

# Predicative Recursion, Ramified Diagonalization and the Elementary Functions

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

**Abstract** The characterization of FP by Bellantoni and Cook is refined and extended to define an  $\omega^2$ -type hierarchy  $\mathcal{T}_\alpha$ , whose finite elements  $\mathcal{T}_c$  characterize the complexity classes  $\text{DTIMEF}(n^c)$ , while those of the form  $\mathcal{T}_{\omega_{k+c}}$  are equivalent to  $\text{DTIMEF}(n_k(n^c))$  ( $k$ -superexponential). The *refinement* allowing to capture the former classes is obtained by restricting substitution and by strengthening meanwhile the *safe predicative recursion* scheme. *Extension* to the latter is by a *diagonalization* scheme.

## 1 Introduction

We define an  $\omega^2$ -type hierarchy  $\mathcal{T}_\alpha$ , whose elements  $\mathcal{T}_c$  characterize the complexity classes  $\text{DTIMEF}(n^c)$ , while  $\mathcal{T}_{\omega_{k+c}}$  is equivalent to  $\text{DTIMEF}(n_k(n^c))$  (for  $n_0(m) = m$ , and  $n_{k+1}(m) = n^{n_k(m)}$ ). This seems to refine and extend the celebrated recursion-theoretical characterization of FP by Bellantoni and Cook [1] (hereinafter B&C).

Our data-type is like theirs: *notations*, and distinction between *normal* and *safe* variables. Below FP we use a weaker substitution, and a stronger recursion; beyond it we add *ramified diagonalization*. That is:

(1) Since our classes are not closed under substitution, this scheme has been drastically restricted to (a) composition between constant-time operators; (b) the substitution mentioned under (3) below.

(2) *Simultaneous* safe recursion is used to define  $n$ -ples of new functions. For  $n = 1$ , this reduces to Predicative Recursion on Notations (PRN).

(3) Transfinite hierarchies are usually obtained by extending a Grzegorzczuk-like sequence  $E_n(x)$  by means of a diagonalization of the form  $E_\lambda(n) = E_{\lambda[n]}(n)$ ; a new class  $\mathcal{E}_\lambda$  is defined by closure under  $\text{PR}_\leq$  (that is PR *limited* by  $E_\lambda$ ). Doubts about  $\text{FP} = \text{PRN}_\leq + \text{smash}$  (cf. sect. 1, §2 of B&C) apply word for word to this method: an operator too strong, limited by an *ad hoc* function. The approach adopted here is different. We get a new class by applying an un-limited operator  $\Delta$  (*ramified diagonalization*) to a previously generated class. Thus  $\mathcal{T}_{\omega_{(k+1)+c}}$  consists of all functions of the form  $f = \Delta(g, e)$  with  $e \in \mathcal{T}_{\omega_{k+c}}$  and  $g \in \mathcal{T}_1$ , where (following Kleene,  $\{n\}$  is the function coded by  $n$ )

$$f(\mathbf{x}) = \{g(e(x))\}(\mathbf{x}) \quad \text{provided that } \Delta \text{ doesn't occur in } \{g(e(y))\}.$$

$\Delta$  works constructively, since it has just to run an enumerator. However, clause “provided ...” is not likely to be decidable.

An initial class  $\mathcal{T}_0$  is defined by closure of the ordinary de/constructors under branching and sequence composition.  $\mathcal{T}_{c+1}$  is the class of all  $n$ -ples which can be defined by a single simultaneous recursion in  $n$ -ples of functions in  $\mathcal{T}_c$ .  $\mathcal{T}_{\omega(k+1)+c}$  is the class of all functions definable by a single diagonalization in an enumerator belonging to  $\mathcal{T}_{\omega k+c}$ .

B&C acknowledge to Leivant [3] the foundational background of their result, and characterizations from both points of view (recursion-theoretical vs provability) of the elementary functions can be found in his recent works [4] and [5]. He used register machines to obtain the classes  $\text{DTIMEF}(n^c)$ , for all even  $c$ , in his [2] which dates back to 1994.

## 2 The Initial Class

**1 Data**  $\mathbf{N}$  is the class of all (*binary*) numerals.  $u, \dots$  are variables defined on  $\mathbf{N}$  or, in certain cases (the safe variables below), on  $\mathbf{N}^* = \mathbf{N} \cup \{\varepsilon\}$ .  $\mathbf{u}, \dots, \mathbf{z}$  are  $n$ -ples of variables ( $n \geq 1$ ).  $x_i$  is the  $i$ -th element of  $\mathbf{x}$ .  $a$  and  $b$  are letters in  $\{0, 1\}$ .

The lengths  $|x|$  and  $|\mathbf{x}|$  are defined in the usual way.  $(u)^n$  is the concatenation of  $n$  copies of word  $u$ , and for  $n = 0$  is  $\varepsilon$  (parentheses omitted when  $u$  is a single letter).  $\underline{n}$  is  $1^n$ .

**2 Functions**  $f(x_1, \dots, x_m; y_1, \dots, y_n)$  maps  $\mathbf{N}^m \times \mathbf{N}^{*n}$  into  $\mathbf{N}$ . Variables  $x_i$  and  $y_j$  are categorized as *normal* and *safe*.

Given  $f(\mathbf{x}; \mathbf{y})$ , we write  $f(\mathbf{x};)$  for  $f(\mathbf{x}; \varepsilon, \dots, \varepsilon)$ , and we say that  $\mathbf{y}$  is *absent*.

$\mathbf{f}, \mathbf{g}, \dots$  will denote  $n$ -ples of functions having the same number of normal and safe variables.  $f_i$  is the  $i$ -th element of  $\mathbf{f}$ . We identify  $f$  with the 1-ple  $\mathbf{g}$  such that  $g_1 = f$ .

