J. Functional Programming **11** (1): 95–116, January 2001. Printed in the United Kingdom

A predicative approach to the classification problem

SALVATORE CAPORASO, EMANUELE COVINO
and GIOVANNI PANI
Dipartimento di Informatica, Università di Bari, Bari, italy

Abstract

We harmonize many time-complexity classes DTIMEF($f(n)$) ($f(n) \geqslant n$) with the PR functions (at and above the elementary level) in a transfinite *hierarchy of classes of functions* $\mathscr{T}_\alpha$. Class $\mathscr{T}_\alpha$ is obtained by means of *unlimited* operators, namely: a variant Π of the *predicative* or *safe* recursion scheme, introduced by Leivant, and by Bellantoni and Cook, if α is a successor; and *constructive diagonalization* if α is a limit. Substitution (SBST) is discarded because the time complexity classes are not closed under this scheme. $\mathscr{T}_\alpha$ is a structure for the PR functions finer than $\mathscr{E}_\alpha$, to the point that we have $\mathscr{T}_{\epsilon_0} = \mathscr{E}_3$ (elementary functions). Although no explicit use is made of hierarchy functions, it is proved that $f(n) \in \mathscr{T}_\alpha$ implies $f(n) \leqslant n^{G_\alpha(n)}$, where G_α belongs to the *slow growing hierarchy* (of functions) studied, in particular, by Girard and Wainer.

1 Introduction

1.1 Context

Since 1990 we have been indebted to Leivant (1991) for pointing out the analogy between: growth of sets, produced by an uncontrolled use of the *comprehension principle* (existence of a set for every description of its elements) together with *impredicative definitions* (in which the *definiendum* occurs inside the *definiens*); and, on the other side, growth of functions defined by nested recursion. A PR definition like $f(x, y + 1) = h(x, y + 1, f(x, y))$ is asking to compute h for a number of times, depending on the entity f being introduced. After the resource-free characterization of PTIMEF by Bellantoni and Cook (1992) and Bellantoni (1992), other complexity classes have been captured by means of variants of *Safe Recursion* (SR) schemes (Leivant, 1994; Leivant and Marion, 1995; Bellantoni, 1995; Caporaso *et al.*, 1997, 2000). Many of them reduce the circularity implicit in all recursions by denying the role of the principal variable to all variables already used as *auxiliary* in a previous recursion; they differ from one another in the choice of the initial functions and by the kind of SR scheme.

In Bellantoni and Niggl (1997), two characterizations of the Grzegorczyk classes $\mathscr{E}_{n+3}$, harmonized with LINSPACEF and, respectively, PTIMEF, are presented, together with a discussion of earlier results, like Leivant (1993) and Niggl (1998). Their classes are defined by closure under SR and substitution (SBST), while the step from one class to the next is impredicative. A characterization harmonizing LINTIMEF and PTIMEF with $\mathscr{E}_{n+3}$, by means of unlimited operators (a variant of TM's), can be found in Caporaso (1996).

1.2 Position of the problem

A *unified predicative taxonomy* collecting and connecting, under a uniform criterion, as many computational complexity classes as possible with other classes of recursive functions is lacking. In addition, little is known about how far can we go if all forms of uncontrolled circularity are avoided.

A method for getting rid of vicious circles in mathematical theories (analysis, set-theory) is based on: a simple, *safe* method; and on a *ramified* construction by *stages*, associated with ordinals – a definition at stage α may use only entities produced at earlier stages β.

The Grzegorczyk extended hierarchy $\mathscr{E}_\alpha$ is an example of ramified (though impredicative) construction. A transfinite sequence E_α of Ackermann-like *hierarchy functions* is obtained first by putting $E_{\alpha+1}(n+1) := E_\alpha^{n+1}(n)$ (as usual), and (*diagonalization*) $E_\lambda(n) := E_{\lambda(n)}(n)$. This allows defining the *hierarchy* $\mathscr{E}_\alpha$ *of classes of functions* by closure of $\bigcup_{\beta<\alpha} \mathscr{E}_\beta + E_\alpha$ under SBST and *limited* PR ($\text{PR}_\leqslant$).

$\mathscr{E}_\alpha$, however, is growing too fast for the complexity classes. In addition, we feel that SBST is incompatible with our aims, since most complexity classes are not closed under this scheme. At higher levels, it obscures the safe recursion phenomenon: in Caporaso *et al.* (1999), we show that all functions in the closure of $\mathscr{E}_2 + n^m$ under SR are dominated by n^{n^n}. Hence, this class does not exhaust $\mathscr{E}_3$, and we prove elsewhere that this kind of phenomenon also holds above the elementary functions: SBST is needed to fill the existing gap between $\mathscr{E}_{n+3}$ and the closure of $\mathscr{E}_{n+2}$ under SR and a single application of PR.

In our view, a *unified predicative taxonomy* should consist of a hierarchy $\mathscr{C}_\alpha$, obtained by (1) discarding the hierarchy functions, (2) keeping the ramified approach, (3) replacing $\text{PR}_\leqslant$ by some sort of SR, and (4) replacing SBST with predicative scheme(s) compatible with the complexity classes.

1.3 Statement of the result

On a ternary word algebra we define a recursion scheme Π, such that $f(x, y, za) = g(f(x, y, z), y, za)$ $(a = 0, 1, 2)$; y and z are the only parameter and principal variable of the recursion; the auxiliary variable x is the place where the previous value of f is stored. No other variables are admitted, and renaming of z by x is not allowed. Ternary words are interpreted and handled as tuples in binary modified form, with the zeroes playing the role of commas. Thus, the parameter and the principal variable may be encoding a potential infinity of binary variables, and Π is essentially equivalent to a form of simultaneous SR. In addition to Π we adopt the following *constructive diagonalization* scheme Δ: assume that the classes $\mathscr{T}_{\lambda(n)}$ are

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

already defined; function f is defined at λ by Δ in the enumerator e if

$$e \in \mathcal{T}_{\lambda(m)}; \quad e(n) = \ulcorner g_n \urcorner; \quad g_n \in \mathcal{T}_{\lambda(n)}; \quad f(n) = g_n(n) \quad \text{(for some } m \text{ and all } n).$$

Given some initial constant-time functions, we call *inessential* SBST (I-SBST) the restriction of SBST, asking that one of the two operands be an initial function. Hierarchy $\mathcal{T}_\alpha$ is defined by taking, as $\mathcal{T}_{\alpha+1}$, the closure under I-SBST of the class of all functions definable by a single Π in $\mathcal{T}_\alpha$; and as $\mathcal{T}_\lambda$, the closure under I-SBST of the functions definable by a single diagonalization in functions $\in \mathcal{T}_{\lambda(n)}$.

There is no circularity in the Δ scheme, which might be regarded as not less predicative than SR. Thus, we feel that hierarchy $\mathcal{T}_\alpha$ might be regarded as a partial predicative answer from below to the Gödel problem of classifying the recursive functions.

For $1 \leqslant k < \omega$ we have $\mathcal{T}_k =$DTIMEF(n^k) – a strengthening of the previous result by Leivant (1994) (which captures the *even* classes DTIMEF(n^{2k})), that, together with Caporaso (1996), might be regarded as a contribution to the discussed question of the robustness of LINTIMEF.

For $\omega \leqslant \alpha < \epsilon_0$ we have $\mathcal{T}_\alpha =$DTIMEF$(n^{coll(\alpha,n+O(1))})$, where $coll(\alpha, m)$ (read: the *collapse* of α at m) is the result of replacing ω with m in Cantor normal form for α (for example, $\mathcal{T}_{\omega^2} = \bigcup_c$DTIMEF$((n+c)^{(n+c)^2})$).

For all α we have $\mathcal{T}_\alpha =$DTIMEF$(n^{G_\alpha(n+O(1))})$, where G_α belongs to the *slow growing hierarchy* (of functions) defended (with respect to E_α) by Girard (1987, pp. 329–345) and thoroughly studied by Girard (1981) and Wainer (1989).

We show that that there is an ordinal ξ (= Bachmann's $\phi_\omega(0)$), such that $\bigcup_{\alpha<\xi} \mathcal{T}_\alpha$ equals the PR functions.

1.4 Further work

It is known that $\mathcal{E}_{\epsilon_0}$ equals the class of all functions provably recursive in **PA**, while stronger theories are reached at higher ordinals. One may then ask whether provably recursive functions exist which are *predicatively inaccessible* (with respect to hierarchies like $\mathcal{T}_\alpha$). That is, whether there is an ordinal β and a theory **T** (Σ_n-IND, **PA** or stronger), such that $\mathcal{E}_\beta$ equals the class of all **T**-provably recursive functions, and properly contains all $\mathcal{T}_\alpha$. A recent result by Weiermann (1999) implies that at Fefermann's Γ_0 we have $\mathcal{T}_{\Gamma_0} = \mathcal{E}(PR)$ (= elementary in Ackermann function). Studying $\mathcal{T}_\alpha$ at higher ordinals requires ordinal notations to be more constructive than those known to us, and might be a sensible sequel of the present research.

2 Recursion-free functions

2.1 B-words, B-functions and codes

Notation

1. a, b are variables ranging over the alphabets introduced in this paper.
2. **B** denotes the binary alphabet $\{1, 2\}$. **B**-*words* denoted by $U, \ldots, Z$ are elements of $\mathbf{B}^*$, i.e. sequences (possibly empty) over **B**. $\overline{0}$ will denote the empty **B**-word. $top(X)$ is the last letter of word X, if any, and is $\overline{0}$ otherwise.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

3. The *binary modified form* $\overline{n}$ for $n = \sum_{(0 \leqslant j \leqslant m)} b_j 2^j > 0\,(b_j = 1,2)$ is $b_0 \dots b_m$ (for example, $\overline{5} = 21$).

