ON TIERED SMALL JUMP OPERATORS

JEAN-YVES MARION

Nancy Université, Loria, INPL-ENSMN, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France, France.
e-mail address: Jean-Yves.Marion@loria.fr

ABSTRACT. Predicative analysis of recursion schema is a method to characterize complexity classes like the class FPTIME of polynomial time computable functions. This analysis comes from the works of Bellantoni and Cook, and Leivant by data tiering. Here, we refine predicative analysis by using a ramified Ackermann's construction of a non-primitive recursive function. We obtain a hierarchy of functions which characterizes exactly functions, which are computed in $O(n^k)$ time over register machine model of computation. For this, we introduce a strict ramification principle. Then, we show how to diagonalize in order to obtain an exponential function and to jump outside deterministic polynomial time. Lastly, we suggest a dependent typed lambda-calculus to represent this construction.

1. INTRODUCTION

Predicative analysis of recursion comes from the works of Bellantoni and Cook [3] and Leivant [11]. This analysis is based on a ramification principle on data which is appealing because its concept is simple and purely syntactic and does not involve parts of its models. Each element of a computation has a tier, which determines its ability to run a recursion. The ramification principle states that a definition by recursion is ramified only if the tier of the recurrence parameter is strictly higher than the tier of the output. This analysis takes its root in the paper of Simmons [17] and Leivant [10]. The results mentioned above characterize the class of polynomial time computable functions using essentially two tiers of data ramification: one for recursion arguments and one for recursion outputs. In this work, we revisit the ramification principle by introducing a *strict ramification principle* which allows getting a characterization of a polynomial time hierarchy of functions. Functions which are defined with k tiers are exactly functions which are computable in $O(n^k)$ steps. The hierarchy is not robust in the sense that it depends on the model of computation which is a register machine model here. So, the result that we suggest is really about intrinsic complexity of functions in the tradition of the recursion Theory. We have tried to understand the mechanism that underpins the suggested classification. Our analysis shows how functions are defined and how we can jump from one class of functions to another one by strict ramified recursion. This leads us to introduce a double recursion operator, which captures each level of the polynomial time hierarchy DTIME(n^k) and escapes them. For

this, we define an exponential function by a diagonalization method, which reveals some analogies with Ackermann [1] construction as it is explained in Chapter 7 of Simmons book [18]. The construction that we propose is a kind of double recursion whose main ideas can be explained by considering the following example.

$$f : \mathbb{N}(1), \mathbb{N}(0) \to \mathbb{N}(0) \qquad \begin{aligned} f(0, y) &= y + 1 \\ f(x + 1, y) &= f(x, f(x, y)) \end{aligned}$$

The function f is defined by nested recursion and satisfies the ramification principle. Indeed, the first argument may be of tier 1 and the second of tier 0. So, the output of f is of tier 0 and f is well typed. However f computes the exponential function : $f(n, m) = 2^n + m$ for all n and m. In $f(x, f(x, y))$, the leftmost occurrence of f calls itself which violates the essence of the ramification principle. Now, we ramify f by assigning to each occurrence of f a tier, and so we obtain the following function sequence.

$$f_0(x, y) = y + 1$$
$$f_{k+1}(0, y) = y$$
$$f_{k+1}(x + 1, y) = f_k(x, f_{k+1}(x, y))$$

where f_1 computes the addition, and f_2 iterates the addition, and so on. We also see that the domain, or the type, of each f_k can be $\mathbb{N}(k), \mathbb{N}(0) \to \mathbb{N}(0)$. If we transform the $(f_k)_{k \in \mathbb{N}}$ sequence of functions into a three place function $\phi(k, x, y)$, we are able to produce by a diagonalization argument a function which eventually dominates each f_k. The type of ϕ depends on its first argument and so would be $\forall k : \mathbb{N}(k), \mathbb{N}(0) \to \mathbb{N}(0)$.

This example is just here to illustrate quickly the ideas that we develop in this paper, which is organized as follows. Section 2 presents the computational models and defines DTIME(n^k). Section 3 focuses on tiered recursion and Leivant's characterization of FPTIME. This Section contains well-known material, and so the paper is self-contained. Section 4 gives the characterization of the polynomial time hierarchy. Section 5 describes how to jump from DTIME(n^k) to DTIME(n^{k+1}) and how to diagonalize in order to escape FPTIME. In the last section, an applied typed lambda-calculus, like in Simmons survey [16], with dependent types is proposed to represent the jump operator presented in the previous Section.

2. Computations and a polynomial time hierarchy

2.1. **Register Machines.** The set of binary words over the alphabet $\{a, b\}$ is $\mathbb{W}$. A register machine, abbreviated RM, works over words of $\mathbb{W}$. A RM consists in

(1) an alphabet $\{a, b\}$.
(2) a finite set $\mathcal{S} = \{s_0, s_1, \ldots, s_k\}$ of states, including a distinct state BEGIN.
(3) a finite list $\mathcal{R} = \{R_1, \ldots, R_m\}$ of registers. Registers store words of $\mathbb{W}$.

(4) a finite function LABEL mapping states to commands which are

$R = a(R)$	*add the letter a to R*
$R = b(R)$	*add the letter b to R*
$R = R'$	*assign the value of R' to R*
$R = \text{PRED}(R)$	*remove the first letter of R*
$\text{BRANCH}(R, s_\epsilon, s_a, s_b)$	*switch to the label s_i following the first letter of R*

A *configuration* of a RM M is given by a pair (s, σ) where s is a state and $\sigma : \mathcal{R} \to \mathbb{W}$ is an environment which stores register values. We guess that the above informal semantics should be enough to understand how register machines work. In particular, after executing one of the four first kinds of instruction, if the state is s_i and $i < k$, then the next state is s_{i+1}. Otherwise, if the state is s_k, then M halts. Lastly, the next step of a branching instruction depends on the value of the register R.

Throughout, we deal with functions which have a co-arity, that is function whose range is $\mathbb{W}^q$ for some q. A function $\phi : \mathbb{W}^p \to \mathbb{W}^q$ is computed by a register machine M if for all $u_1, \ldots, u_p$, $p \leq m$ we have $\phi(u_1, \ldots, u_p) = (v_1, \ldots, v_q)$ then the execution of M starting from the initial configuration (BEGIN, σ_0) ends to a configuration (s, σ_f) such that: for $i \leq p$, $\sigma_0(R_i) = u_i$, otherwise $\sigma_0(R_i) = \epsilon$ and for $j \leq q$, $\sigma_f(R_{m+1-j}) = v_j$.

2.2. **A polynomial time hierarchy.** The time measure corresponds to the number of steps to perform a computation on a register machine. We say that a function $\phi : \mathbb{W}^p \to \mathbb{W}^q$ is computable in $O(n^k)$ if the runtime is bounded by $c.(n_1^k + \ldots + n_p^k) + d$ for some c and d and where for each i, n_i is the size of the *ith* argument. The class $\text{DTIME}(n^k)$ is the set of all functions which are computable in $O(n^k)$. The class FPTIME of polynomial time functions is $\cup_k \text{DTIME}(n^k)$.

In this work, we study the classes $\text{DTIME}(n^k)$ which delineates a polynomial time hierarchy. It is well known that the class FPTIME is robust, which is not the case for polynomial time hierarchies. Indeed, the definition of $\text{DTIME}(n^k)$ is not invariant with respect to another class of computational models. The reason lies on the fact that the simulation of a computational model by another may have a quadratic cost. For example, the runtime of simulations of a two-tape Turing machine by a one-tape Turing machine is quadratic. Such lower bound may be nicely obtained using Kolmogorov complexity. The reader may consult Jones' book [9] for further informations. However, one may use k-tape Turing machines instead of register machines.

3. RAMIFIED PRIMITIVE RECURSION

3.1. **Functions on tiered domains.** We are interested in computational complexity, that is why we focus immediately on words. The domain of reference is the set $\mathbb{W}$ of words over the alphabet $\{a, b\}$. It is generated from the empty word function 0 and two successors A and B. As usual $A(B(0))$ is the word ab.

