This paper is an extended version of a paper presented at DICE 2016, including complete proofs. The 5-paper extended abstract in the DICE proceedings can be found at:

http://cl-informatik.uibk.ac.at/users/kop/dice16.pdf

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 exactly those which can be specified without introducing the list constructor. Put differently, a cons-free program is *read-only*: recursive data cannot be created or altered (beyond taking subterms), only read from the input.

The interest in such programs lies in their applicability to computational complexity: by imposing cons-freeness, the resulting set of programs can only compute functions in a proper subclass of the Turing-computable functions. Jones shows that further limitations to these programs lower the resulting expressivity to known classes. For example, cons-free programs with data order 0 can decide exactly those decision problems which are in PTIME, while tail-recursive cons-free programs with data order 1 characterise PSPACE.

Rather than an artificial functional programming language, it would make a lot of sense to consider *term rewriting* instead. This well-established paradigm, which lies at the heart of functional programming, both offers the expressivity to naturally define cons-free programming, and the simplicity which real-life functional programming languages lack. In addition, term rewriting is natively non-deterministic, which is likely to be useful for characterising the non-deterministic complexity classes.

The authors of [3] explore a first definition of cons-free term rewriting, and prove that this class – when limited to *first-order* term rewriting – characterises PTIME. However, in order to do so they impose a partial linearity restriction on the programs. This restriction is necessary since unrestricted first-order cons-free term rewriting allows for the implementation of arbitrary algorithms operating in $\mathcal{O}(2^{k \cdot n})$ for any k [5]. However, the restriction is not a common one, and the proof is intricate.

The author is supported by the Marie Skłodowska-Curie action "HORIP", program H2020-MSCA-IF-2014, 658162, and partially supported by the Danish Council for Independent Research Sapere Aude grant "Complexity via Logic and Algebra" (COLA)..

In this paper, we will 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 rewriting, as well as a simplification of the linearity restriction in [3].

2. Preliminaries: first-order term rewriting

We assume given an infinite set of variables $\mathcal{V}$ and a finite, disjoint set of symbols $\mathcal{F}$, each $f \in \mathcal{F}$ equipped with an arity $n \in \mathbb{N}$; we will either indicate membership using $f : n \in \mathcal{F}$ or omit the arity and simply denote $f \in \mathcal{F}$. The set of terms $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is given by:

- $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$;
- if $f : n \in \mathcal{F}$, and $\{s_1, \dots, s_n\} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, then $f(s_1, \dots, s_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$.

Let $\mathit{Var}(s)$ be the variables occurring in a term s; a term is *ground* if $\mathit{Var}(s) = \emptyset$. Let $\mathcal{T}(\mathcal{F}) := \mathcal{T}(\mathcal{F}, \emptyset)$ denote the set of ground terms over $\mathcal{F}$. We say t is a subterm of s, notation $s \trianglerighteq t$, if either $s = t$, or $s = f(s_1, \dots, s_n)$ and $s_i \trianglerighteq t$ for some i. Let $s \triangleright t$ if $s \trianglerighteq t$ and $s \neq t$.

A substitution is a function from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$, often denoted $[x_1 := s_1, \dots, x_n := s_n]$ (this substitution is the identity on $y \notin \{x_1, \dots, x_n\}$). For a term s and substitution γ, we let $s\gamma$ be s with all variables x replaced by $\gamma(x)$. The *domain* of γ is the set of variables x such that $\gamma(x) \neq x$.

A *rule* is a pair $\ell \to r$ of terms such that $\mathit{Var}(r) \subseteq \mathit{Var}(\ell)$ and ℓ is not a variable, i.e. ℓ can be written $f(\ell_1, \dots, \ell_n)$; we call f the *root symbol* of the rule. Given a set of rules $\mathcal{R}$, the reduction relation $\to_{\mathcal{R}}$ is inductively defined:

- $\ell\gamma \to_{\mathcal{R}} r\gamma$ for all $\ell \to r \in \mathcal{R}$ and substitutions γ;
- if $s_i \to_{\mathcal{R}} t_i$, then $f(s_1, \dots, s_i, \dots, s_n) \to_{\mathcal{R}} f(s_1, \dots, t_i, \dots, s_n)$.

We assume that $\mathcal{R}$ is finite. Fixing $\mathcal{F}$ and $\mathcal{R}$, we let $\mathcal{D}$ be the set of root symbols of any rule in $\mathcal{R}$, and $\mathcal{C} := \mathcal{F} \setminus \mathcal{D}$; symbols in $\mathcal{D}$ are called *defined symbols* and those in $\mathcal{C}$ *constructors*. A *constructor term* is a term in $\mathcal{T}(\mathcal{C}, \mathcal{V})$, and a *data term* a term in $\mathcal{T}(\mathcal{C})$. A *basic* term is a term $f(s_1, \dots, s_n)$ with $f \in \mathcal{D}$ and all s_i data. The *call-by-value* reduction relation allows root reductions only on basic terms; formally:

- $f(\ell_1, \dots, \ell_n)\gamma \rightsquigarrow_{\mathcal{R}} r\gamma$ for all $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}$ and substitutions γ such that all $\ell_i\gamma$ are in $\mathcal{T}(\mathcal{C})$;
- if $s_i \rightsquigarrow_{\mathcal{R}} t_i$, then $f(s_1, \dots, s_i, \dots, s_n) \rightsquigarrow_{\mathcal{R}} f(s_1, \dots, t_i, \dots, s_n)$.

Clearly, $\rightsquigarrow_{\mathcal{R}} \subseteq \to_{\mathcal{R}}$. A *term rewriting system* (TRS) is a pair $(\mathcal{T}(\mathcal{F}, \mathcal{V}), \to_{\mathcal{R}})$ and a *call-by-value TRS* is a pair $(\mathcal{T}(\mathcal{F}, \mathcal{V}), \rightsquigarrow_{\mathcal{R}})$. Both are typically given just by supplying $(\mathcal{F}, \mathcal{R})$.

3. Cons-free Term Rewriting

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)**.** A set of rules $\mathcal{R}$ is *cons-free* if it is a left-linear constructor-system whose right-hand side introduces no new non-ground constructor terms. Formally, $\mathcal{R}$ is cons-free if for all $\ell \to r \in \mathcal{R}$:

- ℓ is linear; that is, no variable occurs more than once in ℓ;
- ℓ has the form $f(\ell_1, \dots, \ell_n)$ with all ℓ_i constructor terms (including variables);
- if $r \trianglerighteq t$ where $t = c(r_1, \dots, r_m)$ with $c \in \mathcal{C}$, then either $t \in \mathcal{T}(\mathcal{C})$ or $\ell \triangleright t$.

Cons-free term rewriting enjoys many convenient properties. Most importantly, the set of data terms that may be reduced to using cons-free rules 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:

(1) all subterms of s which are data terms;
(2) all subterms of the right-hand side of some rule in $\mathcal{R}$ which are data terms.

$\mathcal{B}_s$ 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 $\mathcal{R}$. A term s is $\mathcal{B}$-*safe* if all its subterms rooted by a constructor are elements of $\mathcal{B}$. Alternatively put:

(1) any term in $\mathcal{B}$ is $\mathcal{B}$-safe;
(2) if $f \in \mathcal{D}$ has arity n and $s_1, \dots, s_n$ are $\mathcal{B}$-safe, then $f(s_1, \dots, s_n)$ is $\mathcal{B}$-safe.

