Higher Type Recursion, Ramification and Polynomial Time

S. Bellantoni* K.-H. Niggl† H. Schwichtenberg

February 1, 1999

Abstract

It is shown how to restrict recursion on notation in all finite types so as to characterize the polynomial time computable functions. The restrictions are obtained by enriching the type structure with the formation of types $!\sigma$, and by adding linear concepts to the lambda calculus.

1 Introduction

Recursion in all finite types was introduced by Hilbert [9] and later became known as the essential part of Gödel's system T [8]. This system has long been viewed as a powerful scheme unsuitable for describing small complexity classes such as polynomial time. Simmons [16] showed that ramification can be used to characterize the primitive recursive functions by higher type recursion, and Leivant and Marion [14] showed that another form of ramification can be used to restrict higher type recursion to PSPACE. However, to characterize the much smaller class of polynomial-time computable functions by higher type recursion, it seems that an additional principle is required. By introducing linearity constraints in conjunction with ramified recursion, we characterize polynomial-time computability while admitting recursion on notation in higher types.

Based on simple types built from the ground type ι of binary numerals by $\to$, *recursion on notation* in type σ is a mapping $\mathcal{R}_\sigma$ of type $\sigma \to (\iota \to \sigma \to \sigma) \to \iota \to \sigma$ defined by

$$\mathcal{R}_\sigma \, g \, h \, \mathbf{0} = g$$
$$\mathcal{R}_\sigma \, g \, h \, (\mathbf{s}_i \mathbf{n}) = h(\mathbf{s}_i \mathbf{n})(\mathcal{R}_\sigma g h \mathbf{n}).$$

Now a single recursion in type $\iota \to \iota$ can define a function of exponential growth:

$$e := \mathcal{R}_{\iota\to\iota} \, \mathbf{s}_1 \, (\lambda u^\iota V^{\iota\to\iota} y^\iota . V(Vy))$$

satisfies $|e(m)(n)| = 2^{|m|} + |n|$. Note that the function e can be assigned a ramified type under the scheme of Leivant [14], in which m is tier 1 and n is tier 0. What this shows is that another requirement, in addition to ramification of the recursion variable, is required to restrict higher type recursion to polytime computability. The culprit seems to lie in the nested, nonlinear use of the

*Research supported by: Graduiertenkolleg "Logik in der Informatik" der DFG, München. Assistance of the Fields Instituted for Research in Mathematical Sciences, Toronto is gratefully acknowledged.

†The research reported here has been supported by a DFG grant, which is gratefully acknowledged.

previous value V. Our approach is to introduce at the same time both ramification of the recursion variable and linearity conditions.

To do so we enrich the type structure with the formation of types $!\sigma$, called *complete types*; all other types are called *incomplete*. Intuitively, objects of complete types are completely known; they can be used as the pattern for a recursion, or if they are of higher type they can be used in a non-linear way. Objects of incomplete types can only be accessed through a few low-order bits, or if they are of higher type can be used in a certain linear way only. Then we define the class RA of ramified affinable terms. The recursor $\mathcal{R}_\sigma$ receives the ramified type $\sigma \to !(!\iota \to \sigma \to \sigma) \to !\iota \to \sigma$ and is admitted for any !-free σ; as well, we require that terms of complete types have no free variables of incomplete types. Input positions of types $!\iota$ and ι correspond to normal / tier 1 and safe / tier 0 input positions, common in earlier work on ramified recursion (cf. [16, 13, 2, 4, 15]). Affinability is central to the system and expresses the linearity constraints for bound variables of incomplete types. Affinability is designed such that the system RA is closed under reduction.

Ordinary lambda terms defined without recursion constants are embeddable in RA. Therefore for a closed term t, the normal form $\mathsf{nf}(t)$ of t may have length superexponential in the length of t. However, this is only a constant with respect to the length of integers $\vec{n}$ at which we may wish to evaluate the denoted function. What we show is that for each closed RA-term t of type level 1, one can find a polynomial p_t such that for all numerals $\vec{n}$, one can compute the normal form $\mathsf{nf}(t\vec{n})$ in time $p_t(|\vec{n}|)$. Thus, t denotes a polynomial time computable function. The converse also holds, as each polynomial time computable function is computed by some RA-term.

Recently, Hofmann [10, 11] used modalities of ramification and of linearity in a lambda calculus, and defined them for all higher types. This interesting work also characterizes polynomial time computability. There, the two modalities are independent; but the present paper makes these concepts coincide. The proof methods of the two papers are also different, as Hofmann uses a category-theoretic approach.

This work has also connections to work in linear logic (see Girard, Scedrov and Scott [6] and Abramsky [1] for background on linear logic), where "!" refers to a knowledge of the input that allows it to be used nonlinearly. However, the logic behind the present work differs significantly from the polytime linear logic in [6], because in the present work there are no explicit polynomials and there is no attribution of work to occurrences of !. There are some connections between the present work and the "light linear logic" of Girard [7]; but due to differing frameworks an exact comparison has not been made. The modality ! used in the present work is different from the linear modality because the ! here includes concepts of ramification from the study of ordinary ramified recursion.

The approach to higher-type functions taken in this work contrasts with Cook and Kapron's well-known Basic Feasible Functions defined by PV^ω terms [5]. There, explicit size bounds are used and the critical value computed during the recursion is of ground type. A further difference can be seen by the fact that the system RA admits the iteration functional $\lambda f^{\iota\to\iota} x^{!\iota} y^{\iota}.f^{(|x|)}(y)$, whose unramified form is not BFF. On the other hand, one intuitively expects that in some suitable sense BFF functions should be definable in RA.

2 Types and terms

The *types* are: ι is a type, and if σ and τ are types, then so are $\sigma \to \tau$ and $!\sigma$. We assume ! binds tighter than $\to$, and $\to$ associates to the right.[1]

Types of the form $!\sigma$ are *complete*; all others are *incomplete*. *Ground* types are the types of level 0, defining level by: $l(\iota) = 0$; $l(!\sigma) = l(\sigma)$; and $l(\sigma \to \tau) = \max\{l(\tau), 1 + l(\sigma)\}$. A *higher* type is any type of level at least 1. For example, $!!\iota$ is a ground type, but $\iota \to \iota$ is a higher type. !-free types are called *safe*. Every ground type is either safe or complete.

