J. Functional Programming **11** (1): 33–53, January 2001. Printed in the United Kingdom

Algorithms with polynomial interpretation termination proof

G. BONFANTE, A. CICHON, J.-Y. MARION

Loria, Calligramme project, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
(*e-mail:* {bonfante,cichon,marionjy}@loria.fr)

H. TOUZET

LIFL - USTL, 59655 Villeneuve d'Ascq Cedex, France
(*e-mail:* touzet@lifl.fr)

Abstract

We study the effect of polynomial interpretation termination proofs of deterministic (resp. non-deterministic) algorithms defined by confluent (resp. non-confluent) rewrite systems over data structures which include strings, lists and trees, and we classify them according to the interpretations of the constructors. This leads to the definition of six function classes which turn out to be exactly the deterministic (resp. non-deterministic) polynomial time, linear exponential time and linear doubly exponential time computable functions when the class is based on confluent (resp. non-confluent) rewrite systems. We also obtain a characterisation of the linear space computable functions. Finally, we demonstrate that functions with exponential interpretation termination proofs are super-elementary.

1 Introduction

Termination orderings on rewrite systems can give rise to characterisations of classes of total functions. Examples of this which occur in the literature are the characterisations of the primitive recursive functions by the Multiset Path Ordering in Hofbauer (192) and of the multiply recursive functions of Péter (1967) by the Lexicographic Path Ordering in Weiermann (1995). Termination proofs by polynomial interpretations were introduced by Lankford (1979). Hofbauer and Lautemann (1988) established that rewrite systems with polynomial interpretation termination proofs can admit derivations doubly exponential in length. The background for our study goes back to the work of Cichon and Lescanne (1992) where it was shown that a particularly important aspect was the interpretations of the constructors. More recently, we have shown (Bonfante *et al.*, 1998) that, according to the interpretations of the successors (i.e. constructors), polynomial interpretation termination proofs for functions give rise to characterisations of the functions computed in polynomial time (PTIME), linear exponential time (ETIME = DTIME($2^{O(n)}$)) and linear doubly exponential time (E_2TIME = DTIME($2^{2^{O(n)}}$)). As a corollary, we established that the functions over $\mathbb{N}$, where $\mathbb{N}$ is the representation of the natural numbers based on one successor, are precisely the LINSPACE functions.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Table 1. *A short summary of results (where* $\exp_\infty(0) = 2$ *and* $\exp_\infty(n+1) = 2^{\exp_\infty(n)}$*)*

n-ary constructor interpretation	Confluent	Non-confluent
Kind 0: $\sum_{i=1}^{n} X_i + \gamma$	PTIME	NPTIME
Kind 1: $\sum_{i=1}^{n} \alpha_i X_i + \gamma$	ETIME	NETIME
Kind 2: $\alpha \prod_{i=1}^{n} X_i^{\beta_i} + R(X_1, \ldots, X_n)$	E_2TIME	NE_2TIME
Kind 3: $\sum_{i=1}^{n} 2^{X_i} + \gamma$	DTIME($\exp_\infty$(n))	DTIME($\exp_\infty$(n))

Following Marion (1998), our investigations in the present paper concern, specifically, the effect of polynomial and exponential interpretation termination proofs for both deterministic and non-deterministic algorithms which are defined by confluent (for deterministic algorithms) and non-confluent (for non-deterministic algorithms) rewrite systems over data structures which include strings, lists and trees. Thus, we are somewhat closer to real programming languages such as those designed in POLO (Giesl, 1995), ORME[1] and LARCH[2]. Our approach incorporates ideas involved in the predicative analysis of recursive definitions which can be found in the works of Simmons (1988), Bellantoni and Cook (1992) and Leivant (1994), and we also exploit more traditional methods from rewriting theory.

We present a notion of function computed by a non-confluent system which appears in Krentel's (1988) work in a different context, and which seems appropriate and robust, as argued by Grädel and Gurevich (1995). The purpose is to provide a basis whereby non-confluent rewrite systems can be seen as a model of non-deterministic computations.

We classify three kinds of polynomial interpretations of constructors and we analyse the computational complexity of algorithms with respect to the kind of interpretation given to the constructors. Hence, we obtain three classes of functions which characterise exactly PTIME, ETIME, E_2TIME when systems are confluent. When systems are non-confluent, we capture their non-deterministic counterparts, that is the class of functions computable in non-deterministic polynomial time (NPTIME), non-deterministic exponential time (NETIME), and lastly non-deterministic doubly exponential time (NE_2TIME).

Machine independent characterisations of complexity classes were originated by Cobham (1962). His approach is by means of 'bounded recursion on notation' in which rates of growth of functions are limited by functions already defined in the class. In contrast, in our work, polynomial interpretations, via a reduction ordering, impose a local condition on each rewrite rule. It is worth mentioning the characteri-

[1] See `http://www.ens -lyon.fr/~plescann/publications.html`
[2] See `http://www.sds.lcs.mit.edu/spd/larch/index.html`

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

sation of PTIME functions over finite models in Sazonov (1980) and Gurevich (1983), since basically the same system is considered: the Herbrand–Gödel equations.

This paper is organised as follows. Sections 2 and 3 define functions computed by rewrite systems with polynomial interpretation termination proofs. The main results with their consequences are presented in Theorems 4.2 and 4.3 of section 4. In section 5, we establish the characterisation of polynomial time computability. In section 6, we examine LINSPACE. Sections 7 and 8 are devoted to exponential time computability, the proofs use results of the previous sections. Finally, section 9 shows that functions with termination proofs using exponential interpretations are super-elementary.

2 Rewrite systems with polynomial interpretations

2.1 Computing with rewrite systems

Let $\mathbb{A}$ be a finite set of symbols of fixed arity. We are concerned with algorithms defined by terminating rewrite systems over a finitely generated free algebra of terms, $\mathcal{T}(\mathbb{A})$. Hence, a symbol f defined by rewrite rules, involving possibly other auxiliary defined symbols, describes an algorithm. Furthermore, only normal forms in $\mathcal{T}(\mathbb{A})$ are considered as meaningful. That is, $\mathcal{T}(\mathbb{A})$-terms serve as the algorithmic data structure, and inputs and outputs also are in $\mathcal{T}(\mathbb{A})$.

Example 1
Let s_0 and s_1 be two unary constructors and ϵ be a constant. The set $\mathbb{W} = \{s_0, s_1, \epsilon\}$ is a set of constructors for binary words.

Example 2
Lists and trees are generated by a binary constructor $*$ (used as an infix operator) and a constant `nil`.

Throughout, we shall always assume that the constructor sets contain at least $\mathbb{D} = \{\mathtt{nil}, \mathtt{tt}, \mathtt{ff}, \epsilon, s_0, s_1, *\}$, where `tt` and `ff` represent the boolean value *true* and *false*. So, lists of binary words are included in the domain of computation. Of course, we could encode all constructor terms with $\{\mathtt{nil}, *\}$ efficiently, so that the size of the encoding is equal to the size of the original term up to a fixed multiplicative factor.