Definition 1
The *initial* **B**-*functions* are the *destructor* Ω and the *constructors* Γ^b

$$\Omega(Xb) = X; \quad \Omega(\overline{0}) = \overline{0}; \quad \Gamma^b(X) = Xb;$$

and the *branching* Ψ^b such that

$$\Psi^b(X,Y,Z) = \textit{ if } top(X) = b \textit{ then } Y \textit{ else } Z.$$

All expressions introduced throughout this paper should be thought of as readable transcriptions of a Polish prefix language over the *united alphabet*

$$\mathbf{U} := \{0,1,2,I,\Omega,\Gamma^b,\Psi^b,\Xi,\Phi,\Upsilon,\Pi,\Pi^*,\Delta,x,y,z,\overline{0},S,+,\phi,\times,\circ\},$$

where $\circ$ is a separator needed in certain special cases. Codes are built-up by juxtaposition from the codes for the letters of **U**, univocity being ensured by the *arity* associated tacitly with each such letter. For example: (1) $\alpha + \beta$ stands for $+\alpha\beta$, the arity of $+$ is 2 and we set, for all α, β, $\ulcorner \alpha + \beta \urcorner = \ulcorner + \urcorner \ulcorner \alpha \urcorner \ulcorner \beta \urcorner$; (2) the code for a function f defined by a scheme Σ in functions g and h is $\ulcorner \Sigma \urcorner \ulcorner g \urcorner \ulcorner h \urcorner$.

Definition 2
1. Let us write $\mathbf{n}$ for $2^{n+1}1$. The code for number i is $\mathbf{i}$, and the code $\ulcorner a \urcorner$ for the ith letter a of **U** is $\mathbf{i}$ too (for example, $\ulcorner \Omega \urcorner = \ulcorner 5 \urcorner = \mathbf{5} = 2^6 1$).
2. Let us write $\langle E_1, \dots, E_n \rangle$ for $\ulcorner E_1 \urcorner \dots \ulcorner E_n \urcorner$. If the arity of $a \in \mathbf{U}$ is n, then $\langle a, E_1, \dots, E_n \rangle$ codes the expression $aE_1 \dots E_n$. Hence, by assigning arity 1 to the letters of **B**, and arity 0 to $\overline{0}$, we have $\ulcorner Y \urcorner = \langle b_1, \langle \dots, \langle b_m, \overline{0} \rangle \rangle \dots \rangle$ for all $Y = b_1 \dots b_m$.
3. If E is a variable defined on a class C of syntactic entities, then E_X is the entity in C coded by X. Thus $Y_X, \alpha_X, f_X, M_X \dots$ are the **B**-word, ordinal, function, TM,... coded by X; however, we often write $\{X\}$ instead of f_X.

Throughout the paper, we shall make tacit use of the identities $\ulcorner \{X\} \urcorner = X$ and $\{\ulcorner f \urcorner\} = f$.

2.2 T-words and T-functions

T denotes the ternary alphabet $\{0,1,2\}$. **T**-*words* denoted by p,q,r,s,t are 0 or sequences over $\mathbf{T}^+$ beginning by 1 or 2. In principle, **T**-words are ternary numbers. In practice, we use them to handle tuples of **B**-words as single objects, according to the following stipulation.

Notation

1. Given a **T**-word s of the form $X_m 0 X_{m-1} 0 \dots X_2 0 X_1$, where X_m begins by 1 or 2, we call X_i the *ith component* of s denoted by $(s)_i$, and s is said to have $\#(s) := m$ components. If s is a **B**-word, then s is its only component; hence $\#(s) = 1$. If s is 0 then $\#(s) = 1$ and $(s)_1 = \overline{0}$. We often display a word s in this form as $X_m, \dots, X_1$.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

2. We write $rpl(X;i;s)$ for the result of replacing $(s)_i$ with X if $i \leqslant \#(s)$; otherwise $rpl(X;i;s)$ is s.

Example 1
We denote 202 and 200 by $2, 2$ and by $2, \overline{0}, \overline{0}$. We display $s = 100220$ by $1, \overline{0}, 22, \overline{0}$; we then have $rpl(11;3;s) = 1, \overline{0}, 11, \overline{0}$.

Variables and functions
An *n-ary* **B**-*function* maps $(\mathbf{B}^*)^n$ $(n < \omega)$ into $\mathbf{B}^*$. An *n-ary* **T**-*function* f takes n **T**-words $(0 < n \leqslant 3)$ into a **T**-word. Unlike $X, s, X_1, s_1 \ldots$ which form a potential infinity of informal variables, x, y, z are three fixed syntactic objects, respectively called the 'auxiliary variable', the 'parameter', and the 'principal variable'. They play a precise and distinct role in the construction of the **T**-functions. $u, v, w, u_1, \ldots$ are variables defined on the syntactic objects x, y, z. With a notation like $f(x, y, z)$ we always admit that some of the indicated variables may be absent (with a bit of common sense, however). $f(s, t, r)$ is the value of $f(x, y, z)$ when the system of values s, t, r is assigned to the variables.

Note
The very role of the arguments s, t, r for $f(x, y, z)$ is handling as single objects an l-ple of variables over $\mathbf{B}^*$, used as initial/auxiliary, an m-ple of parameters, and an n-ple of recursion variables. Accordingly, f should be regarded as a function defined on $(\mathbf{B}^*)^l \times (\mathbf{B}^*)^m \times (\mathbf{B}^*)^n$ rather than on $(\mathbf{T}^+)^3$. Since the number of arguments of a function should be defined, an *intended number of components* is always assigned implicitly to the arguments of a **T**-function (see section 4.3).

Definition 3
The *initial* **T**-*functions* are the *identity* $I(x, y, z) = x$; the *destructors* $\Omega_i(x)$ and *constructors* $\Gamma_i^b(x)$ such that for all s we have

$$\Omega_i(s) = rpl(\Omega((s)_i); i; s); \quad \Gamma_i^b(s) = rpl(\Gamma^b((s)_i); i; s).$$

Definition 4
The *initial class* $\mathscr{T}_0$ of hierarchy $\mathscr{T}_\alpha$ is the closure of the initial **T**-functions under the following *simple schemes*:

1. The *assignment* schemes $\Xi[q, u](g)$ and the *renaming* schemes $\Phi_{uw}(g)$ take function g into the function f which is obtained, respectively, by assigning q to u, and by replacing u with w ($f = g$ if u is absent in g). The only allowed renamings are Φ_{xy}, Φ_{xz} and Φ_{zy}. (In the Note in Section 3.1, we see that an essential point in the present work is that Φ_{zx} is not allowed.)
2. Function $f = \Upsilon(g, h)$ is defined by the *inessential substitution* (I-SBST) scheme if f is defined by SBST in g and h, provided that g or h is an initial function.
3. Function $f = \Psi_i^b(e, g, h)$ is defined by the *ith branching scheme* in functions e, g, h if for all s, t, r we have that $e(s, t, r) = q$ implies

$$f(s, t, r) = \textit{ if } \; i \leqslant \#(q) \textit{ and } top((q)_i) = b \textit{ then } g(s, t, r) \textit{ else } h(s, t, r).$$

Definition 5
A *modifier* is an element of the closure of the initial functions under I-SBST. A function is in *normal form* if it is a modifier or if it is in the form

$$f(x, y, z) = \mathit{if}\ e_1\ \mathit{then}\ g_1\ \mathit{else}\ \mathit{if}\ e_2\ \mathit{then}\ g_2\ \mathit{else}\ \dots g_n,$$

where all g_i are modifiers, and all e_i are expressions of the form 'the kth digit of $(s)_n$ is b', that we will call *tests*.

It can be easily proved that all functions in $\mathscr{T}_0$ can be written in normal form.

3 Recursion and diagonalization

3.1 Predicative recursion

Definition 6
Function $f = \Pi(g, h)$ is defined by the *recursion* scheme Π in the *basis function* $g(x, y, z)$ and in the *step function* $h(x, y, z)$ if we have

$$\begin{cases} f(s, t, a) & = & g(s, t, a) \\ f(s, t, ra) & = & h(f(s, t, r), t, ra). \end{cases}$$

Notation
We write $\Pi^*(g)$ for $\Pi(\Phi_{zy}(g), \Phi_{zy}(g))$ and we let $\ulcorner \Pi^* \urcorner \ulcorner g \urcorner$ code a function in this form (see the next example for the rationale of this clause).

$\mathscr{T}(\omega)$ will denote the closure of $\mathscr{T}_0$ under Π and the simple schemes. The next lemma is an analogue for our system of the *Bounding Theorem* in Bellantoni and Niggl (1997) and Niggl (1997).

Lemma 7
If $f \in \mathscr{T}(\omega)$ is defined by means of d constructors and $b \geqslant 0$ Π's, then

$$|f(s, t, r)| \leqslant |s| + d(|t| + |r|)^b.$$

Proof
Induction on b and on the definition of f. For $b = 0$ we obviously have $|f(s, t, r)| \leqslant |s| + d$.

Assume $b = c + 1$, and define $n := |t| + |r|$. *Case 1.* f begins with a constructor.

The two induction hypotheses give $|f(s, t, r)| \leqslant |s| + (d - 1)n^b + 1$.

