æ

Primitive Recursion on the Partial Continuous Functionals

Helmut Schwichtenberg*

Dedicated to Friedrich Ludwig Bauer on the occasion of his 65th birthday**

We investigate Gödel's notion of a primitive recursive functional of higher type [5] in the context of partial continuous functionals as introduced by Kreisel in [7] and developed mainly by Scott (see [13], [3]). To make this paper readable for people not familiar with the theory of partial continuous functionals we have included a short exposition of the basic material, in a form convenient for our later arguments.

The primitive recursive functionals are defined to be the values of primitive recursive terms, which are built up from constants R_ρ denoting the primitive recursion operator and constants $[\{(u_i, v_i): i \in I\}]$ denoting finite functionals (also called finite approximations or consistent finite sets of data objects, in Scott's terminology). These constants give rise to certain new conversion rules introduced below. It is shown that, for any primitive recursive term of arbitrary type, the leftmost (or standard) reduction sequence terminates. The proof is done by transfinite induction up to ε_0; it uses a method of Howard [6] which in turn is based on earlier work of Sanchis [10] and Diller [2]. Standard machinery from Recursion Theory can then be applied to obtain bounds for the length of the leftmost reduction sequence.

The Sanchis/Diller/Howard method for termination proofs is rather flexible and can be adapted to many situations where it is of (theoretical or practical) interest to have bounds on the lengths of reduction sequences. This is in contrast to Tait's method of computability predicates (employed e. g. by Plotkin in [8]), which uses ordinary (ω–) induction on logically complex predicates and hence does not provide a comparably direct analysis of the recursions involved.

It is somewhat unusual to treat the primitive recursion operators directly when one works with the partial continuous functionals; usually they are defined from the fixed

* Part of the research reported here was done while I was visiting Carnegie–Mellon–University (Pittsburgh, Pennsylvania, USA) in the academic year 1987/88. Thanks are due to the Stiftung Volkswagenwerk for a grant which made this visit possible. I further want to thank Ulrich Berger, Steve Brookes, Daniel Leivant, Frank Pfenning, Dana Scott and Rick Statman for helpful discussions.

** This paper is based on a talk entitled "Programme und Beweise" given June 13th, 1989, in a "Kolloquium über Informatik im Kreuzungspunkt von Numerischer Mathematik, Rechnerentwurf Programmierung, Algebra und Logik" honoring F. L. Bauer. In this talk I have tried to put the result elaborated in the present paper into a broader historical perspective. In particular, I have reviewed work of Gödel, Herbrand and Kleene on recursion equations and associated termination proofs, taking into account an unpublished letter from Herbrand to Gödel. This subject was central for the early development of the theory of computability. Since what I have said on the historical development will essentially appear in [11], I did not include it in the present paper.

point operators Y. Inclusion of the general fixed point operators Y in a term system will, however, always lead to the full notion of computability, since Y corresponds to unbounded search. Hence from the present point of view to give an informative analysis of restricted notions of computability it seems appropriate to replace Y by some of its special cases. æ

1. Finite Functionals

The sets $|D_\rho|$ of partial continuous functionals of type ρ are the proper domains for computable functionals (Kreisel in [7] and Ersov in [3] give convincing arguments for this) and also for the partial primitive recursive functionals we want to study here. In Section 2 we will give a definition of the sets $|D_\rho|$, in a form convenient for our later arguments. The elements of $|D_\rho|$, i.e. the partial continuous functionals of type ρ, can be viewed as limits of certain finite functionals; such finite functionals are the subject of the present Section 1. It seems best to treat them in the context of Scott's information systems of [13].

Definition 1.1. *An information system consists of a set D of (concrete) data objects, a set* Con *of finite subsets of D such that*

$$u \subseteq v \in \mathrm{Con} \Rightarrow u \in \mathrm{Con} \tag{1.1}$$

and for any $X \in D$

$$\{X\} \in \mathrm{Con}, \tag{1.2}$$

and a reflexive and transitive relation $\sqsupseteq$ on Con *such that for all $X_1, \dots, X_m \in D$ and $u \in \mathrm{Con}$*

$$u \sqsupseteq \{X_1, \dots, X_m\} \iff u \sqsupseteq \{X_1\} \wedge \dots \wedge u \sqsupseteq \{X_m\}. \tag{1.3}$$

Note that (1.3) implies that from $u \sqsupseteq v_1, \dots, u \sqsupseteq v_m$ we can conclude $v := v_1 \cup \dots \cup v_m \in \mathrm{Con}$ and $u \sqsupseteq v \sqsupseteq v_i$ for $i = 1, \dots, m$. The $u \in \mathrm{Con}$ are called *consistent finite sets of data objects*, or just (finite) *approximations*. $u \sqsupseteq v$ is read as "*u extends v*".

Our basic information system is D_ι, whose data objects are the natural numbers $0, 1, 2, \dots$, whose approximations are the singletons $\{0\}, \{1\}, \{2\}, \dots$ together with the empty set $\emptyset$, and whose extension relation $\sqsupseteq$ is just the set theoretic inclusion $\supseteq$. Similarly we construct the information system D_o based on the boolean data objects `ff` and `tt`.

Given information systems D and E, we now construct a new information system $D \to E$, as in [13]. Its data objects are the pairs (u, v) with $u \in \mathrm{Con}_D$ and $v \in \mathrm{Con}_E$. A finite set $\{(u_i, v_i) \colon i \in I\}$ of data objects is consistent in $D \to E$ if

$$\forall I' \subseteq I (\bigcup_{i \in I'} u_i \in \mathrm{Con}_D \Rightarrow \bigcup_{i \in I'} v_i \in \mathrm{Con}_E). \tag{1.4}$$

In order to define the extension relation $\sqsupseteq$ for $D \to E$ we first define the result of an *application* of $W = \{(u_i, v_i) \colon i \in I\} \in Con_{D \to E}$ to $u \in \mathrm{Con}_D$:

$$\{(u_i, v_i) \colon i \in I\} u := \bigcup \{v_i \colon u \sqsupseteq u_i\}. \tag{1.5}$$

Then by (1.4) we know that $Wu \in \mathrm{Con}_E$. Obviously, application is monotone in the second argument, i.e.

$$u \sqsupseteq u' \Rightarrow Wu \sqsupseteq Wu'. \tag{1.6}$$

Now define $W \sqsupseteq W'$ by

$$W \sqsupseteq \{(u'_j, v'_j) : j \in J\} :\Longleftrightarrow \forall j \in J. W u'_j \sqsupseteq v'_j. \tag{1.7}$$

Lemma 1.2. *If D and E are information systems, then so is $D \to E$.*

Proof. We first show the transitivity of $\sqsupseteq$. So let

$$W \sqsupseteq \{(u'_j, v'_j) : j \in J\} \sqsupseteq \{(u''_k, v''_k) : k \in K\}.$$

Then we have for all $k \in K$ by (1.6) and (1.7)

$$W u''_k \sqsupseteq \bigcup \{W u'_j : u''_k \sqsupseteq u'_j\} \sqsupseteq \bigcup \{v'_j : u''_k \sqsupseteq u'_j\} \sqsupseteq v''_k.$$

