ON FIRST-ORDER CONS-FREE TERM REWRITING AND PTIME

CYNTHIA KOP

Department of Computer Science, Copenhagen University   *e-mail address*: kop@di.ku.dk

ABSTRACT. In this paper, we prove that (first-order) cons-free term rewriting with a call-by-value reduction strategy exactly characterises the class of PTIME-computable functions. We use this to give an alternative proof of the result by Carvalho and Simonsen which states that cons-free term rewriting with linearity constraints characterises this class.

1. INTRODUCTION

In [4], Jones introduces the notion of *cons-free programming*: working with a small functional programming language, cons-free programs are defined to be *read-only*: recursive data cannot be created or altered, only read from the input. By imposing further restrictions on data order and recursion style, classes of cons-free programs turn out to characterise various deterministic classes in the time and space hierarchies of computational complexity.

Rather than using an artificial language, it would make sense to consider *term rewriting*. The authors of [3] explore a first definition of cons-free first-order term rewriting, and prove that this exactly characterises PTIME, provided a partial linearity restriction is imposed. This restriction is necessary since, without it, we can implement exponential algorithms in a cons-free system [5]. However, the restriction is not a common one, and the proof is intricate.

In this paper, we provide an alternative, simpler proof of this result. We do so by giving some simple syntactical transformations which allow a call-by-value reduction strategy to be imposed, and show that call-by-value cons-free first-order term rewriting characterises PTIME. This incidentally gives a new result with respect to call-by-value cons-free term rewriting, as well as a simplification of the linearity restriction in [3].

2. CONS-FREE TERM REWRITING

We assume the basic notions of first-order term rewriting to be understood. We particularly assume that the set of rules $\mathcal{R}$ is finite, and split the signature $\mathcal{F}$ into $\mathcal{D} \cup \mathcal{C}$ of defined symbols ($\mathcal{D}$) and constructors ($\mathcal{C}$). $\mathcal{T}(\mathcal{F}, \mathcal{V})$ denotes the set of terms built from symbols in $\mathcal{F}$ and variables, and $\mathcal{T}(\mathcal{F})$ the set of *ground* terms over $\mathcal{F}$. Elements of $\mathcal{T}(\mathcal{C})$ (ground constructor terms) are called *data terms*. The *call-by-value* reduction relation is $\leadsto_{\mathcal{R}} \subseteq \to_{\mathcal{R}}$ where a term s may only be reduced at position p if $s|_p$ has the form $f(s_1, \ldots, s_n)$ with all s_i data terms. The subterm relation is denoted $\unrhd$, or $\rhd$ for strict subterms.

Like Jones [4], we will limit interest to *cons-free* rules. To start, we must define what this means in the setting of term rewriting.

Definition 1 (Cons-free Rules). ([3]) A set of rules $\mathcal{R}$ is *cons-free* if for all $\ell \to r \in \mathcal{R}$:
- ℓ is linear (so no variable occurs more than once);
- ℓ has the form $f(\ell_1, \ldots, \ell_n)$ with all ℓ_i constructor terms (including variables);
- if $r \trianglerighteq t$ where $t = c(r_1, \ldots, r_m)$ with $c \in \mathcal{C}$, then either $t \in \mathcal{T}(\mathcal{C})$ or $\ell \rhd t$.

So $\mathcal{R}$ is a left-linear constructor system whose rules introduce no new constructors (besides fixed data). Cons-free term rewriting enjoys many convenient properties. Most importantly, the set of data terms that may be reduced to is limited by the data terms in the start term and the right-hand sides of rules, as described by the following definition.

Definition 2. For a given ground term s, the set $\mathcal{B}_s$ contains all data terms t which occur as (a) a subterm of s, or (b) a subterm of the right-hand side of some rule in $\mathcal{R}$.

$\mathcal{B}_s$ is closed under subterms and, since $\mathcal{R}$ is fixed, has linear size in the size of s. We will see that cons-free reduction, when starting with a term of the right shape, preserves the property of $\mathcal{B}$-*safety*, which limits the constructors that may occur at any position in a term:

Definition 3 ($\mathcal{B}$-safety). Given a set $\mathcal{B}$ of data terms which is closed under subterms, and which contains all data terms occurring in a right-hand side of $\mathcal{R}$:
1. any term in $\mathcal{B}$ is $\mathcal{B}$-safe;
2. if $f \in \mathcal{D}$ has arity n and $s_1, \ldots, s_n$ are $\mathcal{B}$-safe, then $f(s_1, \ldots, s_n)$ is $\mathcal{B}$-safe.