Example 3
The main rules for defining tree-destructors are

$$\begin{array}{llll} \mathtt{hd}(\mathtt{nil}) & \to \mathtt{nil} & \mathtt{tl}(\mathtt{nil}) & \to \mathtt{nil} \\ \mathtt{hd}(c * t) & \to c & \mathtt{tl}(c * t) & \to t \end{array}$$

The conditional is defined as follows:

$$\mathtt{if\text{-}t\text{-}e}(\mathtt{tt}, u, v) \to u \qquad \mathtt{if\text{-}t\text{-}e}(\mathtt{ff}, u, v) \to v$$

The term rewriting notations used throughout are based on Dershowitz and Jouannaud (1990). The relation $\overset{+}{\to}$ ($\overset{*}{\to}$) denotes the transitive (reflexive-transitive) closure of $\to$. If u and v are two terms, we write $u \overset{!}{\to} v$ to mean that $u \overset{*}{\to} v$ and v is in normal form. It is convenient to present a rewrite system as a tuple $\langle \mathcal{R}, \mathbb{F}, \mathbb{A}, f \rangle$, where

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

- $\mathbb{A}$ is the set of constructors,
- $\mathbb{F}$ is the set of defined auxiliary symbols,
- $\mathscr{R}$ is the set of rewrite rules, and
- f is the main symbol of $\mathbb{F}$.

2.2 Polynomial interpretation

A *polynomial interpretation termination proof* for a rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f \rangle$ consists in the assignment to each symbol g of $\mathbb{F} \cup \mathbb{A}$, of 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 in k variables.
- If the arity of g is 0 then $[g] > 0$, otherwise, for all $n_j \geqslant 0$, where $j = 1, \ldots, k$, $[g](n_1, \ldots, n_k) > n_i$, for all $i = 1, \ldots, k$. This condition is useful to guarantee strict monotonicity of the interpretation on terms.

$[\ ]$ is extended canonically to a morphism over terms as follows:

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

Finally, $[\ ]$ must ensure that for all rules $l \to r$ of $\mathscr{R}$, $[l] > [r]$ for all values of variables greater than or equal to the minimum of the interpretations of the constants. The above definition follows Dershowitz (1982) (see also Steinbach, 1995).

Example 4
Whatever the choice of polynomial interpretation for the constructors, the symbols defined in Example 3 admit the following interpretations:

$$[\mathtt{hd}](X) = X + 1 \quad [\mathtt{tl}](X) = X + 1 \quad [\mathtt{if\text{-}t\text{-}e}](X, Y, Z) = X + Y + Z + 1$$

Henceforth, a rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f \rangle$ with a polynomial interpretation, $[\ ]$, is denoted by $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$.

Example 5
The equality predicate on $\mathbb{W} = \{\mathtt{tt}, \mathtt{ff}, \epsilon, s_0, s_1\}$ is computed by

$$\begin{array}{ll} \mathtt{eq?}(\epsilon, \epsilon) \to \mathtt{tt} & \mathtt{eq?}(s_i(u), s_i(v)) \to \mathtt{eq?}(u, v) \\ \mathtt{eq?}(s_i(u), \epsilon) \to \mathtt{ff} & \mathtt{eq?}(s_i(u), s_j(v)) \to \mathtt{ff} \qquad (i \neq j) \\ \mathtt{eq?}(\epsilon, s_j(v)) \to \mathtt{ff} & \end{array}$$

and admits the interpretation $[\mathtt{ff}] = [\mathtt{tt}] = [\epsilon] = 1$, $[s_i](X) = X + 2$ and $[\mathtt{eq?}](X, Y) = X + Y + 1$. Similarly, we define the boolean operator `and` with the interpretation $[\mathtt{and}](X, Y) = X + Y + 1$ as follows:

$$\mathtt{and}(\mathtt{tt}, \mathtt{tt}) \to \mathtt{tt} \qquad \mathtt{and}(\mathtt{ff}, x) \to \mathtt{ff} \qquad \mathtt{and}(x, \mathtt{ff}) \to \mathtt{ff}$$

Example 6
Over $\mathbb{D}$, a graph is represented by a list $V = (s_0(\epsilon) * s_1(\epsilon) * s_0(s_0(\epsilon)) * \cdots * \mathtt{nil})$ of vertices and a list $E = ((u_0 * v_0) * (u_1 * v_1) * \cdots * \mathtt{nil})$ of edges. We first define a

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

predicate `find` which tests whether or not a pair (u, v) of vertices is an edge of the graph:

$$\begin{aligned}\texttt{find}(u, v, \texttt{nil}) &\to \texttt{ff}\\ \texttt{find}(u, v, (u' * v') * E) &\to \texttt{if-t-e}(\texttt{and}(\texttt{eq?}(u, u'), \texttt{eq?}(v, v')), \texttt{tt}, \texttt{find}(u, v, E))\end{aligned}$$

with the polynomial interpretation:

$$[\texttt{nil}] = 1 \quad [*](X, Y) = X + Y + 4 \quad [\texttt{find}](X, Y, Z) = (X + Y + 1)(Z + 1)$$

2.3 Upper bounds on derivation lengths

We end this section by giving some general properties which will be used later. First, 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 termination proof 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
For all closed terms t, $|t| \leqslant [t]$.

Proof
The proof is by induction on $|t|$. The result is immediate if t is a constant because $|t| = 1$ and $[t] > 0$ by assumption on $[\ ]$. Otherwise, t is of the form $f(t_1, \ldots, t_n)$ and so $|f(t_1, \ldots, t_n)| = 1 + \sum_{i=1}^{n} |t_i| \leqslant 1 + \sum_{i=1}^{n} [t_i]$. Since $[f]$ is monotone, for all $n > 0$, $[f](\ldots, n+1, \ldots) \geqslant [f](\ldots, n, \ldots) + 1$. By definition $[f](0, \ldots, 0) > 0$, so we have $[f](X_1, \ldots, X_n) \geqslant \sum_{i=1}^{n} X_i + 1$, for all $X_i \geqslant 0$. Therefore, $|f(t_1, \ldots, t_n)| \leqslant 1 + \sum_{i=1}^{n} [t_i] \leqslant [f(t_1, \ldots, t_n)]$. □

Definition 1
A class $\mathscr{C}$ of unary increasing functions over natural numbers *accommodates polynomials* if, and only if, for all $\phi \in \mathscr{C}$, for all polynomials P with natural number coefficients, there is a function $\phi' \in \mathscr{C}$ such that, for all $x > 0$, $P(\phi(x)) \leqslant \phi'(x)$.

In the sequel, we shall deal with four main classes of functions: polynomials, exponentials $\{2^{\gamma x} \ ; \ \gamma > 0\}$, double exponentials $\{2^{2^{\gamma x}} \ ; \ \gamma > 0\}$, and super elementary functions $\{\exp_\infty(x + \gamma) \ ; \ \gamma > 0\}$, it is clear that these classes accommodate polynomials.

Lemma 3
Let $\mathscr{C}$ be a class of functions which accommodates polynomials. Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ be a rewrite system. Assume that there exists $\phi \in \mathscr{C}$ such that, for all terms t in $\mathscr{T}(\mathbb{A})$, $[t] \leqslant \phi(|t|)$. Then there is $\phi' \in \mathscr{C}$, such that for all terms $t_1, \ldots, t_n \in \mathscr{T}(\mathbb{A})$, the following holds:

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

1. $[f(t_1,\ldots,t_n)] \leqslant \phi'(\max\{|t_i| \ ; \ 1 \leqslant i \leqslant n\})$.
2. The length of any derivation starting from $f(t_1,\ldots,t_n)$ is bounded by
$$\phi'(\max\{|t_i| \ ; \ 1 \leqslant i \leqslant n\}).$$
3. If $f(t_1,\ldots,t_n) \xrightarrow{+}_{\mathscr{R}} v$, then $|v| \leqslant \phi'(\max\{|t_i| \ ; \ 1 \leqslant i \leqslant n\})$.