It remains to show (1.3) for $D \to E$. Since $\Rightarrow$ is obvious we only deal with $\Leftarrow$. So let $\{(u_i, v_i) : i \in I\} \sqsupseteq \{(u'_j, v'_j)\}$ for all $j \in J$. It suffices to show that $\{(u'_j, v'_j) : j \in J\}$ is consistent. So assume $J' \subseteq J$ and $\bigcup_{j \in J'} u'_j \in \mathrm{Con}_D$. By (1.4) we have to show that $\bigcup_{j \in J'} v'_j \in \mathrm{Con}_E$. But this follows from

$$\bigcup \{v_i : \bigcup_{j \in J'} u'_j \sqsupseteq u_i\} \sqsupseteq \bigcup \{v_i : u'_j \sqsupseteq u_i\} \sqsupseteq v'_j.$$

This concludes the proof. $\square$

Note that with the above definition of the extension relation $\sqsupseteq$ in $D \to E$ application is also monotone in the first argument, i.e.

$$W \sqsupseteq W' \Rightarrow W u \sqsupseteq W' u. \tag{1.8}$$

To see this observe that

$$W u \sqsupseteq \bigcup \{W u'_j : u \sqsupseteq u'_j\} \sqsupseteq \bigcup \{v'_j : u \sqsupseteq u'_j\} = W' u.$$

We will exclusively deal with the information systems built up from D_ι and D_o by the $\to$-operation. More formally, define the notion of a *type symbol* and its *level* inductively by the clauses

1. ι and o are type symbols, and $\mathrm{lev}(\iota) = \mathrm{lev}(o) = 0$.
2. If ρ and σ are type symbols, then so is $(\rho \to \sigma)$, and $\mathrm{lev}((\rho \to \sigma)) = \max(\mathrm{lev}(\rho) + 1, \mathrm{lev}(\sigma))$.

As usual we write $\rho_1, \dots, \rho_m \to \sigma$ for $(\rho_1 \to (\rho_2 \to \dots (\rho_m \to \sigma) \dots))$. Note that any type symbol can be written uniquely in the form $\rho_1, \dots, \rho_m \to \iota$ or $\rho_1, \dots, \rho_m \to o$. For any type symbol ρ define the information system D_ρ as follows. D_ι and D_o have already been defined, and $D_{\rho \to \sigma} := D_\rho \to D_\sigma$. The D_ρ are called *standard information systems*.

Note that for standard information systems the exponential test (1.4) for consistency of a finite set of data objects can be replaced by a quadratic test. To see this call an information system *coherent* (see Plotkin [9, p. 210]) if for any finite set $\{X_i : i \in I\}$ of data objects

$$\forall i, j \in I. \{X_i, X_j\} \in \mathrm{Con} \Rightarrow \{X_i : i \in I\} \in \mathrm{Con}. \tag{1.9}$$

Obviously D_ι and D_o are coherent. Now the coherence of all standard information systems D_ρ follows from

Lemma 1.3. *If D and E are information systems and E is coherent, then so is $D \to E$.*

Proof. Let $\{(u_i, v_i): i \in I\}$ be finite and assume

$$\forall i, j \in I.\{(u_i, v_i), (u_j, v_j)\} \in \mathrm{Con}_{D \to E}. \tag{1.10}$$

We have to show $\{(u_i, v_i): i \in I\} \in \mathrm{Con}_{D\to E}$. So, by (1.4), assume $I' \subseteq I$ and $\bigcup_{i \in I'} u_i \in \mathrm{Con}_D$. We have to show $\bigcup_{i \in I'} v_i \in \mathrm{Con}_E$. Now since E is coherent by assumption, it suffices to show $v_i \cup v_j \in \mathrm{Con}_E$ for all $i, j \in I'$. So let $i, j \in I'$. By assumption we have $u_i \cup u_j \in \mathrm{Con}_D$ and hence by (1.10) and the definition of $\mathrm{Con}_{D \to E}$ also $v_i \cup v_j \in \mathrm{Con}_E$. $\square$

The elements of $\mathrm{Con}_\rho := \mathrm{Con}_{D_\rho}$, i.e. the consistent finite sets of data objects or approximations in D_ρ will be called *finite functionals* of type ρ. They are of central importance for our study. In Section 2 they will be used to define the partial continuous functionals as limits of finite functionals. Finite functionals will also be special partial primitive recursive functionals. Furthermore, the primitive recursive terms to be defined in Section 3 to denote partial primitive recursive functionals will contain (constants for) such finite functionals, and consequently they take part in the definition of conversion for such terms, to be given in Section 4.

These conversion rules require at certain points to form the union of finite functionals. This (set theoretic) union will however keep data objects whose information is superseded by others. From the point of view of achieving a reasonably effective implementation of e.g. the leftmost (or standard) reduction strategy it seems necessary to eliminate such superfluous data objects. This can be done as follows.

Write $u \sim u'$ (u is *equivalent* to u') if $u \sqsupseteq u' \sqsupseteq u$. Call any finite functional in D_ι and D_o in normal form, and define a finite functional $\{(u_i, v_i): i \in I\} \in D_{\rho\to\sigma}$ to be in *normal form* if all u_i, v_i are in normal form and for all $i, j \in I$ with $i \neq j$ we have

1. $u_i \not\sim u_j$.
2. $v_j \sqsupset \bigcup\{v_i: u_j \sqsupset u_i\}$.

Here $v \sqsupset v'$ means $v \sqsupseteq v' \not\sqsupseteq v$. Now it is not hard to prove the following (see [12, p.91–92]):

Theorem 1.4. *We have a quadratic algorithm which assigns to any finite functional $w \in \mathrm{Con}_{\rho\to\sigma}$ an equivalent finite functional w^* in normal form. This normal form is uniquely determined, i.e. any two finite functionals in normal form which are equivalent must be equal (as sets).*

The normal forms of finite functionals can be used to extend the natural well–ordering of the finite functionals of the ground types ι and o ($\emptyset$, $\{0\}$, $\{1\}$, $\{2\}, \dots$ and $\emptyset$, $\{\mathtt{ff}\}$, $\{\mathtt{tt}\}$) to higher types, as follows.Assuming that Con_ρ and Con_σ are already well–ordered, first observe — using Theorem 1.4 — that any finite functional of type $\rho \to \sigma$ can be written uniquely as a list of pairs (u, v) with $u \in \mathrm{Con}_\rho$ and $v \in \mathrm{Con}_\sigma$, where the pairs are listed according to their lexicographic order. Now those lists can again be ordered lexicographically. For example, $\wedge_{\mathrm{strict}} \in \mathrm{Con}_{o\to(o\to o)}$ is given by the following list of pairs of 1. constants for finite functionals of type o and 2. lists of pairs of constants for finite functionals of type o:

$$((\mathtt{ff}, ((\mathtt{ff}, \mathtt{ff}), (\mathtt{tt}, \mathtt{ff}))), (\mathtt{tt}, ((\mathtt{ff}, \mathtt{ff}), (\mathtt{tt}, \mathtt{tt}))))$$

and $\wedge_{\mathrm{nonstrict}} \in \mathrm{Con}_{o\to(o\to o)}$ is given by

$$((\bot^o, ((\mathtt{ff}, \mathtt{ff}))), (\mathtt{ff}, ((\bot^o, \mathtt{ff}))), (\mathtt{tt}, ((\mathtt{ff}, \mathtt{ff}), (\mathtt{tt}, \mathtt{tt})))),$$

where all pairs appear in their natural order as just defined. These well–orderings will be used in Section 4 to make our conversion rules unique.

2. Limits of Finite Functionals

We now give the definition (due to Scott [13]) of the partial continuous functionals of type ρ, in a form suitable for our later arguments. They are taken as limits (or, more precisely, as ideals) of finite functionals.