The *constant symbols* are listed below, with their types.

$\mathbf{0}$	ι		
$\mathbf{s}_0$	$\iota \to \iota$		
$\mathbf{s}_1$	$\iota \to \iota$		
$\mathbf{p}$	$\iota \to \iota$		(predecessor)
$\mathbf{d}_\sigma$	$\iota \to (\iota \to \sigma) \to (\iota \to \sigma) \to (\iota \to \sigma) \to \sigma$	for σ safe	(cases)
$\mathcal{R}_\sigma$	$\sigma \to !(!\iota \to \sigma \to \sigma) \to !\iota \to \sigma$	for σ safe	(recursion)

Terms are built from these constants and typed variables x^σ by introduction and elimination rules for the two type forms $\sigma \to \tau$ and $!\sigma$, i.e.

$$(\lambda x^\sigma . r^\tau)^{\sigma \to \tau}, \quad (r^{\sigma \to \tau} s^\sigma)^\tau, \quad (!r^\sigma)^{!\sigma}, \quad (r^{!\sigma}\kappa)^\sigma.$$

A *binary numeral* is either $\mathbf{0}$ or $\mathbf{s}_{i_1} \dots \mathbf{s}_{i_k}\mathbf{s}_1\mathbf{0}$ where $i_j \in \{0,1\}$. In the *conversion rules* below we assume that $\mathbf{s}_i\mathbf{n}$ is a binary numeral (hence distinct from $\mathbf{0}$).

$$
\begin{aligned}
&(\lambda x.r)s \mapsto r[s/x] \\
&(!r)\kappa \mapsto r \\
&\mathbf{s}_0\mathbf{0} \mapsto \mathbf{0} \\
&\mathbf{p0} \mapsto \mathbf{0} \\
&\mathbf{p}(\mathbf{s}_i\mathbf{n}) \mapsto \mathbf{n} \\
&\mathbf{d0}rt_0t_1 \mapsto r\mathbf{0} \\
&\mathbf{d}(\mathbf{s}_i\mathbf{n})rt_0t_1 \mapsto t_i(\mathbf{s}_i\mathbf{n}) \\
&\mathcal{R}\, g\, h\, !\mathbf{0} \mapsto g \\
&\mathcal{R}\, g\, h\, !(\mathbf{s}_i\mathbf{n}) \mapsto h\kappa\ !(\mathbf{s}_i\mathbf{n})\ (\mathcal{R}\, g\, h\, !\mathbf{n})
\end{aligned}
$$

Here an application of ! onto a term associates tighter than other applications, and to the right, while other applications associate to the left. Thus $\mathcal{R}gh!n$ is $(((\mathcal{R}g)h)(!n))$. The conditional $\mathbf{d}$ is based on Hofmann's conditional [10].

The *length* $|t|$ of a term t is defined by $|x| = |c| = 1$; $|\lambda x.r| = |!r| = |r\kappa| = |r|+1$; $|rs| = |r|+|s|+1$. *Redexes* are subterms shown on the left side of conversion rules above. A term is in *normal form* if it does not contain a redex. For every term t there is a unique normal-form term $\mathsf{nf}(t)$ (see e.g. [17, 12] for proofs of normalisation in Gödel's system T). Two terms are *equivalent* if they have the same normal form.

[1] Linear logicians may read "$\to$" as "$\multimap$".

One writes $\mathsf{FV}(t)$ for the set of free variables of t, and $\mathsf{FO}(x,t)$ for the multiset of free occurrences of x in t. Say that a term is *complete*, *incomplete*, *safe*, or *ground* if its type is.

Similar to Gödel's T, types and terms are interpreted over the set theoretical function spaces. Thus, in the semantics we identify objects of type $!\sigma$ with those of type σ, since we are only interested in the computational behaviour of terms. We interpret ι as the non-negative integers. The value $[\![t]\!]_\varphi$ of a term t in an environment φ is defined as usual, where $[\![!r]\!]_\varphi := [\![r]\!]_\varphi$ and $[\![r\kappa]\!]_\varphi := [\![r]\!]_\varphi$. As the value of a closed term t is independent of any environment, we just write $[\![t]\!]$.

3 RA-terms

Two subterms a_i and a_j occurring in a term t are *scope equivalent* if whenever λy binds a variable free in either a_i or a_j, then both a_i and a_j lie within the scope of the λy.

Definition. Let x be an incomplete variable, and let s be a term.

1. An *affination* of x in s is a subterm a^ι with $|\mathsf{FO}(x,s)| = 1$ such that every free occurrence of x in s is in an occurrence of a in s, where the occurrences of a are scope equivalent in s.
2. We call x *affinable* in s if there is an affination of x in s or $|\mathsf{FO}(x,s)| \leq 1$.

Every type ι variable is trivially affinable in every term, because it is an affination of itself.

Definition. r is an RA-*term* (R for ramified, A for affinable) if

(R) every complete subterm contains complete free variables only, and

(A) for every subterm $\lambda x.s$ with x incomplete, the variable x is affinable in a reduct of s.

As pointed out in the Introduction, one intuition is that terms of complete type can be used in a non-linear way, while objects of higher incomplete type can be used only in a certain linear way expressed by (A). Accordingly, (R) requires that complete terms have no incomplete free variables. Following Simmons [16], this and the requirement that *step terms* h in $\mathcal{R}_\sigma ghn$ have complete types rule out non-primitive recursive growth rate. For no *previous value* (V below) of an outer recursion can then be applied to the previous value of an inner recursion. For example, in Gödel's T the function $F_\omega(x) = F_x(x)$, where $F_0(x) := x+1$ and $F_{n+1}(x) := {F_n}^{(x+1)}(x)$, can be defined by

$$t_{F_\omega} := \mathcal{R}_{\iota\to\iota}\, \mathbf{S}\, (\lambda u^\iota V^{\iota\to\iota} y^\iota.\, \mathcal{R}_\iota\, y\, (\lambda u^\iota v^\iota . V v)\, \mathbf{S} y)$$

To obtain polynomial growth rate we additionally require that recursion in type σ is admitted for safe (i.e. !-free) types σ only – recall the type $\sigma \to !(!\iota \to \sigma \to \sigma) \to !\iota \to \sigma$ of $\mathcal{R}_\sigma$ in RA.