We trivially observe:

Lemma 4. *All subterms of a $\mathcal{B}$-safe term are also $\mathcal{B}$-safe.*

The crucial property of cons-free TRSs is that they preserve $\mathcal{B}$-safety:

Lemma 5. *Let $\mathcal{R}$ be cons-free. If s is $\mathcal{B}$-safe and $s \to_{\mathcal{R}}^* t$, then t is $\mathcal{B}$-safe.*

Proof. By induction on the form of s. If the reduction does not take place at the root, then $s = f(s_1, \dots, s_i, \dots, s_n)$, $t = f(s_1, \dots, s_i', \dots, s_n)$ and $s_i \to_{\mathcal{R}} s_i'$. As s_i reduces, it cannot be a data term; therefore, $\mathcal{B}$-safety of s must follow by clause (2), so $f \in \mathcal{D}$ and all s_j are $\mathcal{B}$-safe. By the induction hypothesis, so is s_i', and $\mathcal{B}$-safety of t follows similarly by clause (2).

If the reduction does take place at the root, then $s = \ell\gamma$ and $t = r\gamma$ for some $\ell \to r \in \mathcal{R}$. By Lemma 4, each $\gamma(x)$ is $\mathcal{B}$-safe. By induction on r, this implies $t = r\gamma$ is $\mathcal{B}$-safe:

- if r is a variable, then $r\gamma = \gamma(r)$ is $\mathcal{B}$-safe;
- if $r = f(r_1, \dots, r_n)$ with $f \in \mathcal{D}$, then by the induction hypothesis each $r_i\gamma$ is $\mathcal{B}$-safe, and therefore $\mathcal{B}$-safety of $r\gamma = f(r_1\gamma, \dots, r_n\gamma)$ follows by clause (2);
- if $r = c(r_1, \dots, r_n)$ with $c \in \mathcal{D}$, then either r is a data term – so $r \in \mathcal{B}$ by definition of $\mathcal{B}$ – or $\ell \trianglerighteq r$, so $r\gamma \trianglelefteq \ell\gamma$ is $\mathcal{B}$-safe by Lemma 4. □

Thus, for a decision problem $\mathtt{start}(s_1, \dots, s_n) \to_{\mathcal{R}}^* t$ or $\mathtt{start}(s_1, \dots, s_n) \rightsquigarrow_{\mathcal{R}} t$ (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.

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

Our first result – which will serve as a basis for the remainder – is that any decision problem in PTIME can be accepted by a cons-free TRSs with call-by-value reduction, and vice versa.

To start, we must understand what it means for a TRS to *accept* a decision problem.

Definition 6. 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 $s = s_1 \dots s_n \in \{0,1\}^*$: $s \in A$ if and only if $\mathtt{start}(s_1 :: \dots :: s_n :: \mathtt{nil}) \to_{\mathcal{R}}^* \mathtt{true}$. Similarly, a call-by-value TRS with those symbols accepts A if: $s \in A$ if and only if $\mathtt{start}(s_1 :: \dots :: s_n :: \mathtt{nil}) \rightsquigarrow_{\mathcal{R}}^* \mathtt{true}$.

Note that it is not required that *all* evaluations end in `true`, just that there is such an evaluation – and that there is not if $s \notin A$. This is very relevant as TRSs are not required to be deterministic. We say that a (call-by-value) TRS *decides* a decision problem A if it accepts A and moreover each term has a unique normal form. This corresponds to the notion for (non-deterministic) Turing Machines, where we say that a TM M accepts A if for all $s \in \{0,1\}^*$: some evaluation of M finishes in the accepting state if and only if $s \in A$.

We claim:

Lemma 7. *If a decision problem A is in PTIME – that is, if some deterministic Turing Machine exists which decides A and operates in polynomial time in the length of the input – then there exists a call-by-value cons-free TRS which decides A.*

Proof. We defer to [4]: the algorithm presented there can be seen as a deterministic call-by-value TRS. The only difficulty is that the author admits a pair constructor, which we do not; however, since the algorithm is deterministic this is easily solved by replacing tuples in the left-hand sides of rules by separate arguments, and replacing any function which reduces to a k-tuple by k-functions. For instance, a rule $\mathtt{f}\ (x,y)\ z \to (y,z)$ in the functional program is replaced by the two TRS-rules $\mathtt{f}^1(x,y,z) \to y$ and $\mathtt{f}^2(x,y,z) \to z$. □

Next, we will see that any decision problem that can be accepted by a cons-free call-by-value TRS is in PTIME. This may seem surprising at first, as it implies that the ability to take non-deterministic steps in a TRS adds no extra power. However, this is entirely in line with known results: already in 1973, Cook [2] demonstrated that adding non-determinism to a restricted machine model capable of characterising PTIME does not expand the class. Most relevantly, the author of [1] observes that adding a non-deterministic choice operator to cons-free first-order programming à la Jones does not add any expressive power.

To see that cons-free call-by-value term rewriting is indeed in PTIME, we will use a deterministic algorithm, running in polynomial time, which calculates *all* normal forms of basic terms $f(\vec{s})$ at once.

Algorithm 8. For a given starting term s, let $\mathcal{B} := \mathcal{B}_s$. For all $f : n \in \mathcal{F}$ and for all $s_1, \dots, s_n, t \in \mathcal{B}$, let $\mathsf{Confirmed}^i[f(\vec{s}) \approx t] = \mathtt{NO}$.

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

- if $\mathsf{Confirmed}^i[f(\vec{s}) \approx t] = \mathtt{YES}$, then $\mathsf{Confirmed}^{i+1}[f(\vec{s}) \approx t] := \mathtt{YES}$;
- 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 $\mathsf{Confirmed}^{i+1}[f(\vec{s}) \approx t] := \mathtt{YES}$:
- if neither of the above hold, then $\mathsf{Confirmed}^{i+1}[f(\vec{s}) \approx t] := \mathtt{NO}$.

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 $s = f(s_1, \dots, s_n)$, then let $\mathsf{NF}_i(s) = \bigcup\{u \in \mathcal{B} \mid \exists t_1 \in \mathsf{NF}_i(s_1), \dots, t_n \in \mathsf{NF}_i(s_n).\mathsf{Confirmed}^i[f(t_1, \dots, t_n) \approx u] = \mathtt{YES}\}$.

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 $\mathtt{YES}$ increases in every step, this process ends. The complexity is discussed below, but let us first focus on correctness. We must see two things: that $\mathsf{Confirmed}^I$ captures rewriting, and that it does not capture anything else. These results are explored in the next two lemmas.

First, we see that $\mathsf{Confirmed}^I$ captures rewriting:

Lemma 9. *For $f : n \in \mathcal{D}$ and $s_1, \dots, s_n, t \in \mathcal{B}$: if $f(\vec{s}) \rightsquigarrow^*_{\mathcal{R}} t$, then $\mathsf{Confirmed}^I[f(\vec{s}) \approx t] = \mathtt{YES}$. If s is $\mathcal{B}$-safe, $t \in \mathcal{B}$ and $s \rightsquigarrow^*_{\mathcal{R}} t$, then $t \in \mathsf{NF}^I(s)$.*

Proof. We prove both properties together by mutual induction on the length of the reduction. We start with the first claim, so we can assume it proven when considering the second.