For cons-free $\mathcal{R}$, it is not hard to obtain the following property:

Lemma 4. *Let $\mathcal{R}$ be cons-free. For all s, t: if s is $\mathcal{B}$-safe and $s \to_{\mathcal{R}}^* t$, then t is $\mathcal{B}$-safe.*

Thus, for a decision problem $\mathtt{start}(s_1, \ldots, s_n) \to_{\mathcal{R}}^* t$ or $\mathtt{start}(s_1, \ldots, s_n) \rightsquigarrow_{\mathcal{R}} t$ (where t and all s_i are data terms), all terms occurring in the reduction are $\mathcal{B}$-safe. This insight allows us to limit interest to $\mathcal{B}$-safe terms in most cases, and is instrumental in the following.

3. Call-by-value Cons-free Rewriting Characterises PTIME

For our first result – which will serve as a basis for our simplification of the proof in [3] – we will see that any decision problem in PTIME can be accepted by a cons-free TRSs with call-by-value reduction, and vice versa. First, we define what *accepting* means for a TRS.

Definition 5. A decision problem is a set $A \subseteq \{0, 1\}^*$.

A TRS $(\mathcal{F}, \mathcal{R})$ with nullary constructors $\mathtt{true}, \mathtt{false}, \mathtt{0}, \mathtt{1}$ and $\mathtt{nil}$, a binary constructor $::$ (denoted infix) and a unary defined symbol $\mathtt{start}$ *accepts* A if for all $s = s_1 \ldots s_n \in \{0, 1\}^*$: $s \in A$ if and only if $\mathtt{start}(s_1 :: \cdots :: s_n :: \mathtt{nil}) \to_{\mathcal{R}}^* \mathtt{true}$. Similarly, such a TRS *accepts* A *by call-by-value reduction* if: $s \in A$ if and only if $\mathtt{start}(s_1 :: \cdots :: s_n :: \mathtt{nil}) \rightsquigarrow_{\mathcal{R}}^* \mathtt{true}$.

It is not required that *all* evaluations end in $\mathtt{true}$, just that there is such an evaluation – and that there is not if $s \notin A$. This is important as TRSs are not required to be deterministic. We say that a TRS *decides* A if it accepts A and is moreover deterministic. This also corresponds to the notion for (non-deterministic) Turing Machines. We claim:

Lemma 6. *If a decision problem A is in PTIME, then there exists a cons-free TRS which decides A by call-by-value reduction.*

Proof. It is not hard to adapt the method of [4] which, given a fixed deterministic Turing Machine operating in polynomial time, specifies a cons-free TRS simulating the machine. □

To see that cons-free call-by-value term rewriting *characterises* PTIME, it merely remains to be seen that every decision problem that is accepted by a call-by-value cons-free TRS can be solved by a deterministic Turing Machine – or, equivalently, an algorithm in pseudo code – running in polynomial time. In particular, we consider the following algorithm.

Algorithm 7. For a given starting term s, let $\mathcal{B} := \mathcal{B}_s$. For all $f \in \mathcal{F}$ of arity n and for all $s_1, \ldots, s_n, t \in \mathcal{B}$, let $\mathsf{Confirmed}^i[f(\vec{s}) \approx t] = \mathsf{NO}$.

Now, for $i \in \mathbb{N}$ and f of arity n in $\mathcal{D}, s_1, \ldots, s_n, t \in \mathcal{B}$:

- if $\mathsf{Confirmed}^i[f(\vec{s}) \approx t] = \mathsf{YES}$, then $\mathsf{Confirmed}^{i+1}[f(\vec{s}) \approx t] := \mathsf{YES}$;
- if there is some rule $\ell \to r \in \mathcal{R}$ matching $f(\vec{s})$ and a substitution γ such that $f(\vec{s}) = \ell\gamma$, and if $t \in \mathsf{NF}_i(r\gamma)$, then $\mathsf{Confirmed}^{i+1}[f(\vec{s}) \approx t] := \mathsf{YES}$:
- if neither of the above hold, then $\mathsf{Confirmed}^{i+1}[f(\vec{s}) \approx t] := \mathsf{NO}$.