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

$$\begin{aligned} [f(t_1,\ldots,t_n)] &\leqslant [f](\phi(m),\ldots,\phi(m)) \\ &\leqslant \phi'(m) \qquad\qquad \text{for some } \phi' \text{ in } \mathscr{C} \end{aligned}$$

So (1) is proved. (2) is a consequence of (1) since the length of any derivation is bounded by the polynomial interpretation of the starting term. Finally, Lemma 2.3 implies $|v| \leqslant [v]$. Since $[v] \leqslant [f(t_1,\ldots,t_n)]$, we obtain (3) by applying (1) again. □

3 Calculating semantics

3.1 Deterministic computation

As we have stated earlier, we see the rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\,] \rangle$ as a program over the data structure $\mathscr{T}(\mathbb{A})$. We shall only consider the normal forms, which are in the data set $\mathscr{T}(\mathbb{A})$, as being meaningful. We shall provide a semantics over strings for these programs. This will enable us to effect time simulations over computation models working over strings and to compare expressivities with these models. A reason is that the usage is to measure the time simulation over a computation model working on strings, and to compare the expressivity with those models. All along, Σ will be an alphabet and Σ^* the set of words over Σ. We relate strings of Σ^* to the data structure $\mathscr{T}(\mathbb{A})$ by an encoding function which is always assumed to be suitable. An encoding $\alpha : \Sigma^* \mapsto \mathscr{T}(\mathbb{A})$ is suitable if α is injective and if there is a constant c such that, for all u in Σ^*, $|\alpha(u)| \leqslant c \cdot |u|$. For example, one might define α^{-1} as a 'Polish prefix form' encoding. The semantics of a program is obtained through a pair of encoding functions which expresses the input/output behaviour of the program.

Definition 2
Let (α, β) be a pair of encoding functions. The function computed by $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\,] \rangle$, where f is of arity n, is the function $\{f\} : (\Sigma^*)^n \mapsto \Sigma^*$ defined, for all $w_1,\ldots,w_n, v \in \Sigma^*$

$$f(\alpha(w_1),\ldots,\alpha(w_n)) \xrightarrow{!} \beta(v) \text{ if, and only if, } \{f\}(w_1,\ldots,w_n) = v$$

The pair (α, β) of encoding functions is a key element of our characterisation of exponential and doubly exponential time computation. In fact, it happens that two constructors are needed, while they actually have the same string representation, as it is illustrated in the next example. However, it is worth noticing that we could identify α and β to capture polynomial time computation, hence providing a one-one encoding function. Throughout, we shall omit mention of the encoding pair when it is not absolutely necessary.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Example 7
Consider the following rewrite system which defines E over the set of constructors $\{0, p, q\}$:

$$\begin{array}{ll} \mathtt{D}(0) \to 0 & \mathtt{D}(p(x)) \to p(p(\mathtt{D}(x))) \\ \mathtt{E}(0) \to p(0) & \mathtt{E}(q(x)) \to \mathtt{D}(\mathtt{E}(x)) \end{array}$$

The symbols admit the following polynomial interpretations, $[0] = 1$

$$\begin{array}{ll} [p](X) = X + 4 & [q](X) = 3X + 12 \\ [\mathtt{D}](X) = 3X + 1 & [\mathtt{E}](X) = X + 5 \end{array}$$

Now, define the unary numeral over the tally alphabet $\Sigma = \{|\}$ by $\underline{0} = \epsilon$ and $\underline{n+1} = |\underline{n}$. The function $\underline{n} \mapsto \underline{2^n}$ is computed by E with respect to the pair of encoding functions (α, β) defined:

$$\begin{array}{ll} \alpha(\epsilon) = 0 & \alpha(|u) = q(\alpha(u)) \\ \beta(\epsilon) = 0 & \beta(|u) = p(\beta(u)). \end{array}$$

Therefore, for all n, $E(\alpha(\underline{n})) \xrightarrow{!} \beta(\underline{2^n})$. A consequence of the work of Cichon and Lescanne (1992) is that p and q must have different kinds of interpretations to define a function with an exponential growth rate.

3.2 Non-deterministic computation

We have to decide what we mean when we say that a function is computed by a non-confluent rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\,] \rangle$. Indeed, the computation of f, on arguments in $\mathscr{T}(\mathbb{A})$, leads to several normal forms which depend on the reductions applied. We shall regard a non-confluent rewrite system as a non-deterministic algorithm. Let us illustrate this point of view by defining a procedure which searches for a clique in a graph.

Example 8
Consider a graph G which is represented as in Example 6 by a list V of vertices and a list E of edges. Here the program $\mathtt{clique}(K, E)$ returns the value tt if the set of vertices K is a complete sub-graph of G. To define clique, we use find (Example 6), and complete. The term $\mathtt{complete}(u, V, E)$ checks whether the vertex u is connected to each vertex in V using an edge in E:

$$\begin{array}{rl} \mathtt{complete}(u, \mathtt{nil}, E) & \to \mathtt{tt} \\ \mathtt{complete}(u, v * V, E) & \to \mathtt{if\text{-}t\text{-}e}(\mathtt{find}(u, v, E), \mathtt{complete}(u, V, E), \mathtt{ff}) \\ \mathtt{clique}(\mathtt{nil}, E) & \to \mathtt{tt} \\ \mathtt{clique}(u * K, E) & \to \mathtt{if\text{-}t\text{-}e}(\mathtt{complete}(u, K, E), \mathtt{clique}(K, E), \mathtt{ff}) \end{array}$$

The set of normal forms of $\mathtt{choice}(V, \mathtt{nil}, E)$ is the set of all cliques of G:

$$\begin{array}{rl} \mathtt{choice}(\mathtt{nil}, K, E) & \to \mathtt{if\text{-}t\text{-}e}(\mathtt{clique}(K, E), K, \mathtt{nil}) \\ \mathtt{choice}(u * V, K, E) & \to \mathtt{choice}(V, u * K, E) \\ \mathtt{choice}(u * V, K, E) & \to \mathtt{choice}(V, K, E) \end{array}$$

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

The last two rules are not confluent, and correspond exactly to non-deterministic choice. (By Newman's Lemma, the systems considered are not weakly confluent since they are terminating.) Now, a maximal clique of G is a longest list among all normal forms generated by `choice`-computation. Similarly, when a decision procedure is carried out by a non-deterministic computation, the result, which is 1 or 0 here, is given by the maximal output.

One might check that the system is terminating by the following polynomial interpretations:

$$[\texttt{complete}](X,Y,Z) = (X+1)(Y+1)(Z+1)$$
$$[\texttt{clique}](X,Y) = (X+1)^2 \cdot (Y+1)$$
$$[\texttt{choice}](X,Y,Z) = (2X+Y+1)^2 \cdot (Z+1)$$

Actually, we follow closely the definition introduced by Krentel (1988) and by Grädel and Gurevich (1995) in defining a function computed by a non-confluent system as follows.

Definition 3
Let (α,β) be a pair of encoding functions. The rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ computes the function $\{f\} : (\Sigma^*)^n \mapsto \Sigma^*$ defined by

$$\{f\}(u_1,\ldots,u_n) = \max\{v \ ; \ f(\alpha(u_1),\ldots,\alpha(u_n)) \xrightarrow{!} \beta(v)\}$$

for all $u_1,\ldots,u_n \in \Sigma^*$, and where strings of Σ^* are ordered by length and then lexicographically.