Each  $f$  can be written in *infix* form, and in a *suffix* (Polish, parentheses-free) form. We use the former when readability is our main concern; the latter when  $f$  is the object of some processing like coding, enumeration, or simulation.

$\#(\mathbf{f})$  and  $\#(\mathbf{x})$  will denote the number of elements of  $\mathbf{f}$  and  $\mathbf{x}$ .

**3 Modifiers** The *selectors*, the *de/constructors* and the *constant function* are given by ( $\mathbf{y}$  not absent and  $x \neq \varepsilon$ )

$$U_i(\mathbf{x}; \mathbf{y}) = y_i; \quad C^a(; x) = xa; \quad D(; xa) = x, \quad D(; a) = 1 - a; \quad Q(; x) = 0.$$

The *modifiers* are defined by closure of the selectors under functions  $C^a$ ,  $D$ , and  $Q$ . In the infix form, selectors and constants are replaced by their values, while composition of de/constructors are shown by means of parentheses. If  $g, \dots, h$  are de/constructors or constants, then the suffix form of  $g(\dots(h(U_i(\mathbf{x}; \mathbf{y})))\dots)$  is  $U g \dots h i$ .

$f = g_0 g_1 j B_a$  is defined by *branching* if we have  $(\mathbf{u} = u_1, \dots, u_n; u_{n+1}, \dots, u_m)$

$$f(\mathbf{u}) = \text{if } u_j = u \text{ a with } u \neq \varepsilon \text{ then } g_0(\mathbf{u}) \text{ else } g_1(\mathbf{u}).$$

We get the *conditional modifiers* by closure of the modifiers under scheme  $B$ . An *i-modifier* is a conditional modifier  $f$ , such that we have  $j = i$  for all  $U_j$  occurring in  $f$ .

**4 Simple Tuples**  $\mathbf{f}$  is a *simple tuple* if each  $f_i$  is an *i-modifier*. Thus  $f_i(\mathbf{x}; \mathbf{y})$  modifies  $y_i$  in a way which depends on the less significant bit of all  $\mathbf{x}$  and  $\mathbf{y}$ .

**5 Simultaneous Safe Composition**  $\mathbf{f} = \mathbf{g} \mathbf{h} K$  is defined by scheme  $K$  in  $\mathbf{g}$  and in the simple  $n$ -ple  $\mathbf{h}$  if we have  $\mathbf{f}(\mathbf{x}; \mathbf{y}) = \mathbf{g}(\mathbf{x}; \mathbf{h}(\mathbf{x}; \mathbf{y}))$ ; that is if, for all  $i$ , we have

$$f_i(\mathbf{x}; \mathbf{y}) = g_i(\mathbf{x}; h_1(\mathbf{x}; \mathbf{y}), \dots, h_n(\mathbf{x}; \mathbf{y})).$$

**6 Example** (1) Given  $u = a_1 \dots a_m$ , define  $C_i\langle u \rangle = U C^{a_m} \dots C^{a_1} i$ . We obtain  $C_i\langle u \rangle(\mathbf{x}; \mathbf{y}) = y_i u$ . Define in a similar way the *i-modifier*  $Q_i\langle u \rangle(\mathbf{x}; \mathbf{y}) = u$ .

(2) Let  $\mathbf{C}\langle 1 \rangle$  denote the  $n$ -ple  $C_1\langle 1 \rangle, \dots, C_n\langle 1 \rangle$ . Tuple  $\mathbf{C}\langle 1 \rangle \mathbf{C}\langle 1 \rangle K$  is simple, and its  $i$ -th element yields  $y_i 11$ .

**7 Initial Functions**  $\mathcal{T}_0$  is the closure of the simple tuples under scheme  $K$ . The form of its elements is  $f(\cdot; \mathbf{y})$  with  $\mathbf{y}$  assumed to be not absent.

### 3 The Hierarchy

**8 Simultaneous Safe Recursion**  $\mathbf{f} = \mathbf{h} \mathbf{g} R_i$  is defined by scheme  $R_i$  in the *basis functions*  $\mathbf{g}(\mathbf{u}; \mathbf{w})$  and in the *step functions*  $\mathbf{h}(\mathbf{u}; \mathbf{w})$  if we have

$$\begin{cases} \hat{f}_i(0, \mathbf{x}; \mathbf{y}) &= y_i \\ \hat{f}_i(1, \mathbf{x}; \mathbf{y}) &= g_i(\mathbf{x}; \hat{\mathbf{f}}(0, \mathbf{x}; \mathbf{y})) \\ \hat{f}_i(z a, \mathbf{x}; \mathbf{y}) &= h_i(\mathbf{x}; \hat{\mathbf{f}}(z, \mathbf{x}; \mathbf{y})) \\ \mathbf{f}(\mathbf{x}; \mathbf{y}) &= \hat{\mathbf{f}}(x_i, \mathbf{x}; \mathbf{y}) \\ \mathbf{f}(\mathbf{x}; ) &= \mathbf{f}(\mathbf{x}; \mathbf{x}) \end{cases}$$

with  $1 \leq i \leq \#(\mathbf{f}) = \#(\mathbf{h}) = \#(\mathbf{g}) = \#(\mathbf{x}) = \#(\mathbf{w})$ .

**9 Note** The safe variables may be understood as *program variables* for storage of the intermediate values of the functions being recursed upon. Hence, we need  $\#(\mathbf{f}) = \#(\mathbf{w})$ . The rationale of clause  $\mathbf{f}(\mathbf{x}; ) = \mathbf{f}(\mathbf{x}; \mathbf{x})$  and of the restriction to functions of the form  $f(\mathbf{x}; )$  in Th. 16 is that: (1) the program registers are empty in the input; (2) when the computation of  $g(\mathbf{x}; \mathbf{y})$  occurs in the computation of another function, the values for  $\mathbf{y}$  are assigned by the call to  $g$ ; (3) when a computation begins, the default value of  $y_i$  is the input  $x_i$ .