Definition 2.1. *An ideal x in an information system D (written $x \in |D|$) is a set x of data objects which is consistent in the sense that any finite subset of x is in Con_D, and closed against $\sqsupseteq$, i. e. if $u \sqsupseteq \{X\}$ for some finite subset u of x, then $X \in x$.*

The crucial fact about ideals in $D \to E$ is that they can be identified with *continuous* functions from $|D|$ to $|E|$, defined as follows.

Definition 2.2. *Let $D_1, \ldots, D_m := \vec{D}$ and E be information systems. A function $f\colon |\vec{D}| \to |E|$ is called continuous if it is monotone, i. e. for all $\vec{x}, \vec{y} \in |\vec{D}|$*

$$\vec{x} \subseteq \vec{y} \Rightarrow f(\vec{x}) \subseteq f(\vec{y}) \tag{2.1}$$

and satisfies the approximation property, i. e. for all $\vec{x}, \vec{y} \in |\vec{D}|$ and $v \in \mathrm{Con}_E$

$$v \subseteq f(\vec{x}) \Rightarrow \exists \vec{u} \in \mathrm{Con}_{\vec{D}} (\vec{u} \subseteq \vec{x} \wedge v \subseteq f(\overline{\vec{u}})), \tag{2.2}$$

where $\overline{u}$ denotes the closure of u under $\sqsupseteq$, i. e. $\overline{u} := \{X\colon u \sqsupseteq \{X\}\}$.

It is well known that this notion of continuity is the same as the ordinary one with respect to the *Scott–topologies* of $|\vec{D}|$ and $|E|$, defined as follows. For any consistent (finite or infinite) set y of data objects in an information system D let

$$\check{y} := \{x \in |D| : x \supseteq y\}$$

Then $\{\check{u}\colon u \in \mathrm{Con}_D\}$ is the basis of a T_0–topology (the Scott–topology) on $|D|$, which has the properties

$$\exists u \in \mathrm{Con}_D . x = \overline{u} \iff \check{x} \text{ is open}$$

and

$$x \subseteq y \iff x \in \{y\}^- \iff \forall u \in \mathrm{Con}_D (x \in \check{u} \Rightarrow y \in \check{u}).$$

For the proofs we refer the reader to [13].

We now show how ideals in $D \to E$ can be considered as continuous functions from $|D|$ to $|E|$.

Lemma 2.3. *Let D and E be information systems. To any ideal $z \in |D \to E|$ we can associate a continuous function*

$$\mathrm{fct}_z\colon |D| \to |E|$$

by

$$\mathrm{fct}_z(x) := zx := \bigcup \{v : \exists u \subseteq x . (u, v) \in z\}. \tag{2.3}$$

Also, to any continuous function $f\colon |D| \to |E|$ we can associate an ideal

$$\mathrm{ideal}(f) \in |D \to E|$$

by

$$\mathrm{ideal}(f) := \{(u, v) : v \subseteq f(\overline{u})\}. \tag{2.4}$$

The assignments given by (2.3) and (2.4) are inverse to each other.

The proof is rather straightforward and not given here.

The ideals $x \in |D_\rho|$ are called *partial continuous functionals* of type ρ. Application of $z \in |D_{\rho\to\sigma}|$ to $x \in |D_\rho|$ is given by (2.3). Note that this application operation zx is continuous in both arguments, in the sense of Definition 2.2.

An important consequence of the identification of ideals $z \in |D \to E|$ with continuous functions from $|D|$ to $|E|$ given in Lemma 2.3 is the following *extensionality* property

Lemma 2.4. *Let D and E be information system and $z, z' \in |D \to E|$. Then from $z\overline{u} \subseteq z'\overline{u}$ for all $u \in \mathrm{Con}_D$ we can conclude $z \subseteq z'$.*

Proof. $z = \mathrm{ideal}(\mathrm{fct}_z) = \{(u, v) : v \subseteq z\overline{u}\}$. □

Hence the sets $|D_\rho|$ of partial continuous functionals together with the application operators given by (2.3) form a *pre–structure* in the sense of Friedman [4, p. 23]. (Statman calls this a *frame* in [14, p. 331])

Any $z \in |D \to (E \to F)|$ can be viewed as a binary function $\mathrm{fct}_z^2 : |D|, |E| \to |F|$ defined by

$$\mathrm{fct}_z^2(x, y) := \mathrm{fct}_{\mathrm{fct}_z(x)}(y).$$

We want to characterize the functions which can be obtained in this way. It turns out that these are exactly the binary continuous functions in the sense of Definition 2.2 [1]. This is a consequence of Lemma 2.3 together with the following

Lemma 2.5. *Let $D_1, \dots, D_m, E$ and F be information systems. To any continuous $f\colon |D_1|, \dots, |D_m|, |E| \to |F|$ we can associate a continuous*

$$f_-\colon |D_1|, \dots, |D_m| \to |E \to F|$$

by

$$f_-(x_1, \dots, x_m) = \mathrm{ideal}(f(x_1, \dots, x_m, \cdot)), \tag{2.5}$$

where $f(x_1, \dots, x_m, \cdot)\colon |E| \to |F|$ is defined by $f(x_1, \dots, x_m, \cdot)(y) = f(x_1, \dots, x_m, y)$. Also to any continuous $g : |D_1|, \dots, |D_m| \to |E \to F|$ we can associate a continuous

$$g_+\colon |D_1|, \dots, |D_m|, |E| \to |F|$$

by

$$g_+(x_1, \dots, x_m, y) = \mathrm{fct}_{g(x_1,\dots,x_m)}(y). \tag{2.6}$$

The assignments given by (2.5) and (2.6) are inverse to each other.

Proof. Monotonicity and the approximation property can be verified easily, for f_- as well as for g_+. Furthermore, we have

$$(f_-)_+(\vec{x}, y) = \mathrm{fct}_{f_-(\vec{x})}(y) = \mathrm{fct}_{\mathrm{ideal}(f(\vec{x},\cdot))}(y) = f(\vec{x}, y)$$

and

$$(g_+)_-(\vec{x}) = \mathrm{ideal}(g_+(\vec{x}, \cdot)) = \mathrm{ideal}(\mathrm{fct}_{g(\vec{x})}) = g(\vec{x}),$$

where, in both cases, the last equation follows from Lemma 2.3. □

---

[1] This is the essential step in proving that information systems with continuous functions as morphisms form a cartesian closed category.

We now show that the sets $|D_\rho|$ of partial continuous functionals together with the application operators (2.3) form a *model* of the typed λ–calculus (or a *structure*, in the terminology of Friedman [4, p. 23])[2].

The *terms*, their *types* and their sets of *free variables* are given by

1. Any variable x_i^ρ $(i = 0, 1, 2, \dots)$ is a term of type ρ, $\mathrm{FV}(x_i^\rho) = x_i^\rho$.
2. If r is a term of type σ, then $\lambda x_i^\rho.r$ is a term of type $\rho \to \sigma$, $\mathrm{FV}(\lambda x_i^\rho.r) = \mathrm{FV}(r)\backslash\{x_i^\rho\}$.
3. If t is a term of type $\rho \to \sigma$ and s is a term of type ρ, then (ts) is a term of type σ, $\mathrm{FV}((ts)) = \mathrm{FV}(t) \cup \mathrm{FV}(s)$.

We write $ts_1s_2 \dots s_m$ for $(\dots((ts_1)s_2)\dots s_m)$, and $\lambda x_1x_2\dots x_m.r$ for $\lambda x_1.\lambda x_2.\dots\lambda x_m.r$. A term is called *closed* if $\mathrm{FV}(r) = \emptyset$.

