Complexity classes and rewrite systems with polynomial interpretation

Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, Hélène Touzet

Complexity classes and rewrite systems with polynomial interpretation

G. Bonfante, A. Cichon, J.Y Marion and H. Touzet*

Abstract

We are concerned with functions over words which are computable by a rewrite system admitting polynomial interpretation. We classify them according to a criterion based on the interpretations of successor symbols. This leads to the definition of three classes, which turns out to be exactly the poly-time, linear exponential-time and doubly linear exponential time computable functions. As consequence, we also characterize LINSPACE.

1 Introduction

We are interested in studying the relationship between termination orderings of rewrite systems and feasible computation. One might suspect that polynomial interpretations, introduced by [9], would be a good candidate for the investigation of small complexity classes of functions. Various implementations have been carried out [2, 6, 11, 12]. However, [8] have shown that any rewrite system with a polynomial interpretation termination proof admits doubly-exponential derivations.

The work of [3] initiated an alternative analysis of the effects of termination proofs based on the use of polynomial interpretations. It was shown that a particularly important aspect was the interpretations of the constructors which were restricted to the set $\mathbb{N} = \{0, s\}$, 0 a constant and s a unary successor symbol.

The present paper reconsiders the situation. Firstly, we study functions over strings, that is, rewrite systems based on several successors. Indeed words are the canonical domains for computations. Secondly, we provide a stratification of systems which is related to the kind of polynomials interpreting the successors.

We show that PTIME functions are characterized by rewrite systems admitting polynomial termination proofs where the successors are interpreted by *linear* polynomials with leading coefficient 1. As a consequence, functions over $\mathbb{N}$ turn out to be precisely LINSPACE functions.

Within the same framework, linear exponential time functions, i.e. functions in $\text{ETIME} = \text{DTIME}(2^{0(n)})$, are characterized by rewrite systems admitting polynomial termination proofs where the successors are interpreted by *linear* polynomials. We shall also see that every language in $\text{DTIME}(2^{n^{O(1)}})$ is accepted by this system, with respect to a polynomial reduction.

The linear doubly-exponential time functions, i.e. in $\text{E}_2\text{TIME} = \text{DTIME}(2^{2^{0(n)}})$, are characterized by rewrite systems admitting polynomial termination proofs where the successors are interpreted by *non-linear* polynomials.

Machine independant characterizations of complexity classes were originated by Cobham [4]. This approach is by mean of 'bounded recursion on notation' which imposes global restrictions on the functions' rates of growth. In contrast, in our work, polynomial interpretation imposes a condition on rewrite rules.

*Loria, Callligramme project, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France, {bonfante,cichon,marionjy,touzet}@loria.fr.

Therefore, the different kinds of polynomial interpretations of successors are somewhat akin to the notion of data tiering recently introduced in [1, 10]. It is also worth mentioning the characterization of PTIME functions over finite models in [7], because both consider basically the same system: the Herbrand-Gödel equations.

This paper is organized as follows. Section 2 defines functions computed by rewrite systems with polynomial interpretation termination proofs. Then, the main results with their consequences are presented in Theorem 1. In Section 3, we establish the characterization of PTIME. In Section 4, we examine LINSPACE. The last two sections are devoted to exponential classes and the proofs use results of the previous sections.

2 Computability & polynomial interpretation

The term rewriting notations used throughout are based on [5]. In particular, let $\mathcal{F}$ be a finite set of symbols of fixed arity and $\mathcal{V}$ a denumerable set of variables. $\mathcal{T}(\mathcal{F},\mathcal{V})$ is the term algebra built up from $\mathcal{F}$ and $\mathcal{V}$ and $\mathcal{T}(\mathcal{F})$ is the set of ground terms. The relation $\xrightarrow{+}$ ($\xrightarrow{*}$) denotes the transitive (reflexive-transitive) closure of $\rightarrow$. If u and v are two terms of $\mathcal{T}(\mathcal{F},\mathcal{V})$, we write $u\xrightarrow{!}v$ to mean that $u\xrightarrow{*}v$ and v is in normal form.

2.1 Functions definable by rewrite systems

We shall concentrate on functions over words which are computed by rewrite systems. For this, we specify a set of constructors $\mathbb{W}$ and a disjoint set of defined symbols $\mathbb{F}$. The set of constructors $\mathbb{W}$ contains at least one constant symbol. All other constructors are unary and are called *successors*.

The set of rewrite rules $\mathcal{R}$ over the term algebra $\mathcal{T}(\mathbb{F}\cup\mathbb{W},\mathcal{V})$ defines each function symbol of $\mathbb{F}$. We always assume that $\mathcal{R}$ is terminating (strongly normalizable) and also that $\mathcal{R}$ is confluent and determines for each term a unique normal form in the constructor set $\mathcal{T}(\mathbb{W})$.