For $\sigma = \iota$, this mimics the central idea of ramification via safe (ι) and normal ($!\iota$) input positions (cf. [2]): Previous values in recursions can only be passed to safe input positions, i.e. input positions which do not induce the unfolding of recursions. For example, polynomially-growing functions $\oplus$ satisfying $|m \oplus n| = |m| + |n|$, and $\otimes$ satisfying $|m \otimes n| = |m| \cdot |n|$, are easily definable in RA:

$$\begin{aligned} t_\oplus &:= \lambda x^{!\iota} y^\iota . \mathcal{R}_\iota\, y\, !(\lambda u^{!\iota} v^\iota .\, \mathbf{s}_1 v)\, x \\ t_\otimes &:= \lambda x^{!\iota} y^{!\iota} . \mathcal{R}_\iota\, \mathbf{0}\, !(\lambda u^{!\iota} v^\iota .\, t_\oplus\, x v)\, y \end{aligned}$$

For higher types σ, previous values in recursions can only be passed to affinable input positions such that they will never be applied to objects of complete types – since recursion $\mathcal{R}_\sigma$ is admitted for safe

σ only. Affinability is designed to rule out nested occurrences of previous values in recursions (cf. the definition of e in the Introduction). It requires that whenever we lambda abstract a higher-type incomplete variable x in r, then either $|\mathsf{FO}(x,r)| \leq 1$ or else the free occurrences of x in r can be separated by the occurrences of one and the same ground type context a, the affination of x in r.

Admitting recursion $\mathcal{R}_\sigma$ for incomplete σ would allow one to define proper Kalmar-elementary functions. For example, a function e' satisfying $e'(m,n) \geq n^{2^{|m|}}$ would then have an RA definition

$$\mathcal{R}_{!\iota\to\iota}\,(\lambda y^{!\iota}.y\kappa)\,!(\lambda u^{!\iota} V^{!\iota\to\iota} y^{!\iota}.\,V\,!(t_{\mathsf{sq}}\,y))$$

where $t_{\mathsf{sq}} := \mathcal{R}_\iota\,(\mathbf{s}_1\mathbf{0})\,!(\lambda u^{!\iota}v^\iota.\,\mathbf{s}_0\mathbf{s}_0 v)$ defines the function $\mathsf{sq}(n) = 2^{2|n|}$.

Note that if x is affinable r, and $r \longrightarrow r'$, then x need not to be affinable in r'. To obtain a system closed under reduction, condition (A) requires that x is affinable in a reduct of r.

Note that terms with property (R) are not closed under application, as one may form e.g. $X^{\iota\to!\iota}y^\iota$, or else $(\lambda y^\iota.\,!\mathbf{0})y$. However, if rs is incomplete and r, s satisfy (R), then so does rs. It is also rather immediate that terms with property (R) are closed under reductions. This is also true for RA-terms, as we will prove next.

Theorem 3.1 (Closure under reduction). *Let r be an* RA*-term.*

(1) *If $r \longrightarrow r'$, then r' is an* RA*-term.*

(2) *If x is affinable in r, then x is affinable in $\mathsf{nf}(r)$.*

Proof. By induction on the *height* $h(r)$ of the reduction tree for r, and side induction on r.

For the proof of (1), let r be an RA-term, and assume $r \longrightarrow r'$. Since terms with property (R) are closed under reductions, it suffices to consider a subterm $\lambda x.s'$ of r' where x is incomplete, and prove that x is affinable in a reduct of s'.

Case $s' = s[t/y]$ for a subterm $\lambda x.s$ of r. By assumption x is affinable in a reduct of s. Then x is also affinable in a reduct of s', since $x \notin \mathsf{FV}(t)$.

Case $s \longrightarrow s'$ for a subterm $\lambda x.s$ of r. We distinguish two subcases.

Subcase x is affinable in s. Then $s \longrightarrow s' \longrightarrow^* \mathsf{nf}(s)$, and by the side IH (2) for s, x is affinable in $\mathsf{nf}(s) = \mathsf{nf}(s')$. Hence x is affinable in a reduct of s', namely $\mathsf{nf}(s')$.

Subcase x is affinable in a reduct of s. Then we find ourselves in the situation:

$$\begin{array}{lllllllll} s & \to & s_1 & \to & \dots & \to & s_n, & & x \text{ affinable in } s_n \\ \downarrow & & & & & & & & \\ s' & & & & & & & & \end{array}$$

By the side IH (1) at s, s_1 is an RA-term. Successively applying the IH (1) to $s_1,\dots,s_{n-1}$, one obtains that s_n is an RA-term. Thus, by the IH (2) at s_n, x is affinable in $\mathsf{nf}(s_n) = \mathsf{nf}(s')$.

For the proof of (2), let r be an RA-term and assume that x is affinable in r. If r is normal we are done. So assume $r \longrightarrow r'$. Again, we proceed by distinguishing two cases.

Case there is an affination a of x in r. We may assume that there is a redex in a (otherwise, x is affinable in r' and the claim follows by (1) giving that r' is an RA-term and then IH (2) giving that x is affinable in $\mathsf{nf}(r)$). Let r'' be the reduct of r obtained by replacing all occurrences of a in r with $\mathsf{nf}(a)$. Hence $h(r'') < h(r)$, and r'' is an RA-term by (1). Then the claim follows from the IH for (2), for either x has at most one free occurrence in $\mathsf{nf}(a)$ (then $\mathsf{nf}(a)$ is an affination of x in r''), or else there is an affination b of x in $\mathsf{nf}(a)$ (then b is an affination of x in r'').