For any term r of type σ and any list $\vec{x}$ of variables of types $\vec{\rho}$ containing all the variables free in r we define a continuous function

$$|\vec{x} \mapsto r|\colon |D_{\rho_1}|, \dots, |D_{\rho_m}| \to |D_\sigma|$$

by induction on r, as follows.

1. $|\vec{x} \mapsto x_i|$ is the i–th projection function, which is clearly continuous.
2. $|\vec{x} \mapsto \lambda y.r| := |\vec{x}, y \mapsto r|\_$ (cf. Lemma 2.5).
3. $|\vec{x} \mapsto ts|$ is the result of substituting the functions $|\vec{x} \mapsto t|$ and $|\vec{x} \mapsto s|$ in the continuous binary application function. Clearly the resulting function is continuous.

Now we can define the *value* of a term r with free variables among $\vec{x}$ under an assignment of partial continuous functionals $\vec{\mathbf{x}}$ to the variables $\vec{x}$ to be just $|\vec{x} \mapsto r|\vec{\mathbf{x}}$. In particular, for any closed term r of type ρ we have defined its value $|r| \in |D_\rho|$. æ

3. Primitive Recursion

There are far more continuous functions $f\colon |D_{\vec{\rho}}| \to |D_\sigma|$ than just those given by terms r of the typed λ–calculus. Of special importance (and much studied in the literature, e.g. in [13], [3], [8]) are the *computable* ones, which by definition are those given by recursively enumerable ideals of finite functionals (in $|D_{\vec{\rho}\to\sigma}|$). Here we want to deal with more restricted (and hence better to analyze) notions of computability. As a paradigm we consider Gödel's famous notion of a primitive recursive functional and adapt it to our present context, i.e. we define what it means for a continuous function $f\colon |D_{\vec{\rho}}| \to |D_\sigma|$ to be primitive recursive. Other restricted notions of computability (like polynomial time computability) can be treated similarly.

Now what are primitive recursive functions $f\colon |D_{\vec{\rho}}| \to |D_\sigma|$? It seems best to define them by means of an extension of the notion of a term in the typed λ–calculus.

For any type symbol ρ, let countably many variables x_i^ρ be given. Also, for any finite functional $u \in \mathrm{Con}_\rho$, we introduce a constant $[u]^\rho$. The constant $[m]^\iota$ (denoting the numeral m) is abbreviated by m^ι or just m, and the constant $[\emptyset]^\rho$ (denoting the totally undefined finite functional in Con_ρ) is abbreviated by $\bot^\rho$. We further introduce a constant N for the successor function. Finally, for any type symbol ρ, we introduce a recursion constant R_ρ. The *primitive recursive terms*, their *types* and their sets of *free variables* are given by

[2] This holds generally for cartesian closed categories.

1a. x_i^ρ is a primitive recursive term of type ρ, $\mathrm{FV}(x_i^\rho) = x_i^\rho$.
1b. $[u]^\rho$ is a primitive recursive term of type ρ, $\mathrm{FV}([u]^\rho) = \emptyset$.
1c. N is a primitive recursive term of type $\iota \to \iota$, $\mathrm{FV}(N) = \emptyset$.
1d. R_ρ is a primitive recursive term of type $\iota, \rho, (\iota, \rho \to \rho) \to \rho$, $\mathrm{FV}(R_\rho) = \emptyset$.
2. If r is a primitive recursive term of type σ, then $\lambda x_i^\rho.r$ is a primitive recursive term of type $\rho \to \sigma$, $\mathrm{FV}(\lambda x_i^\rho.r) = \mathrm{FV}(r) \setminus \{x_i^\rho\}$.
3. If t is a primitive recursive term of type $\rho \to \sigma$ and s is a primitive recursive term of type ρ, then (ts) is a primitive recursive term of type σ, $\mathrm{FV}((ts)) = \mathrm{FV}(t) \cup \mathrm{FV}(s)$.

In order to define the *value* of a primitive recursive term we first have to define a value $|R_\rho| \in |D_{\iota,\rho,(i,\rho\to\rho)\to\rho}|$ for each recursion constant R_ρ. This can be done as follows. For any nonnegative integer m, define a function

$$h_m \colon |D_\rho|, |D_{\iota,\rho\to\rho}| \to |D_\rho|$$

by

$$h_0(y, z) = y$$

$$h_{m+1}(y, z) = z\{m\}(h_m(y, z)).$$

Clearly each h_m is continuous , by induction on m. Now define a function

$$f \colon |D_\iota| \to |D_{\rho,(\iota,\rho\to\iota)\to\rho}|$$

by

$$f(\emptyset) = \emptyset$$

$$f(\{m\}) = (h_m)_{--},$$

using Lemma 2.5. Obviously f is continuous. Finally let $|R_\rho| := f_-$. We then have $|R_\rho|_{+++}(x, y, z) = f_{++}(x, y, z)$, and from the definition of f we obtain

$$|R_\rho|_{+++}(\emptyset, y, z) = \emptyset,$$

$$|R_\rho|_{+++}(\{0\}, y, z) = h_0(y, z) = y,$$

$$|R_\rho|_{+++}(\{m+1\}, y, z) = h_{m+1}(y, z) = z\{m\}(|R_\rho|_{+++}(\{m\}, y, z)).$$

Exactly as for terms of the typed λ–calculus (in Section 2) we can now define, for any primitive recursive term r of type σ and any list $\vec{x}$ of variables of types ρ containing all the variables free in r, a continuous function $|\vec{x} \mapsto r| \colon |D_{\rho_1}|, \ldots, |D_{\rho_m}| \to |D_\sigma|$, by induction on r. Just add, in clause 1, that $|\vec{x} \mapsto [u]^\rho|$ is the constant function with value $\overline{u} \in |D_\rho|$, $|\vec{x} \mapsto N|$ is the constant function with value the successor function $\in |D_{\iota\to\iota}|$ (which is clearly continuous), and $|\vec{x} \mapsto R_\rho|$ is the constant function with value $|R_\rho| \in |D_{\iota,\rho,(\iota,\rho\to\rho)\to\rho}|$ defined above.

Now we define the *value* of a primitive recursive term r with the free variables among $\vec{x}$ under an assignment of partial continuous functionals $\vec{\mathbf{x}}$ to the variables $\vec{x}$ to be just $|\vec{x} \mapsto r|\vec{\mathbf{x}}$. In particular, for any closed primitive recursive term r of type ρ we have defined its value $|r| \in |D_\rho|$. Let

$$|D_\rho|^{pr} := \{|r| \colon r \text{ closed primitive recursive term of type } \rho\} \subseteq |D_\rho|.$$

The elements of $|D_\rho|^{pr}$ are called *partial primitive recursive functionals*. Note that any element of $|D_{\rho\to\sigma}|^{pr}$ can be viewed — via Lemma 2.5 — as a continuous function $|D_\rho| \to |D_\sigma|$.

It seems worthwile to also note that any partial primitive recursive functional when viewed as a function $f\colon |D_\rho| \to |D_\sigma|$ is defined on all of $|D_\rho|$, i. e. on all partial continuous functionals of type ρ, not just on the subset $|D_\rho|^{pr}$. This seems to be desirable, since e. g. a primitive recursive operation on the reals like the exponential function e^x should be defined on arbitrary Cauchy sequences of rationals, not just on the primitive recursive ones.