Clause  $\#(\mathbf{f}) = \#(\mathbf{h}) = \#(\mathbf{g})$  says that we have as many defining as defined functions. In practice we may obtain  $1 \leq \#(\mathbf{f}) < \#(\mathbf{x})$  by taking some selectors as step functions. Under these comments we may regard the PRN scheme in B&C as a particular case of the present one.

**10 Iteration** Let us write  $\mathbf{h} I$  for  $\mathbf{h} \mathbf{h} R_1$ . Since we often use definitions of the form  $\mathbf{f} = \mathbf{h} I$ , we find expedient to regard  $I$  as an independent scheme, with its own code, not as a particular case of  $R$ .

**11 Example** Define  $\text{sum} = C_1\langle 1 \rangle I$  and  $\text{sq} = \text{sum} I$  (see 6 for  $C_1\langle 1 \rangle$ ). We have

$$\begin{cases} \widehat{\text{sum}}(0, x; y) &= y \\ \widehat{\text{sum}}(1, x; y) &= C\langle 1 \rangle(; \widehat{\text{sum}}(0, x; y)) = y1 \\ \widehat{\text{sum}}(za, x; y) &= C\langle 1 \rangle(; \widehat{\text{sum}}(z, x; y)) = y|z|1 \\ \text{sum}(x; y) &= y|x|; \quad \text{sum}(\underline{n}) = \underline{2n} \\ \widehat{\text{sq}}(0, x; y) &= y \\ \widehat{\text{sq}}(1, x; y) &= \text{sum}(x; \widehat{\text{sq}}(0, x; y)) = y|x| \\ \widehat{\text{sq}}(za, x; y) &= \text{sum}(x; \widehat{\text{sq}}(z, x; y)) = y|x|(|z| + 1) \\ \text{sq}(x; y) &= y|x| \cdot |y|; \quad \text{sq}(\underline{n}) = \underline{n(n+1)}. \end{cases}$$

**12 Codes** Tuples are built-up from shorter expressions by means of a symbol  $\Sigma$  from a collection  $\Sigma_1, \dots, \Sigma_c$  of symbols  $U, C^a, Q, R, I, \Delta, \dots, T$ , and perhaps of a positive number. Define  $\dot{\Sigma}_i = \dot{i}0$  and  $\dot{n} = \underline{n+c+1}0$ . The code  $\dot{\mathbf{f}}$  for  $\mathbf{f} = \mathbf{g}_1 \dots \mathbf{g}_k l \Sigma$  is  $\dot{\mathbf{g}}_1 \dots \dot{\mathbf{g}}_k \dot{l} \dot{\Sigma}$ . If  $\mathbf{f}$  is a simple tuple then  $\dot{\mathbf{f}}$  is  $\dot{f}_1 \dots \dot{f}_n \dot{T}$ .

**13**  $\{x\}$  and  $\{f(\mathbf{x}; y)\}$  are the tuples coded by  $x$  and by the value of  $f(\mathbf{x}; y)$ .

**14 Ramified Diagonalization**  $\mathbf{f} = g h \Delta$  is defined by scheme  $\Delta$  in  $g(x; )$  and in  $h(x; )$  if we have

$$\mathbf{f}(\mathbf{x}; ) = \{g(h(x_1; ;))\}(\mathbf{x}; ).$$

**15 The Hierarchy** (1)  $\mathcal{T}_\omega$  is the closure of  $\mathcal{T}_0$  under all schemes above but  $\Delta$ .

(2)  $\mathcal{T}_c$  is the subclass of  $\mathcal{T}_\omega$  definable by *recursion depth*  $c$  (that is, all basis and step functions, if any, used to construct  $f \in \mathcal{T}_c$  belong to some  $\mathcal{T}_d$ , with  $d < c$ ).

(3) For all  $k > 1$  define  $\mathcal{T}_{\omega k} = \bigcup_{\alpha < \omega k} \mathcal{T}_\alpha$ .

(4) Class  $\mathcal{T}_{\omega(k+1)+c}$  ( $c > 0$ ) is the class of all functions of the form

$$g h \Delta \quad (g \in \mathcal{T}_1; h \in \mathcal{T}_{\omega k+c}),$$

under the unconstructive restriction that  $\Delta$  should not occur in  $\{g(h(y; ;))\}$  for any  $y$ .

**16 Theorem** For all  $k \geq 0; c > 0$  we have  $\text{DTIMEF}(n_k(O(n^c))) = \{\mathbf{f}(\mathbf{x}; ) \mid \mathbf{f} \in \mathcal{T}_{\omega k+c}\}$ .

*Proof.* In Lemma 20 we show that for every  $\varphi : \mathbf{N}^m \mapsto \mathbf{N}$  which is computed by a TM in time  $O(n_k(O(n^c)))$  there exists an  $m$ -ple  $\mathbf{f} \in \mathcal{T}_{\omega k+c}$  such that  $f_1(\mathbf{x}; ) = \varphi(\mathbf{x})$ .

Conversely, in Sect. 5 we show that for each  $m$  a TM  $IN$  depending on  $m$  can be defined, which plays the role of *interpreter*, in the sense that, by input an appropriate code  $\dot{\mathbf{f}}$  for  $\mathbf{f}(\mathbf{x}; ) \in \mathcal{T}_{\omega k+c}$  and an  $m$ -ple  $\mathbf{x}$ ,  $IN$  returns the values of all  $f_i(\mathbf{x}; )$  in time  $3n_k(|\dot{\mathbf{f}}|^{2k+2}2n^c)$  for  $n = |\mathbf{x}|$ . To handle the diagonalization scheme, an interpreter is needed, instead of mere simulation.