Note that the above notion is a meta-definition. Definability in the system of the max-operation is equivalent to providing a completion procedure which actually amounts to transforming a non-deterministic algorithm into a deterministic one. So, unless PTIME = NPTIME, the above completion can not be carried out inside a $\Delta(i)$-rewrite system, as we shall see.

4 Classes of functions and results

4.1 Computational models

We shall consider non-deterministic multi-stack Turing machines, abbreviated ND-STM, as computational models. Formally, a k-NDSTM, $\mathscr{M}$, is a NDSTM with k stacks which is defined by a tuple $\mathscr{M} = \langle \Sigma, \epsilon, Q, q_0, Q_f, \delta \rangle$, 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, $Q_f \subseteq Q$ the set of final states and finally the transition relation is $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\})^k \times Q \times (\Sigma^*)^k$.

A configuration of the machine is a $k+1$-tuple $\langle q, w_1,\ldots,w_k \rangle$ where $q \in Q$ is the current state, and w_i is the content of the ith stack. Now, for all i, if a_i is the first letter of w_i, or if both a_i and w_i are equal to ϵ, then we may apply the transition $(q, a_1,\ldots,a_k, q', u_1,\ldots,u_k)$ if it belongs to δ. It replaces each letter a_i by the word u_i and switches to the state q' to reach a new configuration. So we obtain the configuration $\langle q', u_1v_1,\ldots,u_kv_k \rangle$ where $w_i = a_iv_i$. Note that δ may yield more

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

than one next configuration. Let $\Rightarrow$ be the relation which provides the set of next configurations provided by δ and $\stackrel{+}{\Rightarrow}$ the transitive closure of $\Rightarrow$.

We have already discussed what it means for a function to be defined by a non-deterministic procedure (see the end of section 3.2). For computation over a NDSTM, we again follow Krentel (1988) and Grädel and Gurevich (1995).

Definition 4
A function $F[\mathscr{M}] : (\Sigma^*)^n \mapsto \Sigma^*$ is computed by a NDSTM $\mathscr{M}$ if $F[\mathscr{M}](u_1, \ldots, u_n) = \max\{v \ ; \ (q_0, u_1, \ldots, u_n, \epsilon, \ldots, \epsilon) \stackrel{+}{\Rightarrow} (q_f, \ldots, v) \text{ where } q_f \in Q_f\}$ where Σ^* is ordered by length and then lexicographically.

Now, if δ is the graph of a $k+1$-ary function from $Q \times (\Sigma \cup \{\epsilon\})^k$ into $Q \times (\Sigma^*)^k$, the computational model is deterministic (STM). In the deterministic case, the max-operation defining ϕ is taken over at most one value, and hence it is a particular case of the above non-deterministic definition.

Two-stack Turing Machines are sufficiently expressive to delineate classes like PTIME and NPTIME. However, we shall need more than two-stacks in section 6 to deal with unary computation in order to capture LINSPACE.

4.2 Classification for deterministic computation

We shall now consider three categories of polynomial interpretation of constructors. This classification extends that of Bonfante *et al.* (1998), and follows studies initiated by Hofbauer and Lautemann (1988) and Cichon and Lescanne (1992).

Kind 0: polynomials of the form $P(X_1, \ldots, X_n) = \sum_{i=1}^{n} X_i + \gamma$, where $\gamma > 0$.
Kind 1: polynomials of the form $P(X_1, \ldots, X_n) = \sum_{i=1}^{n} \alpha_i X_i + \gamma$, where $\alpha_i > 1$, for some i.
Kind 2: polynomials of the form $P(X_1, \ldots, X_n) = \alpha \prod_{i=1}^{n} X_i^{\beta_i} + R(X_1, \ldots, X_n)$, where $\alpha > 0$, $\sum_{i=1}^{n} \beta_i > 1$ and R is any polynomial.

Definition 5
Let $i \in \{0, 1, 2\}$. A *confluent* rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ is $\Pi(i)$ if, for each constructor c of $\mathbb{A}$, the interpretation $[c]$ is of kind less than or equal to i. A function ϕ over Σ^* is $\Pi(i)$-*computable* if it is computed by a $\Pi(i)$-rewrite system.

For instance, Example 8 (`clique`) shows that checking whether a set of vertices is a clique, is a $\Pi(0)$-computable function. Example 7 illustrates the fact that the exponential function is $\Pi(1)$-computable. We now state our main result for deterministic computation.

Theorem 4

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

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Proof
The above theorem is mostly a consequence of the proof of Theorem 5. Corollary 10 simulates $\Pi(0)$-computable functions in PTIME. The converse is stated in Corollary 14. The characterisation of ETIME (resp. E_2TIME) follows from Corollaries 20 and 23 (resp. 27 and 30). □

Now say that a language L is recognised by a rewrite system if the characteristic function of L is computable by the rewrite system. It is obvious that languages recognised in time $2^{O(n)}$ are $\Pi(1)$-computable. Actually, every language recognised in time $2^{O(n^k)}$ is $\Pi(1)$-computable, *with respect to a polynomial time reduction*. To see this, use a standard padding argument. Indeed, let $L \subset \Sigma^*$ be a DTIME(2^{n^k}) language. Suppose that L is *accepted* by the TM M. Suppose that @ is not a letter of Σ. Then, construct $L' = \{x@0^{|x|^k} \ ; \ x \in L\} \subset (\Sigma \cup \{@\})^*$, that is, each word of L' is a word of L padded by extra 0's. Thus, L' is recognised in time $2^{O(n)}$ using M. Then, the characteristic function of L' is $\Pi(1)$-computable.

4.3 Classification for non-deterministic computation

As for confluent systems, non-confluent systems fall into the same three categories.

Definition 6
Let $i \in \{0, 1, 2\}$. A rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ is $\Delta(i)$ if, for each constructor c of $\mathbb{A}$, the interpretation $[c]$ is a polynomial of kind less than or equal to i. A function ϕ over Σ^* is $\Delta(i)$-*computable* if it is computed by a $\Delta(i)$-rewrite system.

It follows that finding a clique (`choice`) in a graph, as defined in Example 8, is a $\Delta(0)$-computable function.

Theorem 5

1. The $\Delta(0)$-computable functions are exactly the NPTIME functions.
2. The $\Delta(1)$-computable functions are exactly the NETIME functions, i.e. the functions computable in non-deterministic time $2^{O(n)}$.
3. The $\Delta(2)$-computable functions are exactly the NE_2TIME functions, i.e. the functions computable in non-deterministic time $2^{2^{O(n)}}$.

Proof
The proof is as follows. Lemma 9 shows how to simulate $\Delta(0)$-computable functions in NPTIME. The converse is established in Lemma 13. The characterisation of NETIME is proved in Lemmas 19 and 22, and the characterisation of NE_2TIME is established in Lemmas 26 and 29. □

4.4 Weak composition properties

We show a property of weak closure under composition of $\Delta(i)$ and $\Pi(i)$-computable functions, enabling us to combine rewrite systems to define functions in a modular way.