The sets $|D_\rho|^{pr} \subseteq |D_\rho|$ are closed against application, since the primitive recursive terms are. Now consider any system $\{\mathfrak{M}_\rho\}$ of sets satisfying $|D_\rho|^{pr} \subseteq \mathfrak{M}_\rho \subseteq |D_\rho|$ and closed against application. By Lemma 2.4 we know that the extensionality condition holds for $\{\mathfrak{M}_\rho\}$, i. e. if $x, y \in \mathfrak{M}_{\rho\to\sigma}$ and $\forall z \in \mathfrak{M}_\rho(xz = yz)$ then $x = y$. Hence the sets $\mathfrak{M}_\rho$ form a pre-structure in the sense of Friedman. They also form a model of the typed λ–calculus, since the $|D_\rho|^{pr}$ do. We view such structures $\{\mathfrak{M}_\rho\}$ as the intended models of theories involving primitive recursive terms.

æ

4. Conversion

We now want to show that any primitive recursive term can be "computed", i.e. transformed by repeated conversions into a normal form where no further conversions are possible. The primitive recursive terms may contain variables and need not be of level 0. If, however, a primitive recursive term is of level 0 and is closed, then the normal form must be a numeral (i.e. a constant for a finite functional in D_ι or D_o) and hence we have computed the numerical value of the term.

Our conversion rules certainly contain *β–conversion*

$$(\lambda x.r)s \mapsto r[s].$$

We do not need α–conversion (i.e. renaming of bound variables) since, following de-Bruijn [1], we can (and do in our implementation) replace bound variables by references to the binding place. We also do not use η–conversion $\lambda x.rx \mapsto r$, for three reasons. First, for closed primitive recursive terms of level 0 η–conversions are not neccessary to transform them into a numeral. Second, for closed primitive recursive terms of type ρ with $\mathrm{lev}(\rho) > 0$ their value $|r| \in |D_\rho|$ is already determined by the values $|r[u_1]\ldots[u_m]|$ of the closed level–0–terms $r[u_1]\ldots[u_m]$ for all finite functionals u_i, which can be computed without η–conversion. Closedness of r can be assumed since we can always λ–abstract free variables. Third, for arbitrary primitive recursive terms we even prefer e.g. $z(\lambda\vec{y}.x\vec{r}\vec{y})$ with $x\vec{r}\vec{y}$ of level 0 over $z(x\vec{r})$, since in the first term the "minimal type" (this corresponds to Gentzen's notion of minimal formula under the Curry–Howard–isomorphism) is of level 0, and this has advantages in certain proof–theoretic arguments.

A special difficulty arises from our inclusion of constants $[\{(u_i, v_i) \colon i \in I\}]$ for finite functionals. We must be able to convert e.g. $[\{(u_i, v_i) \colon i \in I\}]r$, and the result should have as its value the supremum of all v_i with i such that the value of r extends u_i. In order to deal with this difficulty we extend our notion of a primitive recursive term by two more term–forming operations given below.

Here we prove termination of the leftmost (or standard) reduction strategy for primitive recursive terms, by means of transfinite induction of length ε_0. Our proof is by an adaption of a method of Howard [6] to the present situation; ultimately this technique is based on ideas of Sanchis [10] and Diller [2].

We define the *extended primitive recursive terms* r, their *types* and their sets of *free variables* and simultaneously for any list $\vec{x}$ of variables containing all the variables free in r, a continuous function $|\vec{x} \mapsto r|$, by induction on r. The clauses 1a – d, 2 and 3 are as in Sections 2 and 3. We add two more clauses.

1e. Let $s_1, \ldots, s_m$ be extended primitive recursive terms of type o and $v_1, \ldots, v_m \in \mathrm{Con}_\sigma$ $(m \geq 1)$. Assume that for all $\vec{u} \in \mathrm{Con}_{\vec{\rho}}$

$$\bigcup \{v_i \colon |\vec{x} \mapsto s_i|\overline{\vec{u}} = \{\mathtt{tt}\}\} \in \mathrm{Con}_\sigma \tag{4.1}$$

Then $(([v_1]^\sigma \text{ if } s_1) \cup \ldots \cup ([v_m]^\sigma \text{ if } s_m))$ is an extended primitive recursive term of type σ, $\mathrm{FV}((([v_1]^\sigma \text{ if } s_1) \cup \ldots \cup ([v_m]^\sigma \text{ if } s_m))) = \bigcup_{i=1}^m \mathrm{FV}(s_i)$, and

$$|\vec{x} \mapsto (([v_1]^\sigma \text{ if } s_1) \cup \ldots \cup ([v_m]^\sigma \text{ if } s_m))|\vec{\mathbf{x}} = \overline{\bigcup \{v_i \colon |\vec{x} \mapsto s_i|\vec{\mathbf{x}} = \{\mathtt{tt}\}\}}.$$

1f. For any $u \in \mathrm{Con}_\rho$, $(\sqsupseteq [u]^\rho)$ is an extended primitive recursive term of type $\rho \to o$, $\mathrm{FV}((\sqsupseteq [u]^\rho)) = \emptyset$ and $|\vec{x} \mapsto (\sqsupseteq [u]^\rho)|$ is the constant function whose value is the continuous function which takes $\mathbf{x} \in |D_\rho|$ into $\{\mathtt{tt}\} \in |D_o|$ if $\mathbf{x} \supseteq u$, and into $\emptyset \in |D_o|$ otherwise.

For some extended primitive recursive terms, we now define the result of the *conversion* of r, by cases according to the form of r.

a. $(\lambda x.r)s$ converts into $r[s]$.
b. $[\{(u_1, v_1), \ldots, (u_m, v_m)\}]^{\rho \to \sigma} r$ converts into

$$(([v_1]^\sigma \text{ if } (\sqsupseteq [u_1]^\rho)r) \cup \ldots \cup ([v_m]^\sigma \text{ if } (\sqsupseteq [u_m]^\rho)r))$$

c. Nm converts into $m+1$, and $N\bot$ converts into $\bot$.
d. $R0st$ converts into s, $R(m+1)st$ converts into $tm(Rmst)$, and $R\bot st$ converts into $\bot$.
e. $(([v_1]^\sigma \text{ if } b_1) \cup \ldots \cup ([v_m]^\sigma \text{ if } b_m))$ with $b_1, \ldots, b_m \in \{\bot^0, \mathtt{ff}, \mathtt{tt}\}$ (i. e., constants for finite functionals $\in \mathrm{Con}_o$) converts into $[v_{i_1} \cup \ldots \cup v_{i_m}]^\sigma$, where $b_{i_1}, \ldots, b_{i_n}$ are all constants among $b_1, \ldots, b_m$ being $\mathtt{tt}$.
f. $(\sqsupseteq [\{(u, v)\} \cup W]^{\rho \to \sigma})r$ converts into

$$[\wedge_{\text{strict}}]^{o \to (o \to o)}((\sqsupseteq [v]^\sigma)(r[u]^\rho))((\sqsupseteq [W]^{\rho \to \sigma})r).$$

To make this rule unique, we require that (u, v) is the first pair in $\{(u, v)\} \cup W$ (in the well–ordering given in Section 1). $(\sqsupseteq \bot)r$ converts into $\mathtt{tt}$. $(\sqsupseteq m)n$ converts into $\mathtt{tt}$ if $m = n$, and into $\mathtt{ff}$ otherwise. $(\sqsupseteq m)\bot$ converts into $\bot$. $(\sqsupseteq \mathtt{tt})\mathtt{tt}$, $(\sqsupseteq \mathtt{ff})\mathtt{ff}$ convert into $\mathtt{tt}$, $(\sqsupseteq \mathtt{tt})\mathtt{ff}$, $(\sqsupseteq \mathtt{ff})\mathtt{tt}$ convert into $\mathtt{ff}$, and $(\sqsupseteq \mathtt{tt})\bot$, $(\sqsupseteq \mathtt{ff})\bot$ convert into $\bot$.