## 4 Simulation of TM's

**17 Lemma** For all  $\mathbf{h} \in \mathcal{T}_0$  and  $0 < l, c < \omega$ , the *iterators*  $\mathbf{h}_{cl} \in \mathcal{T}_c$  can be defined, such that

$$\mathbf{h}_{cl}(\mathbf{x};) = \mathbf{h} \uparrow l|x_1|^c(\mathbf{x}) \quad (x_1 \neq 0),$$

for  $\mathbf{h} \uparrow 1(\mathbf{x}) = \mathbf{h}(\cdot; \mathbf{x})$  and  $\mathbf{h} \uparrow n + 1(\mathbf{x}) = \mathbf{h}(\cdot; \mathbf{h} \uparrow n(\mathbf{x}))$ .

*Proof.* Define  $\mathbf{h}_{0l} = \mathbf{h} (\mathbf{h} K)^{l-1}$ ;  $\mathbf{h}_{c+1l} = \mathbf{h}_{0l} I^{c+1}$  ( $= \mathbf{h}_{cl} I$ ).

We obviously have  $\mathbf{h}_{0l}(\cdot; \mathbf{x}) = \mathbf{h} \uparrow l(\mathbf{x})$ , and we show that  $\hat{\mathbf{h}}_{cl}(z, \mathbf{x};) = \mathbf{h} \uparrow l|x_1|^{c-1}|z|(\mathbf{x})$ .

Induction on  $c$ . Basis. Like under Ex. 11. Step. Side induction on  $|z|$ . Step. We have

$$\begin{aligned} \hat{\mathbf{h}}_{c+1l}(za, \mathbf{x};) &= \mathbf{h}_{cl}(\mathbf{x}; \hat{\mathbf{h}}_{c+1l}(z, \mathbf{x}; \mathbf{x})) && \text{Def. 10 of } I \\ &= \mathbf{h} \uparrow l|x_1|^c(\hat{\mathbf{h}}_{c+1l}(z, \mathbf{x}; \mathbf{x})) && \text{main I.H. with } |x_1| \text{ as } z \\ &= \mathbf{h} \uparrow l|x_1|^c(\mathbf{h} \uparrow l|x_1|^c|z|(\mathbf{x})) && \text{side I.H.} \\ &= \mathbf{h} \uparrow (l|x_1|^c + l|x_1|^c|z|)(\mathbf{x}) && = \mathbf{h} \uparrow l|x_1|^c|za|(\mathbf{x}). \end{aligned}$$

**18 Note** Last lemma, with the 1-ple  $C_1\langle 1 \rangle$  as  $\mathbf{h}$ , gives  $C_1\langle 1 \rangle_{cl}(x; ) = x \underline{l|x|}^c$ .

**19 Lemma** For all  $\mathbf{h} \in \mathcal{T}_0$ , the *exponential iterators*  $\mathbf{h}_{kcl} \in \mathcal{T}_{\omega k+c}$  can be defined, such that

$$\mathbf{h}_{kcl}(\mathbf{x};) = \mathbf{h} \uparrow m(\mathbf{x}) \quad (\text{for some } m \geq n_k(ln^c) \text{ with } x_1 \neq 0, n = |x_1|).$$

*Proof.* We divide this proof in parts. (I) By defining (see 6 for  $C_i\langle u \rangle$  and  $Q_i\langle u \rangle$ )

$$\hat{\tilde{e}}(0, x; ) = x \quad \hat{\tilde{e}}(1, x; ) = Q_1\langle \dot{\mathbf{h}}_{0l} \dot{I} \rangle(\cdot; x) \quad \hat{\tilde{e}}(ya, x; ) = C_1\langle \dot{I} \rangle(\cdot; \hat{\tilde{e}}(y, x; ))$$

we obtain

$$\tilde{e}(x; ) = \hat{\tilde{e}}(x, x; ) = \dot{\mathbf{h}}_{0l} \dot{I}^{|x|}; \quad \{\tilde{e}(x; )\} = \mathbf{h}_{0l} I^{|x|}. \quad (1)$$

(II) Define (see proof 17 for  $\mathbf{h}_{cl}$  and Note 18 for  $C_1\langle 1 \rangle_{cl}$ ;  $C_1\langle 1 \rangle_{cl}$  is  $\mathbf{h}_{cl}$  for  $\mathbf{h} = C_1\langle 1 \rangle$ )

$$\mathbf{h}_{0cl} = \mathbf{h}_{cl}; \quad \mathbf{h}_{k+1cl} = \tilde{e} C_1\langle 1 \rangle_{kcl} \Delta. \quad (2)$$

(III) We show that the result holds for the construction of part (II).

Induction on  $k$ . The basis is Lemma 17 (with  $m = n_0(ln^c)$ ).

Step. Observe that I. H. (with  $C_1\langle 1 \rangle$  as  $\mathbf{h}$ ) and Note 18 say that we have

$$C_1\langle 1 \rangle_{kcl}(y; ) = C_1\langle 1 \rangle \uparrow M(y) = y \underline{M} \quad \text{for some } M \geq |y|_k(l|y|^c). \quad (3)$$

We then have

$$\begin{aligned} \mathbf{h}_{k+1cl} &= \{\tilde{e}(C_1\langle 1 \rangle_{kcl}(x_1; ))\}(\mathbf{x};) && \text{Def. 14 of } \Delta \text{ and equation (2) above} \\ &= \{\tilde{e}(x_1 \underline{M}; )\}(\mathbf{x};) && \text{I.H. in the form of equation (3)} \\ &= \mathbf{h}_{0l} I^{|x_1|+M}(\mathbf{x};) && \text{equation (1)} \\ &= \mathbf{h}_{n+Ml}(\mathbf{x};) && \text{proof of Lemma 17} \\ &= \mathbf{h} \uparrow n^{l(n+M)}(\mathbf{x}) && \text{Lemma 17.} \end{aligned}$$

The result follows since  $M \geq n_k(ln^c)$  obviously implies  $n^{n+M} \geq n_{k+1}(ln^c)$ .