Here, $\mathsf{NF}_i(s)$ is defined recursively for $\mathcal{B}$-safe terms s by:

- if s is a data term, then $\mathsf{NF}_i(s) = \{s\}$;
- if $s = f(s_1, \ldots, s_n)$, then let $\mathsf{NF}_i(s) =$
  $\bigcup\{u \in \mathcal{B} \mid \exists t_1 \in \mathsf{NF}_i(s_1), \ldots, t_n \in \mathsf{NF}_i(s_n).\mathsf{Confirmed}^i[f(t_1, \ldots, t_n) \approx u] = \mathsf{YES}\}$.

We stop the algorithm at the first index $I > 0$ where for all $f \in \mathcal{F}$ and $\vec{s}, t \in \mathcal{B}$: $\mathsf{Confirmed}^I[f(\vec{s}) \approx t] = \mathsf{Confirmed}^{I-1}[f(\vec{s}) \approx t]$.

As $\mathcal{D}$ and $\mathcal{B}$ are both finite, and the number of positions at which $\mathsf{Confirmed}^i$ is YES increases in every step, this process always ends. What is more, it ends (relatively) fast:

Lemma 8. *Algorithm 7 operates in $O(n^{3k+3})$ steps, where n is the size of the input term s and k the greatest arity in $\mathcal{D}$ (assuming the size and contents of $\mathcal{R}$ and $\mathcal{F}$ constant).*

Moreover, it provides a decision procedure, calculating *all* normal forms at once:

Lemma 9. *For $f \in \mathcal{D}$ of arity n and $s_1, \ldots, s_n, t \in \mathcal{B}$: $\mathsf{Confirmed}^I[f(s_1, \ldots, s_n) \approx t] = \mathsf{YES}$ if and only if $f(s_1, \ldots, s_n) \leadsto^*_{\mathcal{R}} t$.*

Combining these results, we obtain:

Corollary 10. *Cons-free call-by-value term rewriting characterises PTIME.*

Comment: although new, this result is admittedly unsurprising, given the similarity of this result to Jones' work in [4]. Although Jones uses a deterministic language, Bonfante [1] shows (following an early result in [2]) that adding a non-deterministic choice operator to cons-free first-order programs makes no difference in expressivity.

4. "Constrained" Systems

Towards the main topic in this work, we consider the syntactic restriction imposed in [3].

Definition 11. For any non-variable term $f(\ell_1, \ldots, \ell_n)$, let $\mathsf{DV}_{f(\ell_1, \ldots, \ell_n)}$ consist of those ℓ_i which are variables. We say a rule $\ell \to r$ is *semi-linear* if each $x \in \mathsf{DV}_\ell$ occurs at most once in r. A set of rules $\mathcal{R}$ is *constrained* if there exists $\mathcal{A} \subseteq \mathcal{D}$ such that for all $\ell \to r \in \mathcal{R}$:

- if the root symbol of ℓ is an element of $\mathcal{A}$, then $\ell \to r$ is semi-linear;
- for all $x \in \mathsf{DV}_\ell$ and terms t: if $r \unrhd t \rhd x$ then the root symbol of t is in $\mathcal{A}$.

We easily obtain a counterpart of Lemma 6, so to obtain a characterisation result, it suffices if a "constrained" cons-free TRS cannot handle problems outside PTIME. This we show by translating any such system into a cons-free call-by-value TRS, in two steps:

- First, the "constrained" definition is hard to fully oversee. We will consider a simple syntactic transformation to an equivalent system where all rules are semi-linear.
- Second, we add rules to the system to let every ground term reduce to a data term. Having done this, we can safely impose a call-by-value evaluation strategy.

4.1. **Semi-linearity.** It is worth noting that, of the two restrictions, the key one is for rules to be semi-linear. While it is allowed for some rules not to be semi-linear, their variable duplication cannot occur in a recursive way. In practice, this means that the ability to have symbols $f \in \mathcal{D} \setminus \mathcal{A}$ and non-semi-linear rules is little more than syntactic sugar.

To demonstrate this, let us start by a few syntactic changes which transform a "constrained" cons-free TRS into a semi-linear one (that is, one where all rules are semi-linear).