Case 2. We may now assume $f = \Pi(g, h)$ (since otherwise the result follows immediately by the induction hypotheses). We show that we have $|f(s, t, r)| \leqslant |s| + d|r|n^c$. The basis of induction on $|r|$ is shown immediately by the two main induction hypotheses:

$$\begin{array}{rll} |f(s, t, ra)| & = |h(f(s, t, r), t, ra)| & \\ & \leqslant |f(s, t, r)| + d(n + 1)^c & \text{by the two main ind. hyp.} \\ & \leqslant |s| + d|r|n^c + d(n + 1)^c & \text{by the ind. hyp. on } |r| \\ & \leqslant |s| + d(n + 1)^{c+1}. & \end{array}$$

□

Note
Assume given h and a numerical function F such that for all q, t, r we have $|h(q, t, r)| \leqslant |q| + F(|t|, |r|)$. One sees immediately from the proof above that $f = \Pi(g, h)$ implies $|f(s, t, ra)| \leqslant |f(s, t, r)| + F(|t|, |ra|)$.

Example 2
Define a sequence f_n $(n < \omega)$ of functions by

$$g_0 := \Gamma^1_1; \qquad g_{n+1} := \Phi_{zy}(f_{n+1}); \quad f_{n+1} := \Pi(g_n, g_n).$$

We have

$$\begin{cases} f_1(s,t,a) &=& s1 \\ f_1(s,t,ra) &=& f_1(s,t,r)1 \end{cases}; \qquad \begin{cases} f_{n+1}(s,t,a) &=& g_n(s,t) \\ f_{n+1}(s,t,ra) &=& g_n(f_{n+1}(s,t,r),t). \end{cases}$$

Thus, for all **B**-words s we have $f_1(s, t, r) = s1^{|r|}$ and $g_1(s, t) = s1^{|t|}$. An induction on n and r gives $|f_{n+1}(s, t, r)| = |s| + |t|^n|r|$ and $|g_n(s, t)| = |s| + |t|^n$.

Notice that $|\ulcorner f_n \urcorner|$ grows like 2^n. Since this would interfere with further developments (see the conclusion of this example after Definition 12, and the construction of functions g_α in proof of Lemma 24), we observe that functions f_n may be written in the form

$$f_0 := \Gamma^1_1; \quad f_{n+1} := \Pi^*(f_n) \qquad (= \Pi^*(\Pi^*(\ldots(\Pi^*(\Gamma^1_1))\ldots)) \ (n+1 \text{ times})).$$

Note
Assume the renaming scheme Φ_{zx} is available, and let functions h_n be obtained by replacing Φ_{zy} by Φ_{zx} in the functions g_n above. Since f_1 may be regarded as a sum in unary, we see that $h_2(s, t, ra)$ is doubling its value for r, and therefore needs an exponential space. Functions h_n are growing like functions E_n of the fast-growing hierarchy reported in section 3.5. Thus, forbidding Φ_{zx} is essentially equivalent to the semicolon used to keep x *dormant* (Simmons, 1988) or *safe* (Bellantoni and Cook, 1992).

3.2 Ordinals

Notation

1. Greek small letters are ordinal numbers; λ and μ are limit ordinals. λ_n or $\lambda(n)$ is the nth element of the Fundamental Sequence (FS) assigned to $\lambda = \sup(\lambda_n)$ by the assignment of FS's of the next definition.
2. $F^n(E, \ldots)$ denotes the nth iterate of F at E, i.e. $F^0(E, \ldots) = E$ and $F^{n+1}(E, \ldots) = F(F^n(E, \ldots), \ldots)$.

Definition 8
Define simultaneously the *n-critical ordinals* $\phi_n(\alpha)$ and an assignment of FS's by

$$\begin{cases} \phi_0(\alpha) &=& \omega^\alpha \\ \phi_{n+1}(0) &=& \sup_m(\phi^m_n(0)) \\ \phi_{n+1}(\beta+1) &=& \sup_m(\phi^m_n(\phi_{n+1}(\beta)+1)) \\ \phi_n(\lambda) &=& \sup_m(\phi_n(\lambda_m)) \end{cases} \qquad (\text{thus } (\omega^\lambda)_x = \omega^{\lambda_x});$$

the other FS's are given by $(\omega^{\alpha+1})_m = \omega^\alpha \cdot m; \quad (\lambda + \mu)_m = \lambda + \mu_m$.

Example 3
Writing $\omega_0(\alpha)$ for α and $\omega_{n+1}(\alpha)$ for $\omega^{\omega_n(\alpha)}$, we have $\phi_1(0) = \sup(\omega_m(0)) = \epsilon_0$; $\phi_1(1) = \sup(\omega_m(\epsilon_0 + 1)) = \epsilon_1$; and $\phi_2(0) = \sup(\epsilon_0, \epsilon_{\epsilon_0}, \epsilon_{\epsilon_{\epsilon_0}}, \ldots)$.

Ordinals are coded as follows: $\ulcorner 0 \urcorner$ codes 0; $\ulcorner \alpha + 1 \urcorner := \langle S, \alpha \rangle$; $\ulcorner \alpha + \beta \urcorner = \langle +, \alpha, \beta \rangle$; $\ulcorner \alpha \cdot n \urcorner := \langle \times, \alpha, n \rangle$; $\ulcorner \phi_n^m(\alpha) \urcorner := \langle \phi, n, m, \alpha \rangle$.

The following lemma allows us to move in polynomial time from the code for a limit ordinal to the code for the nth element of its FS.

Lemma 9

1. A poly-time TM FS can be defined such that $FS(\ulcorner \lambda \urcorner, Y) = \ulcorner \lambda_{|Y|} \urcorner$.
2. The poly-time TM's SC, LM can be defined, which respectively accept the codes for the successor and for the limit ordinals.

Proof

1. FS has to parse $\ulcorner \lambda \urcorner$ to find a left/innermost $+$, $\times$ or ϕ. By scanning the clauses of Definition 8, we see that we have the worst case (a quadratic time complexity) when case $\phi_0(\phi_0(\ldots(\phi_0 \cdot n_1)\ldots) \cdot n_k) \cdot n_{k+1}$ has to be handled, since, for example, we have $(\omega^{\omega^{\omega \cdot 5 \cdot 4}} \cdot 3)_m = \omega^{\omega^{\omega \cdot 5 \cdot 4}} \cdot 2 + \omega^{\omega^{\omega \cdot 5 \cdot 3 + \omega^{\omega \cdot 4 + m}}}$.
2. Immediate.

□

3.3 Unrestricted diagonalization

Definition 10
Assume a limit ordinal λ and a family of classes $\mathscr{C}_\alpha$ ($\alpha \leqslant \lambda$) of **T**-functions. Function $f = \Delta^u(e, \lambda)$ is defined by *unrestricted diagonalization* of *degree* m in the *enumerator* $e(z)$ at λ if $e(z) \in \mathscr{C}_m$ and if for all s, t, r we have

$$f(s, t, r) = \{e(r)\}(s, t, r); \quad \{e(r)\} \in \mathscr{C}_{\lambda(|r|)}.$$

Example 2 (continued) We use Δ^u for extending to the transfinite the sequence f_n. To this purpose, let us take as $\mathscr{C}_\alpha$ the classes $\mathscr{T}_\alpha$ of Definition 15. Let $\Gamma[Y](x)$ be the function in $\mathscr{T}_1$ such that for all s we have $\Gamma[Y](s) = Ys$. Define in $\mathscr{T}_2$ the function $e^* := \Pi^*(\Gamma[\ulcorner \Pi^* \urcorner])$. We have

$$\begin{cases} e^*(s, a) & = & \ulcorner \Pi^* \urcorner s \\ e^*(s, ra) & = & \ulcorner \Pi^* \urcorner e^*(s, r) & = & \ulcorner \Pi^*(\Pi^*(\ldots(\Pi^*(s))\ldots)) \urcorner \ (|r| + 1 \text{ times}). \end{cases}$$

Define further

$$\begin{aligned} e_0 &:= \Xi[\ulcorner \Gamma_1^1 \urcorner, x](e^*); \\ e_{\omega(k+1)} &:= \Xi[\ulcorner f_{\omega(k+1)} \urcorner, x](e^*); \\ f_{\omega(k+1)} &:= \Delta^u(e_{\omega k}, \omega(k+1)). \end{aligned}$$

Claim 1 For all $k > 0$ we have $|f_{\omega k}(s, t, t)| = |s| + |t|^{k|t|}$.

Proof We show by induction on k that we have (writing c, m, n, l for $|r|, |s|, |t|, |q|$)

$$|\{e_{\omega k}(r)\}(s, t, q)| = m + n^{kn} n^{c-1} l; \tag{1}$$

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

the claim follows, since, for $n = c = l$, we then have

$$|f_{\omega(k+1)}(s,t,t)| = |\{e_{\omega k}(t)\}(s,t,t)| = m + n^{kn}n^{n-1}n = m + n^{(k+1)n}.$$

We have $\{e_0(r)\} = \{e^*(\ulcorner\Gamma_1^1\urcorner, r)\} = f_c$, and, by the first part of this example, $|f_c(s,t,q)| = m + n^{c-1}l$.

