A predicative classification of the elementary functions

Salvatore Caporaso, Giovanni Pani, Emanuele Covino
Dipartimento di Informatica dell'Università di Bari
(caporaso|pani|covino)@ di.uniba.it

Abstract. This is a part of our investigations on the way to extend to larger classes of functions the *Predicative Recurrence/ Implicit Complexity Program* promoted by Leivant and a number of other scholars. It is known that (under the so-called *standard assignment* of fundamental sequences to the ordinals $\leq \epsilon_0$) a fine hierarchy can be defined on the elementary functions by taking as $\mathcal{G}_\alpha$ the closure under limited PR and limited substitution of $G_\alpha + \bigcup_{\beta<\alpha}$ where $\mathcal{G}_\alpha$ is the α*-th* element of the Wainer-Schwichtenberg *slow hierarchy*. However, limited operators are un-decidable, and PR is impredicative. A hierarchy $\mathcal{T}_\alpha$ growing in a way similar to $\mathcal{G}_\alpha$ is defined here by means of unlimited operators (simultaneous predicative recurrence and constructive diagonalization). Thus the robustness result $T = \mathcal{E}_{\epsilon_0} = \mathcal{E}^*$ (Dialectica = Extended Grzegorczyk Hierarchy = provably total in PA) has the predicative analogue $T^* = \mathcal{T}_{\epsilon_0} = \mathcal{E}$ (Beckmann & Weiermann fragment of T = the slow hierarchy revisited with un-limited and predicative operators = elementary functions).

1 Introduction

Context. After Leivant [9, 1991] and Bellantoni & Cook [2, 1992], other forms of *predicative recurrence* have been proposed (see for example Leivant and Marion [12, 1995], and our [5, 6]); under appropriate choices of initial functions and data-type algebras, they yield resource-free characterizations of several complexity classes. Functionals and hierarchies based on ordinals may be used to extend to larger classes this *implicit complexity* approach.

Functionals. Leivant's generalization [9] to finite types of predicative recurrence yields the class $\mathcal{E}$ of all elementary functions, while it is well-known that Gödel's system T of (not predicatively restricted) PR functionals yields the class $\mathcal{E}^*$ of all functions provably total in PA. Recently, $\mathcal{E}$ has been characterized by means of a predicative fragment T^* of T by Beckmann & Weiermann [1].

Hierarchy generating functions. Let λ_n be the *fundamental sequence* for the limit ordinal $\lambda \leq \epsilon_0$ defined by the *standard assignment* S (cf. Def. 10; see Weiermann [17] for surprising consequences of apparently minor and innocent changes to S). The extension to the transfinite of the Ackermann-Grzegorczyk functions is obtained at successors by one definition by PR in the usual way; and by using, at limits λ, the *diagonalization* scheme $E_\lambda(n) := E_{\lambda_n}(n)$. The Wainer-Schwichtenberg *slow hierarchy* [7] G_α is obtained by defining $G_{\alpha+1}(n) := G_\alpha(n)+1$, and, at limits, by the same form of diagonalization.

Hierarchies of classes of functions. The *extended Grzegorczyk hierarchy* is given by

$$\mathcal{E}_\alpha := \text{ closure under substitution and } \textit{limited primitive recursion } (\mathrm{PR}_\leq) \text{ of } E_\alpha + \bigcup_{\beta<\alpha} \mathcal{E}_\alpha.$$

Partially predicative harmonizations of the complexity classes with the finite hierarchy $\mathcal{E}_n$ at the elementary level 3 and up have been obtained by Leivant [10], Caporaso [4], Niggl [13], Bellantoni&Niggl [3], by closure under susbtitution, predicative recurrence, and under an additional *impredicative ingredient*. Other definition schemes, like double recursion or diagonalization, would be needed to reach larger classes. One may then ask whether these schemes are independent.

Another hierarchy of classes of functions $\mathcal{G}_\alpha$ might be defined by: (a) replacing E_α by G_α; (b) replacing full susbstitution by a form of *limited substitution*; and (c) by keeping the $\mathrm{PR}_\leq$ scheme. Since E_3 grows like G_{ϵ_0} (under assignment S) one obtains in this way a fine hierarchy of sub-elementary classes. However, discarding limited operators is fundamental in the implicit complexity program, and one may raise the

Problem 1 It is known that $T = \mathcal{E}_{\epsilon_0} = \mathcal{E}^*$. Define, by means of predicative recurrence and other unlimited operators, a hierarchy $\mathcal{T}_\alpha$ similar to $\mathcal{G}_\alpha$ so that, under assignment S

$$T^* = \mathcal{T}_{\epsilon_0} = \mathcal{E}.$$

1.2 Statement of the result. Let us say that f is defined by *constructive diagonalization in the enumerator* e if we have $f(n) = \{e(n)\}(n)$, where $\{n\}$ is Kleene's notation for the function coded by n. Assume defined a scheme ISBST of *inessential substitution*, such that classes like DTIMEF(n^k) are closed under this scheme. Under assignment S, define a hierarchy $\mathcal{T}_\alpha$ by

1. $\mathcal{T}_1$ is a characterization of DTIMEF(n);
2. $\mathcal{T}_{\alpha+1}$ is the closure under ISBST of the class of all functions obtained by at most one application of (our) predicative recurrence scheme to functions in $\mathcal{T}_\alpha$;
3. $\mathcal{T}_\lambda$ is the closure under ISBST of the class of all functions obtained by at most one application of constructive diagonalization in a given enumerator $e \in \mathcal{T}_{\lambda_1}$ such that $\{e(n)\} \in \mathcal{T}_{\lambda_n}$.