Definition 12. For all $f : n \in \mathcal{D}$, for all indexes i with $1 \leq i \leq n$, we let $\mathtt{count}(f,i) :=$ $\max(\{\mathtt{varcount}(f,i,\rho) \mid \rho \in \mathcal{R}\} \cup \{1\})$, where $\mathtt{varcount}(f,i,g(\ell_1,\ldots,\ell_m) \to r)$ is:
- 1 if $f \neq g$ or ℓ_i is not a variable;
- the number of occurrences of ℓ_i in r if $f = g$ and ℓ_i is a variable.

Note that, by definition of $\mathcal{A}$, $\mathtt{count}(f,i) = 1$ for all i if $f \in \mathcal{A}$. Let the new signature $\mathcal{F}^\bullet := \mathcal{C} \cup \{f : \sum_{i=1}^n \mathtt{count}(f,i) \mid f : i \in \mathcal{D}\}$ (where $f : k$ indicates f has arity k).

In order to transform terms to $\mathcal{T}(\mathcal{F}^\bullet, \mathcal{V})$, we define φ:

Definition 13. For any term s in $\mathcal{T}(\mathcal{F}, \mathcal{V})$, let $\varphi(s)$ in $\mathcal{T}(\mathcal{F}^\bullet, \mathcal{V})$ be inductively defined:
- if s is a variable, then $\varphi(s) := s$;
- if $s = c(\ldots)$ with $c \in \mathcal{C}$, then $\varphi(s) := s$;
- if $s = f(s_1,\ldots,s_n)$ with $f \in \mathcal{D}$, then each s_i is copied $\mathtt{count}(f,i)$ times; that is: $\varphi(s) := f(s_1^{(1)},\ldots,s_1^{(\mathtt{count}(f,1))},\ldots,s_n^{(1)},\ldots,s_n^{(\mathtt{count}(f,n))})$.

We easily obtain that $\varphi(s)$ respects the arities in $\mathcal{F}^\bullet$, provided $s \trianglerighteq c(\ldots)$ with $c \in \mathcal{C}$ implies $c(\ldots) \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ – which is the case in $\mathcal{B}$-safe terms and right-hand sides of rules in $\mathcal{R}$. Moreover, $\mathcal{B}$-safe terms over $\mathcal{F}$ are mapped to $\mathcal{B}$-safe terms over $\mathcal{F}^\bullet$.

Definition 14. We create a new set of rules $\mathcal{R}^\bullet$ containing, for all elements $f(\ell_1,\ldots,\ell_n) \to r \in \mathcal{R}$, a rule $f(\ell_1^1,\ldots,\ell_1^{k_1},\ldots,\ell_n^1,\ldots,\ell_n^{k_n}) \to r''$ where $k_i := \mathtt{count}(f,i)$ for $1 \leq i \leq n$ and:
- for all $1 \leq i \leq n$: $\ell_i^1 = \ell_i$, and all other ℓ_i^j are distinct fresh variables;
- $r'' := \varphi(r')$, where r' is obtained from r by replacing all occurrences of a variable $\ell_i \in \mathsf{DV}_{f(\ell_1,\ldots,\ell_n)}$ by distinct variables from $\ell_i^1,\ldots,\ell_i^{k_i}$.

Using the restrictions and the property that $\mathtt{count}(f,i)$ is always 1 for $f \in \mathcal{A}$, we obtain:

Lemma 15. *The rules in $\mathcal{R}^\bullet$ are well-defined, cons-free and semi-linear.*

Moreover, these altered rules give roughly the same rewrite relation:

Theorem 16. *Let s,t be $\mathcal{B}$-safe terms and u a data term. Then:*
- *if $s \to_\mathcal{R} t$, then $\varphi(s) \to_{\mathcal{R}^\bullet}^+ \varphi(t)$ (an easy induction on the size of s);*
- *if $\varphi(s) \to_{\mathcal{R}^\bullet}^* u$, then $s \to_\mathcal{R}^* u$ (by induction on the length of $\varphi(s) \to_{\mathcal{R}^\bullet}^* u$);*
- *$s \to_\mathcal{R}^* u$ if and only if $\varphi(s) \to_{\mathcal{R}^\bullet}^* u$ (by combining the first two statements).*

To avoid a need to alter the input, we may add further (semi-linear!) rules such as $\mathtt{start}'([]) \to \varphi(\mathtt{start}([]))$, $\mathtt{start}'(x :: y) \to \varphi(\mathtt{start}(x :: y))$. We obtain the corollary that constrained cons-free rewriting characterises PTIME iff semi-linear cons-free rewriting does.