This domain is tiered by duplicating $\mathbb{W}$ into $\mathbb{W}(0), \mathbb{W}(1), \ldots, \mathbb{W}(i), \ldots$ where each $\mathbb{W}(i)$ is an identical copy of $\mathbb{W}$ at tier i. Each domain $\mathbb{W}(i)$ is a set of words over the alphabet $\{a_i, b_i\}$. As previously, there are an empty word function 0_i and two successors A_i and B_i.

In practice, we define functions by specifying their values with respect to tiered domain generators.

There are erasing bijections $\kappa_k : \mathbb{W}(k) \to \mathbb{W}$ for each k which just erase the tier of words. For example, we may represent a function $\phi : \mathbb{W} \to \mathbb{W}$ by $f : \mathbb{W}(k) \to \mathbb{W}(0)$ for some tier k if for each $u \in \mathbb{W}(k)$, $\kappa_0(f(u)) = \phi(\kappa_k(u))$. In this case, we shall just write $f(u) = \phi(u)$.

We always reason with respect to an implicit downcasting principle, which yields that if $u \in \mathbb{W}(k+1)$ then $u \in \mathbb{W}(k)$. Hence, we shall write that $f : \mathbb{W}(k+1) \to \mathbb{W}(0)$ is defined from $h : \mathbb{W}(k+1), \mathbb{W}(k) \to \mathbb{W}(0)$ by $f(x) = h(x, x)$ without mentioning that both occurrences of x are not of the same tier. *Throughout, we shall reason with respect to erasing bijections and implicit downcasting without explicitly mentioning them.*

We consider functions with co-arity. For this, we construct Cartesian product of domains of same tier. We abbreviate $\mathbb{W}(i) \times \ldots \times \mathbb{W}(i)$ by $\mathbb{W}(i)^p$. We have a pairing function $\langle \ , \ \rangle_i$ and both projections π_i^1 and π_i^2, for each tier i.

We often leave out some brackets using familiar conventions and hence we abbreviate $\tau_1 \to (\ldots (\tau_n \to \tau))$ by $\tau_1, \ldots, \tau_n \to \tau$. It is also convenient to have a normal presentation of functions, that we shall always use. We shall write $f : \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(r)^q$ for an n-placed function in such a way that $i_1 \geq \ldots \geq i_n \geq r$. We say that the tier of the jth argument of f is i_j, and the output tier is r. We write $\vec{y}$ to mean $y_1, \ldots, y_n$ where y_i is an element of $\mathbb{W}(i_j)^{p_j}$. The size $|u|$ is the number of letters of the word u. In particular the size of the empty word ϵ is 0. The size of pair of words is inductively defined as follows: $|\langle u, v \rangle_i| = |u| + |v|$ at any tier i.

Conventions that we have described here will be extended to the typed lambda calculus that we suggest at the end in a natural manner.

3.2. Ramified primitive recursion.
A function $f : \mathbb{W}(k+1), \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(r)^q$ is obtained by *ramified primitive recursion* from the functions
$h_\epsilon : \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(r)^q$ and
$h_a, h_b : \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n}, \mathbb{W}(r)^q \to \mathbb{W}(r)^q$ if

$$f(0_{k+1}, \vec{y}) = h_\epsilon(\vec{y}) \tag{3.1}$$

$$f(A_{k+1}(x), \vec{y}) = h_a(\vec{y}, f(x, \vec{y})) \tag{3.2}$$

$$f(B_{k+1}(x), \vec{y}) = h_b(\vec{y}, f(x, \vec{y})) \tag{3.3}$$

where conditions $k+1 \geq i_j$ for any $j \leq n$ and $k \geq r$ hold. We call these last conditions the *ramification principle* based on Leivant's [11]. The first argument is named the recursion argument and its tier is $k+1$. The ramification principle says that the recurrence tier $k+1$ is strictly greater than the output tier r.

3.3. Ramified arithmetic.
In order to compare function growth rate and to illustrate key notions, it is convenient to have an encoding of natural numbers. This encoding will be used in Sections 4.1 and 5.1.

We represent natural numbers by considering both successors A_i and B_i as the same. Hence, we have a single successor that we write S_i, for each tier i. It should be clear that this encoding is non-injective, which is sufficient because we are just interested in the size of the handling values. So in this representation, a word represents a natural number, which

corresponds to its size. Hence, 0_i will refer to zero at tier i, and $S_i(x)$ intuitively increases the size of x by one, which corresponds exactly to the successor operation in unary notation.

We represent in *ramified arithmetic* an arithmetical function $\phi : \mathbb{N}^p \to \mathbb{N}$ by a function $f : \mathbb{W}(i_1), \ldots, \mathbb{W}(i_p) \to \mathbb{W}(r)$ if

$$\phi(n_1, \ldots, n_p) = |f(u_1, \ldots, u_p)| \qquad \text{for each } u_i \text{ such that } |u_i| = n_i \text{ and } i = 1, p$$

Now, we can define below the addition add_k and the multiplication mul_k at tier k.

$add_k : \mathbb{W}(k+1), \mathbb{W}(k) \to \mathbb{W}(k)$ and $|add_k(u,v)| = |u| + |v|$, for all u and v.

$$add_k(0_{k+1}, y) = y$$
$$add_k(S_{k+1}(x), y) = S_k(add_k(x,y)) \qquad\qquad \text{where } S_i = A_i, B_i$$

$mul_k : \mathbb{W}(k+1), \mathbb{W}(k+1) \to \mathbb{W}(k)$ and $|mul_k(u,v)| = |u|.|v|$, for all u and v

$$mul_k(0_{k+1}, y) = 0_k$$
$$mul_k(S_{k+1}(x), y) = add_k(y, mul_k(x,y))$$

Observe that both arguments of mul_k have the same tier $k+1$. We may define polynomials by composition from tiered addition and multiplication, as it is illustrated below.

$cube_k : \mathbb{W}(k+2) \to \mathbb{W}(k)$

$$cube_k(x) = mul_k(x, mul_{k+1}(x,x))$$

We see that we compute the arithmetical function x^3 by composing two multiplications. However, two copies of the multiplication mul_k and mul_{k+1} at different tiers are necessary. Notice also that the tier of the first argument, on the right handside, is lower, which is possible because of the use of a downcasting bijection. Actually, we may define $coerce_k$ by a simple ramified recursion. We may then use it instead of the implicit downcasting, $coerce_k : \mathbb{W}(k+1) \to \mathbb{W}(k)$

$$coerce_k(0_{k+1}) = 0_k$$
$$coerce_k(S_{k+1}(x)) = S_k(coerce_k(x))$$

On the other hand, the ramified principle allows also to define a cubic function $cube'_k : \mathbb{W}(k+1)^3 \to \mathbb{W}(k)$ using only two tiers as follows:

$$mul'_k(0_{k+1}, z, t) = t$$
$$mul'_k(S_{k+1}(y), z, t) = add_k(z, mul'_k(y,z,t))$$
$$cube'_k(0_{k+1}, y, z) = 0_{k+1}$$
$$cube'_k(S_{k+1}(x), y, z) = mul'_k(y, z, cube'_k(x,y,z))$$

3.4. Characterization of FPTIME.
In 1994, Leivant published an elegant characterization [11] of FPTIME, which provides a general framework to study complexity classes. We follow here the main line of his work. So, we begin by introducing a particular kind of recursion, named flat recursion.

A function $f : \mathbb{W}(r), \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(r)$ is obtained by *flat recursion* from the functions

$h_\epsilon : \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(r)$ and
$h_a, h_b : \mathbb{W}(r), \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(r)$ if

$$f(0, \vec{y}) = h_\epsilon(\vec{y}) \tag{3.4}$$

$$f(A_r(x), \vec{y}) = h_a(x, \vec{y}) \tag{3.5}$$

$$f(B_r(x), \vec{y}) = h_b(x, \vec{y}) \tag{3.6}$$

This kind of recursion should be viewed as a mere action on the pattern of the recursive argument. Hence and unlike the ramified principle, the tier of a recurrence argument is not strictly higher that the output tier. The use of flat recursion is essential to define a predecessor over $\mathbb{W}$ and conditional functions.