We then have (writing $clps(\alpha, m)$ for the result of replacing ω by m in Cantor normal form for α)

$$(1) \quad \text{DTIMEF}(n^{clps(\alpha,n)}) \subseteq \mathcal{T}_\alpha \subseteq \text{DTIMEF}((n+4)^{clps(\alpha,n+4)}).$$

Moreover, $\mathcal{T}_k = \text{DTIMEF}(n^k)$ for every finite k. For example (${}^{c+1}n$ stands for $n^{{}^c n}$)

$$\begin{array}{ll} \text{DTIMEF}(n^n) \subseteq \mathcal{T}_\omega \subseteq \text{DTIMEF}((n+4)^{n+4}); & \bigcup_{\beta<\omega} \mathcal{T}_\beta = \text{PTIMEF}; \\ \text{DTIMEF}({}^k n) \subseteq \bigcup_{\beta<\omega_k} \mathcal{T}_\beta \subseteq \text{DTIMEF}({}^k(n+4)); & \bigcup_{\beta<\epsilon_0} \mathcal{T}_\beta = \mathcal{E}. \end{array}$$

By adding some cumbrous clauses to the syntax of our functions the *spread* 4 in (1) can be reduced to 1 (see Note 18).

1.3 Motivation. Using only these three schemes was not only a matter of intellectual parsimony. One may summarize the predicative recurrence program by saying that the amount of computation for a given function should be known *a-priori* and should not depend on intermediate or previous values. PR is then impredicative, since it allows assigning crucial variables of the step function with the previous values of the function being recursed upon. In principle, it is unfair to reject PR as impredicative, and to restore it afterwards, with the *post-hoc* reason that we need it to produce larger functions. In practice, we must take into further account the obvious fact that it grows too fast, and *eats* important differences.

If PR is the avid member of the family, substitution is the coarse one. A logician would not accept that ω^{ω^ω} is the result of the substitution of ω^ω for α in ω^α, if objects like, say, $\omega^{\omega^3+\omega^2}$ were not introduced first. He would probably argue that insight of the well-orderings should come before transfinite computations. Now, let $\mathcal{E}^-$ be the closure of poly-time functions, under any kind of predicative recurrence, ISBST, and a single PR. It can be proved (cf. Lemma 22) that every $f \in \mathcal{E}^-$ is eventually dominated by n^{n^n}. Using substitution to fill such a large gap between $\mathcal{E}^-$ and $\mathcal{E}$ might sound as a laceration of the delicate texture of our understanding the recursion phenomenon.

A sensible sequel of this paper might show how the present result and its extension to larger classes like $\mathcal{E}^*$ depend on a more or less *natural* choice of fundamental sequences, and on the time complexity of the related systems of notations. An essential contribution to this task may come from results by Waiermann like the one mentioned above [17].

2 Diagonalization and other definition schemes

$a, b, a_1, \ldots$ are *digits* of the ternary alphabet $\mathbf{T} = \{0, 1, 2\}$ and $p, \ldots, s, \ldots, w, \ldots$ are words over $\mathbf{T}$; ϵ is the empty word. $u, w, u_1, \ldots$ are variables defined on $\mathbf{T}$. We denote by x, y, z variables previously used as, respectively, *auxiliary variable, parameter, recursion variable*. This convention is tantamount to the use of a semi-colon $(z; x, y)$ to separate safe and unsafe arguments by Simmons [16] and Bellantoni & Cook [3]). When we write $f(u_1, \ldots, u_n)$ we always assume that some of the indicated variables may be absent.

Definition 2 Given a constant s in which n zeroes occur, $(s)_i$ $(1 \leq i \leq n+1)$ is the word (possibly empty) in the alphabet $\{1, 2\}$ which occurs between the $(i-1)$-th and the i-th zeroes of s; if $i > n+1$ then $(s)_i$ does not exist. Given $i \geq 0$ and $a = 1, 2$,

1. the *constructor* $\mathrm{C}_i^a(s)$ adds a digit a at the right of $(s)_i$;
2. the *destructor* $\mathrm{D}_i(s)$ erases the right-most digit of $(s)_i$ if any;

   all these constructors and destructors leave s un-changed if $(s)_i$ doesn't exist;
3. the construct $\mathrm{CASE}_i[f, g, h](s)$ returns $f(s)$ (respectively $g(s)$) if the rightmost digit of $(s)_i$ is 1 (respectively 2); it returns $h(s)$ if $(s)_i$ does not exist or is ϵ.