Lemma 6
Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ be a $\Delta(0)$ (resp. $\Pi(0)$)-rewrite system and let $\langle \mathscr{R}_i, \mathbb{F}_i, \mathbb{A}_i, f_i, [\ ]_i \rangle$

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

be a $\Delta(i)$ (resp. $\Pi(i)$)-rewrite system. Assume that there is an injective morphism $\tau : \mathscr{T}(\mathbb{A}_i) \mapsto \mathscr{T}(\mathbb{A})$ which preserves symbol arity. Then there is a $\Delta(i)$ (resp. $\Pi(i)$)-rewrite system $\langle \mathscr{R}_*, \mathbb{F}_*, \mathbb{A}_*, f_*, [\,]_* \rangle$ such that

$$f_*(\vec{u}, w_1, \ldots, w_n) \xrightarrow{!} v \text{ if, and only if, } f(\tau(f_i(\vec{u})), \tau(w_1), \ldots, \tau(w_n)) \xrightarrow{!} v$$

for all $\vec{u}, w_1, \ldots, w_n \in \mathscr{T}(\mathbb{A}_i)$ and $v \in \mathscr{T}(\mathbb{A})$.

Proof

We give the proof for non-confluent $\Delta(i)$-rewrite systems. Without loss of generality, we may suppose that both $\mathbb{F}$ and $\mathbb{F}_i$ are disjoint and also that $\mathbb{A}$ and $\mathbb{A}_i$ are two disjoint copies of the same set of constructors. The key point of the construction is the introduction of a coercion function $\mathtt{Tr}$ which translates terms of $\mathbb{A}_i$ into terms of $\mathbb{A}$. $\mathtt{Tr}$ is defined as follows:

$$\begin{array}{rll} \mathtt{Tr}(c) & \to \tau(c) & c \in \mathbb{A}_i \text{ is 0-ary} \\ \mathtt{Tr}(c(x_1, \ldots, x_n)) & \to \tau(c)(\mathtt{Tr}(x_1), \ldots, \mathtt{Tr}(x_n)) & c \in \mathbb{A}_i \text{ is } n\text{-ary} \end{array}$$

The interpretation of $\mathtt{Tr}$ is

$$[\mathtt{Tr}]_*(X) = \alpha \cdot X + 1 \tag{1}$$

where $\alpha = \max\{n + \gamma;\ \ [\tau(c)](x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i + \gamma \text{ where } c \in \mathbb{A}_i\}$. Indeed, we have already seen in the proof of Lemma 2 that

$$\sum_{i=1}^{n} [x_i] + 1 \leqslant [c(x_1, \ldots, x_n)] \tag{2}$$

Thus, we have

$$\begin{array}{lll} [\tau(c)(\mathtt{Tr}(x_1), \ldots, \mathtt{Tr}(x_n))] & \leqslant \alpha \cdot \sum_{i=1}^{n} [x_i] + n + \gamma & \tau(c) \in \mathbb{A} \\ & \leqslant \alpha \cdot \left(\sum_{i=1}^{n} [x_i] + 1\right) & \text{by definition of } \alpha \\ & < [\mathtt{Tr}(c(x_1, \ldots, x_n))] & \text{by 1 and 2} \end{array}$$

Define the rewrite system $\langle \mathscr{R}_*, \mathbb{F} \cup \mathbb{F}_i \cup \{f_*\}, \mathbb{A} \cup \mathbb{A}_i, f_*, [\,]_* \rangle$ where the set of rules $\mathscr{R}_*$ contains the set $\mathscr{R} \cup \mathscr{R}_i$, the above rules for $\mathtt{Tr}$ and the rule

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

The polynomial interpretation $[\,]_*$ is an extension of $[\,]$ and $[\,]_i$:

$$\begin{array}{rl} [g]_* = [g], & \text{if } g \in \mathbb{F} \cup \mathbb{A} \\ [g]_* = [g]_i, & \text{if } g \in \mathbb{F}_i \cup \mathbb{A}_i \\ [f_*]_*(\vec{X}, Y_1, \ldots, Y_n) = [f]_*([\mathtt{Tr}]_*([f_i]_*(\vec{X})), [\mathtt{Tr}]_*(Y_1), \ldots, [\mathtt{Tr}]_*(Y_n)) + 1 & \end{array}$$

□

5 Polynomial time computation

5.1 $\Delta(0)$*-computable functions are polynomial time*

Lemma 7

Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\,] \rangle$ be a $\Delta(0)$-rewrite system. Then there is a constant γ such that, for all $t \in \mathscr{T}(\mathbb{A})$, $[t] \leqslant \gamma \cdot |t|$.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Proof
All constructors in $\mathbb{A}$ are of kind 0. Therefore, there exists $\gamma > 0$ such that $[c] \leqslant \gamma$ when c is 0-ary, and $[c](x_1, \ldots, x_n) \leqslant \sum_{i=1}^{n} x_i + \gamma$, when c is n-ary. So, for all $t \in \mathscr{T}(\mathbb{A})$, we have, by composition of interpretations, $[t] \leqslant \gamma \cdot |t|$. □

Corollary 8
Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\,] \rangle$ be a $\Delta(0)$-rewrite system. Then there is a polynomial P_f such that, for all terms $t_1, \ldots, t_n \in \mathscr{T}(\mathbb{A})$,

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

Proof
(1) is a consequence of Lemma 3-(2) and Lemma 7. (2) is a consequence of Lemma 3-(3) and Lemma 7. □

Lemma 9
If ϕ is $\Delta(0)$-computable, then ϕ is in NPTIME.

Proof
Suppose that ϕ is computed by the $\Delta(0)$-rewrite system $\langle \mathscr{R}, \mathbb{A}, \mathbb{F}, f, [\,] \rangle$ and uses the pair (α, β). We shall describe a NDSTM which produces every normal form of $f(u_1, \ldots, u_n)$ for all $u_1, \ldots, u_n$ in $\mathscr{T}(\mathbb{A})$. The machine alphabet is Σ. For this, we use a one-one encoding function on strings $cde : \mathscr{T}(\mathbb{A}) \mapsto \Sigma^*$, as the 'Polish prefix form'. The inputs will be encoded by $(cde(\alpha(u_1)), \ldots, cde(\alpha(u_n))$. At the end, of the computation, the result will be decoded by β^{-1}. Therefore, the result of the computation will correspond exactly to the greatest value with respect to the length lexicographic ordering on Σ^*.

The non-deterministic algorithm works as follows. At each step, we choose non-deterministically a redex of the current term and an $\mathscr{R}$-rule. We apply the corresponding reduction. We stop when no reduction is applicable.

The first key point is that at any derivation step, it follows from (2) of Corollary 8 that the size of terms is always bounded by $O(\max\{|w_i|\}^p)$, for some constant p, where w_i are contents of the stacks of the TM. So a redex position is determined with $O(\log(\max\{|w_i|\}))$ bits, and a constant number of bits is necessary to indicate which $\mathscr{R}$-rule is taken. Hence at each step, we guess $O(\log(\max\{|w_i|\}))$ bits. Because of this, the replacement is performed in at most quadratic time. The size of the term obtained is then bounded by $O(\max\{|w_i|\}^p)$, again by (2) of Corollary 8. Therefore, a reduction step, including the guessing procedure, is carried out in time $O(\max(\{|w_i|\}^{p+3})$.

The second key point is that the length of a derivation path is bounded by $O(\max\{|w_i|\}^q)$, for some constant q, by (1) of Corollary 8. Consequently, the runtime is bounded by $O(\max\{|w_i|\}^{q+p+3})$. □

Corollary 10
If ϕ is $\Pi(0)$-computable, then ϕ is in PTIME.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Proof
Since ϕ is defined by a confluent rewrite system, it follows that any reduction strategy will work. So the proof of Lemma 9 provides a deterministic evaluation procedure which runs in polynomial time. □

5.2 Simulation of time bounded computation

The simulation of a time bounded computation will always proceed as follows. First, we shall construct a rewrite system in $\Delta(0)$ that simulates t steps of the computation of a NDSTM. This will be done in Lemma 11. It will turn out that the system is confluent if the computation is deterministic. Then, it will remain to establish that the time bound is computed by a confluent system. We shall obtain the conclusion by composing both results, following Lemma 6.