Definition 3.1. A function f is in $\mathcal{L}_\omega(\mathbb{W})$ if it is obtained by a finite number of applications of composition, flat recursion and ramified primitive recursion beginning with basic functions 0_k, A_k, B_k, $\langle \_, \_ \rangle_k$, π_k^1 and π_k^2 for each tier k.

Leivant demonstrated in [11] the following result:

Theorem 3.2. *The class of functions $\mathcal{L}_\omega(\mathbb{W})$ is exactly the class FPTIME of the functions which are polynomial time computable.*

In this presentation we use functions with co-arity, unlike Leivant which introduces simultaneous ramified recursion.

Actually, Leivant also showed that only two tiers are sufficient. More generally,

Corollary 3.3. *Let $\mathcal{L}_k(\mathbb{W})$ be the class of functions restricted over $\mathbb{W}(0), \ldots, \mathbb{W}(k)$. For each k, the class of functions $\mathcal{L}_{k+1}(\mathbb{W})$ is exactly the class FPTIME of the functions which are polynomial time computable.*

In the same paper, Leivant shows how to capture $\text{DTIME}(n^k)$ by counting the degree of nested recursions.

3.5. **Other approaches.** The work of Bellantoni and Cook [3] is similar to the Leivant's one. They characterize FPTIME by defining a function algebra in which functions have two kind of arguments: the normal ones which can be used as recursion parameters and the safe ones which cannot be used as recursion parameters.

As we have seen, only two tiers are necessary to characterize FPTIME. Actually, this is also the essence of the characterization by simply typed lambda calculus of [13]. The tier 1 arguments are represented by Church-numerals, and the tier 0 are represented by constant terms of atomic type on which no recursion can be made.

4. STRICT RAMIFIED PRIMITIVE RECURSION

We present the notion *strict ramified primitive recursion* which is central in this study. A function $f : \mathbb{W}(k), \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(0)^q$ is obtained by *k-ramified recursion* from the functions

$h_\epsilon : \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n} \to \mathbb{W}(0)^q$ and
$h_a, h_b : \mathbb{W}(i_1)^{p_1}, \ldots, \mathbb{W}(i_n)^{p_n}, \mathbb{W}(0)^q \to \mathbb{W}(0)^q$ if

$$f(0_k, \vec{y}) = h_\epsilon(\vec{y}) \tag{4.1}$$

$$f(A_k(x), \vec{y}) = h_a(\vec{y}, f(x, \vec{y})) \tag{4.2}$$

$$f(B_k(x), \vec{y}) = h_b(\vec{y}, f(x, \vec{y})) \tag{4.3}$$

where the inequalities between tiers $k > i_j$ for each $j \le n$ and $k > 0$ hold. We call this last condition *the strict ramification principle*.

Definition 4.1. A function f is in $\mathcal{I}_k(\mathbb{W})$ if it is obtained by a finite number of applications of composition, flat recursion and i-ramified recursion, beginning with basic functions 0_i, A_i, B_i, $\langle \ , \ \rangle_i$, π_i^1 and π_i^2 for each tier $i \le k$.

In particular, a function $\mathcal{I}_0(\mathbb{W})$ is not defined by recursion. The notion of 1-ramified recursion was underlying in [14], and the notion of k-ramification is used in order to characterize the NC^k hierarchy in [5].

The difference between the ramification principle and the strict ramification principle is the following:

(1) The recursion argument is *strictly* greater than the other argument tiers,
(2) and the output tier is 0.

Otherwise, we could define the function x^5 with $0, \ldots, 4$ tiers, that is by 4-ramified recursion and composition as follows:

$m_k : \mathbb{W}(k+2), \mathbb{W}(k+1) \to \mathbb{W}(k)$

$$m_k(0_{k+2}, y) = 0_k$$
$$m_k(S_{k+2}(x), y) = add_k(y, m_k(x, y))$$

$five : \mathbb{W}(4) \to \mathbb{W}(0)$ and $|five(x)| = |x|^5$

$$five(x) = m_0(m_2(x, x), m_1(x, m_2(x, x)))$$

The fact that the output tier of an recursion is 0 implies that we cannot defined $coerce_k$ functions. That is why we need to reason modulo downcasting bijections.

4.1. **Strict ramified arithmetic.** We use the same encoding of natural numbers that the one we present in Section 3.3 on ramified arithmetic. However, we slightly modify the way that we represent arithmetical functions to take into account the fact that outputs are of tier 0.

An arithmetical function $\phi : \mathbb{N}^p \to \mathbb{N}$ is represented in *strict ramified arithmetic* by a function $f : \mathbb{W}(i_1), \ldots, \mathbb{W}(i_p) \to \mathbb{W}(0)$ if

$$\phi(n_1, \ldots, n_p) = |f(u_1, \ldots, u_p)| \qquad \text{for each } u_i \text{ such that } |u_i| = n_i, \, i = 1, p$$

The addition function defined in Section 3.3 is defined by 1-ramified recursion, setting $k = 0$. On the other hand, the definition of the multiplication proposed in 3.3 does not satisfy the strict ramification principle because both arguments are of the same tier.

Nevertheless, we can define any polynomial. For this, we present first a sequence $(F_k)_{k \in \mathbb{N}}$ of 3-placed monotonic functions from an initial 1-placed function $g : \mathbb{W}(0)^p \to \mathbb{W}(0)^p$. Intuitively, the function g is iterated a number of steps bounded by a polynomial of degree k. This sequence will play a *crucial role* all along the paper.

$$F_0 : \mathbb{W}(0), \mathbb{W}(0), \mathbb{W}(0)^p \to \mathbb{W}(0)^p$$

$$F_0(t, x, y) = g(y)$$

$$F_{k+1} : \mathbb{W}(k+1), \mathbb{W}(k), \mathbb{W}(0)^p \to \mathbb{W}(0)^p$$

$$F_{k+1}(0_{k+1}, x, y) = y$$
$$F_{k+1}(S_{k+1}(t), x, y) = F_k(x, x, F_{k+1}(t, x, y))$$

It is worth noticing that $(F_k)_{k \in \mathbb{N}}$ is parameterized by the function g. Notice that we use implicitly a downcasting to lower the tier of the second argument on the right hand side of the last equation.

Lemma 4.2. *For any k,u,v and w, we have*

$$F_{k+1}(u, v, w) = g^{m.n^k}(w) \qquad\qquad where\ m = |u|\ and\ n = |v|$$

Proof. The proof is by induction on k.

For $k = 0$, we have by recurrence on the size of the first argument :

$$F_1(0_1, v, w) = w$$
$$F_1(S_1(t), v, w) = F_0(v, v, F_1(t, v, w))$$
$$= g(g^{|t|}(w)) = g^{|t|+1}(w)$$

For $k > 0$, we have again by recurrence on the size of the first argument :

$$F_{k+1}(0_{k+1}, v, w) = w$$
$$F_{k+1}(S_{k+1}(t), v, w) = F_k(v, v, F_{k+1}(t, v, w))$$
$$= g^{n^k}(F_{k+1}(t, v, w)) \qquad\qquad \text{by recurrence on } k$$
$$= g^{n^k}(g^{|t| \times n^k}(w)) \qquad\qquad \text{by recurrence on } t$$
$$= g^{(|t|+1)n^k}(w) \qquad\qquad\qquad \square$$

The sequence of functions $(F_k)_k$ allows us to define polynomial length iterators over $\mathbb{W}(0)$.