For example $\mathrm{D}_2(010) = \mathrm{D}_2(00) = 00$; $\mathrm{D}_1(1) = \epsilon$; $\mathrm{C}_1^1(0) = 01$.

Definition 3 $f = \mathrm{SR}(g, h)$ is defined by *(this paper's) safe recursion* in the *basis function* $g(x, y)$ and in the *step function* $h(x, y, z)$ if for all s, t, r we have

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

An *iteration* is a SR in which parameter and recursion variable are both absent in the step function.

Comments on last two definitions are in Sect. 5.

Definition 4 $f = \mathrm{CDIAG}(e)$ is defined by *(constructive) diagonalization* in the *enumerator* e if for all s, t, r, we have

$$f(s, t, r) = \{e(r)\}(s, t, r).$$

Definition 5

1. $f = \mathrm{ASG}(s, u, g)$ is the result of the *assignment* of s to variable u in g;
2. $f = \mathrm{IDT}_x(g)$ is the result of the *identification* of x as y in g (thus, $f(y, z) = g(y, y, z)$); similarly, $f = \mathrm{IDT}_z(g)$ is the result of the *identification* of z as y in g (thus, $f(x, y) = g(x, y, y)$).
3. $f = \mathrm{SBST}(u, h, g)$ is defined by *substitution* in g and h if it is obtained by substitution of g for u in h.

An essential point is that identification of z as x is not allowed and, therefore, the step function cannot assign the recursion variable with the previous value of the function being defined by safe recursion. (According to Bellantoni&Cook's terminology, in this way z keeps *safe*).

Definition 6 1. Class $\mathcal{T}_0$ is the closure under ASG, IDT_x, IDT_z and SBST of C_i^a, D_i, CASE_i;

2. Class $\mathcal{T}_1$ is the closure under ASG, IDT_x, IDT_z and SBST of functions obtained by at most one iteration from functions in $\mathcal{T}_0$.

In other words, this class is the indicated closure of all functions $f(x,z) = h^{|z|}(x)$ for some $h \in \mathcal{T}_0$.

Definition 7 The *number of components* $\#(f)$ of $f \in \mathcal{T}_1$ is $\max\{i | \text{D}_i \text{ or } \text{C}_i^a \text{ or } \text{CASE}_i \text{ occurs in } f\}$.

Example 8 Define (by iteration of C_0^1) function $g_1 \in \mathcal{T}_1$ such that $g_1(x,y) = x1^{|y|}$. Define further

$$\begin{array}{ll} \left\{ \begin{array}{lcl} f_{n+1}(s,t,\epsilon) & = & s \\ f_{n+1}(s,t,ra) & = & g_n(f_{n+1}(s,t,r),t) \end{array} \right. & (\text{SR}) \\ g_{n+1}(x,y) = f_{n+1}(x,y,y) & (\text{IDT}_z) \end{array}$$

By induction one can verify that $|f_{n+1}(s,ta,rb)| = |s| + |t|^n|r|$ and, therefore, $|g_n(\epsilon,t)| \leq |t|^n$. Assume defined a function $e \in \mathcal{T}_1$ such that $e(r) = \lceil g_{|r|} \rceil$. By setting $f_\omega :=\text{CDIAG}(e)$, we then have $|f_\omega(\epsilon,t,t)| = g_{|t|}(\epsilon,t) = |t|^{|t|}$.

Lemma 9 $\text{DTIMEF}(n) = \mathcal{T}_1$.

Proof. 1. To show that every function in $\mathcal{T}_1$ can be computed in linear time by a TM with input and output on its first tape, let $g \in \mathcal{T}_0$ and $f(s,t) = g^{|t|}(s)$ be given. A TM M_f with $m := \#(f)+1$ tapes can be defined which, by input s on tape 1: (a) copies $(s)_j$ $(j \leq m)$ on tape $j+1$; (b) computes g in constant time; (c) after $|t|$ repetitions collates back the contents of tapes $2,\ldots,m+1$ into tape 1.
2. For every m-tapes TM define in $\mathcal{T}_0$ a function nxt_M which uses two components for the part at the right (read in reverse order) and the part at the left of the observed symbol of each tape, and the last one of its $(2m+1)$ components for the internal state. The behaviour of M by input s for $|t|$ steps can then be simulated by a function $linsim_M(x,z) \in \mathcal{T}_1$, defined by iteration of nxt_M. Let M by input s on its first tape stop operating within $c|s|$ steps. Since $\mathcal{T}_1$ is closed under substitution, we may define $sim_M(s) := linsim_M(s, times_c(s))$, where $times_c(s) = 1^{c|s|}$ is obtained from function g_1 of Ex. 8 by assigning ϵ to x and by some SBST's.