It is convenient to present a rewrite system as a tuple $<\mathcal{R},\mathbb{F},\mathbb{W},f>$, where

- $\mathbb{W}$ is the set of constructors;
- $\mathbb{F}$ is the set of defined function symbols;
- $\mathcal{R}$ is the set of rewrite rules;
- f is the main function symbol of $\mathbb{F}$;

Given an alphabet Σ and a constructor set $\mathbb{W}$, define an *encoding function* α as an injective morphism of $\Sigma\rightarrow\mathbb{W}$ which is extended canonically to Σ^* in the usual manner :

$$\begin{array}{rcll}\alpha(\varepsilon) & = & \epsilon & (\varepsilon \text{ is the empty word of } \Sigma^*).\\ \alpha(iu) & = & \alpha(i)(\alpha(u)) & (i\in\Sigma)\end{array}$$

A pair (α,β) of encoding functions is an encoding pair if whenever $\alpha(i)=\beta(j)$ then $i=j$.

A function $\phi : (\Sigma^*)^n \mapsto \Sigma^*$ is said to be *computed* by the rewrite system $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f >$ if there is a pair (α, β) of encoding functions such that, for all $w_1, \ldots, w_n, v \in \Sigma^*$

$$f(\alpha(w_1), \ldots, \alpha(w_n)) \stackrel{!}{\rightarrow} \beta(v) \quad \Leftrightarrow \quad \phi(w_1, \ldots, w_n) = v$$

This definition allows to differentiate between input constructors and output constructors, a feature which will be necessary soon.

Example 1. The rewrite system $< \mathcal{A}, \{\mathtt{Add}, \mathtt{Mul}, \mathtt{Sq}\}, \mathbb{N}, \mathtt{Sq} >$ where $\mathbb{N}$={0,s} defined the tally square function.

$$\mathcal{A} \left\{ \begin{array}{rcl} \mathtt{Add}(0, y) & \rightarrow & y \\ \mathtt{Add}(sx, y) & \rightarrow & s(\mathtt{Add}(x, y)) \\ \mathtt{Mul}(0, y) & \rightarrow & 0 \\ \mathtt{Mul}(sx, y) & \rightarrow & \mathtt{Add}(y, \mathtt{Mul}(x, y)) \\ \mathtt{Sq}(x) & \rightarrow & \mathtt{Mul}(x, x) \end{array} \right.$$

Each number n is represented in unary, say over $\{|\}^*$, by $\underline{n} = |^n$. Then, $\mathtt{Sq}$ is interpreted by one encoding function $\alpha : \alpha(\epsilon) = 0$ and $\alpha(\underline{n+1}) = s(\alpha(\underline{n}))$. Hence, $\mathtt{Sq}(\alpha(\underline{n})) \stackrel{!}{\rightarrow} \alpha(\underline{n^2})$

2.2 Polynomial interpretation

A *polynomial interpretation termination proof* for a rewrite system $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f >$ consists in the assignment to each function symbol g of $\mathbb{F} \cup \mathbb{W}$, a polynomial $[g]$ with non-negative integer coefficients which satisfies the following conditions. If the arity of g is k then $[g]$ is a polynomial with k variables. $[g]$ is monotonic, that is,

$$x < y \quad \Leftarrow \quad [g](\cdots, x, \cdots) < [g](\cdots, y, \cdots)$$

Also, if the arity of g is 0 then $[g] > 1$, otherwise $[g](x_1, \cdots, x_n) > x_i$, for $i = 1, n$ and for all $x_j > 0$, $j = 1, n$.

[ ] is extended canonically to $\mathcal{T}(\mathbb{F} \cup \mathbb{W})$ as follows

$$[g(t_1, \ldots, t_n)] = [g]([t_1], \ldots, [t_n]).$$

Lastly, [ ] must ensure that for all rules $l \rightarrow r$ of $\mathcal{R}$

$$[l] > [r]$$

for all values of variables greater than the minimum of the interpretations of the constants.

Example 2. The rewrite system of example 2 admits the following interpretation

$$\begin{array}{rcl} [0] & = & 2 \\ [s](x) & = & x + 1 \\ [\mathtt{Add}](x, y) & = & 2x + y, \\ [\mathtt{Mul}](x, y) & = & 3xy \\ [\mathtt{Sq}](x) & = & 3x^2 + 1 \end{array}$$

¿From now on, a rewrite system $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f >$ with a polynomial interpretation $[\,]$ is denoted by $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$.

We end this section by giving some general properties of such systems which will be used later on. Firstly, whenever $u \xrightarrow{+} v$, $[v] < [u]$. This observation implies that the length of any derivation starting from a term t is bounded by $[t]$.

