} and $\tilde{g}(n,m)$ to $1 - T_{2m-n}$.

Stream TRSs – Infinitary reduction

Infinitary R-reduction is defined coinductively.

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

Stream TRSs – Infinitary reduction

Infinitary R-reduction is defined coinductively.

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

Example:

$$f(x) \to_R x :: f(S(x))$$

Then $f(0) \to^\infty 0 :: S(0) :: S^2(0) :: \ldots$

Stream TRSs – Definability

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 $\Sigma^\omega \cup \Sigma^*$.

Stream TRSs – Definability

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 $\Sigma^\omega \cup \Sigma^*$.

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.

LOGSPACE for streams

Jumping Turing Transducer (JTT):

- finitely many states,
- read-only input tape, write-only output tape and finitely many read-write work tapes,
- finitely many cursors on the input tape (may move forward or jump to another cursor) and the work tapes (may move in both directions),
- one cursor on the output tape (may move forward).

LOGSPACE for streams

Jumping Turing Transducer (JTT):

- finitely many states,
- read-only input tape, write-only output tape and finitely many read-write work tapes,
- finitely many cursors on the input tape (may move forward or jump to another cursor) and the work tapes (may move in both directions),
- one cursor on the output tape (may move forward).

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, in the sense of Ramyaa and Leivant, if there is a JTT computing this function which operates in space $O(\log n)$.

Characterisation of LOGSPACE for streams

Theorem

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

Characterisation of LOGSPACE for streams

Theorem

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

($\Leftarrow$) encode any JFT with a local counter in a simple stream TRS, encoding the states as stream function symbols, and the counter as a data term,

Characterisation of LOGSPACE for streams

Theorem

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

($\Leftarrow$) encode any JFT with a local counter in a simple stream TRS, encoding the states as stream function symbols, and the counter as a data term,

($\Rightarrow$) 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 before to compute constructor normal forms of data terms.