Lemma 4.3. *Let $P[X]$ be a polynomial of degree k with natural coefficients and $g : \mathbb{W}(0)^p \to \mathbb{W}(0)^p$. There is a function $\tilde{P} : \mathbb{W}(k), \mathbb{W}(0)^p \to \mathbb{W}(0)^p$ in $\mathcal{I}_k(\mathbb{W})$ such that for each x and y,*

$$\tilde{P}(x, y) = g^{P(|x|)}(y) \qquad\qquad\qquad (4.4)$$

Proof. The proof is done by induction on the degree of the polynomial. The base case is trivial. Suppose that the degree of P is $k + 1$. Hence, $P(x) = c.x^{k+1} + Q(x)$ where the degree of Q is less or equal to k. Suppose that $\tilde{Q}$ satisfies the induction hypothesis wrt Q. We define T_{k+1}^c by composition as follows

$$T_{k+1}^0(x, y) = \tilde{Q}(x, y)$$
$$T_{k+1}^{d+1}(x, y) = F_{k+1}(x, x, T_{k+1}^d(x, y)) \qquad\qquad d < c$$

We set $\tilde{P}(x, y) = T_{k+1}^c(x, y)$. Here $T_{k+1}^c(x, y)$ is defined by c compositions of F_{k+1} where c is given and fixed.

We show by an induction on c that $\tilde{P}(x,y)$ satisfies 4.4. We just show the inductive step below.

$$\tilde{P}(x,y) = T_{k+1}^{d+1}(x,y) = F_k(x,x,T_{k+1}^d(x,y)) \qquad \text{by dfn}$$
$$= F_k(x,x,g^{d.n^{k+1}+Q(n)}(y)) \qquad \text{where } n = |x|$$
$$= g^{n.n^k}(g^{d.n^{k+1}+Q(n)}(y)) \qquad \text{by Lemma 4.2} = g^{(d+1).n^{k+1}+Q(n)}(y) \quad \square$$

Lemma 4.4. *Any polynomial $P[X]$ with natural coefficients is represented in strict ramified arithmetics.*

Proof. We set $\overline{P}(x) = \tilde{P}(x,0_0)$ in which we replace g by the successor A_0. So, we have $\overline{P}(x) = A_0^{P(x)}(0_0)$. $\qquad \square$

We say that a multivariate polynomial $P[X_1,\ldots,X_n]$ with n distinct variables is simple if each monomial of P is of the form $c.X_i^d$ for some natural constants c and d. For example $2x^2 + 3.y^2 + 4y$ is simple, but $2yx^2 + y$ is not. The degree of a simple polynomial is the greatest exponent of P's variables.

Lemma 4.5. *Let $P[X_1,\ldots,X_n]$ be a simple polynomial of degree k and let $g : \mathbb{W}(0)^p \to \mathbb{W}(0)^p$. There is a function $\tilde{P} : \mathbb{W}(k)^n, \mathbb{W}(0)^p \to \mathbb{W}(0)^p$ in $\mathcal{I}_k(\mathbb{W})$ such that for each $x_1,\ldots,x_n$ and y,*

$$\tilde{P}(x_1,\ldots,x_n,y) = g^{P(|x_1|,\ldots,|x_n|)}(y) \qquad (4.5)$$

Proof. The proof is done by induction on the number n of variables. The base case is a consequence of Lemma 4.3. Suppose that the simple polynomial P has $n+1$ variables $X_1,\ldots,X_n,X_{n+1}$. Since P is simple, we write it as the sum $P(X_1,\ldots,X_n,X_{n+1}) = P'(X_1,\ldots,X_n) + P''(X_{n+1})$. Suppose that $\tilde{P}'$ ($\tilde{P}''$) satisfies the induction hypothesis wrt P' (resp. P''). We define $\tilde{P}$ by

$$\tilde{P}(x_1,\ldots,x_{n+1},y) = \tilde{P}'(x_1,\ldots,x_n,\tilde{P}''(x_{n+1},y))$$

Indeed, we have

$$\tilde{P}(x_1,\ldots,x_{n+1},y) = \tilde{P}'(x_1,\ldots,x_n,g^{P''(|x_{n+1}|)}(y)) = g^{P'(|x_1|,\ldots,|x_n|)}(g^{P''(|x_{n+1}|)}(y))$$
$$= g^{P'(|x_1|,\ldots,|x_n|)+P''(|x_{n+1}|)}(y) = g^{P(|x_1|,\ldots,|x_{n+1}|)}(y) \qquad \square$$

4.2. Characterizing a polynomial time hierarchy.

Theorem 4.6. *The set of functions $\mathcal{I}_k(\mathbb{W})$ is exactly $DTIME(n^k)$.*
That is $\mathcal{I}_k(\mathbb{W}) = DTIME(n^k)$.

The demonstration of Theorem 4.6 is a consequence of Lemma 4.7 and 4.8 below.

Lemma 4.7. *Let $\phi : \mathbb{W}^p \to \mathbb{W}^q$ be a function which is computable by a register machine M in time $(c.\sum_{i=1,p} n_i^k) + d$ for some constants c,d and k, where n_i is the size of the ith argument. Then, there is a function $f : \mathbb{W}(k)^p \to \mathbb{W}(0)^q$ of $\mathcal{I}_k(\mathbb{W})$ such that for each u, $f(u) = \phi(u)$.*

Proof. A configuration of M is encoded by a $m + 1$-uplet of $\mathbb{W}(0)$ which represents the state and the value of the m registers of M. Then, it is not difficult to design a function $next : \mathbb{W}(0)^{m+1} \to \mathbb{W}(0)^{m+1}$, which given a configuration, produces the next configuration wrt M. The function $next$ is based on nested flat recursions over $\mathbb{W}(0)$. To illustrate the construction of $next$, consider that the register machine M has just two registers R_1 and R_2. We define the function $next$ for each state of M by using flat recursion in order to match a state and to switch to the right transition. The next configuration depends on the finite function LABEL of M. For example if in state s_i, the value of R_2 is replaced by the value of R_1, and the next state is s_j, we define $next$ by flat recursion such that $next(s_i, R_1, R_2) = (s_j, R_1, R_1)$.

Now, we have to iterate $next$ within the polynomial time bound. For this we use Lemma 4.5 since it is a simple polynomial.

Therefore, there is a function $loop : \mathbb{W}(k)^p, \mathbb{W}(0)^{m+1} \to \mathbb{W}(0)^{m+1}$ such that

$$loop(x_1, \ldots, x_p, \vec{y}) = next^{(c. \sum_{i=1,p} |x_i|^k) + d}(\vec{y}) \ .$$

We conclude by taking $f(x_1, \ldots, x_p) = \theta(loop(x_1, \ldots, x_p, init))$, where θ is a composition of projections and $init$ is the initial configuration, that is $init = (\text{BEGIN}, x_1, \ldots, x_p, 0_0, \ldots, 0_0)$. ☐

Lemma 4.8. *Assume that* $f : \mathbb{W}(k_1)^{p_1}, \ldots, \mathbb{W}(k_n)^{p_n} \to \mathbb{W}(r)^q$ *is in* $\mathcal{I}_k(\mathbb{W})$. *Then there is a polynomial* P *of degree* k, *or less, such that for any* $u_1, \ldots, u_n$, *the computation of* $f(u_1, \ldots, u_n)$, *on register machines, is performed in time bounded by*

$$P(\max\{|u_i| \,|\, where\ the\ tier\ of\ u_i\ is\ greater\ than\ 0,\ that\ is\ k_i > 0\}_{i=1,n})$$

Proof. The proof goes by induction on k. Suppose that $f \in \mathcal{I}_0(\mathbb{W})$. In this case, the definition of $\mathcal{I}_0(\mathbb{W})$ claims that f is not defined by strict ramified recursion. Hence, it is not hard to compute f in constant time.

Now, suppose that $f \in \mathcal{I}_{k+1}(\mathbb{W})$. There are two main cases that we are considering below.