Observe that the conversion rules do not lead out of the class of extended primitive recursive terms, and also preserve the value. Let us verify this for Rules b and f. For Rule b, we must prove (4.1), i. e.

$$\bigcup \{v_i \colon |\vec{x} \mapsto (\sqsupseteq [u_i]^\rho)r|\vec{\mathbf{x}} = \{\mathtt{tt}\}\} \in \mathrm{Con}_\sigma.$$

Now this follows from

$$|\vec{x} \mapsto (\sqsupseteq [u_i]^\rho) r|\vec{\mathbf{x}} = \{\mathtt{tt}\} \iff |\vec{x} \mapsto r|\vec{\mathbf{x}} \supseteq u_i,$$

which also implies the preservation of the value. For Rule f, it suffices to prove that the original term has value $\{\mathtt{tt}\}$ under an assignment of $\vec{\mathbf{x}}$ to $\vec{x}$ iff the resulting term has. This can be seen by an easy computation, using

$$|\vec{x} \mapsto r|\vec{\mathbf{x}} \supseteq \{(u, v)\} \iff |\vec{x} \mapsto r[u]|\vec{\mathbf{x}} \supseteq v.$$

Now let r be an arbitrary extended primitive recursive term. r is called *reducible* if it contains a subterm which is convertible according to the rules above. If r is reducible, we can choose the leftmost one among all convertible subterms and convert it; the result is denoted by $\mathrm{lred}(r)$. The sequence

$$r, \mathrm{lred}(r), \mathrm{lred}(\mathrm{lred}(r)), \ldots$$

is called the *leftmost* (or *standard*) reduction sequence for r. We will show that this sequence terminates, for any primitive recursive term r.

Clearly reducing an extended primitive recursive term does not change its value (since conversion doesn't, and the value of a compound term is defined by means of the values of its constituents only). Hence the final element r^* of the standard reduction sequence for r is a term with the same value as r which does not contain any convertible subterms. If, in particular, r is closed and of level 0, then also r^* is closed and hence must be a constant m, $\bot^\iota$, $\mathtt{tt}$, $\mathtt{ff}$ or $\bot^o$, i. e. we have computed the value of r.

For any primitive recursive term r of level 0, we define a relation $|r|_k \leq \alpha$ (to be read r *has a tree of degree* k *with height* $\leq \alpha$) inductively, by the following rules.

Rule 1. If $|r[s]\vec{t}|_k \leq \alpha_0 < \alpha$, then $|(\lambda x.r)s\vec{t}|_k \leq \alpha$.

Rule 2. If $|t_i\vec{y}_i|_k \leq \alpha_i < \alpha$ for $i = 1, \ldots, m$, then $|xt_1 \ldots t_m|_k \leq \alpha$. In particular, $|x|_k \leq \alpha$ for any α.

Rule 3. If $|r[w_1] \ldots [w_n]|_k \leq \alpha_{w_1 \ldots w_n} < \alpha$ for all full argument sequences (see below) $w_1, \ldots, w_n$ in any of the u_i $(i \in I)$, and $|[\bigcup_{i \in I'} v_i]\vec{t}|_k \leq \alpha_{I'} < \alpha$ for all $I' \subseteq I$ such that $\bigcup_{i \in I'} v_i$ is consistent, then $|[\{(u_i, v_i) \colon i \in I\}]r\vec{t}|_k \leq \alpha$. In particular, $|\bot r\vec{t}|_k \leq \alpha$ for any α. Also, $|c|_k \leq \alpha$ for any constant c of level 0 and any α.

Rule 4. If $|r|_k \leq \alpha_0 < \alpha$, then $|Nr|_k \leq \alpha$.

Rule 5. If $|r|_k \leq \alpha' < \alpha$ and $|Rmst\vec{t}|_k \leq \alpha_m < \alpha$ for all nonnegative integers m, and if r is not a constant of type ι, then $|Rrst\vec{t}|_k \leq \alpha$.

Rule 6. If $|s\vec{t}|_k \leq \alpha_0 < \alpha$, then $|R0st\vec{t}|_k \leq \alpha$.

Rule 7. If $|tm(Rmst)\vec{t}|_k \leq \alpha_0 < \alpha$, then $|R(m+1)st\vec{t}|_k \leq \alpha$.

Rule 8. $|R\bot st\vec{t}|_k \leq \alpha + 1$ for any α.

k–Abbreviation–Rule. If $|r\vec{y}|_k \leq \alpha_0 < \alpha$ and $|t_i\vec{y}_i|_k \leq \alpha_i < \alpha$ for $i = 1, \ldots, m$ $(m \geq 1)$, and furthermore $\mathrm{lev}(r) \leq k$, then $|rt_1 \ldots t_m|_k \leq \alpha$.

The notion of a *full argument sequence* for a finite functional u in normal form is defined inductively, as follows. Any u of level 0 has no full argument sequences, and for $\{(u_1, v_1), \ldots, (u_m, v_m)\}$ the full argument sequences are all sequences $u_iw_{i1} \ldots w_{in}$ where $w_{i1} \ldots w_{in}$ is a full argument sequence for v_i.

We write $|r|_k < \alpha$ to mean that we know a $\beta < \alpha$ such that $|r|_k \leq \beta$.

Now let r be a primitive recursive term, of arbitrary level. We first show that, for sufficiently big k, it is easy to estimate the height of a tree of degree k for $r\vec{y}$.

Lemma 4.1. *For any variable x we have, with an arbitrary k,*

$$|x\vec{y}|_k \leq \operatorname{lev}(x). \tag{4.2}$$

Now let r be any primitive recursive term. If all subterms of r have levels $\leq k$, then

$$|r\vec{y}|_k < \omega \cdot 2. \tag{4.3}$$

Proof. (4.2) can be seen easily by induction on the level of x:

$$|y_i\vec{z_i}|_k \leq \operatorname{lev}(y_i) < \operatorname{lev}(x)$$

for $i = 1, \ldots, m$ by induction hypothesis , and hence

$$|xy_1 \ldots y_m|_k \leq \operatorname{lev}(x).$$

(4.3) is proven by induction on the height (i. e., the number of symbols) of r.

Case x. The claim follows from (4.2).

Case $[u]$. We show $|[u]y\vec{y}|_k < \omega$, by induction on the length of u. This is obvious (by Rule 3) if u is of level 0. Otherwise, $u = \{(u_i, v_i) : i \in I\}$ and we have $|[w_i]\vec{y_i}| < \omega$ for $i = 1, \ldots, m$ by induction hypothesis and hence $|y[w_1] \ldots [w_m]|_k < \omega$ by Rule 2. By Rule 3 we can conclude $|[\{(u_i, v_i) : i \in I\}]y\vec{y}|_k < \omega$.

Case N. We have $|Ny|_k \leq 1$, since $|y|_k \leq 0$.

Case R. We first show $|Rmyz\vec{y}|_k < \omega$, by induction on m. Since $|y\vec{y}|_k \leq \operatorname{lev}(y)$ by (4.2) we have $|R0y\vec{z}\vec{y}|_k \leq \operatorname{lev}(y) + 1 < \omega$ by Rule 6. For the induction step, assume $|Rmyz\vec{y}|_k < \omega$. Then (by Rule 2 and (4.2)) we also have $|zm(Rmyz)\vec{y}|_k < \omega$ and hence $|R(m+1)yz\vec{y}|_k < \omega$ by Rule 7. We now show $|Rxyz\vec{y}|_k \leq \omega$. This follows from $|Rmyz\vec{y}|_k < \omega$ for all m and $|x|_k \leq 0$ by Rule 5.