**20 Lemma**  $\text{DTIMEF}(n_k(O(n^c))) \subseteq \mathcal{T}_{\omega_k+c}$ .

*Proof.* We may assume, without any loss of generality, that each  $\varphi \in \text{DTIMEF}(n_k(O(n^c))) : \mathbf{N}^m \mapsto \mathbf{N}$ , is computed by a TM  $M$  with  $m$  push-down tapes over alphabet  $\{0, 1\}$ , which  $(N = \max_i(|x_i|) - |x_1|; n = |\mathbf{x}|)$

(a) starts operating with the arguments  $x_{i+1}$  for  $\varphi$  in tapes  $i + 1$ ; and with  $x_1 0^N$  in tape 1;  
(b) stops operating with the value of  $\varphi(\mathbf{x})$  in tape 1, in time  $O(n_k(O(n^c)))$ .

We can define: (a) an appropriate representation of the instantaneous descriptions (ID) of  $M$  into  $\mathbf{N}^m$ ; and (b) an  $m$ -ple of functions  $\mathbf{nxt}\langle M \rangle \in \mathcal{T}_0$  taking the ID's into the next ones (we won't give details about this, since it is known from literature that it can be done in any vectorial word-free algebra, built-up by closure under composition and branching of the usual de/constructors). In particular, we assume that  $\mathbf{nxt}\langle M \rangle$  leaves its input un-changed when  $M$  is in its final state.

The result follows by last lemma, since the behaviour of  $M$  is simulated by the  $m$ -ple  $\mathbf{sim}_M = \mathbf{nxt}\langle M \rangle_{kcl}$  for some  $l$ .

## 5 Simulation by TM's

**21** As mentioned in the proof of Th. 16, we now introduce a family of *interpreters*  $IN$  that, by input the code  $\mathbf{f}$  for any  $\mathbf{f} \in \mathcal{T}_\alpha$  ( $\alpha > 0$ ) and  $\mathbf{x}$ , return  $\mathbf{f}(\mathbf{x};)$ , within the promised amount of time. To this purpose, we adopt the following conventions.

(1) Given  $n$ ,  $IN$  is a TM with  $3n + l + 3$  semi-tapes over an alphabet of cardinality  $> 2 \max(n, c)$ , including: a letter  $\hat{\Sigma}$  for each one among the  $c$  letters  $\Sigma = U, \dots, T$  which may occur in the description of  $\mathbf{f}$ ; a digit  $d_i$  for each  $i < c^*$ ; and some *markers*  $\cdot, !, ?, *, \dots$

(2)  $\mathbf{X}$ ,  $\mathbf{Y}$  and  $\mathbf{Y}^*$  are tuples of tapes  $X_0, \dots, X_n, Y_1, \dots, Y_n, Y_1^*, \dots, Y_n^*$ . Some tapes will be used as *stacks*, whose *records*, if data, are separated by a dot. Since a word in a Polish suffix language  $L$  cannot be a suffix of any other word in  $L$ , stacks of function codes don't need separators. On the top of  $Y_i$  there is the current value of the sub-function  $g_i$  of  $\mathbf{f}$  being simulated.  $F$  contains a stack of (codes for) sub-funtions of  $\mathbf{f}$ . If  $\dot{g}$  is on the top of  $F$ , then it is executed and erased, or its sub-functions are un-nested and pushed into  $F$ .  $G, \dots, H$  are  $l$  scratch tapes. See 27 for the role of  $X_0, W$  and  $D$ .

(3) If  $A$  denotes a tape, then  $A u$  means that  $u$  is in the  $|u|$  cells at the left of the observed symbol. Expression  $\mathbf{A u}$  is short for  $A_1 u_1, \dots, A_n u_n$ . A *production* of the form

$$M : A u, B w, \dots \Rightarrow A u^*, B w^*, \dots \quad [T]$$

means that  $M$  takes a *tape configuration*  $A u, B w, \dots$  into a configuration  $A u^*, B w^*, \dots$

(a) without scanning the parts of  $A, B, \dots$  at the left of  $u, w, \dots$ ;

(b) within time  $T$  (one move per tape allowed in a single step).

We omit the tapes having no influence, and  $T$  when not yet estimated.

(4) When  $IN$  starts operating,  $\dot{\mathbf{f}}$  and  $\mathbf{x}$  are in the first cells of  $F$  and  $\mathbf{X}$ , while everything else is empty. When it stops operating  $\cdot\mathbf{f}(\mathbf{x};)$  is on the top of  $\mathbf{Y}$  (cf. Note 31).

**22 Parsing** In several constructions (for example in the implementation of prod. 1 and 2 of 23) we have to parse and copy (or erase) a word  $u$  in a Polish language  $L$ . This takes time  $3|u|$ , since each one of the following tasks requires time  $|u|$ : (a) copying  $u$  into  $G$ , by pushing meanwhile a  $d_i$  in  $H$  at each letter of arity  $i > 0$  (for example,  $d_n$  at each  $\dot{T}$  — cf. 12); (b) decreasing by 1 the top symbol in  $H$  at each arity 0 or popping the sequences of consecutive  $d_i$ 's; (c) copying back  $u$  from  $G$  when  $H$  is empty.