First, f is obtained by $k + 1$-ramified recursion. We compute a loop whose length is bounded by the length of the first argument u_1. We begin by evaluating $v_0 = h_\epsilon(u_2, \ldots, u_n)$. Next we compute $h_\alpha(u_2, \ldots, u_n, v_0)$ where α is the last letter of u_1. And, we repeat this process till we have consumed all letters of the recursion argument u_1. As usual with tiering system, the key point is that the runtime of the auxiliary functions h_a and h_b does not depend on tier 0 values. Hence we associate three polynomials P_ϵ, P_a and P_b satisfying the induction hypothesis. The runtime of f is bounded by $P_\epsilon(\max\{|u_i| \,|\, \text{where } k_i > 0\})) + |u_1| \times \max_{\alpha=a,b}(P_\alpha(\max\{|u_i| \,|\, \text{where } k_i > 0\}))$. Since h_ϵ, h_a, and h_b have domains which have strictly lower tiers than $k + 1$, it follows that degrees of the corresponding polynomials, P_ϵ, P_a and P_b are at most k by induction hypothesis. As a consequence, there is a polynomial which bounds $P_\epsilon(X) + X. \max_{\alpha=a,b}(P_\alpha(X))$ of degree at most $k + 1$. This polynomial is an upper bound on f's runtime.

Second, f is defined by composition. Say that $f(\vec{x}) = h(\vec{x}, g(\vec{x}))$. There are two cases to consider. The first is when the output tier of g is 0. In this case, the runtime of f is bounded by the sum of the runtime of g and h. The second is when the output tier of g is strictly greater than 0. Then, the runtime of g is constant because g cannot be defined by recursion. It follows that the runtime of f is bounded by the runtime of h plus an additive constant (due to g). ☐

5. Diagonalization with dependent tiers

In this section, we consider again the sequence $(F_k)_k$ parameterized by a strictly increasing function g. Recall that, F_k iterates n^k times a function g and is in $\mathcal{I}_k(\mathbb{W})$. Each function of $\mathcal{I}_k(\mathbb{W})$ is eventually dominated by composition of F_k at tier 0. But, F_k is not in $\mathcal{I}_{k+1}(\mathbb{W})$. This leads us to ask two questions: How to jump from $\mathcal{I}_k(\mathbb{W})$ to $\mathcal{I}_{k+1}(\mathbb{W})$? And how to jump outside $\cup_k \mathcal{I}_k(\mathbb{W})$? In other words, this leads us to investigate jump operators, which allows to define $(F_k)_k$ sequence of functions by iteration and to diagonalize it in order to compute a function, which is not in $\cup_k \mathcal{I}_k(\mathbb{W})$.

5.1. **Jumping from $\mathcal{I}_\mathbf{k}(\mathbb{W})$ to $\mathcal{I}_{k+1}(\mathbb{W})$.** In order to answer to the first question, we introduce an operator $\Delta[\_]$ such that for each k, $\Delta[F_k] : \mathbb{W}(k+1), \mathbb{W}(k), \mathbb{W}(0) \to \mathbb{W}(0)$ and

$$\Delta[F_k](0_{k+1}, x, y) = y$$
$$\Delta[F_k](S_{k+1}(r), x, y) = F_k(x, x, \Delta[F_k](r, x, y))$$

From definitions, it is clear that $\Delta[F_k](r, x, y) = F_{k+1}(r, x, y)$. Observe also, that the operator $\Delta[\_]$ respects the strict ramification principle.

Definition 5.1. Let $h : \mathbb{W}(k_1)^{p_1}, \ldots, \mathbb{W}(k_n)^{p_n} \to \mathbb{W}(r)^q$ be an n-placed function and let $f : \mathbb{W}(k) \to \mathbb{W}(0)$ be a 1-placed function. We say that h is dominated by f if $|h(x_1, \ldots, x_n)| \leq |f(x)|$ holds for all $x_1, \ldots, x_n$ and x with $x_1 \leq x, \ldots, x_n \leq x$.

Proposition 5.2. *Each function h of $\mathcal{I}_k(\mathbb{W})$ is dominated by $f_k(x) = \Delta[F_k](a, x, b)$ for some $a \in \mathbb{W}(k+1)$ and $b \in \mathbb{W}(0)$.*

Proof. Since $\mathcal{I}_\mathbf{k}(\mathbb{W}) = \mathrm{DTIME}(n^k)$, there is a' and b' such that for all $x_1, \ldots, x_n$,

$$|h(x_1, \ldots, x_n)| \leq a'(\sum |x_i|^k) + b'$$

Let $a = A_{k+1}^{a'}(0_{k+1})$ and $b = A_0^{b'}(0_0)$ be two words such that $|a| = a'$ and $|b| = b'$. It follows that for all $x_1, \ldots, x_n$ and x with $x_1 \leq x, \ldots, x_n \leq x$, we have $|h(x_1, \ldots, x_n)| \leq \Delta[F_k](a, x, b)$. Indeed,

$$
\begin{aligned}
|h(x_1, \ldots, x_n)| &\leq |g^{|a| \cdot |x|^k}(b)| && \text{since } g \text{ is assumed strictly monotonic} \\
&\leq |F_{k+1}(a, x, b)| && \text{by Lemma 4.2} \\
&= |\Delta[F_k](a, x, b)| && \text{by dfn}
\end{aligned}
$$

$\square$

But, the important point here is that an operator like $\Delta[\_]$ allows to escape $\mathcal{I}_k(\mathbb{W})$ because $\Delta[F_k] = F_{k+1}$ is not in $\mathcal{I}_k(\mathbb{W})$. We now iterate $\Delta[\_]$ starting from F_0 in order to produce the chain of monotonic F_k functions, as follows :

$$\Delta^0[F_0](r, x, y) = F_0(r, x, y)$$
$$\Delta^{k+1}[F_0](r, x, y) = \Delta[\Delta^k[F_0]](r, x, y)$$

We say that the *kth* iterate of F_0 is $\Delta^k[F_0]$.

Proposition 5.3. *For all $k \in \mathbb{N}$, $r \in \mathbf{W}(k)$, $x \in \mathbf{W}(k)$ and $y \in \mathbf{W}(0)$, we have*

$$\Delta^k[F_0](r, x, y) = F_k(r, x, y)$$

Proof. The proof goes by induction on k. The base case is immediate. Next,

$$\begin{aligned}
\Delta^{k+1}[F_0](r,x,y) &= \Delta[\Delta^k[F_0]](r,x,y) && \text{by dfn}\\
&= \Delta[F_k](r,x,y) && \text{Ind. Hyp.}\\
&= F_{k+1}(r,x,y) && \text{by dfn}
\end{aligned}$$

$\square$

Therefore, the $k+1th$ iterate of F_0 is in $\mathcal{I}_{k+1}(\mathbb{W})$ but not in $\mathcal{I}_k(\mathbb{W})$.

Remark 5.4. The jump operator $\Delta[\_]$ can be applied to any function of type

$$\mathbb{W}(k), \mathbb{W}(\max(k-1,0)), \mathbb{W}(0) \to \mathbb{W}(0) \ .$$

5.2. **Jumping outside $\cup_k \mathcal{I}_k(\mathbb{W})$.**
We define next a 4-placed operator Δ^ω based on a double recursion. It is a nested recursion based on lexicographic ordering.

$$\begin{aligned}
\Delta^\omega[g](0,r,x,y) &= g(y) && g : \mathbb{W}(0) \to \mathbb{W}(0)\\
\Delta^\omega[g](k+1,0_{k+1},x,y) &= y\\
\Delta^\omega[g](k+1,S_{k+1}(r),x,y) &= \Delta^\omega[g](k,x,x,\Delta^\omega[g](k+1,r,x,y))
\end{aligned}$$

Here, Δ^ω is parameterized by g.