Lemma 1. *A rewrite system with a polynomial interpretation is terminating.*

Define the *size* $|t|$ of a term t as follows:

$$|t| = \begin{cases} 1 & \text{if } t \text{ is a constant or a variable} \\ \sum_{i=1}^{n} |t_i| + 1 & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Lemma 2. *Let $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ be a rewrite system. For all terms t,*

$$|t| \leq [t].$$

Proof. The proof goes by induction on $|t|$. The result is obvious if t is a constant because $|t| = 1$ and also $[t] > 0$ by assumption on $[\,]$. Otherwise, t is of the form $f(t_1, \ldots, t_n)$ and so

$$\begin{array}{lcll} |f(t_1, \ldots, t_n)| & = & 1 + \sum_{i=1,n} |t_i| & \text{(by size def.)} \\ & \leq & 1 + \sum_{i=1,n} [t_i] & \text{(by ind. assumption)} \end{array}$$

Since, for all $i = 1..n$, we require that for all $x_1, \ldots, x_n$ strictly greater than 0, $[f(x_1, \cdots, x_n)] > x_i$, we deduce that $[f(x_1, \cdots, x_n)] > \sum_{i=1,n} x_i$ So, we obtain

$$|f(t_1, \ldots, t_n)| \quad \leq \quad 1 + \sum_{i=1,n} [t_i] \quad \leq \quad [f(x_1, \cdots, x_n)]$$

□

2.3 Classes of functions and results

We shall now examine the intent of the successor interpretations in computations, as initiated by [8, 3]. Those interpretations will play a central role throughout as the following example shows.

Example 3. On the constructor set $\mathbb{N}_1 = \{0, s, q\}$, we define the factorial function `Fact` by

$$\begin{cases} \texttt{Fact}(0) & \to & s(0) \\ \texttt{Tr}(q(x)) & \to & s(\texttt{Tr}(x)) \\ \texttt{Tr}(0) & \to & 0 \\ \texttt{Fact}(q(x)) & \to & \texttt{Mul}(s(\texttt{Tr}(x)), \texttt{Fact}(x)) \end{cases}$$

The rules for `Mul` are provided in example 1. Clearly, $\texttt{Fact}(q^n(0)) \xrightarrow{+} s^{n!}(0)$, but we have two representations for natural numbers. Inputs are encoded with q whereas outputs with s. To explictly say what the function computed by `fact` is, we provide an encoding pair (α, β). As previously, over $\{|\}^*$, we set $\alpha(|) = q$ and $\beta(|) = s$. Hence, `Fact` represents the factorial function since $\texttt{Fact}(\alpha(\underline{n})) \xrightarrow{!} \beta(\underline{n!})$.

Now, to give a polynomial interpretation of `Fact`, we must interpret both successors s and q in a different way, say by $[s](x) = x+1$ and $[q](x) = 3(x+2)^2$. It is then routine to check that the following interpretation works for `Tr` and `Fact`:

$$\begin{aligned} [\texttt{Tr}](x) &= x+1 \\ [\texttt{Fact}](x) &= x+2 \end{aligned}$$

`Tr` is a coercion function that translates data of some kind to data of a lower kind.

In fact, successor interpretations fall into three categories

kind 0: polynomials of the form $P(X) = X + c$ $(c > 0)$,

kind 1: polynomials of the form $P(X) = aX + b$ $(a > 1, b \geq 0)$

kind 2: polynomials of the form $P(X) = aX^d + R(X)$ $(a > 0,\ d > 1$ and R is a polynomial of degree strictly less than $d)$

This classification allows us to distinguish three classes of rewrite systems.

Definition 1. Let $i \in \{0, 1, 2\}$. A rewrite system $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ is $\Pi(i)$ if, for each successor s of $\mathbb{W}$, the interpretation $[s]$ is a polynomial of kind less than or equal to i. A function ϕ is $\Pi(i)$-*computable* if there is a $\Pi(i)$ rewrite system which computes ϕ.

For instance, example 1 shows that addition and multiplication are $\Pi(0)$-computable functions and example 3 that the factorial function is $\Pi(2)$-computable. We now state our main result.

Theorem 1.

1. *The* $\Pi(0)$*-computable functions are exactly the* PTIME *functions.*
2. *The* $\Pi(1)$*-computable functions are exactly the* ETIME *functions, that is, the functions computable in time* $2^{O(n)}$*.*
3. *The* $\Pi(2)$*-computable functions are exactly the* E_2TIME *functions, that is, the functions computable in time* $2^{2^{O(n)}}$*.*

Proof. The proof goes as follows. Lemma 6 shows how to simulate $\Pi(0)$-computable functions in PTIME. The converse is established in Lemma 9.

The characterization of ETIME is a consequence of Lemma 13 and Lemma 11. Lastly, E_2TIME is obtained by applying Lemmas 17 and 15 □

There are several corollaries worth noting. The first is a characterization of LINSPACE in Section 4. The second is based on a padding argument.

Corollary 1. *There is a* $\Pi(1)$ *rewrite system that recognizes every language* L *which is contained in* DTIME(2^{n^k}) *for some* k*,* with respect to a polynomial time reduction.

Indeed, let L be a DTIME(2^{n^k}) language. Say that L is *accepted* by the TM M. Then, construct $L' = \{x@0^{n^k} \text{ with } x \in L\}$, that is, a word of x is padded by extra 0's to have a length equal to the runtime of M. Therefore L' is decided in time $2^{O(n)}$. The characteristic function of L' is $\Pi(1)$ computable by (2) of Theorem 1.

Finally, a similar result can be established for languages in DTIME$(2^{2^{n^k}})$ using the same padding technique.

2.4 General properties

For use as Lemmas, we now establish some properties of $\Pi(i)$-systems.