In the base case, if $f(\vec{s}) = t$, we have a contradiction because $f \in \mathcal{D}$ and t must be a data term. In the induction case, we have $f(\vec{s}) \rightsquigarrow_{\mathcal{R}} u \rightsquigarrow^*_{\mathcal{R}} t$. Since the only defined symbol is at the root, $f(\vec{s}) = \ell\gamma$ for some rule $\ell \to r$ such that $u = r\gamma$. By the induction hypothesis, $t \in \mathsf{NF}^I(r\gamma) = \mathsf{NF}^{I-1}(r\gamma)$ (as $\mathsf{Confirmed}^I = \mathsf{Confirmed}^{I-1}$ by choice of I). But then $\mathsf{Confirmed}^{(I-1)+1}[f(\vec{s}) \approx t] = \mathtt{YES}$ by definition.

For the second claim, in the base case $s = t$ implies $s \in \mathcal{B}$, so $t \in \{s\} = \mathsf{NF}^I(s)$. In the induction case, s reduces, so $s \notin \mathcal{B}$ and we can write $s = f(s_1, \dots, s_n)$ with $f \in \mathcal{D}$ and all s_i $\mathcal{B}$-safe terms. Note that the reduction $s \rightsquigarrow^*_{\mathcal{R}} t$ must take a root step at some point, since t does not contain the defined symbol f. Thus we can write $s \rightsquigarrow^*_{\mathcal{R}} f(u_1, \dots, u_n) \rightsquigarrow^*_{\mathcal{R}} t$, where $f(\vec{u})$ reduces by a call-by-value root step, so all u_i are data terms; by Lemma 5 they are all in $\mathcal{B}$. By the induction hypothesis, we obtain $u_i \in \mathsf{NF}_I(s_i)$ for all i. By the main claim proven above (for a reduction at most the length of $s \rightsquigarrow^*_{\mathcal{R}} t$), $\mathsf{Confirmed}^I[f(\vec{s}) \approx t] = \mathtt{YES}$. Therefore indeed $t \in \mathsf{NF}_I(s)$. □

Then, we see that $\mathsf{Confirmed}^I$ does not capture anything else.

Lemma 10. *For $f : n \in \mathcal{D}$ and $s_1, \dots, s_n, t \in \mathcal{B}$: if $\mathsf{Confirmed}^I[f(\vec{s}) \approx t] = \mathtt{YES}$, then $f(\vec{s}) \rightsquigarrow^*_{\mathcal{R}} t$. For $\mathcal{B}$-safe s and $t \in \mathsf{NF}^I(s)$: $s \rightsquigarrow^*_{\mathcal{R}} t$.*

Proof. We prove both properties together for all relevant $\mathsf{Confirmed}^i/\mathsf{NF}^i$, by a mutual induction on $0 \leq i \leq I$.

For the first claim, assume $\mathsf{Confirmed}^i[f(\vec{s}) \approx t] = \mathtt{YES}$. This cannot hold for $i = 0$, so (for the induction step) assume that both claims hold for $i' < i$. We are done by the first part of the induction hypothesis if $\mathsf{Confirmed}^{i-1}[f(\vec{s}) \approx t] = \mathtt{YES}$, so assume the alternative. Then there are a rule $\ell \to r \in \mathcal{R}$ and a substitution γ such that $f(\vec{s}) = \ell\gamma$ and $t \in \mathsf{NF}^{i-1}(r\gamma)$. But by the second part of the induction hypothesis this implies that $r\gamma$ reduces to t. We are done because of course $\ell\gamma \to_{\mathcal{R}} r\gamma$.

For the second claim, let s be $\mathcal{B}$-safe, $t \in \mathsf{NF}_i(s)$ and assume that the first claim holds. We will not use the induction hypothesis directly, but will use a second induction on the size of s. If s is a data term, then $\mathsf{NF}_i(s) = \{s\}$, and we have $t = s$; evidently $s \rightsquigarrow^*_{\mathcal{R}} s$. Otherwise $s = f(s_1, \dots, s_n)$ with $f \in \mathcal{D}$, and there are some $u_1 \in \mathsf{NF}_i(s_1), \dots, u_n \in \mathsf{NF}_i(s_n)$ such that $\mathsf{Confirmed}^i[f(\vec{u}) \approx t] = \mathtt{YES}$. By the induction hypothesis on s, each $s_i \rightsquigarrow^*_{\mathcal{R}} u_i$, so indeed $s \rightsquigarrow^*_{\mathcal{R}} f(\vec{u})$. By the first claim, $f(\vec{u}) \rightsquigarrow^*_{\mathcal{R}} t$. □

Thus, the algorithm is correct. To see that is is polynomial, we count the steps.

Lemma 11. *Algorithm 8 operates in $O(n^{3k+3})$ steps, where n is the size of the input term s and k the greatest arity in $\mathcal{R}$ (assuming the size and contents of $\mathcal{R}$ and $\mathcal{F}$ constant).*

Proof. To understand the complexity of this algorithm, consider the calculation cost of the set $\mathsf{NF}^i(s)$, when $\mathsf{Confirmed}^i$ is fully defined. Due to the recursive structure of the definition, $\mathsf{NF}^i(t)$ is calculated exactly once for every subterm t of s rooted by a defined symbol, and for every *topmost* constructor symbol. For the constructor symbols only a single step is done, while for defined symbols f of arity n, we must test for all $\leq |\mathcal{B}|^{n+1}$ combinations $u_1 \in \mathsf{NF}^i(s_1), \dots, u_n \in \mathsf{NF}^i(s_n), v \in \mathcal{B}$ whether $\mathsf{Confirmed}^i[f(\vec{u}) \approx v] = \mathtt{YES}$. Taking into

account that in $r\gamma$ the defined symbols can only occur at positions in r (since each $\gamma(x)$ is a subterm of some s_i and therefore itself an element of $\mathcal{B}$), this means that $\mathsf{NF}^i(r\gamma)$ can be calculated in $O(|r| * |\mathcal{B}|^{k+1})$ steps, where k is the greatest arity in $\mathcal{D}$.

Furthermore, for the full algorithm, note that there are at most $|\mathcal{D}| * |\mathcal{B}|^{k+1}$ indexes in each $\mathsf{Confirmed}^i$. Given the increasing nature of the process, that means $I \leq |\mathcal{D}| * |\mathcal{B}|^{k+1} + 1$. It also means that in every step i, we consider at most $|\mathcal{D}| * |\mathcal{B}|^{k+1}$ positions, each giving at most $|\mathcal{R}| * O(\langle\text{size of largest } r\rangle * |\mathcal{B}|^{k+1})$ operations. In total, writing R for the size of the largest right-hand side in $\mathcal{R}$, we perform $O((|\mathcal{D}| * |\mathcal{B}|^{k+1})^2 * |\mathcal{R}| * R * |\mathcal{B}|^{k+1})$ operations. Given that we consider $\mathcal{R}$ and $\mathcal{F}$ fixed sets, and that $\mathcal{B}$ is linear in the size of the starting term – say n – this algorithm has $O(n^{3k+3})$ complexity; that is, polynomial in n! □

In sum, we obtain:

Theorem 12. *There is a polynomial algorithm to determine whether a cons-free TRS $\mathcal{R}$ with call-by-value reduction reduces a basic term $\mathtt{start}(s_1, \ldots, s_n)$ to $\mathtt{true}$.*

Proof. We let $\mathcal{B} := \mathcal{B}_{\mathtt{start}(\vec{s})}$, use the polynomial algorithm given above to determine $\mathsf{Confirmed}^I$, and look up whether $\mathsf{Confirmed}^I[f(\vec{s}) \approx \mathtt{true}] = \mathtt{YES}$. (If $\mathtt{true} \notin \mathcal{B}$, then we can immediately conclude that $\mathtt{start}(\vec{s}) \not\rightsquigarrow^*_{\mathcal{R}} \mathtt{true}$ by Lemma 5.) If indeed $\mathtt{start}(\vec{s}) \rightsquigarrow^*_{\mathcal{R}} \mathtt{true}$, then this value is $\mathtt{YES}$ by completeness (Lemma 9). If not, then this value can only be $\mathtt{NO}$ by soundness (Lemma 10). □

Taking into account Lemma 7, we thus obtain:

Corollary 13. *Cons-free call-by-value term rewriting characterises PTIME.*

5. "Constrained" Systems

Now, let us move on to the main topic of this work. Carvalho and Simonsen [3] proved a similar result to our Theorem 12, although they did not use call-by-value reduction. Instead, they imposed an additional syntactic restriction on the rewriting rules, considering only *constrained* cons-free TRS:

Definition 14. 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 $r \trianglerighteq t \triangleright x$ then the root symbol of t is in $\mathcal{A}$.

It is not hard to obtain a counterpart to Lemma 7, specifying a "constrained" cons-free TRS which simulates a given deterministic Turing Machine in PTIME. To also see that constrained rewriting cannot decide problems beyond PTIME, the key insight is that we can transform any such TRS into a cons-free call-by-value TRS without significantly altering the TRS's behaviour. This, we shall do 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 (but not affecting the reduction behaviour to data terms over the original signature). This, together with the semi-linearity restriction, is enough to allow a call-by-value strategy to be imposed, simply by eagerly evaluating all inner terms.

5.1. **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 15. 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 $\mathtt{varcount}(f, i, g(\ell_1, \dots, \ell_m) \to r)$ 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 $\mathcal{F}^\bullet := \mathcal{C} \cup \{f : \sum_{i=1}^{n} \mathtt{count}(f, i) \mid f : i \in \mathcal{D}\}$.

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

Definition 16. 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 $s = c(\dots)$ with $c \in \mathcal{C}$, then $\varphi(s) := s$;
- if $s = f(s_1, \dots, s_n)$ with $f \in \mathcal{D}$, then each s_i is copied $\mathtt{count}(f, i)$ times; that is: $\varphi(s) := f(s_1^{(1)}, \dots, s_1^{(\mathtt{count}(f,1))}, \dots, s_n^{(1)}, \dots, s_n^{(\mathtt{count}(f,n))})$.

It is easy to see that indeed $\varphi(s)$ respects the arities in $\mathcal{F}^\bullet$, provided subterms $c(\dots)$ of s which are headed by a constructor are guaranteed to be data terms – which is the case in $\mathcal{B}$-safe terms and right-hand sides of cons-free rules. Moreover, $\mathcal{B}$-safe terms over $\mathcal{F}$ are mapped to $\mathcal{B}$-safe terms over $\mathcal{F}^\bullet$.

Definition 17. We create a new set of rules $\mathcal{R}^\bullet$ containing, for all elements $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}$, a rule $f(\ell_1^1, \dots, \ell_1^{k_1}, \dots, \ell_n^1, \dots, \ell_n^{k_n}) \to r''$ 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 $\ell_i \in \mathsf{DV}_{f(\ell_1,\dots,\ell_n)}$ by distinct variables from $\ell_i^1, \dots, \ell_i^{k_i}$.

Due to the restrictions on the rules, we obtain:

Lemma 18. *The rules in $\mathcal{R}^\bullet$ are well-defined, cons-free and semi-linear.*

Proof. Let $\rho ::= f(\ell_1^1, \dots, \ell_1^{k_1}, \dots, \ell_n^{k_n}) \to r''$ be a rule in $\mathcal{R}^\bullet$, originating from $\ell \to r$.

For well-definedness, we note that f indeed respects the arity in $\mathcal{F}^\bullet$, that all terms ℓ_i^j respect the new arities because they are constructor terms, and that $\varphi(r')$ respects those arities because all constructor-headed subterms $c(\dots)$ of r' are either data terms in $\mathcal{B}$ or (possibly renamed) subterms of some constructor term ℓ_j; we also note that the renaming from r to r' can be done because by definition of $k_i := \mathtt{count}(f, i)$ each variable in DV_ℓ occurs at most as many times on the right as it does in the altered left-hand side.

The rules are also cons-free. Obviously all ℓ_i^j are constructor terms, either because they are immediate arguments of the left-hand side of a rule in $\mathcal{R}$, or because they are variables. Moreover, each subterm $c(\dots)$ of r'' with $c \in \mathcal{C}$ is either in $\mathcal{B}$ or a subterm of some ℓ_i, so of ℓ_i^1: by definition of φ, it is a subterm of r', so a renaming of a subterm of some non-variable argument ℓ_i, but as only the variables directly in $\ell_1, \dots, \ell_n$ are renamed, not those occurring in subterms (and we assumed left-linearity), the renaming has no effect.

As for semi-linearity: r' is linear in the required variables. The mapping φ cannot alter that, as the only duplicated subterms have the form $g(r_1, \dots, r_m)$ with $g \in \mathcal{D} \setminus \mathcal{A}$ (since always $\mathtt{count}(g, i) = 1$ for $g \in \mathcal{A}$), and by definition of $\mathcal{A}$, none of the dangerous variables occur inside such subterms. □

Lemma 19. *Let s, t be $\mathcal{B}$-safe terms. If $s \to_{\mathcal{R}} t$, then $\varphi(s) \to^+_{\mathcal{R}^\bullet} \varphi(t)$.*

Proof. By induction on the position of the redex; since s reduces and is $\mathcal{B}$-safe we can safely write $s = f(s_1, \dots, s_n)$ with $f \in \mathcal{D}$. Let $k_i := \mathtt{count}(f, i)$ for $1 \le i \le n$.

First suppose the reduction takes place in a subterm, so $t = f(\dots, s_i', \dots)$ with $s_i \to_{\mathcal{R}} s_i'$; by the induction hypothesis $\varphi(s_i) \to^+_{\mathcal{R}^\bullet} \varphi(s_i')$. As s_i is not a data term, $f \in \mathcal{D}$ by $\mathcal{B}$-safety of s. Thus

$$\begin{aligned}
\varphi(s) = \;& f(\varphi(s_1^{(1)}), \dots, \varphi(s_1^{(k_1)}), \dots, \varphi(s_i^{(1)}), \dots, \varphi(s_i^{(k_i)}), \dots, \varphi(s_n^{(1)}), \dots, \varphi(s_n^{(k_n)})) \\
\to^+_{\mathcal{R}^\bullet} \;& f(\varphi(s_1^{(1)}), \dots, \varphi(s_1^{(k_1)}), \dots, \varphi(s_i^{(1)'}), \dots, \varphi(s_i^{(k_i)}), \dots, \varphi(s_n^{(1)}), \dots, \varphi(s_n^{(k_n)})) \\
& \dots \\
\to^+_{\mathcal{R}^\bullet} \;& f(\varphi(s_1^{(1)}), \dots, \varphi(s_1^{(k_1)}), \dots, \varphi(s_i^{(1)'}), \dots, \varphi(s_i^{(k_i)'}), \dots, \varphi(s_n^{(1)}), \dots, \varphi(s_n^{(k_n)})) \\
= \;& \varphi(t)
\end{aligned}$$

At least one step is done, because each $k_i \ge 1$.