Proposition 5.5. *For all k,r,x and y, we have*

$$\Delta^\omega[g](k,r,x,y) = \Delta^k[F_0](r,x,y)$$

Proof. By induction on k and r. $\square$

If we fix the first argument k, we iterate on the second argument r of tier k and we compute F_k. Now, if we fix the second argument r, we jump from tier to tier which allows to get outside each function set $\mathcal{I}_k(\mathbb{W})$, computing the successive iterate of F_0. So, Δ^ω allows us to jump outside each $\mathcal{I}_k(\mathbb{W})$ for any k.

Proposition 5.6. *The 4 placed function $\Delta^\omega[A_0]$ is not in $\cup_{k\in\mathbb{N}}\mathcal{I}_k(\mathbb{W})$.*

Proof. We set $\phi(x) = \Delta^\omega[A_0](|x|,x,x,x)$ for all x. We have

$$\begin{aligned}
\phi(x) &= \Delta^\omega[A_0](|x|,x,x,x)\\
&= \Delta^{|x|}[F_0](x,x,x) && \text{by Prop. 5.5 where } g = A_0\\
&= F_{|x|}(x,x,x) && \text{by Prop. 5.3}\\
&= A_0^{|x|^{|x|}}(x) && \text{by Prop. 4.2 when } |x| > 0
\end{aligned}$$

We see that $|\phi(x)| = |x|^{|x|+1} + |x|$ which is clearly not in $\cup_{k\in\mathbb{N}}\mathcal{I}_k(\mathbb{W})$ in which each function is polynomially bounded as it has been established in Theorem 4.6. $\square$

The operator Δ^ω produces a function, which is not in $\cup_{k\in\mathbb{N}}\mathcal{I}_k(\mathbb{W})$. That is, $\Delta^\omega[g]$ is not a ramified function in $\cup_{k\in\mathbb{N}}\mathcal{I}_k(\mathbb{W})$ if g is increasing. However, we may see that intuitively the "domain" depends on the first argument, and so we should write $\Delta^\omega[g]$: $\forall k \in \mathbb{N}.\mathbb{W}(k), \mathbb{W}(k), \mathbb{W}(0) \to \mathbb{W}(0)$. To formalize this idea, we now introduce a typed lambda-calculus with very restricted dependent types and arithmetical gadgets.

6. AN APPLIED LAMBDA-CALCULUS WITH DEPENDENT TYPES

6.1. **Types, terms, and rules.** We propose a typed λ-calculus $\boldsymbol{\lambda}\mathbf{D}^{\boldsymbol{\omega}}$ in which types depend on tiers. For this, we have a base type ω to denote tiers and a unary predicate $\mathbf{W}$ of kind $\omega \Rightarrow *$, which is intended to name words at each tier.

Raw expressions, Kinds, types and terms, are defined following the grammar rules :

(Type constructors)	α	$::=$	$\omega \mid \mathbf{W}$
(Kinds)	κ	$::=$	$* \mid \tau \Rightarrow \kappa$
(Types)	τ	$::=$	$\alpha \mid \tau \times \tau \mid \forall x.\tau \mid \tau\, M$
(Term constructors)	$\mathbf{c}$	$::=$	$\diamond \mid \mathbf{S} \mid \mathbf{0} \mid \mathbf{A} \mid \mathbf{B} \mid \langle \_,\_\rangle\_ \mid \pi^1 \mid \pi^2 \mid \mathbf{flat} \mid \mathbf{diag}$
(Terms)	M	$::=$	$\mathbf{c} \mid x \mid (MM) \mid \lambda x.M$

where x is a variable.

The types assigned to type and term constructors are given in Figure 1. We may omit some brackets of a type or of a term using familiar Currying conventions.

A term M is of type τ, that we write $M : \tau$, if there is a derivation of $\vdash M : \tau$ following the typing rules of Figure 2. We note $\mathrm{dom}(\Gamma)$ the set of (term) variables declared in Γ.

The one step (contextual) reduction $\triangleright$ is defined in Figure 3. The transitive closure of $\triangleright$ is $\triangleright^*$. Here $M[x \leftarrow N]$ means the usual substitution of all free occurrences of x in M by N.

Remark 6.1.
(1) As usual, $\tau \Rightarrow \kappa$ and $\tau \to \tau'$ are short cuts for $\Pi x : \tau.\kappa$ and $\forall x : \tau.\tau'$, when x is not occurring in κ or τ'.
(2) There is no type variable (except type constructors).
(3) In fact, we just consider two kinds $*$ and $\omega \Rightarrow *$, because we have no introduction rules for kinds.
(4) The two previous points imply that in a judgment of the form $\Gamma \vdash M : \sigma$, if $(x : \tau)$ is in M, then τ is either ω or $\mathbf{W}(t)$ for some term t of type ω.

The system $\boldsymbol{\lambda}\mathbf{D}^{\boldsymbol{\omega}}$ can be translated in the system T of Gödel and so it has the Church-Rosser and strong normalization properties.

6.2. **Function representation at a given tier.**

A natural number k is represented by $\underline{k}$ thus:

$$\underline{0} = \diamond \qquad\qquad \underline{x+1} = (\mathbf{S}\ \underline{x})$$

And a word u of $\mathbb{W}$ is represented by $\underline{u}_k$ at tier k thus

$$\underline{\epsilon}_k = (\mathbf{0}\ \underline{k}) \qquad \underline{a(x)}_k = (\mathbf{A}\ \underline{k}\ \underline{x}_k) \qquad \underline{b(x)}_k = (\mathbf{B}\ \underline{k}\ \underline{x}_k)$$

Type constructors

$$\vdash \omega : *$$
$$\vdash \mathbf{W} : \omega \Rightarrow *$$

Terms of type ω

$$\vdash \diamond : \omega$$
$$\vdash \mathbf{S} : \omega \to \omega$$

Tiered words

$$\vdash \mathbf{0} : \forall k.\mathbf{W}(k)$$
$$\vdash \mathbf{A} : \forall k.\mathbf{W}(k) \to \mathbf{W}(k)$$
$$\vdash \mathbf{B} : \forall k.\mathbf{W}(k) \to \mathbf{W}(k)$$

Pairing and projections

$$\vdash \langle \_, \_ \rangle_\_ : \forall k.\mathbf{W}(k) \to \mathbf{W}(k) \to \mathbf{W}(k) \times \mathbf{W}(k)$$
$$\vdash \boldsymbol{\pi}^1 : \forall k.\mathbf{W}(k) \times \mathbf{W}(k) \to \mathbf{W}(k)$$
$$\vdash \boldsymbol{\pi}^2 : \forall k.\mathbf{W}(k) \times \mathbf{W}(k) \to \mathbf{W}(k)$$

Flat recursion

$$\vdash \mathbf{flat}_\tau : \forall k.\tau \to (\mathbf{W}(k) \to \tau)^2 \to \mathbf{W}(k) \to \tau$$

Double tiered recursion

$$\vdash \mathbf{diag}_p : (\mathbf{W}(\diamond)^p \to \mathbf{W}(\diamond)^p) \to \forall k.\mathbf{W}(k) \to \mathbf{W}(k) \to \mathbf{W}(\diamond)^p \to \mathbf{W}(\diamond)^p$$

Figure 1: Types of type and term constructors

Definition 6.2. Let $\phi : \mathbb{W}^{p+p'} \to \mathbb{W}^q$. The function ϕ is represented at tier k if there is a term $M : \mathbf{W}(\underline{k})^p \to \mathbf{W}(\underline{0})^{p'} \to \mathbf{W}(\underline{0})^q$ such that for all $u_1 \ldots u_p$ of $\mathbb{W}^p$ and for all $v_1 \ldots v_{p'}$ of $\mathbb{W}^{p'}$

$$M \ \underline{u_1}_k \ldots \underline{u_p}_k \ \underline{v_1}_0 \ldots \underline{v_{p'}}_0 \ \triangleright^* \ \underline{\phi(u_1, \ldots, u_p, v_1, \ldots, v_{p'})}_0$$

We define $\mathcal{CI}_k(\mathbb{W})$ as the set of functions which are represented at tier k.