Lemma 11
Let $\mathscr{M} = \langle \Sigma, \epsilon, Q, q_0, Q_f, \delta \rangle$ be a NDSTM. Then there is a $\Delta(0)$-computable function ϕ_M such that, if $\mathscr{M}$ halts in less than t steps then $\phi_M(0^t, w_1, \ldots, w_k) = F[\mathscr{M}](w_1, \ldots, w_k)$, for each $w_1, \ldots, w_k$ in Σ^* and $0 \in \Sigma$.

Proof
We construct a rewrite system which computes ϕ_M as follows:

- constructors are $\mathbb{W} = \{s_i \mid i \in \Sigma\} \cup \{\epsilon\}$;
- the defined symbols are $\{\mathsf{q}\ ;\ q \in Q\}$, where 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 $\alpha(\epsilon) = \epsilon$ and $\alpha(i) = s_i$ for all $i \in \Sigma$.

Let s be s_0. The rewrite rules are the following:

- If $(q, a_1, \ldots, a_k, q', u_1, \ldots, u_k) \in \delta$, then
$$\mathsf{q}(s(t), s_{a_1}(x_1), \ldots, s_{a_k}(x_k)) \to \mathsf{q}'(t, \alpha(u_1)(x_1), \ldots, \alpha(u_k)(x_k)),$$
with the the following convention: if the ith stack is empty, that is a_i is ϵ, then $s_{a_i}(x_i)$ is x_i.
- if $q_f \in Q_f$, then $\mathsf{q}_f(s(t), x_1, \ldots, x_k) \to x_k$.

It is straightforward to verify that
$$\langle q, w_1, \ldots, w_k \rangle \Rightarrow \langle q', w_1', \ldots, w_k' \rangle$$
if and only if
$$\mathsf{q}(s(t), \alpha(w_1), \ldots, \alpha(w_k)) \to \mathsf{q}'(t, \alpha(w_1'), \ldots, \alpha(w_k'))$$
and that
$$q \text{ is a final state iff } \mathsf{q}(s(t), \alpha(w_1), \ldots, \alpha(w_k)) \to \alpha(w_k)$$
Therefore, if $\mathscr{M}$ 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 $\mathsf{q}_0(s^t(\epsilon), \alpha(w_1), \ldots, \alpha(w_k))$, which is exactly $F[\mathscr{M}](w_1, \ldots, w_k)$.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Finally, we interpret each function symbol by

$$\begin{aligned} [\epsilon] &= 1 \\ [s_i](X) &= X + 2 && \forall i \in \Sigma \\ [\mathrm{q}](T, X_1, \ldots, X_k) &= k \cdot c \cdot T + X_1 + \cdots + X_k && \forall q \in Q \end{aligned}$$

where the constant c is strictly greater than the interpretation of any word that is involved in the definition of δ. More precisely, associate to the lth transition rule $(q, a_1, \ldots, a_k, q', u_1, \ldots, u_k) \in \delta$ the constant $c_l = \max\{[\alpha(u_j)(\epsilon)] \ ; \ 1 \leqslant j \leqslant k\}$, and define c as the strict supremum of all c_l. We conclude that the system is $\Delta(0)$. □

5.3 *Simulation of polynomial time computation*

Lemma 12
Each polynomial is $\Pi(0)$-computable.

Proof
The following rules and interpretations show that addition (`Add`) and multiplication (`Mul`) are $\Pi(0)$-computable functions:

$$\begin{aligned} \mathtt{Add}(x, 0) &\to x & [0] &= 1 \\ \mathtt{Add}(0, y) &\to y & [s](X) &= X + 2 \\ \mathtt{Add}(s(x), s(y)) &\to s(s(\mathtt{Add}(x, y))) & [\mathtt{Add}](X, Y) &= 2X + Y + 1 \\ \mathtt{Mul}(0, y) &\to 0 & [\mathtt{Mul}](X, Y) &= (X + 1)(Y + 1) \\ \mathtt{Mul}(s(x), y) &\to \mathtt{Add}(y, \mathtt{Mul}(x, y)) \end{aligned}$$

Hence, by Lemma 6, a polynomial is a $\Pi(0)$-computable function since it is obtained by composition of $\Pi(0)$-functions. □

Lemma 13
If ϕ is in NPTIME, then ϕ is $\Delta(0)$-computable.

Proof
Let ϕ be a function computed by a NDSTM such that the time of computation is bounded by a polynomial P. Since ϕ_M, as defined in Lemma 11, is a $\Delta(0)$-computable function and P is a $\Pi(0)$-computable function, then ϕ is also $\Delta(0)$-computable by Lemma 6. □

Corollary 14
If ϕ is in PTIME, then ϕ is $\Pi(0)$-computable.

Proof
Since the transition relation of a deterministic Turing machine is the graph of a function, it follows that the rewrite system described in the proof of Lemma 11 is confluent, and so is $\Pi(0)$. The conclusion follows using Lemmas 12 and 6. □

6 A characterisation of LINSPACE

A function is in LINSPACE if it is computed by a multi-stack Turing machine (STM) over an alphabet with at least two letters running in linear space. Actually, there is an alternative characterisation of LINSPACE due to Gurevich (1983).

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Theorem 15
A function ϕ is computable over a STM over a unary alphabet in polynomial time if, and only if, ϕ is in LINSPACE.

Proof
The proof essentially follows Leivant (1994). 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 $P(w_1, \ldots, w_m)$ 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, $i = 1, \ldots, k$, 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 number 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. However, this is easily performed by N with the spare stack. Thus, the size of each stack of N is bounded by $O(\log(\max\{w_i \ ; \ i = 1, \ldots, m\}))$.

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, $\underline{\epsilon} = 0$, $\underline{au} = 2 \cdot \underline{u} + 1$ and $\underline{bu} = 2 \cdot \underline{u} + 2$). We build a $(k+2)$-stack Turing machine, N, over the unary alphabet $\{|\}$ as follows. If u is in stack i of M then there are $\underline{u}$ |'s in stack i of N. When 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 $2^{c \cdot n}$ where n is the maximum size of the inputs. So, the runtime of N is linear in $2^{c \cdot n}$, i.e. polynomial in the greatest input value. □

Theorem 16
A function is computed by a $\Pi(0)$-rewrite system $\langle \mathscr{R}, \mathbb{F}, \mathbb{N}, f, [\,] \rangle$ over the domain $\mathbb{N} = \{s, 0\}$ if, and only if, it is in LINSPACE.

