A Hierarchy of Imperative Languages for the Feasible Classes DTIMEF(n^k) and for the Superexponential Classes DTIMEF$({}^k n)$

Salvatore Caporaso
Università di Bari
Dipartimento di Informatica
Via Orabona, I-70125 Bari
Italy
caporaso@di.uniba.it

Emanuele Covino
Università di Bari
Dipartimento di Informatica
Via Orabona, I-70125 Bari
Italy
covino@di.uniba.it

Vittoria Cozza
Università di Bari
Dipartimento di Informatica
Via Orabona, I-70125 Bari
Italy
v.cozza@di.uniba.it

Paolo Gissi
Università di Bari
Dipartimento di Informatica
Via Orabona, I-70125 Bari
Italy

Giovanni Pani
Università di Bari
Dipartimento di Informatica
Via Orabona, I-70125 Bari
Italy

Abstract: An imperative programming language is defined by closure of a free word-algebra of de/con-structors under two new operators (simultaneous safe recurrence and constructive diagonalization). By assigning ordinals to its programs a transfinite hierarchy of imperative languages is introduced which singles-out the feasible classes DTIMEF(n^k) and the superexponential classes DTIMEF$({}^k n)$.

Key–Words: Implicit computational complexity, Computational complexity, Elementary functions, Superexponential classes

1 Statement of the result.

Assume defined a scheme of *inessential substitution* (`isbst`), such that classes like DTIMEF(n^k) are closed under this scheme, together with a scheme of *safe recursion* (safe in the sense of [1, 14]) Let us say that f is defined by *constructive diagonalization in the enumerating function* e if we have $f(n) = \{e(n)\}(n)$, where $\{m\}$ is Kleene's notation for the function coded by m. Define a hierarchy $\mathcal{T}_\alpha$ by
$\mathcal{T}_1$ is a characterization of $\texttt{DTIMEF}(n)$;
$\mathcal{T}_{\alpha+1}$ is the closure under `isbst` of the class of all functions obtained by at most one application of (our) safe recurrence scheme to functions in $\mathcal{T}_\alpha$;
$\mathcal{T}_\lambda$ 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 (CNF) for α)

$$\begin{aligned}\text{DTIMEF}(n^{clps(\alpha,n)}) &\subseteq \mathcal{T}_\alpha \\ &\subseteq \text{DTIMEF}((n+4)^{clps(\alpha,n+4)}).\end{aligned} \tag{1}$$

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

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

Under a more heavy syntax, the *spread* 4 in (1) can be reduced to 1 (see Note 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 numerals 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 variables, parameters, recursion variables*. This convention is tantamount to the use of a semicolon $(z; x, y)$ to separate safe and unsafe arguments by Simmons [14] and Bellantoni &

Cook [**?**]). When we write $f(u_1, \ldots, u_n)$ we always assume that some of the indicated variables may be absent.

Definition 1 Given a word s in which n zeroes occur, $(s)_i$ $(1 \leq i \leq n+1)$ is the i-th rightmost word (possibly empty) in the alphabet $\{1, 2\}$ which occurs between two zeroes of s (0 replaced by ϵ when $i = 1, n+1$); if $i > n+1$ then $(s)_i$ does not exist (cf. Schwichtenberg [13] and Sect. 6 for this use of ternary words to represent tuples). Given $i \geq 0$ and $a = 1, 2$,

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

   all these constructors and destructors leave s unchanged if $(s)_i$ doesn't exist;
3. the construct $\mathtt{case}_i[f, g, h](s)$ 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 $\mathtt{d}_2(010) = \mathtt{d}_2(00) = 00$; $\mathtt{d}_1(1) = \epsilon$; $\mathtt{c}_1^1(0) = 01$.

Definition 2 $f = \mathtt{sr}(g, h)$ 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.

Definition 3 $f = \mathtt{cdiag}(e)$ 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 4 1. $f = \mathtt{asg}(s, u, g)$ is the result of the *assignment* of s to variable u in g;

2. $f = \mathtt{idt}_x(g)$ is the result of the *identification* of x as y in g (thus, $f(y, z) = g(y, y, z)$); similarly, $f = \mathtt{idt}_z(g)$ is the result of the *identification* of z as y in g (thus, $f(x, y) = g(x, y, y)$).
3. $f = \mathtt{sbst}(u, h, g)$ 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 5 (1) Class $\mathcal{T}_0$ is the closure under `asg`, $\mathtt{idt}_x$, $\mathtt{idt}_z$ and `sbst` of $\mathtt{c}_i^a$, $\mathtt{d}_i$, $\mathtt{case}_i$;

(2) Class $\mathcal{T}_1$ is the closure under `asg`, $\mathtt{idt}_x$, $\mathtt{idt}_z$ 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 6 The *number of components* $\#(f)$ of $f \in \mathcal{T}_1$ is $\max\{i | \mathtt{d}_i \text{ or } \mathtt{c}_i^a \text{ or } \mathtt{case}_i \text{ occurs in } f\}$.