**23 An interpreter for  $\mathcal{T}_0$**  Define  $(\mathbf{U} = X_0, \dots, X_n, Y_1, \dots, Y_n)$

$IN_0 =$	<i>while</i>	the top symbol of $F$ belongs to the alphabet of $\mathcal{T}_0$	<i>do</i>	
	1	$F \mathbf{g} \mathbf{h} K$	$\Rightarrow$	$F \mathbf{g} ? \mathbf{h}$
	2	$F \dot{g}_0 \dot{g}_1 \dot{l} \dot{B}_a, U_l u$	$\Rightarrow$	$F \dot{g}_b$ $b = 0$ iff $u = w a$ ( $w \neq \varepsilon$ )
	3	$F \dot{g}, Y_i^*$	$\Rightarrow$	$F, Y_i^* \hat{g}$ $g = C^a, D, Q;$
	4	$F \dot{U}$	$\Rightarrow$	$F$
	<i>while</i>	$(F? \text{ or } F = \varepsilon)$	and	$\mathbf{Y}^* \neq \varepsilon$ <i>do</i>
	5	$F \cdot, Y_i^* \hat{C}^a, Y_i$	$\Rightarrow$	$Y_i^*, Y_i a$
	6	$F \cdot, Y_i^* \hat{Q}, Y_i$	$\Rightarrow$	$Y_i^*, Y_i \cdot 0$
	7	$F \cdot, Y_i^* \hat{D}, Y_i ua$	$\Rightarrow$	$Y_i^*, Y_i u$
	8	$F \cdot, Y_i^* \hat{D}, Y_i \cdot a$	$\Rightarrow$	$Y_i^*, Y_i \cdot 1 - a$
	<i>end while</i>	$F?$	$\Rightarrow$	$F$ <i>end while.</i>

**24 Comment** To rule conflicts between data protection and simultaneous processing, bookkeeping of operations to be performed on  $Y_i$  is held in  $Y_i^*$ . Since the complexity of erasing a whole record would interfere with next lemma, a new record  $\cdot 0$  is pushed into  $Y_i$  at each  $Q(U_i(\mathbf{x}))$ . This causes some *obsolete* information to be accumulated in  $\mathbf{Y}$  (inessentiality of this fact is discussed in Note 31).

**25 Lemma** For all  $\mathbf{f} \in \mathcal{T}_0$  we have  $(N = |\dot{\mathbf{f}}|)$

$$IN_0 : F \dot{\mathbf{f}}, \mathbf{X} \mathbf{x}, \mathbf{Y} \cdot \mathbf{y} \Rightarrow F, \mathbf{Y} \mathbf{v} \cdot \mathbf{f}(\mathbf{y}; \mathbf{x}) \quad [5N].$$

where  $v_i$  is absent if no  $Q_i$  has been applied, and conveys obsolete information otherwise.

*Proof.* Observe that (a) Note 22 applies to prod. 1 and 2; (b) prod. 3 requires time  $|\dot{g}|$  (when the *gap* 0 at the end of  $\dot{g}$  is reached, the machine *knows*  $g$ , and can write the single symbol  $\hat{g}$  in the course of its next step); (c) prod. 5-8 require one step for each symbol stored in  $\mathbf{X}^*$ . The result follows, since prod. 1 can be carried out during the simulation of  $\mathbf{h}$ , by using the scratch tape  $K$  (instead of  $H$  as suggested in 22) for its parsing stack.

## 26 The Interpreter Define

$IN =$	0	$\mathbf{X} \mathbf{x}, \mathbf{Y}$	$\Rightarrow$	$\mathbf{X} \mathbf{x}, \mathbf{Y} \cdot \mathbf{x}$
	<i>while</i>	$F$ not empty <i>do</i>		
	1	$F \dot{\mathbf{g}} \in \mathcal{T}_0$	$\Rightarrow$	$IN_0$
	2	$F \dot{\mathbf{g}} \dot{I}$ ,	$\Rightarrow$	$F \dot{\mathbf{g}} \dot{\mathbf{g}} \dot{R}_1$
	3	$F \dot{\mathbf{g}} \dot{R}_i, X_i au, X_0, W$	$\Rightarrow$	$F \dot{R} \dot{\mathbf{g}}^*, X_i au, X_0 \cdot a, W \cdot u$
	4	$F \dot{\mathbf{g}}^*, X_0 \cdot 0$	$\Rightarrow$	$F, X_0 \cdot 0$
	5	$F \dot{\mathbf{g}}^*, X_0 \cdot 1$	$\Rightarrow$	$F \dot{\mathbf{g}}, X_0 \cdot 1$
	6	$F \dot{\mathbf{h}} \dot{R}, W \cdot au, X_0$	$\Rightarrow$	$F \dot{\mathbf{h}} \dot{R} \dot{\mathbf{h}}, W \cdot u, X_0 a$
	7	$F \dot{\mathbf{h}} \dot{R}, W \cdot a, X_0$	$\Rightarrow$	$F ! \dot{\mathbf{h}}, W X_0 a$
	8	$F !, X_0 \cdot u$	$\Rightarrow$	$F, X_0$
	9	$F \dot{g} \dot{h} \Delta$	$\Rightarrow$	$F \dot{g} \hat{\Delta} \dot{h}$
	10	$F \dot{g} \hat{\Delta}, X_1 u, D, Y_1 \cdot w$	$\Rightarrow$	$F + \dot{g}, X_1 w, D u, Y_1$
	11	$F + X_1 w, D u, Y_1 \cdot z$	$\Rightarrow$	$F z, X_1 u, D, Y_1.$

**27 Comment** (1) All  $\mathbf{f} \in \mathcal{T}_c$  are simulated *on site*, in the sense that  $f_i$  is computed on tape  $Y_i$ , without any transfer to other tapes. This explains why the time estimate of next lemma is independent from  $|\mathbf{y}|$ .