Lemma 6.3. *Each function $g : \mathbb{W}(0)^p \to \mathbb{W}(0)^q$ in $\mathcal{I}_0(\mathbb{W})$ is represented at tier 0, and so is in $\mathcal{CI}_0(\mathbb{W})$.*

Proof. The proof is done by induction on the definition of g. $\square$

The construction of a polynomial length iterator in $\boldsymbol{\lambda}\mathbf{D}^{\boldsymbol{\omega}}$ follows closely the lines of the demonstration of Lemma 4.5. It is obtained by composition from F_k functions, which are representable at tier k following the Lemma below.

Lemma 6.4. *For each k, the function F_k parameterized by a function $g : \mathbb{W}(0)^p \to \mathbb{W}(0)^p$, is represented at tier k, and so is in $\mathcal{CI}_k(\mathbb{W})$.*

Kinding rules

$$\frac{\Gamma \vdash \phi : \tau \Rightarrow \kappa \qquad \Gamma \vdash t : \tau}{\Gamma \vdash \phi t : \kappa} \Rightarrow \text{elim}$$

Typing rules

$$\frac{\Gamma \vdash \tau : *}{\Gamma, x : \tau \vdash x : \tau} \text{ Variable, } x \notin \text{dom}(\Gamma)$$

$$\frac{}{\Gamma \vdash \mathbf{c} : \tau} \text{ where } \mathbf{c} \text{ is a type or a term constructor of type } \tau$$
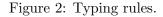
$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \tau \to \sigma} \to \text{intro}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma} \to \text{elim}$$

$$\frac{\Gamma, x : \omega \vdash M : \tau}{\Gamma \vdash \lambda x.M : \forall x.\tau} \forall \text{ intro, and } x \notin \text{dom}(\Gamma)$$

$$\frac{\Gamma \vdash M : \forall x.\tau \qquad \Gamma \vdash k : \omega}{\Gamma \vdash Mk : \tau[x \leftarrow k]} \forall \text{ elim}$$

Weakening rule

$$\frac{\vdash \tau : * \qquad \Gamma \vdash M : \sigma}{\Gamma, x : \tau \vdash M : \sigma} \text{Weakening, } x \notin \text{dom}(\Gamma) \text{ and } M \text{ is a term or a type}$$

Downcasting rule

$$\frac{\Gamma \vdash M : \mathbf{W}(\mathbf{S}(t))}{\Gamma \vdash M : \mathbf{W}(t)} \text{Downcasting}$$

Figure 2: Typing rules.

Proof. The previous lemma 6.3 gives a term $N : \mathbf{W}(\diamond)^p \to \mathbf{W}(\diamond)^p$, which represents g. Now, for each k, we define a sequence of terms $(M_k)_k$ parameterized by N by

$$M_k = \lambda x \lambda y \lambda z.(\mathbf{diag}_p \ N \ \underline{k} \ x \ y \ z) \qquad \text{of type } \mathbf{W}(\underline{k}) \to \mathbf{W}(\underline{k}) \to \mathbf{W}(\diamond)^p \to \mathbf{W}(\diamond)^p$$

We can check that F_k is represented at tier k by M_k by induction on k and the first parameter of M_k:

β-reduction

$$(\lambda x.M)N \triangleright M[x \leftarrow N]$$

projections

$$(\boldsymbol{\pi}^1 k \; \langle M, N \rangle_k) \triangleright M$$
$$(\boldsymbol{\pi}^2 k \; \langle M, N \rangle_k) \triangleright N$$

flat recursion

$$(\mathbf{flat} \; k \; h_\epsilon \; h_a \; h_b \; (\mathbf{0} \; k)) \triangleright h_\epsilon$$
$$(\mathbf{flat} \; k \; h_\epsilon \; h_a \; h_b \; (\mathbf{A} \; k \; x)) \triangleright (h_a \; x)$$
$$(\mathbf{flat} \; k \; h_\epsilon \; h_a \; h_b \; (\mathbf{B} \; k \; x)) \triangleright (h_b \; x)$$

double recursion where $\mathbf{J} = \mathbf{A}, \mathbf{B}$

$$(\mathbf{diag}_p \; g \diamond z \; x \; y) \triangleright (g \; y)$$
$$(\mathbf{diag}_p \; g \; (\mathbf{S} \; k) \; \mathbf{0} \; x \; y) \triangleright y$$
$$(\mathbf{diag}_p \; g \; (\mathbf{S} \; k) \; (\mathbf{J} \; k \; r) \; x \; y) \triangleright (\mathbf{diag}_p \; g \; k \; x \; x \; (\mathbf{diag}_p \; g \; (\mathbf{S} \; k) \; r \; x \; y))$$

Figure 3: Rules of computation

For $k = 0$ and for all u,v and w, we have

$$(M_0 \; \underline{u}_0 \; \underline{v}_0 \; \underline{w}_0) = (\mathbf{diag}_p \; N \; \underline{0} \; \underline{u}_0 \; \underline{v}_0 \; \underline{w}_0)$$
$$\triangleright (N \; \underline{w}_0)$$
$$= \underline{g(w)}_0 = \underline{F_0(u, v, w)}_0$$

For $k + 1$, we proceed by induction on the first argument u of M_{k+1}. First, for all v and w, we have

$$M_{k+1} \; \underline{\epsilon}_{k+1} \; \underline{v}_{k+1} \; \underline{w}_0 = \mathbf{diag}_p \; N \; \underline{k+1} \; \underline{\epsilon}_{k+1} \; \underline{v}_{k+1} \; \underline{w}_0$$
$$\triangleright \underline{w}_0 = \underline{F_{k+1}(\epsilon, v, w)}_0$$

$$M_{k+1} \; \underline{i(u)}_{k+1} \; \underline{v}_{k+1} \; \underline{w}_0 = \mathbf{diag}_p \; N \; \underline{k+1} \; \underline{i(u)}_{k+1} \; \underline{v}_{k+1} \; \underline{w}_0 \qquad\qquad i = a, b$$
$$\triangleright (\mathbf{diag}_p \; N \; \underline{k} \; \underline{v}_k \; \underline{v}_k \; (\mathbf{diag}_p \; N \; \underline{k+1} \; \underline{u}_{k+1} \; \underline{v}_{k+1} \; \underline{w}_0))$$
$$= \underline{F_k(v, v, F_{k+1}(u, v, w))}_0 = \underline{F_{k+1}(i(u), v, w)}_0$$

In conclusion, for all k, u, v and w, we have

$$M_k \; \underline{u}_k \; \underline{v}_k \; \underline{w}_0 = \underline{F_k(u, v, w)}_0$$

It is also worth to see that by Lemma 4.2:

$$M_k \; \underline{u}_k \; \underline{v}_k \; \underline{w}_0 = \underline{g^{|u|.|v|^k}(w)}_0 \qquad\qquad \square$$

The following Lemma corresponds to Lemma 4.2

Lemma 6.5. *Any polynomial $P[X]$ of degree k with natural coefficients is represented at tier k in $\mathcal{CI}_k(\mathbb{W})$. More precisely, assume that $N : \mathbf{W}(0)^p \to \mathbf{W}(0)^p$.*
Then, there is a term $P_k : \mathbf{W}(k), \mathbf{W}(0)^p \to \mathbf{W}(0)^p$ in $\mathcal{CI}_k(\mathbb{W})$ such that for each u and v,

$$P_k \; \underline{u}_k \; \underline{v}_0 \rhd^* (N^{P(|\underline{u}_k|)} \; \underline{v}_0) \tag{6.1}$$