Case $|\mathsf{FO}(x,r)| \leq 1$. By (1) and the IH (2) we may assume $|\mathsf{FO}(x,r')| \geq 2$, i.e. a subterm containing x is duplicated during the reduction. Considering all reductions, the only ones which can duplicate a subterm are $\mathcal{R}$ reductions and β reductions. But in the former case, the duplicated subterm of r has complete type, hence by the property (R) cannot contain x. In the latter case, there is a redex $(\lambda y.s)t$ in r with $x \in \mathsf{FV}(t)$ such that r' is formed by replacing $(\lambda y.s)t$ with $s[t/y]$. Since r satisfies (R) and t contains the incomplete variable x, it must be that t and hence y is incomplete. As r satisfies (A), y is affinable in a reduct s' of s. So let r'' be obtained from r' by replacing $s[t/y]$ with $s'[t/y]$. Then r'' is an RA-term by (1), and $r' \longrightarrow^* r''$. Furthermore, x is affinable in r'', for if $|\mathsf{FO}(y,s')| \leq 1$, then $|\mathsf{FO}(x,r'')| \leq 1$, and if b is an affination of y in s', then $b[t/y]$ is an affination of x in r''. Therefore, applying the induction hypothesis (2) to r'', we obtain that x is affinable in $\mathsf{nf}(r'') = \mathsf{nf}(r)$. ∎

4 RS-terms

In our final result we will only be interested in ground type terms t whose free variables are of ground type. We first observe that – due to the typing of our constants – in the normal form of any such term all variables are safe or ground.

Lemma 4.1. *Let t be a ground type term whose free variables are of ground type. Then in $\mathsf{nf}(t)$ all variables are safe or ground.*

Proof. Suppose a variable x^σ with σ neither safe nor ground occurs in $\mathsf{nf}(t)$. It must be bound in a subterm $(\lambda x^\sigma.r)^{\sigma\to\tau}$ of $\mathsf{nf}(t)$. Now from the structure of normal derivations in the system of propositional logic consisting of introduction and elimination rules for $\to$ and ! it follows (cf. [17, p.84]) that $\sigma \to \tau$ either occurs positively in the type of $\mathsf{nf}(t)$, or else negatively in the type of one of the constants or free variables of $\mathsf{nf}(t)$. The former is impossible since t is of ground type, and the latter by inspection of the types of the constants. ∎

Now if a *normal* ground type term with only ground free variables satisfies (A), then for every subterm $\lambda x.s$ with x higher type (and hence safe), the variable x is affinable in s (since each reduct of s is s itself). Therefore by a kind of sharing construct we can obtain an equivalent term containing each higher type variable at most once: if e.g. s is $\dots a \dots a \dots$ with a an affination of x in s, replace $\lambda x. \dots a \dots a \dots$ by $\lambda x.(\lambda y^\iota. \dots y \dots y \dots)a$.

Lemma 4.2 (Sharing). *Let t be a term such that for every subterm $\lambda x.s$ with x higher-type incomplete, the variable x is affinable in s. Then by repeated η_ι-expansions $r[a/y^\iota] \mapsto (\lambda y.r)a$ one can construct a term $\eta(t)$ from t (hence $\eta(t) \longrightarrow^* t$) such that for every subterm $\lambda x.s$ in $\eta(t)$ with x higher-type incomplete, $|\mathsf{FO}(x,s)| \leq 1$.*

Proof. By induction on the number of occurrences of bound higher-type incomplete variables. Consider an outermost subterm $\lambda x.r$ with x higher-type incomplete and $|\mathsf{FO}(x,r)| \geq 2$. By assumption x has an affination a in r. Let u be the minimal subterm of r such that u contains all occurrences of a in r. Now let t' result from t by replacing u with $(\lambda y.u[y/a])a$ for some new variable y.[2]

[2] We write $u[y/a]$ for u with all occurrences of a simultaneously replaced by y.

To apply the induction hypothesis to t', one must show that every affination in t inside $\lambda x.r$ results in an affination in t'. To see this, let $\lambda z.s$ be a subterm of r such that z has an affination b in s. If a has no occurrence in s, then b is still an affination of z in t'. Otherwise by scope equivalence, either all occurrences of a are in s and $z \in \mathsf{FV}(a)$, or else no occurrence of a in s has a free occurrence of z. In the latter case, either a occurs in b, in which case $b[y/a]$ is an affination of z in t', or else a, b are separated, in which case b is still an affination of z in t'. In the former case, the minimality of u implies that u is in s. By construction t' results from t by replacing the subterm u of s with $(\lambda y.u[y/a])a$ for some new y. Since z has a free occurrence in both a and b, there are two cases. If a is a subterm of b, then each occurrence of b contains exactly one occurrence of a, for b is an affination of z in s. By construction it follows that a is an affination of z in t'. Otherwise if b is a subterm of a, then by construction b is still an affination of z in t'. □

This might motivate why it will be useful to consider a subset of the set of RA-terms, to be called RS-terms, where S stands for *sharing*.

Definition. An RA-term is an RS-*term* if it has safe or ground variables only, and

(S) every higher type variable occurs at most once.

Every term t can be written uniquely in *head form*, being of the form $U\vec{r}$, where U is a variable, a constant, $!s$ or $s\kappa$, or else U is $\lambda x.s$ with $|\mathsf{FO}(x,s)| \leq 1$, or $|\mathsf{FO}(x,s)| > 1$ and x ground. Call $\vec{r}$, s, x, and U the *components* of t. The number of components is not considered to be part of the head form. Components are specified by numbering them in order from left to right. A general *term formation* is an operation on two terms, resulting in the formation of a term t, $t\vec{r}$, $t\kappa$, $!t$, $\lambda x.t$, $t[s/x]$, or $(t[s/x])\vec{r}$, where t, x and s are any components of the two terms specified by constant numbers, and $\vec{r}$ are all the remaining components of one of the two terms specified by the number of r_1.