3 The hierarchy

Definition 10 We adopt the following assignment of fundamental sequences to all $\lambda < \epsilon_0$

$$\lambda_n = \left\{ \begin{array}{ll} n & \text{if } n \leq 1 \text{ and } \lambda < \omega^2 \\ \omega^{\mu_n} & \text{if Cantor normal form for } \lambda \text{ is } \omega^\mu \\ \omega^\alpha n & \text{if Cantor normal form for } \lambda \text{ is } \omega^{\alpha+1} \\ \mu + (\omega^\alpha)_n & \text{if Cantor normal form for } \lambda \text{ is } \mu + \omega^\alpha. \end{array} \right.$$

Comment. Assignment S of last definition differs from the traditional standard assignment S^* (see [14, p. 78]) at line 1 (adopted to simplify the proof of Lemma 17 — cf. Case 2 of the induction); and at line 3, where we multiply by n instead of $n+1$ in order to cope with the fact that, as we shall see, n nested SR's grow like $|t|^n$, not as $|t|^{n+1}$. When defined with respect to S the slow hierarchy G_α (cf. §1.1) is obviously dominated by the one defined with respect to S^*. On the other side the former hierarchy is not collapsing since the Bachmann property ($\lambda_n < \lambda_{n+1}$ for all λ and n) keeps holding for S (see [14, Th. 3.6]).

Definition 11 $f = \text{ISBST}(X, h, g)$ is defined by *inessential substitution* in g and h if it is defined by substitution in g and h and $h \in \mathcal{T}_1$.

We may now define the following hierarchy of classes of functions.

Definition 12 (a) $\mathcal{T}_{\alpha+1}$ ($\alpha > 0$) is the closure under ASG, IDT_x, IDT_z, ISBST of the class of all functions obtained by at most one application of SR to $\mathcal{T}_\alpha$;
(b) $\mathcal{T}_\lambda$ is the closure under ASG, IDT_x, IDT_z, ISBST of the functions of the form $\text{CDIAG}(e)$, for $e \in \mathcal{T}_{\lambda_1}$, and such that $\{e(r)\} \in \mathcal{T}_{\lambda_{|r|}}$.

Notation 13 $B_\alpha(n) := \max(2, n)^{clps(\alpha, n)}$

By induction on α one sees that for all $n \geq 2$ we have $B_\alpha(n) = n^{G_\alpha(n)}$.

Theorem 14 1. For all $\omega \leq \alpha < \epsilon_0$ we have $\text{DTIMEF}(B_\alpha(n)) \subseteq \mathcal{T}_\alpha \subseteq \text{DTIMEF}(B_\alpha(n+4))$.
2. For all finite k, $\text{DTIME}(n^k) = \mathcal{T}_k$.

Proof. **1.** By proof of Lemma 9, there is a function $sim_M(s,t) \in \mathcal{T}_1$ returning the instantaneous description of the TM M after $|t|$ steps. By Lemma 21, for all $\alpha < \epsilon_0$ we can define in $\mathcal{T}_\alpha$ a function g_α which *computes in unary* $B_\alpha(|t|)$. The first inclusion then follows by ISBST of $g_\alpha(s)$ for t in sim.

The second inclusion and part **2.** follow by Lemma 17, in which an interpreter is defined, instead of mere simulation, in order to handle diagonalization.

4 Proofs

4.1 Codes

All expressions introduced throughout this paper may be thought of as transcriptions of a Polish prefix language over a *united alphabet* $\mathbf{U} = \{0, 1, 2, \text{ISBST}, \text{SR}, \text{CDIAG}, \text{ITRT}, \ldots\}$. Codes are built-up by juxtaposition from the codes for the letters of $\mathbf{U}$, unique parsing being ensured by the *arity* associated tacitly with each such letter.

Definition 15 The code $\lceil L \rceil$ for the i-th letter L of $\mathbf{U}$ is $2^{i+1}1$. Let us write $\langle E_1, \ldots, E_n \rangle$ for $\lceil E_1 \rceil \ldots \lceil E_n \rceil$. If the arity of $L \in \mathbf{U}$ is n then $\langle L, E_1, \ldots, E_n \rangle$ codes the expression $LE_1 \ldots E_n$.

Example 16 If $f(x,z) \in \mathcal{T}_1$ is the $|z|$-th iterate of function $e \in \mathcal{T}_0$,then its code is $\langle \text{ITRT}, e \rangle$. The code for functions g_{n+1} of Ex.8 is $\langle \text{IDT}_z, \langle \text{SR}, x, g_n \rangle \rangle$, where $\lceil g_1 \rceil = \langle \text{IDT}_z, \langle \text{ITRT}, \text{C}_0^1 \rangle \rangle$.

4.2 Simulation by TM's

Lemma 17 $\mathcal{T}_\alpha \subseteq \text{DTIMEF}(B_\alpha(n+4_\alpha))$, where $4_\alpha = 0$ if $\alpha < \omega$ and $4_\alpha = 4$ otherwise.

Proof. We first associate each integer d with an interpreter INT_d. By input $\lceil f \rceil, s, t, r$ it returns $f(s,t,r)$ provided that the following *d-condition* holds: (a) $|s|+|t|+|r| \leq d$ or (b) no CDIAG occurs in f and $\#(g) \leq d$, for each $g \in \mathcal{T}_1$ used in the construction of f.

When the d-condition holds, since by Def. 6 the number of zeroes doesn't increase during the computation of f, the number of tapes needed to store the parts of the arguments which can be modified by f in a number of tapes which depends only on d. In the final part of this proof, we will reduce the family INT_d to a single interpreter using only two tapes.