Proof. The proof goes by induction. Suppose that the degree of P is $k+1$. Hence, $P(x) = c.x^{k+1} + Q(x)$ where the degree of Q is less or equal to k. Suppose that M' satisfies the induction hypothesis wrt Q. We define M_{k+1}^c by composition as follows

$$M_{k+1}^0 \; x \; y = (M'x \; y)$$
$$M_{k+1}^{d+1} \; x \; y = (M_{k+1} \; x \; x \; (M_{k+1}^d \; x \; y)) \qquad\qquad d < c$$

where $(M_k)_k$ is the sequence of terms defined in the demonstration of the previous Lemma, and computes $(F_k)_k$. The type of M_{k+1}^d is $\mathbf{W}(k+1) \to \mathbf{W}(0)^p \to \mathbf{W}(0)^p$ for any d. We set $P_{k+1} = M_{k+1}^c$. $\qquad\square$

As a direct consequence of the above Lemma, we have a result which is analogous to Lemma 4.5:

Corollary 6.6. *Let $P[X_1, \ldots, X_n]$ be a simple polynomial of degree k.*
There is a term $\mathbf{P} : \mathbf{W}(k)^n, \mathbf{W}(0)^p \to \mathbf{W}(0)^p$ such that for each $u_1, \ldots, u_n$, and v,

$$(\mathbf{P} \; \underline{u_1}_k \ldots \; \underline{u_n}_k \; \underline{v}_0) \rhd^* (N^{P(|u_1|, \ldots, |u_n|)} \; \underline{v}_0)$$

where $N : \mathbf{W}(0)^p \to \mathbf{W}(0)^p$.

Proof. The proof is done by induction on the number of variables. The base case is a consequence of Lemma 6.5. Suppose that the simple polynomial P has $n+1$ variables $X_1, \ldots, X_n, X_{n+1}$. Since P is simple, we write it as the sum $P(X_1, \ldots, X_n, X_{n+1}) = P'(X_1, \ldots, X_n) + P''(X_{n+1})$. Suppose that $\mathbf{P}'$ ($\mathbf{P}''$) satisfies the induction hypothesis wrt P' (resp. P''). We define $\mathbf{P}$ by

$$(\mathbf{P} \; x_1 \ldots \; x_{n+1}) = (\mathbf{P}' \; x_1 \ldots \; x_n(\mathbf{P}'' \; x_{n+1} \; y)) \qquad\square$$

Theorem 6.7. *The set of functions $\mathcal{I}_k(\mathbb{W})$ is exactly the set $\mathcal{CI}_k(\mathbb{W})$, that is the class $DTIME(n^k)$.*

Proof. First, we establish that $DTIME(n^k) \subseteq \mathcal{CI}_k(\mathbb{W})$. For this, observe that the transition function *next*, which is defined in the proof of Lemma 4.7, is represented at tier 0, by a term of type $\mathbf{W}(0)^{m+1} \to \mathbf{W}(0)^{m+1}$. We iterate *next* by using Corollary 6.6.

Conversely, we show that $\mathcal{CI}_k(\mathbb{W}) \subseteq \mathcal{I}_k(\mathbb{W})$. For this, let f be a function represented at tier k by a term M. In other words, there is a normal derivation ∇ such that $\vdash M : \mathbf{W}(\underline{k})^p \to \mathbf{W}(\underline{0})^q$. Observe that if x is a variable of M, the type of x is $\mathbf{W}(k')^{p'}$, $k' \leq k$ and $p' \leq p$. So, a subterm t of M of type ω does not contain a variable (of type ω) and so represents a natural number, that is $t = \underline{r}$ for some r. Therefore, the term M denotes a function of $\mathcal{I}_k(\mathbb{W})$. The proof is complete by Theorem 4.6. $\qquad\square$

6.3. **Jumping outside.** Let $\phi : \mathbb{W} \to \mathbb{W}$. The function ϕ is represented at tier ω if there is a term $M : \forall k.\mathbf{W}(k)^p \to \mathbf{W}(\diamond)^q$ such that for all u,

$$M \ \underline{k} \ \underline{u}_k \rhd^* \underline{\phi(u)}_0 \qquad\qquad \text{where } k = |u|$$

We define $\mathcal{CI}_\omega(\mathbb{W})$ as the set of functions which are represented at tier ω.

Proposition 6.8. *There is a function represented at tier ω which is not representable at tier k, for any k. In other words, this function is not $\cup_k \mathcal{I}_k(\mathbb{W})$.*

Proof. The function $\Delta^\omega[A_0]$ is representable at tier ω. As the consequence, we can define the exponential as follows: $E = \lambda k \lambda x.(\mathbf{diag}_1(\mathbf{A} \diamond) \ k \ x \ x \ x)$ of type $\forall k.\mathbb{W}(k) \to \mathbb{W}(0)$. We have $E \ \underline{|u|} \ \underline{u}_k \rhd^* e$ and $|e| \geq |u|^{|u|+1} + |u|$, for all u. $\qquad\square$

6.4. **Other ways to jump.** We have presented a manner of constructing an exponential function by diagonalizing functions defined by strict ramified recursion. There are other approaches. In [12], Leivant ramifies the system T of Gödel [8] by introducing an atomic type constructor $\Omega(\tau)$ which allows to perform recursion over type τ terms. Thus, he obtains a characterization of FPTIME and of the elementary functions.

Bellantoni and Niggl [4] characterized the Grzegorczyk hierarchy starting from the class FPTIME. For this, they define a rank function which, roughly speaking, is a bound on the number of nested recursions. The work of Caporaso, Covino and Pani seems also related to the research presented in this paper, see [6]. We are also aware of other related works like the one of Oitavem [15] or the one of Beckmann and Weiermann [2]. Finally, Danner [7] proposed a ramified Gödel system T with a dependant typing system to study primitive recursive functions.

References

[1] W. Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Math. annalen*, 99:118–133, 1928.

[2] A. Beckmann and A. Weiermann. Characterizing the elementary recursive functions by a fragment of Gödel's T. *Archive for Mathematical Logic*, 1996. to appear.

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

[4] S. Bellantoni and K-H Niggl. Ranking primitive recursions: The low Grzegorczyk classes revisited. *SIAM Journal on Computing*, 29(2):401–415, 1999.

[5] Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem. Recursion schemata for nck. In Michael Kaminski and Simone Martini, editors, *22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213, pages 49–63, Italie Bertinoro, 2008. Springer.

[6] S. Caporaso, E. Covino, and G. Pani. A predicative approach to the classification problem. *J. Funct. Program.*, 11(1):95–116, 2001.

[7] N. Danner Ramified Recurrence with dependent types. In S. Abramsky, editor, *Typed-Lambda calculi and applications*, volume 2044, pages 91–105, 2001. Springer.

[8] K. Gödel. Über eine bisher noch nicht benüte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958. Republished with English translation and explanatory notes by A. S. Troelstra in *Kurt Gödel: Collected Works*, Vol. II. S. Feferman, ed. Oxford University Press, 1990.

[9] N. Jones. *Computability and complexity, from a programming perspective*. MIT press, 1997.

[10] D. Leivant. A foundational delineation of computational feasiblity. In *Proceedings of the Sixth IEEE Symposium on Logic in Computer Science (LICS'91)*, 1991.

[11] D. Leivant. Predicative recurrence and computational complexity I: Word recurrence and poly-time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

[12] D. Leivant. Ramified recurrence and computational complexity III: Higher type recurrence and elementary complexity. *Annals of Pure and Applied Logic*, 96(1-3):209–229, 1999.

[13] D. Leivant and J-Y Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19(1,2):167,184, September 1993.

[14] D. Leivant and J-Y Marion. A characterization of alternating log time by ramified recurrence. *Theoretical Computer Science*, 236(1-2):192–208, Apr 2000.

[15] I. Oitavem. New reursive characterization of the elementary functions and the functions computable in polynomial space. *Revista Matemática de la universidad complutense de Madrid*, 10(1), 1997.

[16] H. Simmon. Tiering as a recursion technique. *Bulletin of Symbolic Logic*, 11(3):321–350, 2005.

[17] H. Simmons. The realm of primitive recursion. *Archive for Mathematical Logic*, 27:177–188, 1988.

[18] H. Simmons. *Derivation and Computation*, volume 51 of *Tracts in theoretical computer science*. Cambridge, 2000.