For the base case, suppose $s = \ell\gamma$ and $t = r\gamma$ for some rule $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}$. Then $\mathcal{R}^\bullet$ has a rule $\ell' := f(\ell_1^{(1)}, \dots, \ell_1^{(k_1)}, \dots, \ell_n^{(1)}, \dots, \ell_n^{(k_n)}) \to \varphi(r') =: r''$. In addition, we can write $\varphi(s) = f(\varphi(\ell_1\gamma)^{(1)}, \dots, \varphi(\ell_1\gamma)^{(k_1)}, \dots, \varphi(\ell_n\gamma)^{(1)}, \dots, \varphi(\ell_n\gamma)^{(k_n)})$.

Now, for $1 \le i \le n$, consider ℓ_i. If ℓ_i is not a variable, then it must be a constructor term; by $\mathcal{B}$-safety of s, $\ell_i\gamma \in \mathcal{B}$. For all $x \in \mathit{Var}(\ell_i)$, we thus let $\delta(x) := \gamma(x) \in \mathcal{T}(\mathcal{C}) \subseteq \mathcal{T}(\mathcal{F}^\bullet, \emptyset)$. In addition, we let $\delta(\ell_i^j) := \varphi(\ell_i\gamma)$ for $j > 1$. If ℓ_i *is* a variable, then let $\delta(\ell_i^j) := \varphi(\gamma(\ell_i))$ for all $1 \le j \le k_i$. Then δ maps all variables in ℓ' to $\mathcal{T}(\mathcal{F}^\bullet, \emptyset)$, and $\varphi(s) = \varphi(\ell\gamma) = \ell'\delta$.

We are done if also $\varphi(t) = \varphi(r')\delta$. Noting that r' is obtained from r by replacing some variables x by variables y with $\delta(x) = \delta(y)$ – and therefore $\varphi(r')$ is obtained from $\varphi(r)$ in the same way – it suffices to prove that $\varphi(r\gamma) = \varphi(r)\delta$. We prove this by a straightforward induction on the form of r, using that $\delta(x) = \gamma(x)$ for all variables occurring below a constructor in r, and $\delta(x) = \varphi(\gamma(x))$ for the remainder. □

This gives us one direction: $\to_{\mathcal{R}^\bullet}$ can simulate $\to_{\mathcal{R}}$. Now we must see the converse:

Lemma 20. *Let s be $\mathcal{B}$-safe and t a data term such that $\varphi(s) \to^*_{\mathcal{R}^\bullet} t$. Then $s \to^*_{\mathcal{R}} t$.*

Proof. By induction on the length of the reduction $\varphi(s) \to^*_{\mathcal{R}^\bullet} t$.

First suppose the reduction contains no steps at the root. Since t is a data term, s must have a constructor as root symbol, so by $\mathcal{B}$-safety s itself is a data term; the reduction is empty. As $\varphi(s) = t$, obviously also $s = t$.

Alternatively, we split the reduction in the first part (without root steps) and the remainder; we may write $s = f(s_1, \dots, s_n)$ and there is a rule $f(\ell_1, \dots, \ell_n) \to r$ in $\mathcal{R}$ and a corresponding rule $\ell' := f(\ell_1^1, \dots, \ell_1^{k_1}, \dots, \ell_n^1, \dots, \ell_n^{k_n}) \to \varphi(r') \in \mathcal{R}^\bullet$, a substitution γ and numbers N, M such that:

- $\varphi(s) = f(\varphi(s_1)^{(1)}, \dots, \varphi(s_n)^{(k_n)}) \to^N_{\mathcal{R}^\bullet} \ell'\gamma$ without root steps;
- $\varphi(r')\gamma \to^M_{\mathcal{R}^\bullet} t$;
- the length of the reduction is $N + M + 1$;

As the first part uses no root steps, we can split it further, letting $N = N_1^1 + \dots + N_1^{k_1} + \dots + N_n^1 + \dots + N_n^{k_n}$ where N_i^j is the number of steps in the reduction $\varphi(s_i) \to^*_{\mathcal{R}^\bullet} \ell_i^j\gamma$.

Now, let $1 \leq i \leq n$. If ℓ_i is a variable, we define $\delta(\ell_i) := s_i$, so we certainly have $s_i \to_{\mathcal{R}}^* \ell_i\delta$. Otherwise, $\ell_i = \ell_i^1$ is a constructor term, and by $\mathcal{B}$-safety (Lemma 5), this implies $\ell_i^1\gamma$ is a constructor term; we define $\delta(x) := \gamma(x)$ for all $x \in Var(\ell_i)$. Then also $s_i \to_{\mathcal{R}}^* \ell_i\delta$ by the induction hypothesis (as $\varphi(s_i) \to_{\mathcal{R}}^* \ell_i^1\gamma = \ell_i\delta$ in $N_i^1 \leq N < N + M + 1$ steps). Thus, $s = f(s_1, \ldots, s_n) \to_{\mathcal{R}}^* f(\ell_1, \ldots, \ell_n)\delta = \ell\delta \to_{\mathcal{R}} r\delta$.

We will see that moreover $\varphi(r\delta) \to_{\mathcal{R}^\bullet}^* \varphi(r')\gamma$ in at most N steps. Then we have $\varphi(r\delta) \to_{\mathcal{R}^\bullet}^* t$ in at most $N + M < N + M + 1$ steps, so $s \to_{\mathcal{R}}^* r\delta \to_{\mathcal{R}}^* t$, and we are done.

To see this, we observe that $\varphi(r\delta) = \varphi(r)\delta^\varphi$, where $\delta^\varphi(x) = \varphi(\delta(x))$ for all x; this holds by a trivial induction on r. Moreover defining η as the extension of δ^φ with mappings $\ell_i^j := \delta^\varphi(\ell_i^1)$ for all i, j such that ℓ_i is a variable and $1 < j \leq k_i$, we have $\varphi(r\delta) = \varphi(r)\delta^\varphi = \varphi(r')\eta$. Thus, it suffices if each $\eta(x) \to_{\mathcal{R}^\bullet}^* \gamma(x)$, using in total no more than N steps.