Definition of the interpreter We have to avoid the waste of time resulting from moving back and forth the value of the function being recursed upon from the storage which are reserved to the data to that containing the current results. To this purpose, the interpreter INT_d (see Fig. 1) uses the following stacks:
(a) T^x, T^y, T^z, to store the values of x, y, z during the computation; T^x consists of d tapes, one for each of the modifiable parts of the value assigned to x;
(b) T^u, to store the value of the principal variable of the current enumeration or recursion;
(c) T_f, to store (the codes for) some sub-functions of f;
the initial contents of T^x, T^y, T^z, T_f are, respectively, the input values s, t, r, and $\lceil f \rceil$; T^u is initially empty. At the end of the computation INT collates the components of the result into the first of tapes T^x.

INT_d repeats, until T_f is not empty, the following cycle
- it pops a function k from the top of T_f, 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;
- if the form of j is ITRT(g) with $g \in \mathcal{T}_0$, it calls an interpreter $ITRT$ for $\mathcal{T}_1$ which simulates g on T^x for $|t|$ times, where t is the top record of T^y;
- in all other cases, it pushes into T_f an information of the form $j\, MRK\, k$, where MRK is a mark informing about the outermost scheme used to define j.

Time complexity We first show that for all f, s, t, r respecting the d-condition we have

$$f \in \mathcal{T}_\alpha \text{ implies } INT_d(\lceil f \rceil, s, t, r) = f(s,t,r) \quad \text{within time } \ |s| + |\lceil f \rceil| B_\alpha(|t|+|r|+1_\alpha),$$

where $1_\alpha = 0$ if $\alpha < \omega$ and $1_\alpha = 1$ otherwise. The result follows, since every function $f \in \mathcal{T}_\alpha$ is then computed in DTIMEF$(B_\alpha(n+1_\alpha))$ by the sequence composition of the constant-time TM writing the code for f with INT_d.

Define $m := |s|,\ n := |t|+|r|;\ X := \lceil f \rceil;\ c := |X|$. We show that, for all $f \in \mathcal{T}_\alpha$, INT_d moves within $m + cB_\alpha(n+1_\alpha)$ steps from an istantaneous description of the form

$$T_f = ZX;\ T^x = s_0 s;\ T^y = t_0 t;\ T^z = r_0 r;\ T^u = q,$$

to a new istantaneous description of the form

$$T_f = Z;\ T^x = s_0(\{X\}(s,t,r));\ T^y = t_0 t;\ T^z = r_0 r;\ T^u = q.$$

Induction on α and on the construction of f. Basis. $\alpha = 1$. We have $1_\alpha = 0$. The complexity of $ITRT$ is obviously $< m + cn$.
Step. Case 1. $f = \text{SR}(g, h)$. We have $\alpha = \beta + 1$; let r be the word $a_{|r|} \dots a_1$. By the induction on α, INT_d needs time $\leq m + |\lceil g \rceil| B_\beta(n + 1_\alpha)$ to produce the istantaneous description

$$T_f = ZX\ RC;\ T^x = s_0\ g(s, t, a_1);\ T^y = t_0 t;\ T^z = r_0 r a_1;\ T^u = qr.$$

If $|r| > 1$ then INT_d puts $T_f := Z\ X\ RC\ \lceil h \rceil$ and $T^z := r_0 r a_2 a_1$, and calls itself in order to compute h and the next value of f. Again by the induction on α we have that INT_d needs time $\leq |g(s, t, a_1)| + |\lceil h \rceil| B_\beta(n + 1_\alpha)$ to produce an istantaneous description of the form

$$T_f = ZX\ RC;\ T^x = s_0\ (h(g(s, t, a_1), t, a_2 a_1));\ T^y = t_0 t;\ T^z = r_0 r a_2 a_1;\ T^u = qr.$$

After $|r| - 1$ simulations of h we obtain the promised istantaneous description within an overall amount of time

$$m + |r| \max(|\lceil g \rceil|, |\lceil h \rceil|) B_\beta(n + 1_\alpha) \leq m + |r| c B_\beta(n + 1_\alpha) \leq m + c B_\alpha(n + 1_\alpha),$$

where, since $\alpha \geq 2$, in these evaluations we may compensate the quadratic amount of time needed to copy r and its digits with the difference between c and $\max(|\lceil g \rceil|, |\lceil h \rceil|)$.
Case 2. $f = \text{CDIAG}(h) \in \mathcal{T}_\lambda$. We have $h \in \mathcal{T}_{\lambda_1}$, $1_\alpha = 1$, and (recall that $\lambda_1 = 1$ when $\lambda \leq \omega^2$)

(2) $\quad B_{\lambda_1}(n + 1) \cdot B_{\lambda_n}(n + 1) \leq B_{\lambda_{n+1}}(n + 1) = B_\lambda(n + 1).$

INT_d computes $h(r)$, understands from the mark DG that the result is the code for the function to be computed, and, accordingly, pushes it into T_f.