Induction on c and l. We have

$$\begin{array}{ll}
|\{e_{\omega k}(a)\}(s,t,b)| & = |\Pi^*(f_{\omega k})(s,t,b)| = |f_{\omega k}(s,t,t)| = (\text{ind. on } k)\ m + n^{kn};\\
|\{e_{\omega k}(a)\}(s,t,qb)| & = |f_{\omega k}(\{e_{\omega k}(a)\}(s,t,q),t,t)|\\
& = (\text{ind on } k \text{ and } l, \text{ since } c-1=0)\ m + n^{kn} + n^{kn}l;\\
|\{e_{\omega k}(ra)\}(s,t,b)| & = |\Pi^*(\{e_{\omega k}(r)\})(s,t,b)| \quad (\text{by definition of } e^*)\\
& = |\{e_{\omega k}(r)\}(s,t,t)| = m + n^{kn}n^{c-1};\\
|\{e_{\omega k}(ra)\}(s,t,qb)| & = |\{e_{\omega k}(ra)\}(s,t,q)| + |\{e_{\omega k}(r)\}(s,t,t)| \ (\text{Note 3.1})\\
& = m + n^{kn}n^{c-1}l + n^{kn}n^{c-1}.
\end{array}$$

We now show a function which *computes in unary* n^{n^2}. To this purpose, define (here only, for simplicity, we code by $\ulcorner\Delta\urcorner\ \ulcorner e\urcorner$ a function in the form $\Delta^u(e,\lambda)$)

$$\begin{cases} e^{**}(s,a) & = \quad s\\ e^{**}(s,ra) & = \quad \ulcorner\Delta\urcorner\ \ulcorner\Xi\urcorner\ e^{**}(s,r)\ \ulcorner x\urcorner\ \ulcorner e^*\urcorner \end{cases}$$
$$e_{\omega^2} := \Xi[\ulcorner f_\omega\urcorner, x](e^{**})$$
$$f_{\omega^2} := \Delta(e_{\omega^2}, \omega^2).$$

Claim 2 We have $|f_{\omega^2}(s,t,t)| = |s| + |t|^{|t|^2}$.

Proof
Notations as under Claim 1. We show by induction on c that, for all r, q, s, t

$$\{e_{\omega^2}(r)\}(s,t,q) = f_{\omega c}(s,t,q).$$

$e_{\omega^2}(a) = \Xi[\ulcorner f_\omega\urcorner, x](e^{**})(s) = \ulcorner f_\omega\urcorner$.

We have

$$\{e_{\omega^2}(ra)\}(s,t,q) = \{\ulcorner\Delta\urcorner\ \ulcorner\Xi[e^{**}(r),x](e^*)\urcorner\}(s,t,q) = \{\ulcorner\Delta(e^*(\ulcorner f_{\omega c})\urcorner, z)\urcorner\}(s,t,q)$$

(by the induction hypotheses) $= \{\ulcorner\Delta(e_{\omega c})\urcorner\}(s,t,q)$ (by definition of $e_{\omega k}$) $= f_{\omega(c+1)}(s,t,q)$ by def. of f_k. $\square$

It is not clear to these authors how far one can go with this approach *from below* (ϵ_0 is reached in Caporaso *et al.* (1999) without making use of the recursion theorem).

3.4 Restricted diagonalization

Definition 11
The *length* $lh(f)$ of function f is given by

$$\begin{cases} 1 & \text{if } f \text{ is an initial function}\\ |q| + lh(g) + 1 & \text{if } f = \Xi[q,u](g)\\ lh(e) + 1 & \text{if } f \text{ is defined by unrestricted diagonalization in } e\\ 2lh(h) + 3 & \text{if } f = \Pi^*(h)\\ \sum_i lh(g_i) + 1 & \text{if } f = \Sigma(g_1,\ldots,g_n), \text{ with } n \leqslant 2, \text{ and } \Sigma \text{ is an other scheme.} \end{cases}$$

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

The *degree* $dg(f)$ of function f is given by

$$\begin{cases} 0 & \text{if } f \text{ is an initial function} \\ \sup(m, \max_z(dg(\{e(|z|)\}))) & \text{if } f \text{ defined by unrestricted diagonalization of} \\ & \text{degree } m \text{ in } e \\ \sup_i(dg(g_i)) & \text{if } f = \Sigma(g_1, \ldots, g_n), \text{ and } \Sigma \text{ is another scheme.} \end{cases}$$

Definition 12
Function $f(x, y, z) = \Delta(e, \lambda)$ is defined by *(restricted) diagonalization* if its degree is finite. The code for $f = \Delta(e, \alpha)$ is $\langle \Delta, e, \alpha \rangle$.

Example 2 (concluded) All functions of this example are defined by restricted diagonalization, since they are defined by diagonalizations of degrees $\leqslant 3$ in functions which enumerate functions whose degrees, in turn, are $\leqslant 2$.

3.5 A fast-growing and two slow-growing hierarchies

The *fast-growing hierarchy* E_α and the *slow-growing hierarchy* G_α mentioned in the Introduction are the following transfinite sequences of functions:

$$\begin{array}{rclrclrcl} E_0(n) & = & n+1; & E_{\alpha+1}(n) & = & E_\alpha^n(n); & E_\lambda(n) & = & E_{\lambda_n}(n) \\ G_0(n) & = & 0; & G_{\alpha+1}(n) & = & G_\alpha(n)+1; & G_\lambda(n) & = & G_{\lambda_n}(n). \end{array}$$

We now define a variant of the slow-growing hierarchy which better copes with the complexity classes.

Definition 13
The *slow-growing hierarchy* $B_\alpha(n)$ is given by

$$B_0(n) = 1; \qquad B_{\alpha+1}(n) = nB_\alpha(n); \qquad B_\lambda(n) = B_{\lambda_n}(n).$$

Note

1. By induction on $\alpha < \phi_\omega(0)$ one sees that, for $\beta \leqslant \alpha$, we have
$$G_{\beta+\alpha}(n) = G_\beta(n) + G_\alpha(n); G_{\omega^\alpha}(n) = n^{G_\alpha(n)} = B_\alpha(n); B_{\alpha+\beta}(n) = B_\alpha(n)B_\beta(n).$$
2. Thus, $\alpha_k \geqslant \ldots \geqslant \alpha_1$ implies $B_{\omega^{\alpha_k}+\ldots+\omega^{\alpha_1}} = B_{\omega^{\alpha_k}}(n) \cdot \ldots \cdot B_{\omega^{\alpha_1}}(n)$.
3. Hence $B_m(n) = n^m, B_{\omega \cdot c}(n) = n^{cn}, B_{\omega^c}(n) = n^{n^c}, B_{\omega_c}(n) = n^{n^{\cdot^{\cdot^{n}}}}$ (c times).
4. On the other hand, we have $E_1(n) = 2n$ and $E_2(n) = 2^n n$.

Note
The connection between hierarchies $E_\alpha(n)$ and $B_\alpha(n)$ provided by the next lemma is easily proved via a result by Cichon and Wainer (1983) that we now report. A two-places variant $F_\alpha(n, m)$ of the fast hierarchy is defined, such that, for all $n > 2$ we have (Chichon and Wainer, 1983, p. 402)

$$E_{2+\alpha}(n-1) \leqslant F_\alpha(n, n) \leqslant E_{2+\alpha}(2(n+1)^2). \tag{2}$$

Now for all finite α (Chichon and Wainer, 1983, p. 406, since, for all finite α their f_{γ_α} equals F_α)

$$G_{\phi_a(\alpha)}(n) = F_a(G_\alpha(n), n). \tag{3}$$

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

Lemma 14

1. For all $a, b < \omega$ and $n > 2$ we have $B_{\phi_a^b(0)}(n) \leqslant E_{a+2}^{b+1}(n)$.
2. For all $a, b < \omega$ and $n > 2$ we have $E_{a+2}^{b}(n) \leqslant B_{\phi_a^{2b}(0)}(n+b)$.

Proof

1. We have

$$
\begin{array}{lll}
B_{\phi_a^b(0)}(n) & = n^{G_{\phi_a^b(0)}(n)} \leqslant G_{\phi_0(\phi_a^b(0))}(n) & \text{Note 3.5} \\
 & \leqslant G_{\phi_a^{b+1}(0)}(n) \leqslant G_{\phi_a^{b+1}(\omega)}(n) & \\
 & \leqslant F_a(G_{\phi_a^b(\omega)}(n), n) & \text{Note 3.5, (3)} \\
 & \leqslant E_{a+2}(2(G_{\phi_a^b(\omega)}(n)+1)^2) & \text{Note 3.5, (2)} \\
 & \leqslant E_{a+2}^2(G_{\phi_a^b(\omega)}(n)) & \text{since } E_{a+2}(2(n^2+1)) \\
 & & \leqslant E_{a+2}(E_2(n)) \text{ for } n > 2 \\
 & \leqslant E_{a+2}^b(F_a(G_\omega(n), n)) & \text{by repeating for } b-1 \text{ times} \\
 & \leqslant E_{a+2}^b(F_a(n,n)) \leqslant E_{a+2}^{b+1}(n) & \text{since } G_\omega(n) = n
\end{array}
$$

2. Induction on b. By Note 3.5 and definitions of G_a and B_α, we have

$$E_{a+2}(n) \leqslant F_a(n+1, n+1)$$

$$\leqslant F_a(G_\omega(n+1), n+1) \leqslant G_{\phi_a(\omega)}(n+1) \leqslant B_{\phi_a(\omega)}(n+1) \leqslant B_{\phi_a^2(0)}(n+1).$$

We have $E_{a+2}^{b+1}(n) \leqslant E_{a+2}^{b}(B_{\phi_a^2(0)}(n+1))$ (same arguments as under the basis of the induction) $\leqslant B_{\phi_a^{2b}(0)}(B_{\phi_a^2(0)}(n+1)+b)$ (induction hypothesis) $\leqslant B_{\phi_a^{2(b+1)}(0)}(n+b+1)$.

□

3.6 The hierarchy $\mathscr{T}_\alpha$

The *extended Grzegorczyk hierarchy* consists of the classes $\mathscr{E}_\alpha$ of functions obtained by closure under $\mathrm{PR}_{\leqslant}$ and ordinary SBST of $\{E_\alpha\} \cup \bigcup_{\beta<\alpha} \mathscr{E}_\beta$.

We follow here another approach: a hierarchy $\mathscr{T}_\alpha$ is defined by means of *unlimited* operators and, therefore, without any explicit reference to hierarchy functions; the variant B_α of the slow hierarchy is then used to discuss the size of the elements of hierarchy $\mathscr{T}_\alpha$.