4.2. **Call-by-value Reduction.** Now, to draw the connection with Corollary 10, we cannot simply impose a call-by-value strategy and expect to obtain the same normal forms; an immediate counterexample is the TRS with rules $\mathsf{a} \to \mathsf{a}$ and $\mathsf{f}(x) \to \mathsf{b}$: we have $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}}^* \mathsf{b}$, but this normal form is never reached using call-by-value rewriting.

Thus, we will use another simple syntactic adaptation:

Definition 17. We let $\mathcal{F}_\perp^\bullet := \mathcal{F}^\bullet \cup \{\perp\}$, and let $\mathcal{R}_\perp^\bullet := \mathcal{R}^\bullet \cup \{f(x_1, \ldots, x_n) \to \perp \mid f : n \in \mathcal{D}\}$. We also include $\perp$ in $\mathcal{B}$.

After this modification, every ground term reduces to a data term, which allows a call-by-value strategy to work optimally. Otherwise, the extra rules have little effect:

Lemma 18. *Let s be a $\mathcal{B}$-safe term in $\mathcal{T}(\mathcal{F}^\bullet)$ and $\perp \neq t \in \mathcal{T}(\mathcal{C})$. Then $s \to_{\mathcal{R}^\bullet}^* t$ iff $s \to_{\mathcal{R}_\perp^\bullet}^* t$.*

On this TRS, we may safely impose call-by-value strategy.

Lemma 19. *Let s be a $\mathcal{B}$-safe term and t a data term such that $s \to_{\mathcal{R}^\bullet}^* t$. Then $s \leadsto_{\mathcal{R}_\perp^\bullet}^* t$.*

Proof. The core idea is to trace descendants: if $C[u] \to_{\mathcal{R}^\bullet}^* q$ by reductions in C and u is not data, then because of semi-linearity, q has at most one copy of u: say $q = C'[u]$ with $C[] \to_{\mathcal{R}^\bullet}^* C'[]$. Any subsequent reduction in u might as well be done immediately in $C[u]$. $\square$

Binding Lemmas 18 and 19 together, we obtain:

Corollary 20. *For every $\mathcal{B}$-safe term $s \in \mathcal{T}(\mathcal{F}^\bullet)$ and data term $t \neq \perp$: $s \to_{\mathcal{R}^\bullet}^* t$ iff $s \leadsto_{\mathcal{R}_\perp^\bullet}^* t$.*

5. CONCLUSION

Putting the transformations and Algorithm 7 together, we thus obtain an alternative proof for the result in [3]. But we have done a bit more than that: we have also seen that both call-by-value and semi-linear cons-free term rewriting characterise PTIME. Moreover, through these transformations we have demonstrated that, at least in the first-order setting, there is little advantage to be gained by considering constrained or semi-linear rewriting over the (arguably simpler) approach of imposing an evaluation strategy.

Although we have used a call-by-value strategy here for simplicity, it would not be hard to adapt the results to use the more common (in rewriting) *innermost* strategy instead. An interesting future work would be to test whether the parallel with Jones' work extends to higher orders, i.e. whether innermost k^{th}-order rewriting characterises $\text{EXP}^{k-1}\text{TIME}$ – and whether instead using semi-linearity restrictions does add expressivity in this setting.

A longer version of this work containing complete proofs is available at:

http://cl-informatik.uibk.ac.at/users/kop/dice16long.pdf

REFERENCES

[1] G. Bonfante. Some programming languages for logspace and ptime. In M. Johnson, editor, *AMAST '06*, volume 4019 of *LNCS*, pages 66–80, 2006.

[2] S.A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *ACM*, 18(1):4–18, 1971.

[3] D. de Carvalho and J. Simonsen. An implicit characterization of the polynomial-time decidable sets by cons-free rewriting. In G. Dowek, editor, *RTA-TLCA '14*, volume 8560 of *LNCS*, pages 179–193, 2014.

[4] N. Jones. Life without cons. *JFP*, 11(1):5–94, 2001.

[5] C. Kop and J. Simonsen. Complexity hierarchies and higher-order cons-free rewriting. Unpublished; http://cl-informatik.uibk.ac.at/users/kop/fscd16.pdf, 2016.