But this is easy. For all variables x occurring in r', either $x \in Var(\ell_i)$ for some constructor term ℓ_i, or $x = \ell_i^j$ for some i, j. In the former case, $\eta(x) = \delta^\varphi(x) = \gamma(x)$, so this reduction is free. In the latter case, $\eta(x) = \varphi(\delta(\ell_i)) = \varphi(s_i) \to_{\mathcal{R}^\bullet}^* \ell_i^j\gamma$ in N_i^j steps by definition of N_i^j. As every ℓ_i^j can occur at most once (following the semi-linearity observation of Lemma 18), we obtain in total at most N steps. □

Combining both lemmas, we obtain:

Corollary 21. *For every $\mathcal{B}$-safe term $s \in \mathcal{T}(\mathcal{F})$ and every data term t: $s \to_{\mathcal{R}}^* t$ if and only if $\varphi(s) \to_{\mathcal{R}^\bullet}^* t$.*

The relevance of this is evident: to determine whether $\mathtt{start}(s_1, \ldots, s_n) \to_{\mathcal{R}}^* \mathtt{true}$ for some $s_1, \ldots, s_n$, we simply calculate $\mathcal{R}^\bullet$ (which requires only a few purely syntactic changes, and is independent of $\vec{s}$ so can be done in *constant* time) and need only determine whether the semi-linear TRS $\mathcal{R}^\bullet$ admits the reduction $\varphi(\mathtt{start}(s_1, \ldots, s_n)) \to_{\mathcal{R}^\bullet}^* \mathtt{true}$. To further obtain a characterisation result, we merely have to add a new start symbol:

Theorem 22. *"Constrained" cons-free term rewriting characterises PTIME if and only if semi-linear cons-free term rewriting does.*

Proof. We observe:

- For every semi-linear cons-free TRS $(\mathcal{F}^\bullet, \mathcal{R}^\bullet)$ there is a constrained cons-free TRS $(\mathcal{F}, \mathcal{R})$ such that for all lists $s \in \mathcal{T}(\{0, 1, [], ::\})$: $\mathtt{start}(s) \to_{\mathcal{R}^\bullet}^* \mathtt{true}$ if and only if $\mathtt{start}(s) \to_{\mathcal{R}}^* \mathtt{true}$. This follows trivially because every semi-linear TRS *is* constrained, by choosing $\mathcal{A} := \mathcal{D}$.
- For every constrained cons-free TRS $(\mathcal{F}, \mathcal{R})$ there is a constrained cons-free TRS $(\mathcal{F}^{\bullet\prime}, \mathcal{R}^{\bullet\prime})$ such that for all lists $s \in \mathcal{T}(\{0, 1, [], ::\})$: $\mathtt{start}(s) \to_{\mathcal{R}}^* \mathtt{true}$ if and only if $\mathtt{start}'(s) \to_{\mathcal{R}^{\bullet\prime}}^* \mathtt{true}$. This follows by using the transformations described above, and adding a symbol $\mathtt{start}' : 1$ and semi-linear rules $\mathtt{start}'([]) \to \varphi(\mathtt{start}([]))$ and $\mathtt{start}'(x :: y) \to \varphi(\mathtt{start}(x :: y))$. Thus, we do not even need to alter the input to the starting symbol.

Knowing this, every algorithm implemented in one of the styles immediately transfers to an algorithm of the other, as does every algorithm determining the outcome of a reduction in one of the styles. □

5.2. **Call-by-value Reduction.** Now, to draw the connection with Theorem 12, we cannot simply impose a call-by-value strategy and expect to obtain the same normal forms; an immediate counterexample is the following TRS.

$$\mathtt{a} \;\to\; \mathtt{a} \qquad\qquad \mathtt{f}(x) \;\to\; \mathtt{b}$$

Then $\mathtt{f}(\mathtt{a}) \to^*_{\mathcal{R}} \mathtt{b}$, but this normal form is never reached using call-by-value rewriting.

Thus, we will use another simple syntactic adaptation:

Definition 23. We let $\mathcal{F}^\bullet_\bot := \mathcal{F}^\bullet \cup \{\bot\}$, and let $\mathcal{R}^\bullet_\bot := \mathcal{R}^\bullet \cup \{f(x_1,\dots,x_n) \to \bot \mid f : n \in \mathcal{D}\}$. We also include $\bot$ in $\mathcal{B}$.

Including the $\bot$-symbols makes sure that every ground term reduces to a data term, so allows a call-by-value strategy to work even in a non-terminating setting. Otherwise, it has little effect, at least on the "x reduces to data term y" property we are interested in.

In the following, for purposes of induction, let $\mathtt{cost}(s \to^*_{\mathcal{R}^\bullet_\bot} t)$ be the number of $\to^*_{\mathcal{R}^\bullet}$ steps in the given reduction; that is, the length of the reduction when not counting any steps with a rule $f(\dots) \to \bot$.

Lemma 24. *Let s be a $\mathcal{B}$-safe term in $\mathcal{T}(\mathcal{F}^\bullet)$ and t a data term other than $\bot$. Then for any number $N \in \mathbb{N}$: $s \to^*_{\mathcal{R}^\bullet} t$ at cost N if and only if $s \to^*_{\mathcal{R}^\bullet_\bot} t$ at cost N.*

Proof. If $s \to^*_{\mathcal{R}^\bullet} t$, then obviously $s \to^*_{\mathcal{R}^\bullet_\bot} t$ at the same cost, as $\mathcal{R}^\bullet \subseteq \mathcal{R}^\bullet_\bot$. For the other direction, we let $B := \mathcal{R}^\bullet_\bot \setminus \mathcal{R}^\bullet$, so the set of the new rules $f(\vec{x}) \to \bot$, and show that (**) if $u \to^*_B v \to_{\mathcal{R}^\bullet} v'$ for $\mathcal{B}$-safe $u \in \mathcal{T}(\mathcal{F}^\bullet)$ and $v, v' \in \mathcal{T}(\mathcal{F}^\bullet_\bot)$, then exists some $u' \in \mathcal{T}(\mathcal{F}^\bullet)$ such that $u \to_{\mathcal{R}^\bullet} u' \to^*_B v'$. If we have this, then all $\to_{\mathcal{R}^\bullet}$ steps in the reduction $s \to^*_{\mathcal{R}^\bullet_\bot} t$ can be pushed to the left: using induction on the length of the reduction,

- if $s \to^*_B t$ then $s = t$ (since t does not contain $\bot$);
- otherwise, $s \to^*_B u \to_{\mathcal{R}^\bullet} v \to^*_{\mathcal{R}^\bullet_\bot} t$ for some u, v, which implies $s \to_{\mathcal{R}^\bullet} s' \to^*_B v \to^*_{\mathcal{R}^\bullet_\bot} t$ (a reduction of the same cost), with s' also being $\mathcal{B}$-safe by Lemma 5 and in $\mathcal{T}(\mathcal{F}^\bullet)$ because the rules in $\mathcal{R}^\bullet$ do not introduce $\bot$; we obtain $s' \to^*_{\mathcal{R}^\bullet} t$ at cost $N-1$ by the induction hypothesis, s $s \to^*_{\mathcal{R}^\bullet} t$ at ocst N.

It remains to prove the claim (**), so let $u \to^*_B v \to_{\mathcal{R}^\bullet} v'$. By $\mathcal{B}$-safety of u, we can write $u = f(u_1,\dots,u_n)$ with $f \in \mathcal{D}$ (as otherwise $v = u$ does not reduce), and the reduction $u \to^*_B v$ does not take root steps (as otherwise $v = \bot$ does not reduce). Thus, also $v = f(v_1,\dots,v_n)$ with $u_i \to^*_B v_i$ for all i. We find u' by induction on the size of v.