Definition 15

The *hierarchy* $\mathscr{T}_\alpha$ is given by (see Definition 4 for $\mathscr{T}_0$)

1. $\mathscr{T}_{\alpha+1}$ is the closure of the functions defined by Π in $\mathscr{T}_\alpha$ under the simple schemes.
2. $\mathscr{T}_\lambda$ is the closure of the functions defined by Δ in $\mathscr{T}_{\lambda_n}$ under the simple schemes.

Sometimes we write $\mathscr{T}(\alpha)$ for $\bigcup_{\beta<\alpha} \mathscr{T}_\beta$.

3.7 Main result

Notation

DTIMEF$^*(f(n))$ is the class DTIMEF$(f(n+O(1)))$.

Notice that if $n \cdot n^n$ is an upper time bound for f, we have $f \in$DTIMEF$^*(n^n)$, though $f \notin$DTIMEF(n^n). However the small classes are not disturbed by the distinction between DTIMEF$^*(f)$ and DTIMEF(f), since DTIMEF$^*(B_m(n))$=DTIMEF $(B_m(n))$ =DTIMEF(n^m).

The next theorem holds under the notion of equivalence of Definition 17.

Theorem 16

1. For all α we have DTIMEF$^*(B_\alpha(n)) = \mathscr{T}_\alpha$.
2. For all n we have $\mathscr{T}(\phi_{n+1}(0)) = \mathscr{E}_{n+3}$.

For example, $\mathscr{T}(\omega)$ and $\mathscr{T}(\epsilon_0)$ are equivalent to PTIMEF and to the elementary functions.

Proof

1. By Lemmas 20 and 25.
2. By part 1 and Lemma 14, since it is known that a function is in $\mathscr{E}_{n+3}$ iff (Rose, 1984, p. 77) it can be computed by a TM in time bounded by a function in $\mathscr{E}_{n+3}$ iff it is dominated by E^c_{n+2} for a constant c (in the application of part 2 of Lemma 14 one uses the notation above). □

4 Three classes of Turing Machines

4.1 Ordinary TMs

Simulation of **T**-functions will be performed by means of *ordinary* many-tapes TM's over a finite alphabet $\mathbf{U}_1$, obtained by adding to **U** the symbol # (to be used as *blank*), and a number of *markers* PSI,DELTA, ...

A *while*-ordinary TM is in the form $M := while\ M_1\ do\ M_2$ where the states of M_1 and M_2 are disjoint, M_1 is an acceptor with final states q_{yes}, q_{no}, and M_2 has to be repeated while M_1 accepts. Time for M is the overall time taken by the repetitions of M_1 and M_2, without any extra-charge, since moving from M_1 to M_2 and back is ruled by means of changes in the states, which do not require any action on the tapes.

4.2 A restricted form of TM

For simulations by **T**-functions we will restrict ourselves to *push-down binary* TM's. Such a TM M has $m+1$ states and k *push-down tapes* over **B**. Its final (initial) state is 0(1). The behaviour of M at each step only depends on the current state i, and on the top symbol of a tape defined by a special function $j(i)$. M consists of m *rows* (one for each state $i \neq 0$) of the form $(i, j(i), i_1, j_1, I_1, i_2, j_2, I_2, i_3)$. Each such row should be understood as

if the current state is i then
if $top(j(i)) = 1$ then enter state i_1 and apply I_1 to j_1;
if $top(j(i)) = 2$ or $j(i)$ is empty then enter state i_2 and apply I_2 to j_2
if $j(i)$ is empty then enter state i_3,

where I_1 and I_2 may be the commands *pop, push* 1 *and push* 2.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

Note

All classes DTIMEF($F(n)$) considered in this paper are robust with respect to the distinction between ordinary and binary push-down TM's. Indeed, let an ordinary n tapes TM M be given, and assume that its alphabet has already been reduced to $\mathbf{B}$. M is simulated by a TM N with $2n$ push-down tapes, which stores in (its tape) $2i-1$ the contents at the left of the scanned symbol of M's ith tape, and on $2i$ the part at the right, read in reverse order. A move left (right) on i by M corresponds to a pop (push) on $2i$ and to a push (pop) on $2i+1$.

4.3 Equivalence between T-functions and Turing-computable functions

Notation

Given a binary TM M with k push-down tapes, $T_i = X$ means that the contents of tape T_i is the $\mathbf{B}$-word (possibly empty) X.

M by input $s = X_1, \ldots, X_n$ *standard computes* $q = Y_1, \ldots, Y_m$ if M starts operating with $T_i = X_i$ ($1 \leqslant i \leqslant n$), and stops operating with $T_j = Y_j$ ($1 \leqslant j \leqslant m$), leaving un-changed all other $k - \max(n,m)$ tapes.

Notation

$M(s) =_{sc} q$; and $M(s) =_{sc} M_1(s)$ for $M(s) =_{sc} q$ iff $M_1(s) =_{sc} q$.

$G : (\mathbf{B}^*)^m \mapsto \mathbf{B}^*$ is *standard computed* by M if $G(s) = q$ implies $M(s) =_{sc} q$.

Definition 17

DTIMEF($F(n)$) $\subseteq \mathscr{T}_\alpha$ if for all G standard computed by a binary push-down TM M_G in time $F(|s|)$, there is $f \in \mathscr{T}_\alpha$ such that $M_G(s) =_{sc} f(s)$.

Conversely, $\mathscr{T}_\alpha \subseteq$DTIMEF($F(n)$) if for all $f \in \mathscr{T}_\alpha$ and for all $m > 0$, there is an ordinary TM $M_{(m)}$ which, by input s, t, r such that $\#(s), \#(t), \#(r) \leqslant m$, yields $f(s,t,r)$ within time $F(|s| + |t| + |r|)$.

Comment The restriction of the second part of this definition to arguments whose number of components is pre-asigned plays an essential role in the proof of Lemma 20. See the Note in section 2.2 for a justification of this restriction.

4.4 Poly-time TMs

Definition 18

An *explicitly poly-time* TM (EPTM) is a triple $p = (M, a, b)$, where M is a binary push-down TM; and where a and b are numbers such that M by input q runs in time $(a + |q|)^b$. We call a the *additive term* and b the *main term*. (a, b) is an *appropriate bound* for M if (M, a, b) is an EPTM.

$(M, a, b)(s) =_{sc} q$ means that $M(s) =_{sc} q$ and (a, b) is an appropriate bound for M.

The code for an EPTM (M, a, b) is $\langle M, \overline{a}, \circ, \overline{b} \rangle$ (where $\overline{n}$ is n in modified binary, and $\circ$ is needed to separate $\ulcorner \overline{a} \urcorner$ from $\ulcorner \overline{b} \urcorner$). p_Z is the EPTM coded by Z.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

5 Simulation by TMs

Definition 19
Function $g(x, y, z) \in \mathscr{T}_1$ is *simple* if it is obtained by modifying x only. More precisely, if all modifiers occurring in g are in the form $h(x)$ and if no renaming occurs in the basis and step functions of all recursions used to define g. A function f is simple if all functions in $\mathscr{T}_1$ used to define it are simple.

Comment
Let $g \in \mathscr{T}_1$ be simple, and let it begin with Π. Since x is used to store the previous values of g, since all its modifiers are changing x only, and since no renaming occurs in g, it does not happen that, during the recursive computation of $g(s, t, r)$, there is a value qa of the recursion variable such that g *forgets* its value for q, to re-start with a value for qa, obtained by modifying t or qa. Notice that all functions of next section are simple; hence, simplicity does not interfere with the simulation of TMs.

Notation
τ (possibly with superscripts) is a tuple of tapes over $\mathbf{U}_1$ (see section 4.1). $\tau_i^{\cdots}$ is the i-th tape of tuple $\tau^{\cdots}$.

Since every $\mathbf{T}$-function f may be described by a word E_f over $\mathbf{U}$, we identify the input-functions f for the interpreter of next lemma with their representations E_f over its tapes.

Lemma 20
$\mathscr{T}_\alpha \subseteq DTIMEF^*(B_\alpha(n))$.

Proof
Define $/f/ := lh(f) + dg(f)$. We show that for all $N > 0$ there exists an ordinary TM $INT_{(N)}$ such that, for all simple $f \in \mathscr{T}_\alpha$ and for all s, t, r whose number of components is $\leqslant N$ (cf. Definition 17) we have (writing c for $/f/$, and n for $|t| + |r|$)

$$INT_{(N)}(f, s, t, r) = f(s, t, r) \quad \text{within time} \quad c^2 B_\alpha(n + c^2). \tag{4}$$

Hence, every $f \in \mathscr{T}_\alpha$ is simulated in time $O(B_\alpha(n+O(1)))$ by the sequence composition of the constant-time TM writing (the word over $\mathbf{U}$ describing) f with $INT_{(N)}$.

Construction
In addition to some working tapes the TM $INT_{(N)}$ sketched-down in figure 1 uses as stacks the following n-ples of ordinary tapes (subscript $_{(N)}$ omitted hereinafter):

- τ^x, τ^y, τ^z, to store the intermediate computed values associated with x, y, z;
- τ^u, to store the current value of the principal variable of the current enumeration or recursion;
- τ, to store some sub-functions of f;
  the initial contents of $\tau^x, \tau^y, \tau^z, \tau$ are the input values s, t, r, and f, while τ^u is empty.

INT repeats, until τ is not empty, the following cycle (the terms *pop, push* should be understood as sequences of writing/erasing instructions, not as elementary commands of a push-down TM):

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

- it pops a function k from the top of τ, and un-nests the outermost sub-function j of k;
- according to the form of j, it carries out a different action on the stacks;
- in all other cases, it pushes into τ an information of the form $j\ MK\ k$, where MK is a mark (belonging to $\mathbf{U}_1$) informing about the outermost scheme used to define j.