The algorithms nf and rf described below use a register machine model of computation, where each register may contain a term. One has an unlimited supply of registers u, v, w etc. A primitive computation *step* is any of the following operations: Test on the head form of t, and copying any component of t into a register; test on $(\lambda x.s)r\vec{r}$ with $|\mathsf{FO}(x,s)| > 1$, and formation of r and $s\vec{r}$; test on $(\lambda x.s)r\vec{r}$ with $|\mathsf{FO}(x,s)| \leq 1$, and formation of $s[r/x]\vec{r}$; test on $\mathbf{d}_\sigma t_1t_2t_3t_4\vec{r}$, and formation of t_1 and $t_jx\vec{r}$ for some $j \in \{2,3,4\}$ and variable x; test on $(!s)\kappa\vec{r}$, and formation of $s\vec{r}$.

Each of these operations can be simulated by a Turing machine in time polynomial in the length of the terms in the registers; therefore, a program which is computable in polynomial time on a register machine using polynomially-large terms in the registers, is polynomial-time computable on a Turing machine. Here one refers to an encoding of terms on the Turing machine tape.

One associates a unique *environment register* u_x with each variable x. A *numeral* is a binary numeral preceeded by any number of !'s. An *environment* is a list $\vec{n};\vec{x} := n_1,\ldots,n_k; x_1,\ldots,x_k$ where each n_i is an RS-term of the same type as x_i. A *numeral environment* is an environment $\vec{n};\vec{x}$ such that each n_i is a numeral.

Theorem 4.3. *For every $\mathcal{R}$-free* RS-*term t of ground type and numeral environment $\vec{n};\vec{x}$ such that $\mathsf{FV}(t) \subseteq \vec{x}$,*

(i) one can compute $\mathsf{nf}(t[\vec{n}/\vec{x}])$ in at most $2|t|$ steps,

(ii) the number of used registers is $\leq |t| + \#\vec{n}$, and

(*iii*) *every term* s *occurring in the computation satisfies* $|s| \leq |t| + \max |n_i|$.

Proof. We describe the algorithm nf, which at input $t, \vec{n}; \vec{x}$ outputs $\mathsf{nf}(t, \vec{n}; \vec{x}) = \mathsf{nf}(t[\vec{n}/\vec{x}])$ in the input register of t, by induction on $|t|$. For type reasons, t is of the form $U\vec{r}$ where U is either a variable among $\vec{x}$ or a constant or !, or else U is $(\lambda x.s)r$, where $|\mathsf{FO}(x, s)| \leq 1$ or x is of ground type.

If t is $\mathbf{0}$, then output $\mathbf{0}$. We have performed two steps, and (ii), (iii) are obvious.

If t is $x_i\kappa \cdots \kappa$ with k occurrences of κ, then delete k leading !'s from the content of u_{x_i} and output the resulting numeral. We have performed $k + 2 \leq 2|t|$ steps, using $2 \leq |t| + \#\vec{n}$ registers, and (iii) is obvious.

If t is Ur where U is a symbol $\mathsf{s}_1, !$, first compute $n := \mathsf{nf}(r, \vec{n}; \vec{x})$, then form Un. We have performed $\leq 2 + 2|r| \leq 2|t|$ steps, using $\leq 1 + |r| + \#\vec{n} \leq |t| + \#\vec{n}$ registers, and (iii) follows.

If t is $\mathsf{s}_0 r$, first compute $n := \mathsf{nf}(r, \vec{n}; \vec{x})$, then output $\mathbf{0}$ if $n = \mathbf{0}$, otherwise form $\mathsf{s}_0 n$. We have performed $\leq 3 + 2|r| \leq 2|t|$ steps, using $\leq 1 + |r| + \#\vec{n} \leq |t| + \#\vec{n}$ registers. As for (iii), observe $|\mathsf{s}_0 n| \leq 2 + |r| + \max |n_i| = |t| + \max |n_i|$.

Similarly, if t is $\mathbf{p}r$, first compute $n := \mathsf{nf}(r, \vec{n}; \vec{x})$, then form n' if $n = \mathsf{s}_i n'$, else output $\mathbf{0}$.

If t is $(!s)\kappa\vec{r}$, compute $\mathsf{nf}(s\vec{r}, \vec{n}; \vec{x})$, in $\leq 4 + 2|s\vec{r}| \leq 2|t|$ steps, using $\leq |s\vec{r}| + \#\vec{n}$ registers. (iii) follows directly from the induction hypothesis on $s\vec{r}$.

If t is $\mathbf{d}_\sigma t_1 t_2 t_3 t_4 \vec{r}$, first compute $n := \mathsf{nf}(t_1, \vec{n}; \vec{x})$, find a new variable x, copy n to u_x, then compute $\mathsf{nf}(t_j x\vec{r}, \vec{n}, n; \vec{x}, x)$ where $j := 2$ if $n = \mathbf{0}$, and $j := i + 3$ if $n = \mathsf{s}_i n'$. We have performed $\leq 4 + 2|t_1| + 2|t_j x\vec{r}| \leq 2|t|$ steps, using $\leq 1 + \max(|r| + \#\vec{n}, |t_j x\vec{r}| + \#\vec{n} + 1) \leq |t| + \#\vec{n}$ registers. As for (iii), observe $|\mathsf{nf}(t_j x\vec{r}, \vec{n}, n; \vec{x}, x)| \leq |t_j x\vec{r}| + \max(|n_i|, |n|) \leq |t_j x\vec{r}| + |t_1| + \max |n_i| \leq |t| + \max |n_i|$.

If t is $(\lambda x.s)r\vec{r}$ with $|\mathsf{FO}(x, s)| > 1$, then x has ground type. First compute $n := \mathsf{nf}(r, \vec{n}; \vec{x})$, copy n to u_x, then compute $\mathsf{nf}(s\vec{r}, \vec{n}, n; \vec{x}, x)$. Observe that r is of ground type, and $\vec{n}, n; \vec{x}, x$ is a numeral environment such that $\mathsf{FV}(s\vec{r}) \subseteq \vec{x}, x$. We have performed $\leq 2 + 2|r| + 2|s\vec{r}| \leq 2|t|$ steps, and (ii), (iii) follow as in the previous case.