Example 7 Define (by iteration of $\mathtt{c}_0^1$) function $g_1 \in \mathcal{T}_1$ such that $g_1(x, y) = x1^{|y|}$. Define further

$$\begin{cases} f_{n+1}(s, t, \epsilon) &= s \\ f_{n+1}(s, t, ra) &= g_n(f_{n+1}(s, t, r), t) \end{cases} \quad (\mathtt{sr})$$
$$g_{n+1}(x, y) = f_{n+1}(x, y, y) \quad (\mathtt{idt}_z)$$

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

$|f_\omega(\epsilon, t, ta)| = g_{|t|}(\epsilon, ta) = |t|^{|t|}$.

Lemma 8 $\mathtt{DTIMEF}(n) = \mathcal{T}_1$.

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` M 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. 7 by assigning ϵ to x and by some `sbst`'s.

3 The hierarchy

Definition 9 We adopt the following assignment of fundamental sequences to all $\lambda < \epsilon_0$

$$\lambda_n = \begin{cases} n & \text{if } n \leq 1 \text{ and } \lambda < \omega^2 \\ \omega^{\mu_n} & \text{if CNF for } \lambda \text{ is } \omega^\mu \\ \omega^\alpha n & \text{if CNF for } \lambda \text{ is } \omega^{\alpha+1} \\ \mu + (\omega^\alpha)_n & \text{if CNF for } \lambda \text{ is } \mu + \omega^\alpha. \end{cases}$$

Comment. Assignment S of last definition differs from the traditional standard assignment S^* (see [11, p. 78]) at line 1 (adopted to simplify the proof of Lemma 16 — 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 $|t|^{n+1}$. The slow hierarchy G_α, when defined with respect to S (cf. §1.1), is obviously dominated by the one defined with respect to S^*. On the other hand 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 [11, Th. 3.6]).

Definition 10 $f = \texttt{isbst}(u,h,g)$ 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 11 (a) $\mathcal{T}_{\alpha+1}$ ($\alpha > 0$) is the closure under `asg`, $\texttt{idt}_x$, $\texttt{idt}_z$, `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`, $\texttt{idt}_x$, $\texttt{idt}_z$, `isbst` of the functions of the form $\texttt{cdiag}(e)$, for $e \in \mathcal{T}_{\lambda_1}$, and such that $\{e(r)\} \in \mathcal{T}_{\lambda_{|r|}}$.

Notation 12 $B_\alpha(n) := \max(2,n)^{clps(\alpha,n)}$

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

Theorem 13 1. For all $\omega \leq \alpha < \epsilon_0$ we have $\texttt{DTIMEF}(B_\alpha(n)) \subseteq \mathcal{T}_\alpha \subseteq \texttt{DTIMEF}(B_\alpha(n+4))$.
2. For all finite k, $\texttt{dtime}(n^k) = \mathcal{T}_k$.

Proof. **1.** By proof of Lemma 8, 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 20, 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 16, 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* $\mathbf{U} = \{0,1,2,\texttt{isbst},\texttt{sr},\texttt{cdiag},\texttt{itrt},\ldots\}$. 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 14 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 15 If $f(x,z) \in \mathcal{T}_1$ is the $|z|$-th iterate of function $e \in \mathcal{T}_0$,then its code is $\langle \texttt{itrt}, e\rangle$. The code for functions g_{n+1} of Ex.7 is $\langle \texttt{idt}_z, \langle \texttt{sr}, x, g_n\rangle\rangle$, where $\lceil g_1 \rceil = \langle \texttt{idt}_z, \langle \texttt{itrt}, \texttt{c}_0^1\rangle\rangle$.

4.2 Simulation by TM's

Lemma 16 For all $\alpha \geq \omega$ we have $\mathcal{T}_\alpha \subseteq \texttt{DTIMEF}(B_\alpha(n+4))$; for all finite m, $\mathcal{T}_m \subseteq \texttt{DTIMEF}(B_m(n))$.