(2) Tapes  $X_0$  and  $W$  contain two stacks of  $m$  current values for the principal variable of the sequence of  $m \leq c$  nested recursions being simulated. Recursions are simulated *from below*: for  $z = uaw$ , when  $\mathbf{f}(u, \dots)$  has been simulated we have  $aw$  on the top of  $W$ ,  $u$  on the top of  $X_0$ , and in next repetition  $a$  moves from  $W$  to  $X_0$ .

(3) Assume  $\mathbf{f} = \mathbf{g} \mathbf{h} \Delta$ .  $\dot{\mathbf{h}}$  is un-nested by prod. 9 and its value  $w$  is computed and pushed into  $Y_1$ . Since this  $w$  is the input for  $\mathbf{g}$ , prod. 10 saves  $x_1$  in tape  $D$ , and stores  $w$  in  $X_1$ . Finally, prod. 11 pushes the code  $z = g(w; ) = g(h(x_1; ))$  into  $F$ .

(4) Tapes  $X_0, W$  and  $\mathbf{Y}$  are used as stacks, while at most one record may be stored in the other tapes; presence/absence of  $\cdot$  in some of the productions depends on this.

**28 Lemma** For all  $\mathbf{f} \in \mathcal{T}_c$  ( $0 < c < \omega$ ), and for all  $\mathbf{x}, \mathbf{y}, u$  we have ( $N = |\dot{\mathbf{f}}|; l = \max(2, |\mathbf{x}|)$ )

$$IN : F \dot{\mathbf{f}}, \mathbf{X} \mathbf{x}, \mathbf{Y} \cdot \mathbf{y}, X_0 \cdot u \Rightarrow F, \mathbf{Y} \mathbf{v} \cdot \mathbf{f}(u, \mathbf{x}; \mathbf{y}) \quad [|\dot{\mathbf{f}}|^{2l^d}],$$

where  $u$  may be absent in  $\mathbf{e}$  and  $X_0$ , and  $\mathbf{v}$  is absent or conveys obsolete information.

*Proof.* Induction on  $c$ . Basis and step.  $IN$  produces the following configurations

1	$F \dot{\mathbf{h}} \dot{\mathbf{g}} \dot{R}_i, X_i 1u, X_0, W$	$\Rightarrow_{3,5}$	$F \dot{\mathbf{h}} \dot{R} \dot{\mathbf{g}}, X_0 \cdot 1, W \cdot u$
2	$F \dot{\mathbf{h}} \dot{R} \dot{\mathbf{g}}, X_0 \cdot 1, \mathbf{X} \mathbf{x}, \mathbf{Y} \cdot \mathbf{x}$	$\Rightarrow_{\text{I.H.}}$	$F \dot{\mathbf{h}} \dot{R} \dot{\mathbf{g}}, \mathbf{Y} \cdot \mathbf{g}(1, \mathbf{x}; \mathbf{x})$
3	$F \dot{\mathbf{h}} \dot{R}, W \cdot au, X_0 \cdot 1$	$\Rightarrow_6$	$F \dot{\mathbf{h}} \dot{R} \dot{\mathbf{h}}, W \cdot u, X_0 \cdot 1a$
4	$F \dot{\mathbf{h}} \dot{R} \dot{\mathbf{h}}, X_0 \cdot 1a, \mathbf{X} \mathbf{x}, \mathbf{Y} \cdot \mathbf{g}(1, \mathbf{x}; \mathbf{x})$	$\Rightarrow_{\text{I.H.}}$	$F \dot{\mathbf{h}} \dot{R}, \mathbf{Y} \cdot \mathbf{h}(1a, \mathbf{x}; \mathbf{g}(1, \mathbf{x}; \mathbf{x}))$
5	...	$\Rightarrow$	prod. 6 and I.H. for $ x_i  - 3$ times
6	$F \dot{\mathbf{h}} \dot{R}, W \cdot a, X_0 \cdot w$	$\Rightarrow_7$	$F ! \dot{\mathbf{h}}, W \cdot, X_0 \cdot x_i$
7	$F ! \dot{\mathbf{h}}, X_0 x_i, \mathbf{X} \mathbf{x}, \mathbf{Y} \cdot \hat{\mathbf{f}}(w, \mathbf{x}; )$	$\Rightarrow_{\text{I.H.}}$	$F !, \mathbf{Y} \cdot \mathbf{h}(x_i, \mathbf{x}; \hat{\mathbf{f}}(w, \mathbf{x}))$
8	$F !, X_0 \cdot x_i$	$\Rightarrow_8$	$F, X_0,$

where (a) it is assumed that the form of  $\mathbf{f}$  is  $\mathbf{h} \mathbf{g} R_i$ , ( $\mathbf{g}, \mathbf{h} \in \mathcal{T}_c$ ), and that  $x_i = 1u$  (since this case is worst than  $x_i = 0u$ );

(b) the numerical subscripts to  $\Rightarrow$  refer to the applied production; and subscript I.H. should be replaced by Lemma 25 for  $c = 1$ ;

(c) tapes that remain un-changed are omitted at the right-side of  $\Rightarrow$ .

*Complexity.* Prod. 3, 4, 5, and 8 are applied at most once in time  $T_1 \leq 3N + 3l$ . For  $|x_i| \leq l$  times we then apply prod. 6 and we call  $IN$  with  $\dot{\mathbf{g}}$  or  $\dot{\mathbf{h}}$  on the top of  $F$ ; by I.H. (or, for  $c = 1$ , by Lemma 25), since  $|\dot{\mathbf{h}}|, |\dot{\mathbf{g}}| \leq N - 5$ , each repetition takes time  $T_2 \leq 3(N - 5) + (N - 5)^2 l^d$ . By summing-up we obtain  $T_a + lT_b \leq 3N + 3l + l(3(N - 5) + (N - 5)^2 l^d) < N^2 l^{d+1}$ .