If t is $(\lambda x.s)r\vec{r}$ with $|\mathsf{FO}(x, s)| \leq 1$, compute $\mathsf{nf}(s[r/x]\vec{r}, \vec{n}; \vec{x})$. Since $|s[r/x]\vec{r}| < |t|$, we have performed $\leq 1 + 2|s[r/x]\vec{r}| \leq 2|t|$ steps, using $\leq |s[r/x]\vec{r}| + \#\vec{n}$ registers, and (iii) is obvious. □

Corollary 4.4 (Base Normalisation). *Let* t *be a closed* $\mathcal{R}$*-free* RS*-term of ground type. Then the numeral* $\mathsf{nf}(t)$ *can be computed in at most* $2|t|$ *steps using* $\leq |t|$ *registers, and every term* s *occurring in the computation satisfies* $|s| \leq |t|$. □

In order to compute $\mathcal{R}$-free RS-terms t, we slightly generalise the technique above.

Theorem 4.5 ($\mathcal{R}$-Elimination). *Let* t *be an* RS*-term of safe or ground type. There is a polynomial* q_t *such that: if* $\vec{n}$ *are closed ground type* $\mathcal{R}$*-free* RS*-terms with* $\mathsf{FV}(t[\vec{n}/\vec{x}])$ *all safe, then one can compute an* $\mathcal{R}$*-free* RS*-term* $\mathsf{rf}(t, \vec{n}; \vec{x})$ *equivalent to* $t[\vec{n}/\vec{x}]$ *such that the number of steps, the number of used registers and the length of every term occurring in the computation all are* $\leq q_t(\sum |n_i|)$.

Proof. By induction on $|t|$. Let $m := \sum |n_i|$. We write #steps, #registers and maxlength for the three quantities above, and call their maximum *bound.* Of course, the computed term $\mathsf{rf}(t, \vec{n}; \vec{x})$ will be such that no new free variables are produced, i.e. $\mathsf{FV}(\mathsf{rf}(t, \vec{n}; \vec{x})) \subseteq \mathsf{FV}(t[\vec{n}/\vec{x}])$.

If t is $\lambda z.r$, then compute $r^* := \mathsf{rf}(r, \vec{n}; \vec{x})$ and form $t^* := \lambda z.r^*$. Observe that z and r are safe because t has safe type, hence $r[\vec{n}/\vec{x}]$ has safe free variables only. By the induction hypothesis the $\mathcal{R}$-free RS-term r^* is obtained with bound $q_r(m)$. Hence t^* is an $\mathcal{R}$-free RS-term obtained with bound $|t| + q_r(m)$.

If t is $Ur_1 \dots r_l$ with U a variable $y \neq x_i$ or one of the constants $\mathbf{0}, \mathsf{s}_0, \mathsf{s}_1, \mathbf{p}, \mathbf{d}_\sigma$, then each r_i is a safe or ground type term or else is κ. Apply the induction hypothesis to all RS-terms r_i to obtain suitable $\mathcal{R}$-free RS-terms $r_i^* := \mathsf{rf}(r_i, \vec{n}; \vec{x})$. Then form $t^* := Ur_1^* \dots r_l^*$ and rename t^* so as to obtain an RS-term. Here we need that the $\vec{n}$ are closed, for otherwise a free variable in $\vec{n}$ might be duplicated, thus violating the (S) property. Using the induction hypothesis, t^* is obtained with bound $|t| + 1 + \sum q_{r_i}(m)$.

If t is $(\lambda x.s)r\vec{r}$ with $|\mathsf{FO}(x, s)| > 1$, then x must be of ground type, since t satisfies (S). We distinguish two cases: If x is safe, we first rename t, then form r and $s\vec{r}$ (in one step), and compute $s^* := \mathsf{rf}(s\vec{r}, \vec{n}; \vec{x})$ and $r^* := \mathsf{rf}(r, \vec{n}; \vec{x})$. Finally, we form $(\lambda x.s^*)r^*$, and rename it so as to obtain an RS-term. Using the induction hypothesis the result term is obtained with bound $|t| + 6 + q_{s\vec{r}}(m) + q_r(m)$. Otherwise if x is complete, first form r and $s\vec{r}$ (in one step) and compute $n := \mathsf{rf}(r, \vec{n}; \vec{x})$, then copy n to u_x and compute $\mathsf{rf}(s\vec{r}, \vec{n}, n; \vec{x}, x)$. Using the induction hypothesis the result term is obtained with bound is $|t| + q_r(m) + q_{s\vec{r}}(m + q_r(m))$.

If t is $(\lambda x.s)r\vec{r}$ with $|\mathsf{FO}(x, s)| \leq 1$, form $t' := s[r/x]\vec{r}$ (in one step) and compute $\mathsf{rf}(t', \vec{n}; \vec{x})$. Using the induction hypothesis the result term is obtained with bound $|t| + q_{t'}(m)$.

The case $(!s)\kappa\vec{r}$ is treated similarly to the previous case, and the case $t = x_i$ is obvious.

Because t is safe, the only remaining case is where t is of the form $\mathcal{R}ghnr_1 \dots r_l$. Then we will output a renamed version of the term

$$(T_0(T_1 \dots (T_{k-1}g^*) \dots))r_1^* \dots r_l^*$$

with $g^* := \mathsf{rf}(g, \vec{n}; \vec{x})$, $k := [\![N]\!]$ where $!N := \mathsf{nf}(\mathsf{rf}(n, \vec{n}; \vec{x}))$, $T_i := \mathsf{rf}(h\kappa z, \vec{n}, !N_i; \vec{x}, z)$ for some new variable z, where N_i is obtained from N by deleting the first i leading constants s_0 or s_1, and $r_j^* := \mathsf{rf}(r_j, \vec{n}; \vec{x})$.