Firstly, we show a property of weak closure by composition of $\Pi(i)$-computable functions, enabling us to combine systems to define functions in a modular way.

Lemma 3. *Let ϕ_i be a $\Pi(i)$-computable function and ϕ be a $\Pi(0)$-computable function. The function ψ defined by $\psi(\vec{x},\vec{y}) = \phi(\phi_i(\vec{x}),\vec{y})$ is $\Pi(i)$-computable.*

Proof. Assume that ϕ is computed by the $\Pi(0)$-rewrite system $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ with the pair (α,β) of encoding functions. Assume that ϕ_i is computed by the $\Pi(i)$-rewrite system $< \mathcal{R}_i, \mathbb{F}_i, \mathbb{W}_i, f_i, [\,]_i >$ with the encoding (α_i,β_i) verifying Without loss of generality, we also suppose that both $\mathbb{F}$ and $\mathbb{F}_i$ are disjoint and also that $\mathbb{W}$ and $\mathbb{W}_i$ are two disjoint copies of the same set of constructors. This means that there is a one-one mapping τ from $\mathbb{W}_i$ to $\mathbb{W}$ which respect constructor arities.

The key point of the construction is the introduction of a coercion function $\mathtt{Tr}$ that translates terms of $\mathbb{W}_i$ into terms of $\mathbb{W}$. $\mathtt{Tr}$ is defined thus :

$$\begin{array}{rcll} \mathtt{Tr}(\epsilon) & \rightarrow & \tau(\epsilon) & \epsilon \in \mathbb{W}_i \text{ is 0-ary} \\ \mathtt{Tr}(s(x)) & \rightarrow & \tau(s)\mathtt{Tr}(x) & s \in \mathbb{W}_i \text{ is a successor} \end{array}$$

where the function τ verifies $\alpha = \tau \circ \alpha_i$ and $\alpha = \tau \circ \beta_i$.

The function ψ is computed by $< \mathcal{R}_*, \mathbb{F} \cup \mathbb{F}_i, \mathbb{W} \cup \mathbb{W}_i, f_*, [\,]_* >$ where the set of rules $\mathcal{R}_*$ just contains the set $\mathcal{R} \cup \mathcal{R}_i$, the above rules for $\mathtt{Tr}$ and the following rule for f:

$$f_*(\vec{x}, y_1, \ldots, y_n) \rightarrow f(\mathtt{Tr}(f_i(\vec{x})), \mathtt{Tr}(y_1), \ldots, \mathtt{Tr}(y_n))$$

The encoding pair interpreting f_* is (α_i,β).

The polynomial interpretation $[\,]_*$ is an extension of $[\,]$ and $[\,]_i$. Let $a = \max\{c;\ [\epsilon] = c \text{ or } [s](x) = x + c \text{ where } \epsilon, s \in \mathbb{W}\} + 1$.

$$\begin{array}{rcl} [X]_* & = & [X], \text{ if } X \in \mathbb{F} \cup \mathbb{W} \\ [X]_* & = & [X]_i, \text{ if } X \in \mathbb{F}_i \cup \mathbb{W}_i \\ [\mathtt{Tr}]_*(x) & = & ax \\ [f_*]_*(\vec{x}, y_1, \ldots, y_n) & = & [f]_*([\mathtt{Tr}]_*[f_i]_*(\vec{x}), [\mathtt{Tr}]_*(y_1), \cdots, [\mathtt{Tr}]_*(y_n)) + 1 \end{array}$$

□

The next Lemma is convenient for determining upper bounds on the runtime of $\Pi(i)$-computable functions. The proposition claims that both (1) the length of the derivation and (2) the size of any terms involved during the reduction process are polynomially bounded in the interpretation of the inputs.

Definition 2. A class $\mathcal{C}$ of unary increasing functions over natural numbers *accommodates polynomials* iff for all $\phi \in \mathcal{C}$, for all polynomials P with natural number coefficients, there is a function $\phi' \in \mathcal{C}$ such that $P(\phi(x)) < \phi'(x)$, for all $x > 0$.

In the sequel, we shall deal with three main classes of functions: polynomials, exponentials $\{2^{cx}\ ;\ c > 0\}$ and doubly exponentials $\{2^{2^{cx}}\ ;\ c > 0\}$. It is clear that these classes accommodate polynomials.

Lemma 4. *Let $\mathcal{C}$ be a class of functions which accommodates polynomials. Let $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ be a rewrite system. Assume that there is $\phi \in \mathcal{C}$ such that, for all terms t in $\mathcal{T}(\mathbb{W})$, $[t] \leq \phi(|t|)$. Then there is $\phi' \in \mathcal{C}$, such that for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$ and $f \in \mathbb{F}$, the following holds:*

(i) $[f(t_1, \cdots, t_n)] \leq \phi'(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$.

(ii) *The length of any derivation starting from $f(t_1, \cdots, t_n)$ is bounded by* $\phi'(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$..