To compute $h(r)$ and $\{h(r)\}(s, t, r)$ the interpreter INT_d needs, by the induction on α, time $\leq m + |\lceil h \rceil| B_{\lambda(1)}(|r| + 1) + |\{h(r)\}| B_{\lambda(|r|)}(n + 1) \leq$ by (2) $m + |\lceil h \rceil| B_\lambda(n + 1) \leq m + c B_\lambda(n + 1)$.

```
INT(X, s, t, r):=
T_f := X; T^x := s; T^y := t; T^z := r;
while T_f not empty do A := last record(s) of T_f;
case
  A = ISBST(X, g, h)     then   X := g h; push g X in T_f
  A = X := x             then   copy last record of T^x into T^X
  A = REN(X, Y, h)       then   push h in T_f; copy last record of T^X into T^Y
  A = DIAG(h)            then   push DG h into T_f; copy last record of T^x into T^u
  A = DG                 then   pop T_f; pop last record of T^x and push it into T_f;
                                pop last record from T^u and push it into T^x
  A = SREC(g, h)         then   push A RC g into T_f; copy last record of T^z into T^u
                                push last digit of T^u into T^z
  A = SREC(g, h) RC      then   if T^u = T^z then pop T_f; pop T^u; pop T^z
                                else push h into T_f; pop last digit of T^u and push it into T^z
  A = ITRT(g)            then   call ITRT.
end case; end while.
```

Fig. 1

Reduction to 2 tapes Let e be an enumerator such that for all d there is r such that $\#(\{e\}(r)) > d$. If such an e occurs in the computation of f, then $f(s,t,r)$ is simulated by a TM $INT_c(\ulcorner f \urcorner, s, t, r)$ ($c = |s|+|t|+|r|$) which obviously depends on the arguments for f. However, we know from [8] that a k-tapes TM with time bound $T(n)$ can be simulated by a 2-tapes TM INT in time $kT(n)\log(T(n))$. Hence we can define a single 2-tapes TM INT which behaves like the INT_c's. For $\alpha \geq \omega$, runtime for INT is $T(n) = nB_\alpha(n+1)\log(B_\alpha(n+1)) \leq nB_\alpha(n+2)\log(n+1) \leq B_\alpha(n+4)$.

Note 18 We see from the proof above that the spread 4 in the statement of last lemma is reduced to 1 if we have $\#(g) \leq d$ for some d and for all g involved in the computation of f. This can be obtained by adding some cumbrous syntactic clauses to Definitions 2 and 4.

4.3 Simulation of TM's

Lemma 19 Given
(a) a function $\{p\}$ such that, for a numerical function $F(n)$, we have $|\{p\}(s,t)| = |s| + F(|t|)$;
(b) the function h in $\mathcal{T}_1$ defined as follows

$$\begin{cases} h(x,\epsilon) & = & x \\ h(x,ya) & = & \langle \text{IDT}_z, \langle \text{SR}, x, \{h(x,y)\}\rangle\rangle \end{cases}$$

we have that for all t, q: $|\{h(p,q)\}(\epsilon,t)| = F(|t|)|t|^{|q|}$.

Example 20 For $p_1 :=^{\ulcorner} \text{C}_1^{1\urcorner}$, we have $|\{p_1\}(s,t)| = |s| + 1$. Last lemma (for $F(n) = 1$) says that $|\{h(p_1,q)\}(\epsilon,t)| = |t|^{|q|}$. Hence $\text{CDIAG}(\text{ASG}(p_1,x,h))$ is the f_ω of Ex.8, which *computes in unary* B_ω. Define further

$$\begin{cases} k(p,\epsilon) & = & p \\ k(p,qa) & = & \langle \text{CDIAG}, \langle \text{ASG}, k(p,q), x, h(x,y)\rangle\rangle \end{cases}$$

By last lemma and induction on $|q|$ we may prove that, if p_ω codes f_ω, we have $|\{k(p_\omega,q)\}(\epsilon,t)| = B_{\omega\cdot|q|}(|t|)$. Hence, $\text{CDIAG}(k(p_\omega,y))$ computes in unary B_{ω^2}.
Notice that all these enumerators are obtained by a safe recursion which adds at each step a constant information to its previous value, and, therefore, they belong to $\mathcal{T}_1$. A uniform way to build codes of functions $\{p_\alpha\}$ computing in unary B_α for every $\alpha \leq \omega^\omega$ can be defined similarly.

Proof of Lemma 19. Note that the first occurrence of x in "$\langle \text{IDT}_z, \langle \text{SR}, x, \{h(x,y)\}\rangle\rangle$" is the code for a variable, while the second is an argument (cf. Ex.16). Function $h(p,q)$ yields the codes for q nestings of IDT_z's and SR's over $\{p\}$. We show by induction on $|q|$ that we have

(3) $\quad |\{h(p,q)\}(s,t)| = |s| + F(|t|)|t|^{|q|}$.