$INT(f,s,t,r)$:=
$\tau := f;\ \tau^x := s;\ \tau^y := t;\ \tau^z := r;$
while τ not empty *do* $A := \text{pop}(\tau)$;
case

$A = \Upsilon(g,h)$	*then*	push $g\#h$ in τ
$A = \Phi_{vw}(h)$	*then*	push h in τ; copy last record of τ^w into τ^v
$A = \Psi_i^b(e,g,h)$	*then*	push $A\ PSI$ into τ; copy last record of τ^x into τ^u
$A = \Psi_i^b(e,g,h)\#PSI$	*then*	pop τ; if $top(\tau_i^x) = b$ *then* push g into τ *else* push h into τ; pop last record from τ^x; pop last record from τ^u and push it into τ^x
$A = \Delta(h,\alpha)$	*then*	push $DELTA\#h$ into τ; copy last record of τ^x into τ^u
$A = DELTA$	*then*	pop τ; pop last record of τ^x and push it into τ; pop last record from τ^u and push it into τ^x
$A = \Pi(g,h)$	*then*	push $A\#PI\#g$ into τ; copy last record of τ^z into τ^u push last digit of τ^u into τ^z
$A = \Pi(g,h)\#PI$	*then*	*if* $\tau^u = \tau^z$ *then* pop τ; pop τ^u; pop τ^z *else* push h into τ; pop last digit of τ^u and push it into τ^z
$A = \Pi^*(h)$	*then*	push $\Pi(\Phi_{zy}(h), \Phi_{zy}(h))$ into τ
$A = \Omega_i$	*then*	cancel and move left on τ_i^x
$A = \Gamma_i^b$	*then*	write b on τ_i^x and move right.

end *case*; end *while*.

Fig. 1

Time complexity
We show, by induction on α and c that, for all $f \in \mathscr{T}_\alpha$ (not beginning with Π^*), INT moves within $c^2 B_\alpha(n+c^2)$ steps from a configuration of the form

$$\tau = e\#f;\ \tau^x = s_0\#s;\ \tau^y = t_0\#t;\ \tau^z = r_0\#r;\ \tau^u = q,$$

to a configuration of the form

$$\tau = e;\ \tau^x = s_0\#f(s,t,r);\ \tau^y = t_0\#t;\ \tau^z = r_0\#r;\ \tau^u = q.$$

Basis $\alpha = 0$ and f is a constructor or destructor. The simulation is performed in one step.

Step Case 1
$f = \Psi_i^b(g_0, g_1, g_2)$. We need a time T_1 to copy g_0 and one of the g_{i+1}s on the top of τ; and a time T_2 for the execution of g_1, and of one of the g_{i+1}s. We have $T_1 \leqslant 3\sum_{0\leqslant j\leqslant 2} //g_i$ (to copy g_0) $+2\sum_{0\leqslant j\leqslant 2} //g_i$ to replace f by the g_{i+1} to be simulated next) $\leqslant \sum_{j\neq j^*} 2/g_j/\ /g_{j^*}/$. Hence, by applying the induction hypothesis on c to the g's we obtain $T_1 + T_2 \leqslant \sum_{j\neq j^*} 2/g_j/\ /g_{j^*}/ + \sum_{0\leqslant j\leqslant 2} /g_i/^2 B_\alpha(n+c^2) \leqslant (\sum_{0\leqslant j\leqslant 2} /g_i/ + 1)^2 B_\alpha(n+c^2) = c^2 B_\alpha(n+c^2)$.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

Case 2
$f = \Upsilon(g, h)$. Similarly.

Case 3
$f = \Pi(g, h)$. We have $\alpha = \beta + 1$. Let the form of r be $a_{|r|} \dots a_1$. INT needs (i) a time T_1 to copy r into τ^u; and for r *preparatory steps* which: un-nest g and ($|r| - 1$ times) h; copy back, bit by bit, r in τ^z; and (ii) a time T_2 to simulate g and h. Since we now have $B_\alpha(n + c^2) \geqslant n + c^2 > |r|$, by arguments like under Case 1, we obtain $T_1 \leqslant 5(/g/ + /h/)B_\alpha(n + c^2)$.

We now evaluate T_2. By the induction on α, INT needs time $\leqslant m + /g/^2 B_\beta(n+c^2)$ to compute g, thus moving from the first to the second of the two following configurations:

$$\begin{array}{lllll} \tau = e\#f\#PI\#g & \tau^x = s_0\#s & \tau^y = t_0\#t & \tau^z = r_0\#r\#a_{|r|} & \tau^u = q\#r \\ \tau = e\#f\#PI & \tau^x = s_0\#g(s,t,a_1) & \tau^y = t_0\#t & \tau^z = r_0\#r\#a_{|r|} & \tau^u = q\#r. \end{array}$$

If $|r| > 1$ then INT puts $\tau := e\#f\#PI\#h$; $\tau^z := r_0\#r\#a_{|r|}a_{|r|-1}$, and computes h for $|r| - 1$ times, producing the configurations $\tau = e\#f\#PI$ (each time)

$$\begin{array}{llll} \tau^x = s_0\#h(f(s,t,a_{|r|}), t, a_{|r|}a_{|r|-1}) & \tau^y = t_0\#t & \tau^z = r_0\#r\#a_{|r|}a_{|r|-1} & \tau^u = q\#r; \\ \dots & & & \\ \tau^x = s_0\#h(f(s,t,a_{|r|}\dots a_{|r|-i}), t, a_{|r|}a_{|r|-i-1}) & \tau^y = t_0\#t & \tau^z = r_0\#r\#a_{|r|}\dots a_{|r|-i-1} & \tau^u = q\#r; \end{array}$$

By applying $|r| - 1$ times the induction hypotheses on α we obtain (since $c \geqslant lh(h) + lh(g) + 1$)

$$\begin{array}{rcl} T_2 & \leqslant & /g/^2 B_\beta(n + c^2) + (|r| - 1)/h/^2 B_\beta(n + c^2) \\ T_1 + T_2 & \leqslant & (5(/g/ + /h/) + \max(/g/^2, /h/^2))/r/B_\beta(n + c^2) \leqslant c^2 B_\alpha(n + c^2). \end{array}$$

Case 4
$f = \Delta(h, \lambda)$. INT computes $h(r)$, understands from the mark DELTA that the result is giving the function to be computed next, and accordingly, pushes it into τ. To compute $h(r)$ and $\{h(r)\}(s,t,r)$ INT needs, by the induction hypotheses on α and by Lemma 7 (twice), time $\leqslant /h/n^c + /\{h(r)\}/^2 B_{\lambda(|r|)}(n + c^2) \leqslant /h/n^c + /h/^2 n^{2c} B_{\lambda(|r|)}(n + c^2) \leqslant c^2 n^{2c} B_{\lambda(|r|)}(n + c^2) \leqslant c^2 B_{\lambda(|r|+c^2)}(n + c^2) = c^2 B_\lambda(n + c^2)$, where, to believe the last inequality, observe that we have the worst for $\lambda = \omega$, where $c^2 n^{2c} B_{\omega(|r|)}(n + c^2) \leqslant c^2 n^{2c}(n + c^2)^n \leqslant c^2(n + c^2)^{n+c^2} \leqslant c^2 B_{\omega(|r|+c^2)}(n + c^2)$. $\square$

Note
Assume that $f(x, y, z) \in \mathscr{T}_1$ is not simple. It may then happen that, in the computation of $f(s, t, r)$ at values $q_1, \dots, q_n$, the step function modifies y or z, instead of the previous value of f. Copying t or the current value of z into τ^x for n times may require a quadratic time. A long and tedious way to face this difficulty uses the fact that only $|f|(|q_{i+1}| - |q_i|)$ digits may have been added or killed in the *phase* of the computation between $z := q_i$ and $z := q_{i+1}$; and that the overall amount of such digits during the whole computation is $\leqslant |h||r|$. By using additional tapes $\tau^{u,h}, \tau^{u,t}, \tau^{u,c}$ $(u = y, z)$ one may store: in $\tau^{u,h}$ the *head* of u shared in common by the original value of u and the current value of f; in $\tau^{u,t}$ the original *tail* of u; and in $\tau^{u,c}$ the current tail of f. At the end of each phase, the value of f may be destroyed and $\tau^{u,t}$ may be copied in τ^h, thus recovering the original value of u.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

6 Simulation of TMs

Definition 21
Let M be a binary push-down TM with k tapes, and $m+1$ states:

1. The code for M is $\langle R_1, \ldots, R_m \rangle$, where R_i is its ith row, coded, in turn, by
$$\langle i, j, i_1, j_1, I_1, i_2, j_2, I_2, i_3 \rangle.$$
2. An *instantaneous description* (id) of M is a $\mathbf{T}$-word $s = X_1\mathbf{i}, \ldots, X_k$, where X_j is the current contents of tape j and i is the current state.

Lemma 22
For every TM M, a function $nxt^M(x)$ can be defined in $\mathscr{T}_0$, which, for all s coding an id of M, returns s if the state is 0, and the next id otherwise.