(iii) *If $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq \phi'(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$.*

Proof. Let $m = \max\{|t_i|, 1 \leq i \leq n\}$. By hypothesis, we have

$$\begin{array}{rcl} [f(t_1, \ldots, t_n)] & \leq & [f](\phi(m), \ldots, \phi(m)) \\ & \leq & \phi'(m) \text{ for some } \phi' \text{ in } \mathcal{C} \end{array}$$

So (i) is proved. (ii) is a consequence of (i) since the length of any derivation is bounded by the polynomial interpretation of the term reduced. Finally, Lemma 2 implies $|v| \leq [v]$. And since $[v] \leq [f(t_1, \cdots, t_n)]$, we complete (iii) by applying (i) again. □

3 PTIME is $\Pi(0)$

3.1 $\Pi(0)$-computable functions are PTIME

Lemma 5. *Let $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ be a $\Pi(0)$ rewrite system. Then there is a constant c such that for all $t \in \mathcal{T}(\mathbb{W})$ $[t] \leq c.|t|$.*

Proof. By construction of $\Pi(0)$, for all constructors in $\mathbb{W}$, there exists $c > 0$ such that $[\epsilon] \leq c$ and $[s](x) \leq x + c$. So, for all $t \in \mathcal{T}(\mathbb{W})$, we have, by composition of interpretations, $[t] \leq c.|t|$. □

Corollary 2. *Let $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ be a $\Pi(0)$ rewrite system. Then there is a polynomial P_f such that for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$,*

1. *the length of any derivation starting from $f(t_1, \ldots, t_n)$ is bounded by $P_f(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$.,*
2. *if $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq P_f(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$.*

Proof. (1) is a consequence of Lemma 4-(ii) and Lemma 5. (2) is a consequence of Lemma 4-(iii) and Lemma 5. □

Lemma 6. *If ϕ is $\Pi(0)$-computable, then ϕ is in* PTIME.

Proof. Let Σ be the alphabet for ϕ. Suppose that ϕ is computed by the system $< \mathcal{R}, \mathbb{W}, \mathbb{F}, f, [\,] >$ in $\Pi(0)$ and let $w_1, \ldots, w_n$ be words in Σ^*. Since all derivations starting from $f(\alpha(w_1), \ldots, \alpha(w_n))$ lead to the same normal form, we take any strategy to compute the normal form of $f(\alpha(w_1), \ldots, \alpha(w_n))$. The key point is that at any step the size of the term is bounded by $O(\max(|w_i|)^p)$ ((iii) of Lemma 2). So, in time bounded by $O(\max(|w_i|)^p)$, we select a redex and replace it by the right hand side of the corresponding rule of $\mathcal{R}$. The reduction lasts for at most $O(\max(|w_i|)^q)$ (see (ii) of Lemma 2). It follows that the runtime of the computation of $\phi(w_1, \ldots, w_n)$ is bounded by $O(\max(|w_i|)^{p+q})$. □

3.2 PTIME functions are $\Pi(0)$-computable

In this section we show that PTIME functions are $\Pi(0)$-computable by simulating a Turing Machine by a rewrite system with polynomial interpretations. The outline of the proof is as follows. First of all, we construct a rewrite system in $\Pi(0)$ that simulates T steps of the computation of a Turing machine. This will be done in Lemma 7. This part is independant from PTIME: it does not take advantage of the polynomial bound on the computation runtime. Then we show in Lemma 8 that polynomials can be computed by $\Pi(0)$ rewrite systems. Finally, we conclude by composing both results.

3.2.1 Encoding a time bounded Turing machine

We consider multi-stack Turing machines, abbreviated STM. Of course, this computational model delineates the same computational complexity classes as, say, multi-tape Turing machines. But, this model makes the proof easier and is convenient for the discussion, further on in Section 4, of computation on a unary alphabet.

Formally, a k-stack TM, $\mathcal{M}$, is defined by a tuple $\mathcal{M} =< \Sigma, \epsilon, Q, q_0, Q_f, \delta >$ where Σ is the alphabet and ϵ is the bottom stack symbol; Q is the set of states with $q_0 \in Q$ as initial state and $Q_f \subseteq Q$ is the set of final states; and finally the transition function is $\delta : Q \times (\Sigma \cup \{\epsilon\})^k \to Q \times (\Sigma^*)^k$.

The meaning of $\delta(q, a_1, \cdots, a_k) = (q', u_1, \cdots, u_k)$ is the following. The STM is in state q and the letter on the top of the ith stack is a_i. Then, the STM replaces each a_i by the word u_i and switches to the state q'.

A configuration of the machine is a tuple $< q, w_1, \cdots, w_k >$ where $q \in Q$ is the current state, and w_i is the content of the ith stack. Let $\Rightarrow$ be the relation which provides the next configuration of δ. We define for $\mathcal{M}$ a function $F[\mathcal{M}] : (\Sigma^*)^k \mapsto \Sigma$ by $F[\mathcal{M}](w_1, \cdots, w_k) = r_k$ iff the normal form of $(q_0, w_1, \cdots, w_k)$ is r_k.