Basis. $q = \epsilon$. We have

$$\begin{array}{lll} |\{h(p,\epsilon)\}(s,t)| & = |\{p\}(s,t)| & \text{(by the definition of } h) \\ & = |s| + F(|t|)|t|^0 & \text{(by the hypothesis on } \{p\}). \end{array}$$

Step. Define $e :=\text{SR}(x, \{h(p,q)\})$. We show by induction on $|r|$ that we have

(4) $\quad |e(s,t,r)| = |s| + F(|t|)|t|^{|q|}|r|.$

$$\begin{array}{lll} |e(s,t,rb)| & = |\{h(p,q)\}(e(s,t,r),t)| & \text{by definition of } e \\ & = |e(s,t,r)| + F(|t|)|t|^{|q|} & \text{by (3), that is by the induction on } q \\ & = |s| + F(|t|)|t|^{|q|}|r| + |F(|t|)|t|^{|q|} & \text{by (4), that is by the induction on } |r| \\ & = |s| + F(|t|)|t|^{|q|}(|r|+1). & \end{array}$$

The result follows from (4) since we have

$$|\{h(p,qa)\}(s,t)| = |e(s,t,t)| = |s| + F(|t|)|t|^{|q|+1}.$$

Lemma 21 1. For all $\alpha < \epsilon_0$ there exists $g_\alpha \in \mathcal{T}_\alpha$ such that $|g_\alpha(s)| = B_\alpha(|s|)$.

2. Every TM whose runtime is bounded above by B_α can be simulated in $\mathcal{T}_\alpha$.

Proof. 1. Case 1. $\alpha = \beta + 1$. Define $g_\alpha = \{h(\lceil g_\beta \rceil, 1)\}$, where g_β is granted by the induction and h is given by last lemma.
Case 2. $\alpha = \omega^\beta$. Define in $\mathcal{T}_\beta$ by ISBST $e_\alpha = \{h(\lceil \text{C}_1^1 \rceil, g_\beta(y))\}$, where $g_\beta \in \mathcal{T}_\beta$ is given by the induction. The induction and Lemma 19 give

$$|\{e_\alpha(s)\}(s)| = |s|^{B_\beta(|s|)}.$$

Since $\beta \leq (\omega^\beta)_1$, we may define $g_\alpha(s) :=\text{CDIAG}(e_\alpha)$ in $\mathcal{T}_\alpha$. By the induction and last lemma

$$|g_\alpha(s)| = |\{e_\alpha(s)\}(s)| = |s|^{B_\beta(|s|)}.$$

The result follows since $clps(\omega^\beta, n) = n^{clps(\alpha,n)}$ implies $B_{\omega^\beta}(n) = n^{B_\beta(n)}$.
Case 3. $\alpha = \lambda + \omega^\beta$ $(\lambda \geq \omega^\beta)$. Define in $\mathcal{T}_\beta$ by ISBST $e_\alpha(s) := \{h(\lceil g_\lambda \rceil, g_\beta(y))\}$. Since $\omega^\beta \leq \alpha_1$, we may define $g_\alpha(s) :=\text{CDIAG}(e_\alpha)$ in $\mathcal{T}_\alpha$. By the induction and last lemma we have

$$|\{g_\alpha(s)\}(s)| = B_\lambda(|s|)|s|^{B_\beta(|s|)} = B_\lambda(|s|)B_{\omega^\beta}(|s|).$$

The result follows since $clps(\lambda+\omega^\beta, n) = clps(\lambda, n)+clps(\omega^\beta, n)$ implies $B_{\lambda+\omega^\beta}(n) = B_\lambda(n)B_{\omega^\beta}(n)$.
2. The behaviour of the TM M for $B_\alpha(|t|)$ steps is returned by $linsim_M(t, g_\alpha(t))$. This function is in $\mathcal{T}_\alpha$ since is defined by IDT_x and by ISBST of the function g_α (in $\mathcal{T}_\alpha$, by part 1) inside the function *linsim* (in $\mathcal{T}_1$, by lemma 1).

5 On the adopted form of predicative recursion

Our safe recursion may be regarded as a scheme of *simultaneous safe primitive recursion on (binary) notations*. Following a method used by Schwichtenberg, let us read all zeroes occurring in a word $s \in \mathbf{T}^*$ as *commas* and all words over $\mathbf{B} := \{1, 2\}$ as numbers in binary modified form, with ϵ representing 0 (for example read 1100220 as $3, 0, 6, 0$). It is then natural to write s in the form $\vec{s}$,

where s_i is the $(s)_i$ of Def. 2. At this point, one may replace our D_i,C_i^a,CASE_i with the ordinary de/con-structors and CASE on binary notations. Defining $f = \mathrm{SR}(g,h)$ with $\#(f) = d$ is then tantamount to a sequence of functions f_i $(i \leq d)$ defined by simultaneous primitive recursion on notations

$$\begin{cases} f_i(\vec{s},\vec{t},\epsilon) &= g_i(\vec{s},\vec{t},\epsilon) \\ f_i(\vec{s},\vec{t},za) &= h_i(f_1(\vec{s},\vec{t},z),\ldots,f_d(\vec{s},\vec{t},z),\vec{t},z). \end{cases}$$

The distinction between x, y, z keeps the recursion *safe*. Separation of the auxiliary variable of an outer cycle from the variable already used as principal in an inner cycle is obtained by just using different variables: no bookkeeping of their *history* along the construction of f has to be taken.