Since n is a complete subterm of a term satisfying (R), all free variables of n are complete. Hence $n[\vec{n}/\vec{x}]$ is closed, since all free variables of $t[\vec{n}/\vec{x}]$ are safe. Therefore, $\mathsf{nf}(n[\vec{n}/\vec{x}])$ is a numeral. One obtains $\mathsf{rf}(n, \vec{n}; \vec{x})$ with bound $\leq q_n(m)$ by the induction hypothesis. Then by Base Normalization (4.4) one obtains the numeral $!N := \mathsf{nf}(\mathsf{rf}(n, \vec{n}; \vec{x})) = \mathsf{nf}(n[\vec{n}/\vec{x}])$ with

$$\#\text{steps} \leq 2|\mathsf{rf}(n, \vec{n}; \vec{x})| \leq 2q_n(m), \quad \#\text{registers} \leq q_n(m), \quad \text{maxlength} \leq q_n(m).$$

We now compute the term $T_0(T_1 \dots (T_{k-1}g^*) \dots)$ by an obvious loop with $k \leq |N| \leq q_n(m)$ rounds. However, to obtain an estimate on our bound, we need to look into some details. First pick a new variable z and write $h\kappa z$ into a fixed register v. Then, compute $g^* := \mathsf{rf}(g, \vec{n}; \vec{x})$ with bound $\leq q_g(m)$ in a result register u, and consider the register w holding $N = N_0$ as counter. If w holds $N_i = 0$, output the content of register u, i.e. g^*. Otherwise, w holds $N_i \neq 0$ and u holds $(T_{k-i} \dots (T_{k-1}g^*) \dots)$. Compute $!N_{k-i-1}$ from N and N_i in the environment register u_z; this clearly is possible with some bound $q_1(|N|) \leq q_1(q_n(m))$ for some polynomial q_1. Compute $T_{k-i-1} := \mathsf{rf}(h\kappa z, \vec{n}, !N_{k-i-1}; \vec{x}, z)$ in v, with bound $q_{h\kappa z}(m + |N_{k-i-1}|) \leq q_{h\kappa z}(m + q_n(m))$. Update u by applying the content of v onto u's original content. This gives $T_{k-i-1}(T_{k-i} \dots (T_{k-1}g^*) \dots)$ in one step, with no additional register and maxlength increased by $|T_{k-i-1}| \leq q_{h\kappa z}(m + q_n(m))$. Finally, update w to hold N_{i+1} and go to the initial test of the loop.

Let us now estimate our bound. We go $k \leq |N| \leq q_n(m)$ times through the loop. The number of steps in each round is

$$\leq 1 + q_1(|N|) + q_{h\kappa z}(m + |N_{k-i-1}|) + 2 \leq 1 + q_1(q_n(m)) + q_{h\kappa z}(m + q_n(m)) + 2.$$

The number of register used is 3 (for v, u, u_z) plus $q_1(q_n(m))$ (to compute $!N_{k-i-1}$) plus $q_{h\kappa z}(m + q_n(m))$ (to compute T_{k-i-1}), and the maximum length of a term is

$$q_n(m) + q_{h\kappa z}(m + q_n(m)).$$

Hence the total bound for this part of the computation is

$$\big(3 + q_n(m) + q_{h\kappa z}(m + q_n(m))\big) \cdot q_n(m).$$

Finally, in a loop with l rounds, compute $r_j^* := \mathsf{rf}(r_j, \vec{n}; \vec{x})$ with bound $q_{r_j}(m)$, assuming u holds $(T_0(T_1 \dots (T_{k-1}g^*) \dots))r_1^* \dots r_{j-1}^*$, and update u by applying this term to r_j^*.

The total number of steps, used registers and lengths of used terms are therefore $\leq q_t(m)$ with a polynomial q_t explicitely definable from q_n, $q_{h\kappa z}$, q_g and all $q_{r_j^*}$. □

5 Polynomial time computable functions

In this last section we complete the proof that the number theoretical functions definable by RA-terms are exactly the polynomial-time computable functions.

Theorem 5.1 (Bounding). *Let t be a closed* RA*-term of type $\vec{\sigma} \to \iota$, where each σ_i is a ground type. Then t denotes a function computable in polynomial time on a Turing machine.*

Proof. One must find a polynomial p_t such that for all numerals $\vec{n}$ of types $\vec{\sigma}$, one can compute $\mathsf{nf}(t\vec{n})$ in time $p_t(m)$ with $m := \sum_i |n_i|$. Let $\vec{x}$ be new variables of types $\vec{\sigma}$. Since t is an RA-term, so is $t\vec{x}$ (note that t has type $\vec{\sigma} \to \iota$).

The normal form of $t\vec{x}$ is computed in a number of steps that is large but still only a constant with respect to $\vec{n}$. By closure under reduction (3.1) this is an RA-term, with only ground free variables. Note that by (4.1) all variables in $\mathsf{nf}(t\vec{x})$ are safe or ground. Since $\mathsf{nf}(t\vec{x})$ is a normal term satisfying (A), for every subterm $\lambda x.s$ with s higher type the variable x is affinable in s. Hence by Sharing (4.2) one obtains an RS-term $t' := \eta(\mathsf{nf}(t\vec{x})) \to^* t\vec{x}$. Let c be the number of steps needed to compute t'. By $\mathcal{R}$-Elimination (4.5) one obtains an $\mathcal{R}$-free RS-term $\mathsf{rf}(t', \vec{n}; \vec{x})$ equivalent to $t'[\vec{n}/\vec{x}]$ and hence to $t\vec{n}$. This requires at most $q_{t'}(m)q_{\mathsf{time}}(q_{t'}(m))$ steps, and uses at most this many registers of this size. As the output is in a register, this also bounds the length $|\mathsf{rf}(t', \vec{n}; \vec{x})|$. Using Base Normalization (4.4) one obtains $\mathsf{nf}(t\vec{n}) = \mathsf{nf}(\mathsf{rf}(t', \vec{n}; \vec{x}))$ in a total of $p_t(m) := c + 3q_{t'}(m)q_{\mathsf{time}}(q_{t'}(m))$ steps using registers of at most the same size.