Lemma 7. *Let $\mathcal{M} =< \Sigma, \epsilon, Q, q_0, Q_f, \delta >$ be a STM. Then there is a $\Pi(0)$-computable function ϕ_M such that, for each input $(w_1, \cdots, w_k)$, if $\mathcal{M}$ halts in less than t steps then $\phi_M(t, w_1, \cdots, w_k) = F[\mathcal{M}](w_1, \cdots, w_k)$.*

Proof. We construct a rewrite system $\mathcal{R}_\delta$ which computes ϕ_M as follows:

- Constructors are $\mathbb{W} = \{s_i \quad i \in \Sigma\} \cup \{\epsilon\}$,
- the defined symbols are $\{\mathtt{q}\ ;\ q \in Q\}$, where, for all $q \in Q$, $\mathtt{q}$ is a function symbol of arity $k+1$. The first parameter corresponds to the remaining computation runtime and the k other parameters to stacks,
- the encoding pair is (α, α) with $\alpha(i) = s_i$ for all $i \in \Sigma$.

The rewrite rules are given by the following template:
If $\delta(q, a_1, \cdots, a_k) = (q', u_1, \cdots, u_k)$ then

$$\mathtt{q}(st, \alpha(a_1)x_1, \cdots, \alpha(a_k)x_k) \quad \to \quad \mathtt{q}'(t, \alpha(u_1)(x_1), \cdots, \alpha(u_1)x_k)$$

with the convention that $\alpha(\epsilon)x_i = \epsilon$, $\alpha(a_i)x_i = s_i(x_i)$ if $a_i \in \Sigma$, and $\alpha(a_i v)x_i = s_i(\alpha(v)(x_i))$. Otherwise, $q_f \in Q_f$ and $\mathtt{q}_f(st, x_1, \cdots, x_k) \to x_k$. It is straightforward to verify that

$$< q, w_1, \cdots, w_k > \Rightarrow < q', w'_1, \cdots, w'_k >$$

if and only if

$$\mathtt{q}(st, \alpha(w_1), \cdots, \alpha(w_k)) \to \mathtt{q}'(t, \alpha(w'_1), \cdots, \alpha(w'_k))$$

and that

$$q \text{ is a final state} \quad \text{iff} \quad \mathtt{q}(t, \alpha(w_1), \cdots, \alpha(w_k)) \to \alpha(w_k)$$

Therefore, if $\mathcal{M}$ halts in less than t steps on inputs $w_1, \ldots, w_k$ then the result of the computation is provided by the normal form of $\mathtt{q}_0(t, \alpha(w_1), \cdots, \alpha(w_k))$, which is exactly $F[\mathcal{M}](w_1, \cdots, w_k)$. So ϕ_M is represented by f_M, thus

$$f_M(t, x_1, \cdots, x_k) \to \mathtt{q}_0(t, x_1, \cdots, x_k)$$

Lastly, we interpret each function symbol by

$$\begin{array}{rcll} [\epsilon] & = & 2 & \\ [s_i](x) & = & x+1 & \forall i \in \Sigma \\ [\mathtt{q}](t, x_1, \cdots, x_k) & = & k.c.t + x_1 + \cdots + x_k & \forall q \in Q \\ [f_M] & = & [\mathtt{q}_0] + 1 & \end{array}$$

where the constant c is strictly greater than the interpretation of any word that is involved in the definition of δ. We conclude that the system is $\Pi(0)$. □

3.2.2 Simulating Poly-time

Lemma 8. *Each polynomial is* $\Pi(0)$*-computable.*

Proof. Each polynomial is defined by composition of addition and multiplication. Example 1 shows that addition and multiplication are $\Pi(0)$-computable functions. Lemma 3 leads then to conclusion. □

Lemma 9. *If* ϕ *is in* PTIME*, then* ϕ *is* $\Pi(0)$*-computable.*

Proof. Let ϕ be a function, computed by a Turing machine $\mathcal{M}$, such that the time of computation is bounded by a polynomial P. The function ϕ is obtained by composing ϕ_M, as defined in Lemma 7, and P. Since ϕ_M and P are both $\Pi(0)$-computable, ϕ is also $\Pi(0)$-computable by lemma 3. □

4 A characterization of LINSPACE

A function is in Linspace if it is computed by a multi-stack Turing machine over an alphabet with at least two letters running in linear space. The proof follows [10] which depends essentially on [7].

Theorem 2. *A function* ϕ *is computable over a multi-stack Turing machine over a unary alphabet in polynomial-time iff* ϕ *is in* LINSPACE*.*

Proof. Let ϕ be an m-ary function. Assume that ϕ is computed by a k-stack Turing machine M which works on the unary alphabet $\{|\}$ and whose runtime is bounded by $P(w_1, \cdots, w_m)$ for some polynomial P and for all inputs w_1, $\ldots$,w_m. From M, we construct a $(k+1)$-stack Turing machine N over the binary alphabet $\{0, 1\}$. The stack i, $i = 1, k$, of N contains the binary representation

of the number of |'s in the ith stack of M. Now, observe that M's operations just consist in adding or removing some fixed amount of |'s. So when, M adds c |'s to some stack, for example, N will also add c to the same stack in base two. But, this is easily performed by N with the spare stack. Thus, the size of each stack of N is bounded by $O(\log(\max_{i=1,m}(w_i)))$.