6 On susbtitution

Let $\mathcal{E}_3^{\mathrm{ISB}}$ be the class of all functions definable by closure of $\mathcal{T}_1$ under safe recursion, inessential substitution and at most one application of the primitive recursion scheme in $g(y)$ and in $h(x,y,z)$

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

Let $lh(f)$ be the number of safe recursions, primitive recursions and inessential substitution occurring in the definition of f. The following lemma shows that there is a gap between $\mathcal{E}_3^{\mathrm{ISB}}$ and $\mathcal{E}_4$; this means that full substitution (instead of our inessential substitution) is needed to cover the gap.

Lemma 22 For all $f \in \mathcal{E}_3^{\mathrm{ISB}}$ we have $|f(s,t,r)| \leq |s| + lh(f)(|t|+|r|)^{lh(f)^{|t|+|r|}}$.

Proof. Define $c := lh(f)$, $m := |s|$, $n := |t| + |r|$ and $i \oplus j := i^{j^i}$.
We show by induction on c and n that we have $|f(s,t,r)| \leq m + c(n \oplus c)$.
Step. Assume that f is defined by safe recursion or primitive recursion in g and h, since else the result is an obvious consequence of the ind. hyp. Case 1. Safe recursion. We have $|f(s,t,ra)| = |h(f(s,t,r),t,ra)| \leq$

$$\begin{array}{ll} \leq |f(s,t,r)| + (c-1)((n+1) \oplus (c-1)) & \text{by induction on } c \\ \leq m + c((n+1) \oplus c) + (c-1)((n+1) \oplus (c-1)) & \text{by induction on } r \\ \leq m + c((n+1) \oplus c) & \text{by definition of } \oplus \text{ and computations.} \end{array}$$

Case 2. Primitive recursion. We have $h(y,z) \in \mathcal{T}_{c-1}$ and (cf. proof of Lemma 17)

(5) $\quad |h(p,q,u)| \leq |p| + (c-1)(|q|+|u|)^{c-1}$.

We have $|f(t,ra)| = |h(ra,t,f(t,r))| \leq$ (assuming for simplicity that y is absent)

$$\begin{array}{ll} \leq n + 1 + (c-1)|f(r)|^{(c-1)} & \text{by (5)} \\ \leq n + 1 + (c-1)c(n \oplus c) & \text{by the induction on } c \\ \leq c((n+1) \oplus c) & \text{by definition of } \oplus \text{ and immediate computations.} \end{array}$$

References

[1] A. Beckmann and A. Weiermann, *Characterizing the elementary recursive functions by a fragment of Gödel T*— To appear

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

[3] S.J.Bellantoni and K.-H.Niggl,*Ranking primitive recursion: the low Grzegorczyck classes revisited* (1998, to appear on SIAM J. Computing).

[4] S. Caporaso, *Safe TM' s, Grzegorczyk classes and Polytime*, Intern. J. Found. Comp. Sc.,7.3(1996)241-252.

[5] S. Caporaso, M. Zito, N. Galesi, A predicative and decidable characterization of the polynomial classes of languages — To appear on Theoretical Comp. Sc.

[6] S. Caporaso, M. Zito, N. Galesi, E. Covino, Syntactic characterization in Lisp of the polynomial complexity classes and hierarchies. In G. Bongiovanni, D.P. Bovet, G. Di Battista (eds) Algorithms and Complexity. LNCS 1203. (Springer, Berlin, 1997).61-73.

[7] M. Fairtlough and S.S. Wainer, *Hierarchies of provably recursive functions*, in S. Buss (ed.) Handbook of proof theory. (Elsevier, Amsterdam 1998).

[8] F.C.Hennie and R.E.Stearns, *Two-tape simulation of multi-tapes TM's.*

[9] D. Leivant,*Predicative recurrence in finite types*, in A. Nerode and Y. V. Matiyasevich (eds.), Logical foundations of computer science, Springer lecture notes in computer science 813(1994), 227-239.

[10] D. Leivant, Stratified functional programs and computational complexity. In Conference records of the 20th annual ACM symposium on Principle of programming languaged. New York, 1993.

[11] D. Leivant, Stratified functional programs and computational complexity. In Conference records of the 20th annual ACM symposium on Principle of programming languaged. New York, 1993.

[12] D. Leivant and J.-Y. Marion, Ramified recurrence and computational complexity II: substitution and polyspace. In J. Tiuryn and L. Pacholski (eds), Computer Science Logic (Springer LNCS 933(1995) 486-500).

[13] K.-H. Niggl,*The μ-measure as a tool for classifying computational complexity.* Logic Coll. 97, Leeds 1997.

[14] H.E. Rose, *Subrecursion: Functions and hierarchies* (Oxford University Press, Oxford, 1984).

[15] K. Schütte, *Proof Theory* (Springer, 1977).

[16] H. Simmons,*The realm of primitive recursion*, Arch.Math. Logic, 27(1988), 177-188.

[17] A. Weiermann, *What makes a (pointwise) subrecursive hierarchy slow growing?* To appear.