Case $\lambda x.r$. By induction hypothesis we have $|r[y]\vec{y}|_k < \omega \cdot 2$, and hence $|(\lambda x.r)y\vec{y}|_k < \omega \cdot 2$ by Rule 1.

Case rs. By induction hypothesis we have $|ryy_1 \ldots y_m|_k < \omega \cdot 2$ and $|s\vec{z}|_k < \omega \cdot 2$, and by (4.2) $|y_i\vec{z_i}|_k \leq \operatorname{lev}(y_i) < \omega$ for $i = 1, \ldots, m$. By the k–Abbreviation–Rule we can conclude $|rsy_1 \ldots y_m|_k < \omega \cdot 2$. □

We now want to show how from an estimate for the height of a tree of degree k we can obtain an estimate for the height of the tree of degree 0.

Lemma 4.2. *Let $r, s_1, \ldots, s_m$ be primitive recursive terms with* $\operatorname{lev}(r) = 0$. *If $|r|_k \leq \alpha$ and $|s_j\vec{y_j}|_k \leq \beta$ for $j = 1, \ldots, m$, and if $s_1, \ldots, s_m$ have levels $\leq k$, then*

$$|r_{x_1,\ldots,x_m}[s_1, \ldots, s_m]|_k \leq \beta + \alpha.$$

Proof. We use induction on the generation of $|r|_k \leq \alpha$, and — for readability — write t^* for $|t_{x_1,\ldots,x_m}[s_1, \ldots, s_m]|$.

Rule 1. $|r^*[s^*]\vec{t^*}|_k \leq \beta + \alpha_0 < \beta + \alpha$ by induction hypothesis, hence $|(\lambda x.r^*)s^*\vec{t^*}|_k \leq \beta + \alpha$ by Rule 1.

Rule 2. $|t_i^*\vec{y_i^*}|_k \leq \beta + \alpha_i < \beta + \alpha$ for $i = 1, \ldots, m$ by induction hypothesis, hence $|xt_1^* \ldots t_m^*|_k \leq \beta + \alpha$. Now if x is one of the variables x_j to be substituted by s_j, we must use the k–Abbreviation–Rule instead of Rule 2. This is possible since by hypothesis s_j has level $\leq k$, and since also by hypothesis $|s_j\vec{z_j}|_k \leq \beta$. Then (if $m > 0$) the k–Abbreviation–Rule yields $|s_jt_1^* \ldots t_m^*|_k \leq \beta + \alpha$, as required. In case $m = 0$ there are no t_i's and we have used Rule 2 to generate $|x_j|_k \leq \alpha$. But then $|s_j|_k \leq \beta + \alpha$ holds by hypothesis.

Rule 3. $|r^*[w_1] \ldots [w_m]|_k \leq \beta + \alpha_{w_1 \ldots w_m} < \beta + \alpha$ for all $w_1, \ldots, w_m$ and $|[\bigcup_{i \in I'} v_i]\vec{t^*}|_k \leq \beta + \alpha_{I'} < \beta + \alpha$ for all $I' \subseteq I$ such that $\bigcup_{i \in I'} v_i$ is consistent by induction hypothesis, hence $|[\{(u_i, v_i) : i \in I\}]r\vec{t}|_k \leq \beta + \alpha$ by Rule 3.

Rules 4–8 and the *k–Abbreviation–Rule* can be treated similarly (i. e. as Rules 1 and 3); the claim always follows by induction hypothesis and the same rule. □

Lemma 4.3. *Let r be a primitive recursive term of level 0. If $|r|_{k+1} \leq \alpha$, then $|r|_k \leq 2^\alpha$.*

Proof. We again use induction on the generation of $|r|_{k+1} \leq \alpha$. The only case where the claim does not follow immediately from the induction hypothesis is where $|rt_1 \dots t_m|_{k+1} \leq \alpha$ was generated from $|r\vec{y}|_{k+1} \leq \alpha_0 < \alpha$ and $|t_i\vec{y_i}|_{k+1} \leq \alpha_i < \alpha$ for $i = 1, \dots, m$ by the k–Abbreviation–Rule, and $\mathrm{lev}(r) \leq k+1$. By induction hypothesis we then have $|r\vec{y}|_k \leq 2^{\alpha_0}$ and $|t_i\vec{y_i}|_k \leq 2^{\alpha_i}$ for $i = 1, \dots, m$. Using Lemma 4.2 we can conclude $|rt_1 \dots t_m|_k \leq 2^{\max(\alpha_1,\dots,\alpha_m)} + 2^{\alpha_0} \leq 2^\alpha$. □

We now show that, from an estimate α for the height of the tree of degree 0 of a primitive recursive term $r\vec{y}$, we can conclude — by an application of transfinite induction of the same length α — that the leftmost reduction sequence for r terminates.

Lemma 4.4. *Let r be a primitive recursive term of arbitrary level. If $|r\vec{y}|_0 \leq \alpha$, then the leftmost (or standard) reduction sequence for r terminates.*

Proof. We use induction on the generation of $|r\vec{y}|_0 \leq \alpha$, and write $t\downarrow$ to mean that the leftmost reduction sequence for t terminates.

If r is a variable or a constant, then the claim is trivial.

Case $\lambda x.r$. Then $|[\lambda x.r]y\vec{y}|_0 \leq \alpha$ was generated from $|r[y]\vec{y}|_0 \leq \alpha_0 < \alpha$ by Rule 1. By induction hypothesis we know $r[y]\downarrow$, and this obviously implies $(\lambda x.r)\downarrow$.

Case $xt_1 \dots t_m$. Then $|xt_1 \dots t_m\vec{y}|_0 \leq \alpha$ was generated from $|t_i\vec{y_i}|_0 \leq \alpha_i < \alpha$ and $|y_j\vec{z_j}|_0 \leq \beta_j < \alpha$ by Rule 2. By induction hypothesis we know $t_i\downarrow$, and this obviously implies $(xt_1 \dots t_m)\downarrow$.

Case $[\{(u_i, v_i) : i \in I\}]r\vec{t}$. Then $|[\{(u_i, v_i) : i \in I\}]r\vec{t}\vec{y}|_0 \leq \alpha$ was generated by Rule 3 from $|r[w_1] \dots [w_m]|_0 \leq \alpha_{w_1 \dots w_m} < \alpha$ for all full argument sequences $w_1, \dots, w_m$ in any of the u_i $(i \in I)$, and from $|[\bigcup_{i \in I'} v_i]\vec{t}\vec{y}|_0 \leq \alpha_{I'} < \alpha$ for all $I' \subseteq I$ such that $\bigcup_{i \in I'} v_i$ is consistent. By induction hypothesis we know $(r[w_1] \dots [w_m])\downarrow$ for all $w_1, \dots, w_m$, and also $([\bigcup_{i \in I'} v_i]\vec{t})\downarrow$ for all $I' \subseteq I$ such that $\bigcup_{i \in I'} v_i$ is consistent. Now from the definition of the leftmost reduction procedure it can be seen easily that this implies $([\{(u_i, v_i) : i \in I\}]r\vec{t}))\downarrow$.