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. 5 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 $\mathtt{itrt}(g)$ 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 $f \in \mathcal{T}_\alpha$ implies

$$INT_d(\lceil f \rceil, s, t, r) = f(s, t, r)$$

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 $\mathtt{DTIMEF}(B_\alpha(n + 1_\alpha))$ by the sequence composition of the constant-time $\mathtt{TM}$ 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. $f = \mathtt{sr}(g, h)$. 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 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. $f = \mathtt{cdiag}(h) \in \mathcal{T}_\lambda$. We have $h \in \mathcal{T}_{\lambda_1}$, $1_\alpha = 1$, and (recall that $\lambda_1 = 1$ when $\lambda \leq \omega^2$)

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

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 eq. (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 [7] 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 17 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 1 and 3.

4.3 Simulation of TM's

Lemma 18 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

$$\begin{cases} h(x,\epsilon) &= x \\ h(x,ya) &= \langle \mathtt{idt}_z, \langle \mathtt{sr}, x, \{h(x,y)\}\rangle\rangle \end{cases}$$

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

Example 19 For $p_1 :=^{\ulcorner} \mathtt{c}_1^1 \urcorner$, 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 $\mathtt{cdiag}(\mathtt{asg}(p_1,x,h))$ is the f_ω of Ex.7, which *computes in unary* B_ω. Define further

$$\begin{cases} k(p,\epsilon) &= p \\ k(p,qa) &= \langle \mathtt{cdiag}, \langle \mathtt{asg}, k(p,q), x, h(x,y)\rangle\rangle \end{cases}$$

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, $\mathtt{cdiag}(k(p_\omega,y))$ 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 18. Note that the first occurrence of x in "$\langle \mathtt{idt}_z, \langle \mathtt{sr}, x, \{h(x,y)\}\rangle\rangle$" is the code for a variable, while the second is an argument (cf. Ex.15). Function $h(p,q)$ yields the codes for q nestings of $\mathtt{idt}_z$'s and $\mathtt{sr}$'s over $\{p\}$. We show by induction on $|q|$ that we have

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

Basis. $q = \epsilon$. We have

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

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

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

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

The result follows from (4) since we have

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

Lemma 20
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 $\mathtt{isbst}\ e_\alpha = \{h(\ulcorner \mathtt{c}_1^1 \urcorner, g_\beta(y))\}$, where $g_\beta \in \mathcal{T}_\beta$ is given by the induction. The induction and Lemma 18 give

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

Since $\beta \leq (\omega^\beta)_1$, we may define $g_\alpha(s) :=\mathtt{cdiag}(e_\alpha)$ 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 $\mathtt{isbst}\ e_\alpha(s) := \{h(\lceil g_\lambda \rceil, g_\beta(y))\}$. Since $\omega^\beta \leq \alpha_1$,

we may define $g_\alpha(s) :=\mathtt{cdiag}(e_\alpha)$ 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 $\mathtt{idt}_x$ and by $\mathtt{isbst}$ 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 dating back to Schwichtenberg [13, 1967], 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. 1. At this point, one may replace our $\mathtt{d}_i$,$\mathtt{c}^a_i$,$\mathtt{case}_i$ with the ordinary de/constructors and $\mathtt{case}$ on binary notations. Defining $f =\mathtt{sr}(g, h)$ 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: by doing so, no bookkeeping of their *history* along the construction of f has to be taken.

References:

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

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

[3] S. Caporaso, M. Zito, N. Galesi, *A predicative and decidable characterization of the polynomial classes of languages* Theoretical Comp. Sc. 250(2001)83-99

[4] 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.

[5] S. Caporaso, E. Covino, *A predicative approach to the classification problem* Journal of Functionnal Programming 11.1(2001)95-116.

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

[7] F.C.Hennie and R.E.Stearns, *Two-tape simulation of multi-tapes TM's*. J.ACM 13.4 (1966) 533-546.

[8] D. Leivant, *A foundational delineation of computational feasibility.* In Proc. of the Sixth IEEE Conference on Logic in Computer Science. (IEEE Computer Society Press, 1991, p.2-18).

[9] D. Leivant,*Predicative recurrence in finite types*, in A. Nerode and Y. V. Matiyasevich (eds.), Logical foundations of computer science, LNCS 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 languages. New York, 1993.

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

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

[13] H. Schwichtenberg, *Eine Klassification der ϵ_0-rekursiven Funktionen,* Zeitschr. math. Logik u. Grundl. d. Math. 17(1971)61-74.

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