Proof
Let ϕ be the function computed by $\langle \mathscr{R}, \mathbb{F}, \mathbb{N}, f, [\,] \rangle$. By Corollary 5.1, ϕ is computable in polynomial time over a unary alphabet. So, by Theorem 6, ϕ is in LINSPACE. Conversely, if ϕ is in LINSPACE, then Theorem 6 yields that ϕ is computable in polynomial time on some STM which works on a unary alphabet. Therefore, by Corollary 5.3, ϕ is $\Pi(0)$-computable. □

7 Exponential time

7.1 $\Delta(1)$*-computable functions are exponential time*

Lemma 17
Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\,] \rangle$ be a $\Delta(1)$-rewrite system. Then there is a constant γ such that, for all $t \in \mathscr{T}(\mathbb{A})$, $[t] \leqslant 2^{\gamma \cdot |t|}$.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Proof
By definition of $\Delta(1)$, for all constructors in $\mathbb{A}$, there is a constant γ such that $[c](X_1,\ldots,X_n) \leqslant \gamma \cdot (\sum_{i=1}^{n} X_i)$, where $X_i > 0$. It is clear that

$$[c(t_1,\ldots,t_n)] \leqslant \gamma \cdot \left(\sum_{i=1}^{n}[t_i]\right) \leqslant \gamma \cdot \left(\sum_{i=1}^{n} 2^{\gamma\cdot|t_i|}\right) \leqslant 2^{\gamma\cdot|c(t_1,\ldots,t_n)|}.$$

□

Corollary 18
Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ]\rangle$ be a $\Delta(1)$-rewrite system. Then there is a constant γ such that, for all terms $t_1,\ldots,t_n \in \mathscr{T}(\mathbb{A})$,

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

Proof
This is a consequence of Lemma 3 and Lemma 17. □

Lemma 19
If ϕ is $\Delta(1)$-computable, then ϕ is in NETIME.

Proof
The argument follows the proof of Lemma 9. Suppose that ϕ is computed by $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ]\rangle$ in $\Delta(1)$. It follows that for each term $t \in \mathscr{T}(\mathbb{A})$, $[t] \leqslant 2^{O(|t|)}$. At each reduction step, we choose a redex non-deterministically. By Corollary 18-(2), throughout the reduction process, expressions have a size bounded by $2^{O(|t|)}$. So, guessing a redex requires at most $O(|t|)$ bits. Finally, if follows again by Lemma 18-(1), that the maximal length of a derivation is bounded by $2^{O(|t|)}$. We conclude that the computation is carried out in non-deterministic time bounded by $2^{O(|t|)}$. □

Corollary 20
If ϕ is $\Pi(1)$-computable, then ϕ is in ETIME.

Proof
Any reduction strategy will work with a confluent system. □

7.2 Simulation of exponential time computation

Lemma 21
Let γ be a constant. The function $\lambda n.\ 2^{\gamma\cdot n}$ is $\Pi(1)$-computable

Proof
The function $\lambda n.\ 2^{\gamma\cdot n}$ is represented by $\mathtt{E}_\gamma$ which is defined as follows.

$$\mathtt{E}_\gamma(0) \to s(0)$$
$$\mathtt{E}_\gamma(r(x)) \to \mathtt{Add}(\mathtt{E}_\gamma(x),\ldots\mathtt{Add}(\mathtt{E}_\gamma(x),\mathtt{E}_\gamma(x))\ldots) \qquad (2^\gamma - 1 \text{ occurrences of } \mathtt{Add})$$

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

`Add` is defined and interpreted as in Lemma 12. The function $\lambda n.\ 2^{\gamma \cdot n}$ is $\Pi(1)$-computable because we can assign the following polynomial interpretations:

$$[r](X) = 2^{\gamma+1}(X+4) \qquad [\mathrm{E}_\gamma](X) = X + 3$$

□

Lemma 22
If ϕ is in NETIME, then ϕ is $\Delta(1)$-computable.

Proof
Let ϕ be a function computed by a NDSTM such that the time of computation is bounded by an exponential $2^{\gamma \cdot n}$ for some constant γ. The function ϕ can be obtained by composing ϕ_M as defined in Lemma 5.2 and $\lambda n.\ 2^{\gamma \cdot n}$. Since ϕ_M is $\Delta(0)$ and $\lambda n.\ 2^{\gamma \cdot n}$ is $\Pi(1)$, ϕ is $\Delta(1)$-computable by Lemma 6. □

Corollary 23 If ϕ is in ETIME, then ϕ is $\Pi(1)$-computable.

Proof
In the proof of the previous lemma, observe that ϕ_M turns out to be in $\Pi(0)$. So Lemma 6 yields that ϕ is $\Pi(1)$-computable. □

8 Doubly exponential time

8.1 $\Delta(2)$*-computable functions are doubly exponential time*

Lemma 24
Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ be a $\Delta(2)$-rewrite system. Then there is a constant $\gamma > 0$ such that, for all $t \in \mathscr{T}(\mathbb{A})$, $[t] \leqslant 2^{2^{\gamma \cdot |t|}}$.

Proof
By definition of $\Delta(2)$, there is a constant a such that for every constructor c, $[c](X_1, \ldots, X_n) \leqslant a \cdot (\prod_{i=1}^n X_i)^a$ where $X_i > 0$. Define $\gamma = 2a$. It is easy to verify that

$$[c(t_1, \ldots, t_n)] \leqslant a \cdot \left(\prod_{i=1}^n [t_i] \right)^a \leqslant a \cdot (2^{2^{\gamma \cdot (\sum_{i=1}^n |t_i|)}})^a \leqslant (2^{2^{\gamma \cdot (\sum_{i=1}^n |t_i|)}})^{2a}$$

We have that the interpretation is bounded by $2^{2^{\gamma \cdot (\sum_{i=1}^n |t_i| + 1)}}$. □

Corollary 25
Let $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$ be a $\Pi(2)$-rewrite system. Then there is a constant γ such that, for all terms $t_1, \ldots, t_n \in \mathscr{T}(\mathbb{A})$,

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

Proof
These are consequences of Lemmas 3 and 24. □

Lemma 26
If ϕ is $\Delta(2)$-computable, then ϕ is in NE_2TIME.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

Proof
By definition of $\Delta(2)$, for each term $t \in \mathscr{T}(\mathbb{A})$, we have $[t] \leqslant 2^{2^{O(|t|)}}$. So it follows from Corollary 25, that the evaluation procedure described in the proof of Lemma 9, runs in time bounded by $2^{2^{O(|t|)}}$. □

Corollary 27
If ϕ is $\Pi(2)$-computable, then ϕ is in $\mathrm{E}_2\mathrm{TIME}$.

8.2 Simulation of doubly exponential time computation

Lemma 28
Let γ be a constant. The function $\lambda n.\, 2^{2^{\gamma \cdot n}}$ is $\Pi(2)$-computable.

Proof
The following rules define $\mathtt{DE}$ which denotes the function $\lambda n.2^{2^{\gamma n}}$:

$$\begin{aligned} \mathtt{DE}(0) &\to s(s(0)) \\ \mathtt{DE}(r(x)) &\to \mathtt{Mul}(\mathtt{DE}(x), \ldots \mathtt{Mul}(\mathtt{DE}(x), \mathtt{DE}(x)) \ldots) \qquad (2^\gamma - 1 \text{ occurrences of } \mathtt{Mul}) \end{aligned}$$

$\mathtt{Mul}$ is defined and interpreted as in Lemma 12. Since these rules admit the polynomial interpretation

$$\begin{aligned} [r](X) &= (X+6)^{2^\gamma+1} \\ [\mathtt{DE}](X) &= X + 5, \end{aligned}$$

we conclude that the function $\lambda n.2^{2^{\gamma n}}$ is $\Pi(2)$-computable. □

Lemma 29
If ϕ is in $\mathrm{NE}_2\mathrm{TIME}$, then ϕ is $\Delta(2)$-computable.

Proof
Let ϕ be a function computed by a NDSTM such that the time of computation is bounded by a doubly exponential function, $\mathtt{DE}$. The function ϕ is obtained by composing ϕ_M as defined in Lemma 11 and $\mathtt{DE}$. Since ϕ_M is $\Delta(0)$-computable and $\mathtt{DE}$ is $\Pi(2)$-computable, Lemma 6 implies that ϕ is $\Delta(2)$-computable. □

Corollary 30
If ϕ is in $\mathrm{E}_2\mathrm{TIME}$, then ϕ is $\Pi(2)$-computable.