Case Nr. Then $|Nr|_0 \leq \alpha$ was generated from $|r|_0 \leq \alpha_0 < \alpha$ by Rule 4. By induction hypothesis we know $r\downarrow$, and this obviously implies $(Nr)\downarrow$.

Case $Rrst\vec{t}$ with r not a constant. Then $|Rrst\vec{t}\vec{y}|_0 \leq \alpha$ was generated by Rule 5 from $|r|_0 \leq \alpha' < \alpha$ and $|Rmst\vec{t}\vec{y}|_0 \leq \alpha_m < \alpha$ for all nonnegative integers m. By induction hypothesis we know $r\downarrow$ and $(Rmst\vec{t})\downarrow$ for all m, and this obviously implies $(Rrst\vec{t})\downarrow$.

Case Rrs with r not a constant. Then $|Rrsz\vec{y}|_0 \leq \alpha$ was generated by Rule 5 from $|r|_0 \leq \alpha' < \alpha$ and $|Rmsz\vec{y}|_0 \leq \alpha_m < \alpha$ for all nonnegative integers m. We also know that $|R0sz\vec{y}|_0 \leq \alpha_0$ was generated by Rule 6 from $|s\vec{y}|_0 \leq \alpha'_0 < \alpha_0$. By induction hypothesis we know $r\downarrow$ and $s\downarrow$, and this clearly implies $(Rrs)\downarrow$.

Case Rr with r not a constant. Then $|Ryz\vec{y}|_0 \leq \alpha$ was generated by Rule 5 from $|r|_0 \leq \alpha' < \alpha$ and $|Rmyz\vec{y}|_0 \leq \alpha_m < \alpha$ for all nonnegative integers m. By induction hypothesis we know $r\downarrow$, and this clearly implies $(Rr)\downarrow$.

Case $R0st\vec{t}$. Then $|R0st\vec{t}\vec{y}|_0 \leq \alpha$ was generated from $|s\vec{t}\vec{y}|_0 \leq \alpha_0 < \alpha$ by Rule 6. By induction hypothesis we know $(s\vec{t})\downarrow$. Since $\mathrm{lred}(R0st\vec{t}) = s\vec{t}$, we can conclude $(R0st\vec{t})\downarrow$.

Case $R(m+1)st\vec{t}$. Then $|R(m+1)st\vec{t}\vec{y}|_0 \leq \alpha$ was generated from $|tm(Rmst)\vec{t}\vec{y}|_0 \leq \alpha_0 < \alpha$ by Rule 7. By induction hypothesis we know $(tm(Rmst)\vec{t})\downarrow$. Since $\mathrm{lred}(R(m+1)st\vec{t}) = tm(Rmst)\vec{t}$, we can conclude $(R(m+1)st\vec{t})\downarrow$.

Case Rms. We first show, by induction on m, that in the generation of $|Rmsz\vec{y}_m|_0 \leq \beta_m$ there must occur $|R0sz\vec{y}_0|_0 \leq \beta_0$ for some $\beta_0 \leq \beta_m$. For $m = 0$ there is nothing to show, and for $m > 0$ $|Rmsz\vec{y}_m|_0 \leq \beta_m$ was generated from $|z(m-1)(R(m-1)sz)\vec{y}_m|_0 \leq \beta'_m < \beta_m$ by Rule 7, and this in turn was generated from (among others) $|R(m-$

$1)sz\vec{y}_{m-1}|_0 \leq \beta_{m-1} < \beta'_m$, whence by induction hypothesis our claim follows.—Hence from $|Rmsz\vec{y}|_0 \leq \alpha$ we can infer $|R0sz\vec{y}|_0 \leq \alpha$. But $|R0sz\vec{y}|_0 \leq \alpha$ was generated from $|s\vec{y}|_0 \leq \alpha_0 < \alpha$ by Rule 6. By induction hypothesis we know $s\downarrow$, and this obviously implies $(Rms)\downarrow$.

Case Rm. This term does not contain a convertible subterm, and hence the claim is trivial.

Cases $R\bot st\vec{t}$, $R\bot s$, $R\bot$. The leftmost reduction sequences for all these terms clarly terminate with $\bot$.

Case $(\lambda x.r)s\vec{t}$. Then $|(\lambda x.r)s\vec{t}\vec{y}|_0 \leq \alpha$ was generated from $|r[s]\vec{t}\vec{y}|_0 \leq \alpha_0 < \alpha$ by Rule 1. By induction hypothesis we know $(r[s]\vec{t})\downarrow$. Since $\mathrm{lred}((\lambda x.r)s\vec{t}) = r[s]\vec{t}$, we can conclude $((\lambda x.r)s\vec{t})\downarrow$. □

To summarize, we have proved the following result.

Theorem 4.5. *Let r be a primitive recursive term of arbitrary level, possibly containing free variables and constants for finite functionals. Then we can find an ordinal $\alpha < \varepsilon_0$ (by the constructions in Lemmas 4.1 and 4.3) such that the termination of the leftmost (or standard) reduction sequence for r is provable in elementary arithmetic (more precisely: primitive recursive arithmetic in Skolem's sense) plus tranfinite induction up to α.*

By a well–known result this implies that the length of the leftmost reduction sequence for r is bounded by an α–recursive function in the length of r.

æ

Bibliography

1. de Bruijn, N. G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. Indagationes Math. *34*, 381–392 (1972)

2. Diller, J.: Zur Berechenbarkeit primitiv–rekursiver Funktionale endlicher Typen. In K. Schütte (ed.): Contributions to Mathematical Logic. North–Holland, Amsterdam 1968, pp. 109–120

3. Ershov, Yu. L.: Model C of partial continuous functionals. In R. Gandy and M. Hyland (eds.): Logic Colloquium 1976. North Holland, Amsterdam 1977, pp. 455–467

4. Friedman, H.: Equality between functionals. In R. Parikh (ed.): Logic Colloquium, Lecture Notes in Math 453. Springer, Berlin 1975, pp. 22–37

5. Gödel, K.: Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. Dialectica *12*, 280–287 (1958)

6. Howard, W. A.: Ordinal analysis of terms of finite type. The Journal of Symbolic Logic, *45* (3), 493–504 (1980)

7. Kreisel, G.: Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting (ed.): Constructivity in Mathematics. North Holland, Amsterdam 1959, pp. 101–128

8. Plotkin, G. D.: LCF considered as a programming language. Theoretical Computer Science. *5*, 223–255 (1977)

9. Plotkin, G. D.: $\mathbf{T}^{\omega}$ as a universal domain. Journal of Computer and System Sciences *17*, 209–236 (1978)

10. Sanchis, L. E.: Functionals defined by recursion. Notre Dame Journal of Formal Logic *8*, 161–174 (1967)

11. Schütte, K., Schwichtenberg, H.: Mathematische Logik. To appear in W. Scharlau (ed.): "100 Jahre DMV", Vieweg, Braunschweig 1990

12. Schwichtenberg, H.: Eine Normalform für endliche Approximationen von partiellen stetigen Funktionalen. In J. Diller (ed.): Logik und Grundlagenforschung, Festkolloquium zum 100. Geburtstag von Heinrich Scholz. Aschendorff, Münster 1986, pp. 89–95

13. Scott, D. S.: Domains for denotational semantics. In M. Nielsen, E. M. Schmidt (eds.): Automata, Languages and Programming. Lecture Notes in Computer Science 150, Springer, Berlin 1982, pp. 577–613

14. Statman, R.: Equality between functionals revisited. In L. A. Harrington et al. (eds.): Harvey Friedman's Research on the Foundations of Mathematics. North Holland, Amsterdam 1985, pp. 331–338
æ