The computation can be simulated on a Turing machine within a polynomial factor. □

It remains to embed the polynomial time computable functions into the system of RA-terms. One could use any of the resource-free function algebra characterizations [2, 4, 15] of the polynomial time computable functions; we pick [2].

Theorem 5.2. *Every function f computable in polynomial time on a Turing machine, is denoted by an RA term t_f.*

Proof. In [2] the polynomial time computable functions are characterized by a function algebra B based on the schemata of safe recursion and safe composition. There every function is written in the form $f(\vec{x};\vec{y})$ where $\vec{x};\vec{y}$ denotes a kind of bookkeeping of those variables $\vec{x}$ involved in a safe recursion in the definition of f, whereas $\vec{y}$ denotes those variables on which no recursion has been performed. We proceed by induction on the definition of $f(\vec{x};\vec{y})$ in B, associating to f a closed RA-term t_f of type $!\vec{\iota};\vec{\iota} \to \iota$ such that t_f is denoting f, i.e. $[\![t_f]\!] = f$.

If f is an initial function $\mathbf{0}$, $\mathbf{s}_i(;y)$ or $\mathbf{p}(;y)$, then $t_f := f$. If f is a projection $\pi_j^{m,n}$ satisfying $\pi_j^{m,n}(x_1,\ldots,x_m;x_{m+1},\ldots,x_{m+n}) = x_j$, then we define $t_f := \lambda x_1 \ldots x_{m+n}.u_j$ where $u_j := x_j\kappa$ if $j \le m$, and $u_j := x_j$ otherwise. If f is the conditional c satisfying $c(;y_1,y_2,y_3) = y_2$ if $y_1 \,\mathsf{mod}\, 2 = 0$, and $c(;y_1,y_2,y_3) = y_3$ otherwise, then $t_f := \lambda y_1 y_2 y_3.\, \mathbf{d}_\iota\, y_1\, (\lambda z.y_2)\, (\lambda z.y_2)\, (\lambda z.y_3)$ where all $\vec{y}, z$ are of type ι.

If f is defined by safe composition from $g, \vec{g}, \vec{h}$, that is, $f(\vec{x};\vec{y}) := g(\vec{g}(\vec{x};);\vec{h}(\vec{x};\vec{y}))$ where $\#\vec{g} =: m$ and $\#\vec{h} =: n$, then define $t_f := \lambda \vec{x}^{!\vec{\iota}} \vec{y}^{\vec{\iota}}.\, t_g!(t_{g_1}\vec{x}) \ldots !(t_{g_m}\vec{x})(t_{h_1}\vec{x}\vec{y}) \ldots (t_{h_n}\vec{x}\vec{y})$.

Finally, if f is defined by safe recursion from g, h_0, and h_1, then $f(0,\vec{x};\vec{y}) = g(\vec{x};\vec{y})$ and $f(\mathbf{s}_i x,\vec{x};\vec{y}) = h_i(x,\vec{x};\vec{y},f(x,\vec{x};\vec{y}))$ for $\mathbf{s}_i x \neq 0$. Using the induction hypothesis to obtain t_{h_0} and t_{h_1}, first define $t_h = \lambda n\vec{x}\vec{y}.\, \mathbf{d}\, n\, (\lambda z.0)\, (\lambda z.t_{h_0}(\mathbf{p}n)\vec{x}\vec{y})\, (\lambda z.t_{h_1}(\mathbf{p}n)\vec{x}\vec{y})$. The case is finished by defining

$$t_f := \lambda x^{!\iota} \vec{x}^{!\vec{\iota}}.\mathcal{R}_{\vec{\iota}\to\iota}\, (t_g\vec{x})\, !(\lambda u^{!\iota} V^{\vec{\iota}\to\iota} \vec{y}^{\vec{\iota}}.\, t_h u\vec{x}\vec{y}(V\vec{y}))\, x.$$

In each case one easily verifies $[\![t_f]\!] = f$. $\square$

References

[1] S. Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3-57, 1993.

[2] S.J. Bellantoni and S. Cook. A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity*, 2:97–110, 1992.

[3] S.J. Bellantoni. Characterizing parallel polylog time using type 2 recursions with polynomial output length. *Logic and Computational Complexity*, D. Leivant ed., Springer Lecture Notes in Computer Science, n. 960 (1995).

[4] S.J. Bellantoni and K.-H. Niggl. Ranking Primitive Recursions: The Low Grzegorczyk Classes Revisited. *SIAM Journal of Computing*, 1998. To appear.

[5] S.A. Cook and B.M. Kapron. Characterizations of the basic feasible functionals of finite type. *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhäuser, Boston, 1990.

[6] J.Y. Girard, A. Scedrov, and P.J. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97:1-66, 1992.

[7] J.Y. Girard. Light Linear Logic. *Information and Computation*, v. 143, 1998.

[8] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[9] D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1925.

[10] M. Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. *CSL '97*, Springer Lecture Notes in Computer Science, to appear.

[11] M. Hofmann. New results on linear/modal lambda calculus: Higher Result types and recursion on trees. Draft 1998.

[12] F. Joachimski and R. Matthes. Short proofs of normalisation for the simply-typed λ-calculus, permutative conversions and Gödel's T. Submitted, 1998

[13] D. Leivant. Subrecursion and lambda representation over free algebras. *Feasible Mathematics*, S. Buss and P. Scott, eds., Birkhäuser 1990.

[14] D. Leivant and J.Y. Marion. Ramified Recurrence and Computational Complexity IV: Predicative Functionals and Poly-space. *Information and Computation*, to appear.

[15] K.-H. Niggl. The μ-Measure as a Tool for Classifying Computational Complexity. Submitted, September 1997.

[16] H. Simmons. The Realm of Primitive Recursion. *Archive for Mathematical Logic*, 27:177–188, 1988.

[17] A.S. Troelstra and H. Schwichtenberg. Basic Proof Theory, Cambridge University Press, 1996.