If the step $v \to_{\mathcal{R}^\bullet} v'$ does not take place at the root, then $v' = f(v_1,\dots,v_i',\dots,v_n)$ with $u_i \to^*_B v_i \to_{\mathcal{R}^\bullet} v_i'$. By the induction hypothesis we find u_i' such that $u_i \to_{\mathcal{R}^\bullet} u_i' \to^*_B v_i'$. We are done choosing $u' := f(u_1,\dots,u_i',\dots,u_n)$. Otherwise, $v = \ell\gamma$ and $v' = r\gamma$ for some $\ell \to r \in \mathcal{R}^\bullet$ and substitution γ. Now, for each $1 \le i \le n$:

- if ℓ_i is a variable, let $\delta(\ell_i) = u_i$; then $\delta(x) \to^*_B \gamma(x)$ for all $x \in \mathit{Var}(\ell_i)$;
- otherwise, $\bot \neq \ell_i$ is a constructor term; by $\mathcal{B}$-safety $\bot \neq v_i \in \mathcal{B}$, which gives $u_i = v_i$ since $\mathcal{B}$ has no elements with $\bot$ as a strict subterm; let $\delta(x) = \gamma(x)$ for $x \in \mathit{Var}(\ell_i)$.

By left-linearity, this is well-defined, giving a substitution δ such that $u = \ell\delta$ and each $\delta(x) \to^*_B \gamma(x)$. But then $u \to_{\mathcal{R}^\bullet} r\delta \to^*_B v'$; we are done choosing $u' := r\delta$. □

More importantly, when we consider reductions with $\mathcal{R}^\bullet_\bot$, we are allowed to use a call-by-value strategy.

Lemma 25. *Let s be a $\mathcal{B}$-safe term and t a data term such that $s \to^*_{\mathcal{R}^\bullet} t$. Then $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\bot} t$.*

The core idea is that we can trace the descendant of every non-data subterm t of s in a reduction: due to the semi-linearity of $\mathcal{R}^\bullet$, there will always be at most one copy of t, so there is only one term which t is reduced to. But then, we can do this reduction immediately.

Proof. We prove three claims, by simultaneous induction on (N, M) ordered lexicographically:

(1) if $M = 0$ and $s \to^*_{\mathcal{R}^\bullet} t$ at cost N, then $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ at cost $\leq N$;

(2) if there are terms u and v such that:
- $s \to^*_{\mathcal{R}^\bullet} u$;
- $u \to_{\mathcal{R}^\bullet} v$ by a reduction step at the root;
- $v \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$;
- $M = \mathtt{cost}(s \to^*_{\mathcal{R}^\bullet} u) + 2$ and $N = \mathtt{cost}(s \to^*_{\mathcal{R}^\bullet} u) + \mathtt{cost}(v \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t)$

then $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ by a call-by-value reduction of cost $\leq N + 1$;

(3) if $M = 1$ and $f(s_1, \ldots, s_n) \to^*_{\mathcal{R}^\bullet} \ell\gamma$ by reductions below the root, $\ell \to r \in \mathcal{R}^\bullet$ and $r\gamma \rightsquigarrow_{\mathcal{R}^\bullet_\perp} t$ such that $N = \mathtt{cost}(f(\vec{s}) \to^*_{\mathcal{R}^\bullet} \ell\gamma) + \mathtt{cost}(r\gamma \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t)$, then $f(s_1, \ldots, s_n) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ by a call-by-value reduction of cost $\leq N + 1$.

Consider statement (1). If $N = 0$, so $s = t$, then we are done. Otherwise, the last step of the reduction must be at the root: if $s \to^*_{\mathcal{R}^\bullet} u \to_{\mathcal{R}^\bullet} t$, then by Lemma 5 also u is $\mathcal{B}$-safe, so its root symbol is defined, so if the reduction were not at the root then t would still have a defined root symbol, contradiction. Thus, we conclude with the induction hypothesis, part (2) for $(N - 1, N + 1)$, taking $v = t$: this gives a reduction $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ of cost $\leq N - 1 + 1 = N$.

Consider statement (2). Let $\ell \to r \in \mathcal{R}^\bullet$ and γ be the relevant rule and substitution respectively for the root reduction $u \to_{\mathcal{R}^\bullet} v$. We can write $\ell = f(\ell_1, \ldots, \ell_n)$ and $u = f(\ell_1\gamma, \ldots, \ell_n\gamma)$. Now, if the reduction $s \to^*_{\mathcal{R}^\bullet} u$ uses no root steps, then we can write $s = f(s_1, \ldots, s_n)$ and conclude with the induction hypothesis, part (3) for $(N, 1)$, since $M \geq 2$. If there is a root step, we can write $s \to^*_{\mathcal{R}^\bullet} w$ for some w at cost K_1, $w \to_{\mathcal{R}^\bullet} f(s_1, \ldots, s_n)$ by a root step, $f(s_1, \ldots, s_n) \to^*_{\mathcal{R}^\bullet} u$ by reduction steps below the root at cost K_2 and $v \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ at cost K_3, with $M = K_1 + K_2 + 3$ and $N = K_1 + K_2 + K_3 + 1$.

By the induction hypothesis, part (3) for $(K_2 + K_3, 1)$, $f(\vec{s}) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ by a call-by-value reduction at cost $C \leq K_2 + K_3 + 1$. Then, by the induction hypothesis, part (2) for $(K_1 + C, K_1 + 2)$ – where $K_1 + C \leq K_1 + K_2 + K_3 + 1 = N$ and $K_1 + 2 < M$ – $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ by a call-by-value reduction of cost $\leq K_1 + C + 1 \leq N + 1$ as required.

Consider statement (3). We can reshuffle the reduction $r\gamma \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ such that the steps at the variable positions of r are all performed before any other step (due to the call-by-value nature of the reduction: if the subterm at any of these positions is not yet normalised, then it is not a data term, so other reduction steps can only occur at disjoint positions, so can be postponed). This reshuffling does not affect the cost of the reduction. As r is linear in all variables for which $\gamma(x)$ is not a data term, and all normal forms of $\mathcal{R}^\bullet_\perp$ are data terms (the $\perp$-rules guarantee this), there is a substitution δ mapping $Var(r)$ to data terms, such that the reduction $r\gamma \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$ can safely be assumed to have the form $r\gamma \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} r\delta \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$, where each $\gamma(x) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \delta(x)$. But then there is also such a substitution on $Var(\ell)$: we choose $\delta'(x) := \delta(x)$ for $x \in Var(r)$, $\delta'(x) := \gamma(x)$ for $x \in Var(\ell) \setminus Var(r)$ if $\gamma(x)$ is a data term, and $\delta'(x) := \perp$ if $x \in Var(\ell) \setminus Var(r)$ and $\gamma(x)$ is headed by a defined symbol (these are the only possibilities by $\mathcal{B}$-safety of $\ell\gamma$).