Conversely, assume that ϕ is computed by a k-stack Turing machine M over, say, $\{a,b\}^*$. Define $\underline{u}$ to be the dyadic representation of the word $u \in \{a,b\}^*$. (That is $\underline{\epsilon} = 0$, $\underline{au} = 2.\underline{u} + 1$ and $\underline{bu} = 2.\underline{u} + 2.$) We build a $(k+2)$-stack Turing machine N on the unary alphabet $\{|\}$ as follows. If there is u in the stack i of M then there are $\underline{u}$ |'s in the stack i of N. Say that M pushes a onto a stack, N doubles the number of |'s in this stack and then adds |. To multiply by two, N uses the two extra stacks to duplicate the stack. N proceeds similarly to push or pop a word given by the transition function of M. The runtime of M is bounded by $2^{c.n}$ where n is maximum size of the inputs. So, the runtime of N is linear in $2^{c.n}$, that is, polynomial in the greatest input value. □

Theorem 3. *A function is computed by a* $\Pi(0)$ *rewrite-system* $< \mathcal{R}, \mathbb{F}, \mathbb{N}, f, [\,] >$ *over the domain* $\mathbb{N} = \{s, 0\}$ *iff it is* LINSPACE.

Proof. Let ϕ be a function computed by the $\Pi(0)$ rewrite-system $< \mathcal{R}, \mathbb{F}, \mathbb{N}, f, [\,] >$. By Lemma 6, ϕ is computable in polynomial time over a unary alphabet. So, by Theorem 2, ϕ is in LINSPACE. Conversely, if ϕ is in LINSPACE, then Theorem 2 yields that ϕ is computable in polynomial time on some stack TM which works on a unary alphabet. Therefore, by Lemma 9, ϕ is $\Pi(0)$-computable. □

5 ETIME is $\Pi(1)$

Define ETIME as the class of functions which are computable in time bounded by $2^{O(n)}$ on Turing machines where n is the maximum size of the inputs.

5.1 $\Pi(1)$-computable functions are ETIME

Lemma 10. *Let* $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ *be a* $\Pi(1)$ *rewrite system. Then there is a constant* c *such that, for all* $t \in \mathcal{T}(\mathbb{W})$, $[t] \leq 2^{c|t|}$.

Proof. By definition of $\Pi(1)$, for all constructors in $\mathbb{W}$, there is a constant c such that $[\epsilon] \leq c$ and and $[s](x) \leq c.x$, when $x > 0$. It is clear that

$$[s(t)] \quad \leq \quad c.[t] \qquad \leq \quad c.2^{c.|t|} \qquad \leq \quad 2^{c(|t|+1)} = 2^{c|st|}$$

□

Corollary 3. *Let* $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ *be a* $\Pi(1)$ *rewrite system. Then there is a constant* c *such that, for all terms* $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$,

1. *the length of any derivation starting from* $f(t_1, \ldots, t_n)$ *is bounded by* $2^{c \max\{|t_i| \,;\, 1 \leq i \leq n\}}$,
2. *if* $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, *then* $|v| \leq 2^{c \max\{|t_i| \,;\, 1 \leq i \leq n\}}$.

Proof. Consequence of Lemma 4 and Lemma 10. □

Lemma 11. *If* ϕ *is* $\Pi(1)$*-computable, then* ϕ *is in* ETIME.

Proof. See the proof of Lemma 6. □

5.2 ETIME functions are $\Pi(1)$-computable

Lemma 12. *Let c be a constant. The function $\lambda n.\ 2^{c.n}$ is $\Pi(1)$-computable*

Proof. Let $< \mathcal{A}, \{\mathtt{Add}, \mathtt{Mul}, \mathtt{E}, \{0, s, q\}, \mathtt{E} >$ be the rewrite system defined as in example 1 with the following rules :

$$\begin{cases} \mathtt{E}(0) & \to & s0 \\ \mathtt{E}(qx) & \to & \mathtt{Add}(\mathtt{E}(x), \ldots \mathtt{Add}(\mathtt{E}(x), 0)) \quad (2^c \text{ occurrences of } \mathtt{Add}) \end{cases}$$

$\mathtt{E}$ represents $\lambda n.\ 2^{cn}$ through the encoding pair (α, β) where $\alpha(1) = q$ and $\beta(1) = s$. The rewrite system is $\Pi(1)$:

$$\begin{array}{rcl} [q](x) & = & 2^{c+1}(x+2) \\ [\mathtt{E}](x) & = & x+2 \end{array}$$

□

Lemma 13. *If ϕ is in* ETIME*, then ϕ is $\Pi(1)$-computable.*

Proof. Let ϕ be a function, computed by a Turing machine $\mathcal{M}$, such that the time of computation is bounded by an exponential 2^{cn} for some constant c. The function ϕ can be obtained by composing ϕ_M as defined in Lemma 7 and $\lambda n\ 2^{cn}$. Since ϕ_M is $\Pi(0)$ and $\lambda n\ 2^{cn}$ is $\Pi(1)$, ϕ is $\Pi(1)$-computable by Lemma 3. □