Proof
Let a binary push-down TM M be given. For every i, a test $st[i](x)$ can be defined, such that $st[i](s)$ is true iff i is the state of the id coded by s. Define

$$\begin{aligned} nxt(x) = {} & \textit{if } st[0](x) \textit{ then } x \textit{ else if } st[1](x) \textit{ and } top(j(1)) = 1 \textit{ then } E_{11} \textit{ else} \\ & \textit{if } st[1](x) \textit{ and } top(j(1)) \neq 1 \textit{ then } E_{12} \textit{ else} \ldots \textit{if } st[m](x) \\ & \textit{and } top(j(m)) \neq 1 \textit{ then } E_{m2}, \end{aligned}$$

where E_{ib} is a modifier up-dating the state and applying the appropriate *push*/*pop* to j_b. $\square$

To prove that the iteration for $B_\alpha(n)$ times of every function $f \in \mathscr{T}_0$ is in $\mathscr{T}_\alpha$, we need a version of the recursion theorem, allowing the iterator of f at level $\mathscr{T}_\lambda$ to call itself at level $\mathscr{T}_{\lambda(n)}$. Since the core of the recursion theorem is a self-referential substitution, and since the implementation of full SBST in our classes is cumbersome, we first prove a version for poly-time TM's of this theorem, and we then import it in $\mathscr{T}(\omega)$.

Lemma 23

1. (Uniform composition of EPTM's) For all $l, m, n \geqslant 0$ there exist the EPTM's C_{lmn} such that if X_1 and X_2 are codes of EPTM's, then $C_{lmn}(X_1, X_2)$ codes an EPTM satisfying
$$\begin{aligned} & p_{C_{lmn}(X_1,X_2)}(Y_1, \ldots, Y_{m+n+l}) \\ & =_{sc} p_{X_1}(Y_1, \ldots, Y_l, p_{X_2}(Y_{l+1}, \ldots, Y_{l+m}), Y_{l+m+1}, \ldots, Y_{l+m+n}). \end{aligned}$$
2. (The S_n^m theorem) For all m, n, there exist the EPTM's S_n^m such that such that if X codes an EPTM then for all B-words $Y_1, ..., Y_m$ we have
$$p_{S_n^m(X,Y_1,...,Y_m)}(Y_{m+1}, \ldots, Y_{m+n}) =_{sc} p_X(Y_1, \ldots, Y_{m+n});$$
$$p_{S_n^0(X)}(Y_1, \ldots, Y_n) =_{sc} p_X(X, Y_1, \ldots, Y_n),$$
and the main term in the bound for $p_{S_n^m(X,Y_1,...,Y_m)}$ depends on X, and not on the Y_i.
3. (The recursion theorem) For each EPTM $p(Z, Y)$ there is U such that $p_U(Y) =_{sc} p(U, Y)$.
4. There is a function $sim \in \mathscr{T}(\omega)$ such that for all EPTM p_Y, s and q
$$p_Y(s) =_{sc} q \quad \text{iff} \quad \{sim(Y)\}(s) = q.$$

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

Proof

1. Let M_{lmn} be a TM which, by input $X_i = \ulcorner(M_i, a_i, b_i)\urcorner$ $(i = 1, 2)$:
   (i) recovers the number N_i of tapes used by M_i;
   (ii) writes the code for a $(N_1 + N_2)$-tapes TM M whose rows consist of
      (A) instructions copying the first l and the last n tapes into (tapes) $N_2 + 1, \dots, N_2 + l + n$; and copying $Y_{l+1}, \dots, Y_{l+m}$ into $1, \dots, l$;
      (B) the rows of M_2;
      (C) instructions copying 1 in $l + 1$ and $N_2 + 1, \dots, N_2 + l + n$ in $1, \dots, l, l + 2, \dots, l + n + 1$;
      (D) the rows of M_1, with the state-numbers re-assigned in order to avoid confusion with the previous lines;
   (iii) writes the code for a bound (a, b) such that $\overline{a}$ and $\overline{b}$ consist respectively of $|\overline{a_1}| + |\overline{a_2}| + 2$ two's, and of $|\overline{b_1}| + |\overline{b_2}| + 2$ two's.

   Observe that (a, b) is appropriate for M, since we have $(a+n)^b \geqslant (a_1a_2+1+n)^{b_1b_2}$ (1 is added to the additive term in order to consider the copying back and forth mentioned under parts (A) and (C)); and since we have $\overline{m} < 2^{|\overline{m}|+1}$, and, therefore, $\overline{mn} < 2^{|\overline{m}|+|\overline{n}|+2}$.
   Parts (i)–(iii) can obviously be performed in polynomial time, and we may take as C_{lmn} the result of adding an appropriate bound to M_{lmn}.
2. (i) Let us write $\mathbf{Y}$ and $\mathbf{Z}$ for $Y_1, \dots, Y_m$ and $Y_{m+1}, \dots, Y_{m+n}$. Define $l := |\mathbf{Y}|$.
   (ii) Let $M_n^m[\mathbf{Y}]$ be the TM's (one for each $\mathbf{Y}, m, n$) which copy $\mathbf{Z}$ into tapes $m + 1, \dots, m + n$ and write $\mathbf{Y}$ on tapes $1, \dots, m$. Observe that $(l, 2)$ is appropriate for $M_n^m[\mathbf{Y}](\mathbf{Z})$, and define $WR_n^m[\mathbf{Y}] := (M_n^m[\mathbf{Y}], l, 2)$.
   (iii) We now take uniformly $\mathbf{Y}$ into $\ulcorner WR_n^m[\mathbf{Y}]\urcorner$. To this purpose, observe that a TM U_n^m can be defined which, by input $\mathbf{Y}$ yields $\ulcorner WR_n^m[\mathbf{Y}]\urcorner$; and that, since the length of $M_n^m[\mathbf{Y}]$ is linear in l, there is a constant c, depending only on m and n, such that $(c, 2)$ is appropriate for U_n^m. Define $UWR_n^m := (U_n^m, c, 2)$.
   (iv) Define an EPTM by
$$S_n^m := p_{C_{1m0}(\ulcorner C_{0m0}\urcorner, \ulcorner UWR_n^m\urcorner)}.$$
   By part 1, we have
$$S_n^m(X, \mathbf{Y}) = C_{0m0}(X, UWR_n^m(\mathbf{Y})). \tag{5}$$
   By (5) we have
$$p_{S_n^m(X,\mathbf{Y})}(\mathbf{Z}) = p_{C_{0m0}(X,UWR_n^m(\mathbf{Y}))}(\mathbf{Z})$$
$$= p_X(p_{UWR_n^m(\mathbf{Y})}(\mathbf{Z})) = p_X(WR_n^m[\mathbf{Y}](\mathbf{Z})) = p_X(\mathbf{Y}, \mathbf{Z}).$$
   (v) For $m = 0$, use a duplicating TM and composition to define $S_n^0(X) := S_n^1(X, X)$.
   (vi) To see that the main term of $p_{S_{\dots}}$ is independent from $\mathbf{Y}$, observe that: by parts (ii) and (iii), l is only contributing to certain additive terms; and that, by arguments at the end of the proof of part 1, in compositions of the form $C_{\dots}$ the additive terms don't contribute to the main terms.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

3. Given p_X, define $W := C_{011}(X, \ulcorner S_1^0 \urcorner)$. We have $p_W(Y, Z) =_{sc} p_X(S_1^0(Y), Z)$. Define further $U := S_1^0(W)$. We obtain $p_U(Z) =_{sc} p_{S_1^0(W)}(Z) =_{sc} p_W(W, Z) =_{sc} p_X(S_1^0(W), Z) =_{sc} p_X(U, Z)$.
4. *sim* by input $Z = \langle M, a, \circ, b \rangle$, returns the required result by taking $\ulcorner M \urcorner$ into $\ulcorner nxt^M \urcorner$, and by producing the code for a sequence of the form

$$\Pi(I, \Phi_{zy}(\Pi(I, \ldots, \Phi_{zy}(\Pi(I, nxt^M))\ldots)))$$

($|\overline{a}| + |\overline{b}|$ times). Both these tasks are straightforward and can be performed within a time linear in $|Z|$. Notice that the output of *sim* is the code for a function $\in \mathscr{T}(\omega)$.

$\square$

Example 4
In the proof of next lemma we need a uniform way to move from the codes for an EPTM p and a limit ordinal λ to the code for an EPTM q such that $q(U) = p(\ulcorner \lambda(|U|) \urcorner)$. To this purpose, let fs be the EPTM which is obtained by adding an adequate bound to the EPTM FS of Lemma 9. Define an EPTM by $G(Z, Y) := S_1^1(C_{020}(Z, \ulcorner fs \urcorner), Y)$. For all p_Z and λ, we have

$$p_{G(Z, \ulcorner \lambda \urcorner)}(U) = p_{C_{020}(Z, \ulcorner fs \urcorner)}(\ulcorner \lambda \urcorner, U) = p_Z(fs(\ulcorner \lambda \urcorner, U)) = p_Z(\ulcorner \lambda(|U|) \urcorner).$$

By the last statement of part 2 of Lemma 23, if p_Z is in DTIMEF(n^c) then $p_{G(Z,Y)}$ is in a class DTIMEF(n^d), where d only depends upon c.

Lemma 24
For all $h(x) \in \mathscr{T}_0$ and for all ordinal $\alpha \leqslant \phi_\omega(0)$ there exists $g_\alpha \in \mathscr{T}_\alpha$ such that $g_\alpha(s, t) = h^{B_\alpha(|t|)}(s)$.

Construction of g_α
Given h define a TM satisfying

$$M(Z, Y) = \begin{cases} \ulcorner h \urcorner & \text{if } Y \text{ codes the ordinal } 0 \\ \ulcorner \Pi^* \urcorner M(Z, \ulcorner \alpha \urcorner) & \text{if } Y \text{ codes a successor } \alpha + 1 \\ \ulcorner \Delta \urcorner sim(G(Z, Y))Y & \text{if } Y \text{ codes a limit ordinal} \\ 1 & \text{otherwise,} \end{cases}$$

where (i) the different cases are decided by the poly-time TM's SC and LM of Lemma 9; (ii) expressions $\ulcorner \Pi^* \urcorner M(Z, \ulcorner \alpha \urcorner)$ and $\ulcorner \Delta \urcorner sim(G(Z, Y))Y$ should be understood as mere concatenations of the indicated codes with the values of $M(\ldots)$ and, respectively, of $sim(\ldots)$; and (iii) the values of the EPTM G of the last example and of the function *sim* are obtained by including in the finite control of M a copy of G and of the EPTM which simulates *sim*.