**29 Claim** Runtime for  $IN$  by input any  $\mathbf{f}(\mathbf{x};) \in \mathcal{T}_\omega$  is  $\leq l^N$  ( $N = |\dot{\mathbf{f}}|$ ;  $l = \max(2, |\mathbf{x}|)$ ). Indeed, by 12, we have  $|\dot{I}|, |\dot{R}| > 5$ , and, therefore,  $\mathbf{f} \in \mathcal{T}_{\lfloor N/5 \rfloor}$ . Our claim now follows since by last lemma we have  $N^2 l^{\lfloor N/5 \rfloor} \leq l^{\lfloor N/5 \rfloor + 2 \log(N)} \leq l^N$ .

**30 Lemma** For all  $\mathbf{f} \in \mathcal{T}_{\omega_{k+c}}$  we have ( $N = |\dot{\mathbf{f}}|$ ;  $l = \max(2, |\mathbf{x}|)$ )

$$IN : F \dot{\mathbf{f}}, \mathbf{X} \mathbf{x} \Rightarrow F \mathbf{Y} \cdot \mathbf{f}(\mathbf{x};) [l_k(N^{2(k+1)}l^c)].$$

*Proof.* Induction on  $k$ , with the basis ( $k = 0$ ) already proved by last lemma. Step. The form of  $\mathbf{f}$  is  $g h \Delta$ .  $IN$  produces the configurations listed below

$$\begin{array}{ll} 1 & F \dot{g} \dot{h} \Delta \quad \Rightarrow_9 \quad F \dot{g} \hat{\Delta} \dot{h} \\ 2 & F \dot{g} \hat{\Delta} \dot{h}, X_1 x_1, Y_1 \quad \Rightarrow_{I.H.} \quad F \dot{g} \hat{\Delta}, Y_1 \cdot h(x_1;) \\ 3 & F \dot{g} \hat{\Delta}, X_1 x_1, D, Y_1 \cdot h(x_1;) \quad \Rightarrow_{10} \quad F + \dot{g}, X_1 h(x_1;), D x_1, Y_1 \\ 4 & F + \dot{g}, X_1 h(x_1;), Y_1 \quad \Rightarrow_{I.H.} \quad F +, Y_1 \cdot g(h(x_1;);) \\ 5 & F +, X_1 h(x_1;), D \cdot x_1, Y_1 \cdot g(h(x_1;);) \quad \Rightarrow_{11} \quad F g(h(x_1;);), X_1 x_1, Y_1, D \\ 6 & F g(h(x_1;);), \mathbf{X} \mathbf{x} \quad \Rightarrow_{I.H.} \quad F, \mathbf{Y} \cdot \{g(h(x_1;);)\}(\mathbf{x}). \end{array}$$

*Complexity.* Define  $M = \max(|\dot{\mathbf{g}}|, |\dot{\mathbf{h}}|)$ . Time  $T_i$  ( $i = 1, \dots, 6$ ) for line  $i$  is given by:

$$\begin{array}{ll} T_1 = 3N & \text{by Note 22} \\ T_2 = l_k(M^{2(k+1)}l^c) & \text{by I.H. applied to } h, \text{ with } k \geq 0 \\ T_3 = 2T_2 & \text{since we may infix the mark } + \text{ while copying the value of } h(x_1;) \\ T_4 = M^2 T_2 & \text{by I.H. applied to } g, \text{ with } k = 0, c = 1, \text{ and } |x| < T_2 \\ T_5 = 2T_4 & \text{to transfer } g(h(x_1;);) \\ T_6 = l^{T_4} & \text{by Claim 29, since, by Def. 15, } \Delta \text{ cannot occur in } \{g(h(x_1;);)\}. \end{array}$$

By summing-up, since all other terms are neglectable with respect to  $T_6$ , we obtain a runtime bounded above by  $2T_6 = 2l^{M^2 T_2} \leq l^{M^2 l_k(M^{2(k+1)}l^c)+1} \leq l_{k+1}(N^2 M^2 M^{2(k+1)}l^c) \leq l_{k+1}(N^{2(k+2)}l^c)$ .

**31 Note** Requiring the result in  $\mathbf{X}$  and all other tapes empty would increase the complexity of  $IN$  by a factor 3, with respect to the time estimate  $T$  of Lemma 30. Indeed the length of the information stored in  $\mathbf{Y}$  to be erased or copied into  $\mathbf{X}$  cannot exceed  $T$ . We conclude that all  $\mathbf{f} \in \mathcal{T}_{\omega_{k+c}}$  can be simulated in time  $O(n_k(O(n^c)))$ .

## References

- [1] S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of the poly-time functions*, Computational Complexity 2(1992)97-110.
- [2] D. Leivant, *Ramified recurrence and computational complexity I: word recurrence and polytime*. In P. Clote and J. Remmel (eds), Feasible mathematics II, p. 320-343. (Birkhäuser, 1994).
- [3] D. Leivant, *A foundational delineation of computational feasibility*. Proc. of the 6th Annual IEEE symposium on Logic in computer science. (IEEE Computer Society Press, 1991).2-18.
- [4] D. Leivant, *Ramified recursion and computational complexity III: Higher type recurrence and elementary complexity*. Ann. p. and appl. Logic 96(1999) 209-229.
- [5] D. Leivant, *Calibrating computational feasibility by abstraction rank*. To appear.