6 $\Pi(2)$ is E_2TIME

6.1 $\Pi(2)$-computable functions are E_2TIME

Lemma 14. *Let $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ be a $\Pi(2)$ rewrite system. Then there is a constant $c > 0$ such that, for all $t \in \mathcal{T}(\mathbb{W})$, $[t] \leq 2^{2^{c|t|}}$.*

Proof. By definition of $\Pi(2)$, there is a constant a such that $[\epsilon] \leq a$ and $[s](x) \leq ax^a$, for $x > 0$ and for all constructors. Define $c = 2a$. It is easy to verify that

$$[s(t)] \quad \leq \quad a.[t]^a \quad \leq \quad a.2^{2^{c.|t|^a}} \quad \leq \quad 2^{2^{c.|t|}2a} \quad \leq \quad 2^{2^{c.|t|}+2a} \quad \leq \quad 2^{2^{c.(|t|+1)}}$$

□

Corollary 4. *Let $< \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] >$ be a $\Pi(2)$ rewrite system. Then there is a constant c such that, for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$,*

1. *the length of any derivation starting from $f(t_1, \ldots, t_n)$ is bounded by $2^{2^{c \max\{|t_i|\ ;\ 1 \leq i \leq n\}}}$.*
2. *if $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq 2^{2^{c.\ \max\{|t_i|\ ;\ 1 \leq i \leq n\}}}$.*

Proof. Consequences of Lemmas 4 and 14. □

Lemma 15. *If ϕ is $\Pi(2)$-computable, then ϕ is in* E_2TIME*.*

6.2 E_2TIME functions are $\Pi(2)$-computable

Lemma 16. *Let c be a constant. The function $\lambda n\ 2^{2^{cn}}$ is $\Pi(2)$-computable.*

Proof. As in example 1, we define $< \mathcal{A}, \{\mathtt{Add}, \mathtt{Mul}, \mathtt{D}\}, \{0, s, q\}, \mathtt{D} >$ where $\mathtt{D}$ is defined thus,

$$\begin{cases} \mathtt{D}(0) & \rightarrow & ss0 \\ \mathtt{D}(qx) & \rightarrow & \mathtt{Mul}(\mathtt{D}(x), \ldots \mathtt{Mul}(\mathtt{D}(x), s0) \quad (2^c \text{ occurrences of } \mathtt{Mul}) \end{cases}$$

$\mathtt{D}$ represents $\lambda n\ 2^{2^{cn}}$ through the encoding pair (α, β) where $\alpha(1) = q$ and $\beta(1) = s$. The rewrite system is $\Pi(2)$:

$$\begin{array}{rcl} [q](x) & = & (3^{2^c}2)(x+3)^{2^c} \\ [\mathtt{D}](x) & = & x+3 \end{array}$$

□

Lemma 17. *If ϕ is in E_2TIME, then ϕ is $\Pi(2)$-computable.*

Proof. Let ϕ be a function computed by a Turing machine $\mathcal{M}$, such that the time of computation is bounded by a doubly exponential function, D. The function ϕ is obtained by composing ϕ_M as defined in Lemma 7 and D. Since ϕ_M is $\Pi(0)$-computable and D is $\Pi(2)$-computable, Lemma 3 implies that ϕ is $\Pi(2)$-computable. □

References

[1] S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of the poly-time functions*, Computational Complexity,2, 1992, p. 97–110.

[2] A. Ben Cherifa and P. Lescanne, *Termination of rewriting systems by polynomial interpretations and its implementation.* Science of computer Programming 9 (1987), p. 131-159.

[3] E.A Cichon and P. Lescanne, *Polynomial interpretations and the complexity of algorithms.* CADE'11 (1992), p 139-147.

[4] A. Cobham, *The intrinsic computational difficulty of functions*, ed. Y. Bar-Hillel, Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science, North-Holland, Amsterdam, 1962, p. 24-30.

[5] N. Dershowitz and J.P. Jouannaud, *Rewrite systems.* Handbook of Theoretical Computer Science vol.B, North-Holland.

[6] J. Giesl, *Generating polynomial orderings for termination proofs.* RTA-95, Lecture Notes in Computer Science 914, p. 427-431.

[7] Y. Gurevich, *Algebras of feasible functions.*, Twenty Fourth Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1983, p. 210–214.

[8] D. Hofbauer and C. Lautemann, *Termination proofs and the length of derivations.* RTA-88, Lecture Notes in Computer Science 355.

[9] D.S. Lankford, *On proving term rewriting systems are Noetherien.* Technical Report Memo MTP-3, louisiana Technical University, Ruston, LA (1979).

[10] D. Leivant, *Predicative recurrence and computational complexity I: Word recurrence and poly-time*, Feasible Mathematrics II, ed. Peter Clote and Jeffery Remmel, Birkhauser-Boston, 1994.

[11] P. Lescanne, *Computer experiments with the REVE term rewriting system generator.* Tenth ACM symposium on principles of programming languages, Austin, Texas (1983), p. 99-108

[12] J. Steinbach, *Generating polynomial orderings.* Information Processing Letters 49 (1994), p. 85-93.