Time for M is polynomial, since this TM is defined by composition of EPTM's and by a loop which adds a string of a constant length at each repetition.

Let p be the EPTM which is obtained by adding an appropriate bound (a, b) to M. By Lemma 23, there exists a fixed-point U such that $p_U(Y) = p(U, Y)$, with $p_U \in DTIMEF(n^d)$ for some d. Define

$$f^* = \{sim(U)\}; \quad f_\alpha = \{f^*(\ulcorner \alpha \urcorner)\}; \quad g_\alpha(x, y) = f_\alpha(x, y, y). \tag{6}$$

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

Proof of the lemma

We show by transfinite induction on α that $f_\alpha \in \mathscr{T}_\alpha$, and

$$f_{\beta+1}(s,t,r) = h^{|r|B_\beta(|t|)}(s); \qquad f_\lambda(s,t,t) = h^{B_{\lambda(|t|)}(|t|)}(s); \tag{7}$$

hence $f_{\beta+1}(s,t,t) = h^{B_{\beta+1}(|t|)}(s)$.

Basis $\alpha = 0$. We have $f_0 = \{f^*(\ulcorner 0 \urcorner)\}$ (by (6))$= \{\ulcorner h \urcorner\}$ (by first line in the definition of $M) = h$.

Step Case 1

$\alpha = \beta + 1$. We have

$$\begin{array}{lll} f_\alpha & = \{f^*(\ulcorner\alpha\urcorner)\} & \text{by (6)} \\ & = \{\{sim(U)\}(\ulcorner\alpha\urcorner)\} & \text{again by (6)} \\ & = \{p_U(\ulcorner\alpha\urcorner)\} & \text{by part 4 of Lemma 23} \\ & = \{p(U, \ulcorner\beta+1\urcorner)\} & \text{by part 3 of the same lemma, and definition of } U \\ & = \Pi(\Phi_{zy}(f_\beta), \Phi_{zy}(f_\beta)) & \text{by definition of } p \text{ and } \Pi^*. \end{array}$$

$f_\alpha \in \mathscr{T}_\alpha$ follows by the induction hypothesis and by closure of $\mathscr{T}_\beta$ under renaming.

We now prove (7) by induction on $|r|$.

$$\begin{array}{llll} \text{Basis.} & f_\alpha(s,t,a) & = \Pi(\Phi_{zy}(f_\beta), \Phi_{zy}(f_\beta))(s,t,a) & \text{last equality above} \\ & & = \Phi_{zy}(f_\beta)(s,t,a) & \text{by definition of } \Pi \\ & & = f_\beta(s,t,t) = h^{|a|B_\beta(|t|)}(s) & \text{ind.hyp. on } \alpha. \\ \text{Step.} & f_\alpha(s,t,ra) & = \Pi(\Phi_{zy}(f_\beta), \Phi_{zy}(f_\beta))(s,t,ra) & \text{see line 1} \\ & & = f_\beta(f_\alpha(s,t,r),t,t) & \text{definition of } \Pi \text{ and } \Phi_{zy} \\ & & = h^{B_\beta(|t|)}(f_\alpha(s,t,r)) & \text{ind. hyp. on } \alpha \\ & & = h^{B_\beta(|t|)}(h^{|r|B_\beta(|t|)}(s)) & \text{ind. hyp. on } r \\ & & = h^{(|r|+1)B_\beta(|t|)}(s) & \text{computations.} \end{array}$$

Case 2

α is the limit ordinal λ. We have $f_\lambda = \{f^*(\ulcorner\lambda\urcorner)\} = \Delta(\{sim(G(U, \ulcorner\lambda\urcorner))\}, \lambda)$. Notice that $\{sim(G(U, \ulcorner\lambda\urcorner))\} \in \mathscr{T}(\omega)$, by definition of G and *sim*. Hence, by definition of the diagonalization scheme, the result follows by the induction after checking that, for all r, we have

$$\{\{sim(G(U, \ulcorner\lambda\urcorner))\}(r)\} = f_{\lambda(|r|)}.$$

Indeed, we have

$$\begin{array}{lll} \{\{sim(G(U, \ulcorner\lambda\urcorner))\}(r)\} & = \{p_{G(U,\ulcorner\lambda\urcorner)}(r)\} & \text{by part 4 of Lemma 23} \\ & = \{p_U(\ulcorner\lambda(|r|)\urcorner)\} & \text{by example 4} \\ & = \{\{sim(U)\}(\ulcorner\lambda(|r|)\urcorner)\} & \text{by part 4 of Lemma 23} \\ & = \{f^*(\ulcorner\lambda(|r|\urcorner))\} & \text{by (6)} \\ & = f_{\lambda(|r|)} & \text{by (6) again.} \end{array}$$

To see that the degree of all g_α is finite, recall that runtime for p_U is bounded above by n^d for some d. By arguments repeatedly used in this proof, for all λ we have that the form of f_λ is $\Delta(sim(G(U, \ulcorner\lambda\urcorner)), \lambda)$. By the last remark in Example 4, we have $p_{G(U,\ulcorner\lambda\urcorner)} \in$DTIMEF$(n^b)$ for some b depending only on d. Thus, by Lemma 20, the degree of every f_λ is b.

https://doi.org/10.1017/S0956796800003853 Published online by Cambridge University Press

Lemma 25
$\text{DTIMEF}^*(B_\alpha(n)) \subseteq \mathscr{T}_\alpha$.

Proof
Let M compute function $F(s)$ in time $B_\alpha(|s|+c)$. By the last lemma there is a function $g^M(x,y)$ such that $g^M(s,t)$ returns the $B_\alpha(|t|)$-th iterate of $nxt^M(s)$. By means of one Φ_{xy} and of c functions Γ_1^1, we can define a function returning the $B_\alpha(|s|+c)$-th iterate of $nxt^M(s)$. $\square$

Acknowledgements

The comments of one of the referees were so constructive and adherent to our ideas that we regard her/him as a moral co-author.

References

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

Bellantoni, S. J. (1992) Predicative recursion and computational complexity. *PhD thesis*, Toronto.

Bellantoni, S. J. (1995) Predicative recursion and the polytime hierarchy. In: P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pp. 15–29. (Birkhäuser.

Bellantoni, S. J. and Niggl, L.-H. (1997) Ranking primitive recursion: the low Grzegorczyck classes revisited (submitted).

Caporaso, S. (1996) Safe Turing machines, Grzegorczyk classes and Polytime, *Int. J. Found. Comp. Sci.*, **7**(3), 241–252.

Caporaso, S., Zito, M. and Galesi, N. (November 2000) A predicative and decidable characterization of the polynomial classes of languages. *Theoretical Comput. Sci.* (to appear).

Caporaso, S., Zito, M., Galesi, N. and Covino, E. (1997) Syntactic characterization in Lisp of the polynomial complexity classes and hierarchies. In: G. Bongiovanni, D. P. Bovet and G. Di Battista, editors, *Algorithms and Complexity: Lecture Notes in Computer Science 1203*, pp. 61–73. Springer-Verlag.

Caporaso, S., Pani, G. and Covino, E. (1999) Predicative recursion, constructive diagonalization and the elementary functions. *Implicit Computational Complexity 99 (a workshop affiliated with LICS99*, Trento, Italy.

Cichon, E. A. and Wainer, S. S. (1983) The slow-growing and the Grzegorczyk hierarchies. *J. Symbolic Logic*, **48**(2), 399–408.

Cobham, A. (1965) The intrinsic computational difficulty of functions. In: Y. Bar Hillel, editor, *Proc. Int. Cong. Logic, Methodology and Philosophy Sci*, pp. 24–30. North-Holland.

Girard, J.-Y. (1981) Π_2^1 logic. *Ann. Math. Logic*, **21**, 75–219.

Girard, J.-Y. (1987) *Proof Theory and Logical Complexity*. Bibliopolis, Naples.

Leivant, D. (1991) A foundational delineation of computational feasibility. *Proc. 6th Annual IEEE Symposium on Logic in Computer Science*, pp. 2–18. IEEE Press.

Leivant, D. (1993) Stratified functional programs and computational complexity. *Conference Record 20th Annual ACM Symposium on Principle of Programming Languages*. New York.

Leivant, D. (1994) Ramified recurrence and computational complexity I: word recurrence and polytime. In: P. Clote and J. Remmel, editors, *Feasible mathematics II*, pp. 320–343. Birkhäuser.

Leivant, D. and Marion, J.-Y. (1995) Ramified recurrence and computational complexity II: substitution and polyspace. In: J. Tiuryn and L. Pacholski, editors, *Computer Science Logic: Lecture Notes in Computer Science 933*, pp. 486–500. Springer-Verlag.

Niggl, K.-H. (1999) The μ-measure as a tool for classifying computational complexity. (Logic Colloquium 97, Leeds.) *Bull. Symb. Log.*, **4**(1), 100–101 (Abstract).

Rose, H. E. (1984) *Subrecursion: Functions and Hierarchies*. (Oxford University Press.

Simmons, H. (1988) The realm of primitive recursion. *Arch. Math. Logic*, **27**, 177–188.

Wainer, S. S. (1989) Slow-growing versus fast-growing. *J. Symbolic Logic*, **54**(2), 608–614.

Weiermann, A. (1999) Γ_0 may be recursively inaccessible. Preprint, Münster.