9 Exponential interpretation termination proof

We now attack the problem of showing that a function, admitting a termination proof with an exponential interpretation, is super-elementary. Exponential interpretations were considered by Lescanne (1992) in ORME. Henceforth a symbol f of arity n is interpreted by a function

$$[f](X_1, \ldots, X_n) = P(X_1, \ldots, X_n, 2^{X_1}, \ldots, 2^{X_n})$$

where P is a monotone polynomial, with positive integer coefficients, which satisfies $P(X_1, \ldots, X_n, 2^{X_1}, \ldots, 2^{X_n}) > X_i$, for $i = 1, \ldots, n$. Hence, exponential interpretations

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

provide a simplification order. Note that, the set of functions on which the interpretation is based is not closed under composition.

Say that an interpretation P is of **kind** 3, if

$$P(X_1, \ldots, X_n) = \sum_{i=1}^{n} 2^{X_i} + \gamma$$

Define $\Pi(3)$-computable functions as in Definition 5. Let us now illustrate the computational power of these functions.

Example 9

Recall Example 7. The successors p and q were interpreted by $[p](X) = X + 4$ and $[q](X) = 3 \cdot X + 12$ We define $\mathtt{Tr}$ which converts a numeral $p^n(0)$ of kind 0 into a numeral $q^n(0)$ of kind 1.

$$\mathtt{Tr}(0) \to 0$$
$$\mathtt{Tr}(p(x)) \to q(\mathtt{Tr}(x))$$

It is easy to check that the interpretation $[\mathtt{Tr}](X) = 2^X$ gives a termination proof. So, $\Delta(0)$ and $\Delta(1)$-computable functions collapse when exponential interpretations of function symbols are allowed.

Example 10

The super-elementary function $\exp_\infty$ defined by $\exp_\infty(0) = 2$ and $\exp_\infty(n+1) = 2^{\exp_\infty(n)}$ is computed by $\mathtt{exp}$ as follows

$$\mathtt{exp}(0) \to p(p(0))$$
$$\mathtt{exp}(r(x)) \to \mathtt{E}(\mathtt{Tr}(\mathtt{exp}(x)))$$

where $\mathtt{E}$ denotes the exponential function as defined in Example 7.

So, $\mathtt{exp}(r^n(0)) \overset{!}{\to} p^{\exp_\infty(n)}(0)$. The system has the following interpretation:

$$[r](X) = 2^X + 8 \qquad [\mathtt{exp}](X) = 2^X + 8$$

Theorem 31

The class of functions which are $\Pi(3)$-computable, is exactly the functions which are in $\textsc{Dtime}(\exp_\infty(n + O(1)))$.

Proof

Assume that ϕ is a unary $\Pi(3)$-function computed by $\langle \mathscr{R}, \mathbb{F}, \mathbb{A}, f, [\ ] \rangle$. For each term $t \in \mathscr{T}(\mathbb{A})$, we have $[t] \leqslant \exp_\infty(|t| + \gamma')$, for some constant γ'. Let n be the maximal size of the inputs. Lemma 3 implies that the derivation length is bounded by $\exp_\infty(n + \gamma)$ and also that each expression in the reduction process is of size bounded by $\exp_\infty(n + \gamma)$. So, ϕ is computed in time bounded by $\exp_\infty(n + \gamma + 1)$.

Conversely, to simulate a function ϕ running in time bounded by $\exp_\infty(n + \gamma)$, we use the construction in Lemma 11, where we insert the time bound provided by $\mathtt{exp}(r^{n+\gamma}(0))$. □

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press

In conclusion, it is well known that $\text{NDTIME}(T(n)) = \text{DTIME}(k^{T(n)})$. It follows that $\text{NDTIME}(\exp_\infty(n+O(1))) = \text{DTIME}(\exp_\infty(n+O(1)))$. This means that a function which is computed by a non-confluent $\Delta(3)$-rewrite system is also definable by a confluent $\Pi(3)$-rewrite system.

References

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

Bonfante, G., Cichon, A., Marion, J.-Y. and Touzet, H. (1998) Complexity classes and rewrite systems with polynomial interpretation. *CSL: Lecture Notes in Computer Science 1584*, pp. 372–384. Springer-Verlag.

Cichon, E. A. and Lescanne, P. (1992) Polynomial interpretations and the complexity of algorithms. *CADE'11: Lecture Notes in Artificial Intelligence 607*, pp. 139–147. Springer-Verlag.

Cobham, A. (1962) The intrinsic computational difficulty of functions. In: Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pp. 24–30. North-Holland.

Dershowitz, N. (1982) Orderings for term-rewriting systems. *Theor. Comput. Sci.*, **17**(3), 279–301.

Dershowitz, N. and Jouannaud, J. P. (1990) *Handbook of Theoretical Computer Science vol.B*. Elsevier.

Grädel, E. and Gurevich, Y. (1995) Tailoring recursion for complexity. *J. Symbolic Logic*, **60**(3), 952–969.

Giesl, J. (1995) Generating polynomial orderings for termination proofs. *RTA: Lecture Notes in Computer Science 914*, pp. 427–431. Springer-Verlag.

Gurevich, Y. (1983) Algebras of feasible functions. *Twenty Fourth Symposium on Foundations of Computer Science*, pp. 210–214. IEEE Press.

Hofbauer, D. and Lautemann, C. (1988) Termination proofs and the length of derivations. *RTA: Lecture Notes in Computer Science 355*. Springer-Verlag.

Hofbauer, D. (1992) Termination proofs with multiset path orderings imply primitive recursive derivation lengths. *Theor. Comput. Sci.*, **105**(1), 129–140.

Krentel, M. (1988) The complexity of optimization problems. *J. Computer & System Sci.*, **36**, 490–519.

Lankford, D. S. (1979) On proving term rewriting systems are noetherien. *Technical Report MTP-3*, Louisiana Technical University.

Leivant, D. (1994) Predicative recurrence and computational complexity I: Word recurrence and poly-time. In: P. Clote and J. Remmel, editors, *Feasible Mathematrics II*. Birkhäuser.

Lescanne, P. (1992) Termination of rewrite systems by elementary interpretations. In: H. Kirchner and G. Levi, editors, *3rd International Conference on Algebraic and Logic Programming: Lecture Notes in Computer Science 632*, pp. 21–36. Springer-Verlag.

Marion, J.-Y. (1998) An hierarchy of terminating algorithms with semantic interpretation termination proofs. *Rapport de recherche 98-R-273*, LORIA.

Péter, R. (1967) *Rekursive Funktionen*. Akadémiai Kiadó, Budapest. (English translation: *Recursive Functions*, Academic Press, New York, 1967.)

Sazonov, V. (1980) Polynomial computability and recursivity in finite domains. *Elektronische Informationsverarbeitung und Kybernetik*, **7**, 319–323.

Simmons, H. (1988) The realm of primitive recursion. *Archive for Mathematical Logic*, **27**, 177–188.

Steinbach, J. (1995) Simplification orderings: history of results. *Fundamenta Informaticae*, **24**, 47–87.

Weiermann, A. (1995) Termination proofs by lexicographic path orderings yield multiply recursive derivation lengths. *Theor. Comput. Sci.*, **139**, 335–362.

https://doi.org/10.1017/S0956796800003877 Published online by Cambridge University Press