Now, as the reduction from $f(\vec{s})$ to $\ell\gamma$ is fully below the root, we can write $\ell = f(\ell_1, \dots, \ell_n)$, and the reduction $f(\vec{s}) \to^*_{\mathcal{R}^\bullet} \ell\gamma$ consists of the n reductions $s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma$. Thus,

$$N = \left(\sum_{i=1}^{n} \mathtt{cost}(s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma)\right) + \left(\sum_{x \in \mathsf{DV}_{f(\vec{\ell})} \cap Var(r)} \mathtt{cost}(\gamma(x) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \delta(x))\right) + \mathtt{cost}(r\delta \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t)$$

For $1 \le i \le n$, we consider three cases:

- If ℓ_i is a variable with $\delta'(\ell_i) = \perp$, then note that this can only occur if $\ell_i\gamma$ is not a data term (as $\gamma(\ell_i) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \delta(\ell_i)$ and $\gamma(\ell_i) \in \mathcal{T}(\mathcal{F}^\bullet)$). If $\ell_i\gamma$ is not a data term, then neither can s_i be. Thus, by stepwise replacing the innermost non-data subterm by $\perp$, we obtain $s_i \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \perp = \ell_i\delta'$ at cost $0 \le \mathtt{cost}(s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma)$.
- If ℓ_i is a variable, but $\delta'(\ell_i) \neq \perp$ and also $\ell_i\gamma$ is not a data term, then by Lemma 24 we have $s_i \to^*_{\mathcal{R}^\bullet} \delta'(\ell_i)$ at cost $C_i := \mathtt{cost}(s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma = \gamma(\ell_i)) + \mathtt{cost}(\gamma(\ell_i) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \delta'(\ell_i)) \le N$. By the induction hypothesis, part (1) for $(C_i, 0) < (N, 1)$ therefore $s_i \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \ell_i\delta'$ by a call-by-value reduction of at most the same cost.
- If $\ell_i\gamma$ is a data term – which, by $\mathcal{B}$-safety, is always the case if ℓ_i is not a variable – then by the induction hypothesis, case (1) for $(\mathtt{cost}(s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma), 0) < (N, 1)$, $s_i \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \ell_i\gamma$ by a call-by-value reduction of at most the same cost as $s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma$. As $\ell_i\gamma = \ell_i\delta'$ in this case, we obtain $\mathtt{cost}(s_i \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \ell_i\delta') \le \mathtt{cost}(s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma)$.

Thus, overall, $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \ell\delta'$ by a call-by-value reduction of at most cost

$$\left(\sum_{i=1}^{n} \mathtt{cost}(s_i \to^*_{\mathcal{R}^\bullet} \ell_i\gamma)\right) + \left(\sum_{x \in \mathsf{DV}_{f(\vec{\ell})} \cap Var(r)} \mathtt{cost}(\gamma(x) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \delta(x))\right) = N - \mathtt{cost}(r\delta' \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t)$$

As all $\delta'(x)$ are data terms, the step $\ell\delta' \to_{\mathcal{R}^\bullet_\perp} r\delta'$ is also call-by-value (and has cost 1), so we obtain $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \ell\delta' \rightsquigarrow_{\mathcal{R}^\bullet_\perp} r\delta' \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} r\gamma$ at cost $\le N + 1$. □

Binding Lemmas 24 and 25 together, we obtain:

Corollary 26. *For every $\mathcal{B}$-safe term $s \in \mathcal{T}(\mathcal{F}^\bullet)$ and data term t: $s \to^*_{\mathcal{R}^\bullet} t$ iff $s \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} t$.*

6. Conclusion

Combining the results, we thus obtain the statement of [3] with an alternative proof:

Theorem 27. *There is a polynomial algorithm to determine whether a constrained cons-free TRS reduces a basic term $\mathtt{start}(\vec{s})$ to $\mathtt{true}$.*

Proof. Using Definition 15–17, convert the TRS $(\mathcal{F}, \mathcal{R})$ to a semi-linear TRS $(\mathcal{F}^\bullet, \mathcal{R}^\bullet)$ such that $\mathtt{start}(\vec{s}) \to^*_{\mathcal{R}} \mathtt{true}$ if and only if $\varphi(\mathtt{start}(\vec{s})) \to^*_{\mathcal{R}^\bullet} \mathtt{true}$. Use Definition 23 to obtain $(\mathcal{F}^\bullet_\perp, \mathcal{R}^\bullet_\perp)$ such that $\varphi(\mathtt{start}(\vec{s})) \to^*_{\mathcal{R}^\bullet} \mathtt{true}$ if and only if $\varphi(\mathtt{start}(\vec{s})) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \mathtt{true}$. Then, if $\mathtt{start}$ has arity n in the original signature $\mathcal{F}$, add a symbol $\mathtt{start}' : n$ and a rule $\mathtt{start}'(x_1, \dots, x_n) \to \varphi(\mathtt{start}(x_1, \dots, x_n))$, creating the set of rules $\mathcal{R}^\bullet_\perp{}'$. Then we clearly have: $\mathtt{start}(\vec{s}) \to^*_{\mathcal{R}} \mathtt{true}$ if and only if $\mathtt{start}'(\vec{s}) \rightsquigarrow^*_{\mathcal{R}^\bullet_\perp} \mathtt{true}$. By Theorem 12, there is an algorithm to determine whether this holds. □

However, we have done a little more than this:

- we have shown that call-by-value cons-free term rewriting characterises PTIME;
- we have simplified the restrictions, showing that semi-linear cons-free term rewriting characterises PTIME;

In addition, we have shown that, at least in the first-order setting, the question of expressivity of semi-linear TRSs can be directly reduced to the same question for call-by-value TRSs. This is particularly relevant with an eye on future extensions: it may be possible to use the freedom with respect to evaluation strategy inherent in term rewriting to characterise complexity classes we cannot easily handle with call-by-value programs, but in order to succeed we must avoid definitions which admit this transformation.

It should be said that *call-by-value* is not a common strategy in the field of term rewriting; it is more useful to instead impose an *innermost* evaluation strategy, where a term may only be reduced when its immediate subterms are irreducible, rather than data. The difference is evident in those cases where a ground normal form exists which is not a data term; for instance:

$$\mathtt{a}(\mathtt{0}) \;\to\; \mathtt{0} \qquad\qquad \mathtt{b}(x) \;\to\; \mathtt{0}$$

Here, $\mathtt{b}(\mathtt{a}(\mathtt{1}))$ reduces at the root using innermost reduction, but does not reduce at all using call-by-value reduction.

Although not proved in this paper, we can also define a polynomial algorithm to determine whether a basic term reduces to `true` when an innermost strategy is given. We obtain the innermost variant of Corollary 26 immediately:

Theorem 28. *For every $\mathcal{B}$-safe term $s \in \mathcal{T}(\mathcal{F}^\bullet)$ and data term t: $s \to^*_{\mathcal{R}^\bullet} t$ if and only if $s \to^*_{\mathcal{R}^\bullet_\perp} t$ by an innermost reduction.*

Proof. Every call-by-value reduction is obviously an innermost reduction, giving the *only if* direction using Corollary 26. Observing that in $\to_{\mathcal{R}^\bullet_\perp}$, a ground term is in normal form if and only if it is a data term, and therefore every innermost reduction *is* a call-by-value reduction, we obtain the other direction. □

In future work, it would be interesting to see whether the parallel to Jones' result holds up for higher-order, i.e. whether k^{th}-order innermost term rewriting characterises $\text{EXP}^{k-1}\text{TIME}$. In addition, we might consider whether higher-order extensions of semi-linearity do give a truly different way of expressing algorithms than imposing an innermost or call-by-value evaluation strategy.

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.