A Simple Ordinal Recursive Normalization of Gödel's T

Paul J. Voda

Institute of Informatics, Comenius University Bratislava,
Mlynská dolina, 842 15 Bratislava, Slovakia,
email: voda@fmph.uniba.sk homepage: www.fmph.uniba.sk/~voda/

1 Introduction

Primitive recursive functionals of finite types, also known as Gödel's T (see [4,11]) in logic, and as typed lambda calculus with primitive recursion in higher types in computer science, are quite important in both areas. In the former they are used in the proof-theoretic studies of Peano arithmetic, in the latter they can be used to give the formal semantics to the modern kind of (functional) computer programming languages.

The reduction of typed terms of T which denote the functionals can be used to give the so-called operational semantics (the way to compute) to the functions defined in a functional programming language. It has been for a long time a matter of folklore in logic that the (typed) terms normalize (i.e. reduce to unique irreducible terms) and this has been also proved many times and in many different ways since. Perhaps the simplest proof is by the method of the so-called strong computability predicates of Tait [16,10,3]. From such a proof we can extract a recursive normalization function $N(\ulcorner t \urcorner)$ operating on codes of terms.

We can get more information about the nature of the normalization function by a finer analysis of the normalization process by means of ordinals. It turns out that the function is ε_0-recursive. The ordinal analysis of T and of its fragments T_n (where the recursion is restricted to types of level n) is again quite important in both logic and computer science. In the former we can use it to obtain the proof-theoretic strength of Peano arithmetic and of its fragments (where induction is restricted to the formulas of certain quantifier complexity). In the latter we can analyze the computational complexity of functions defined in a functional programming language. In this respect the fragment T_0 is of extreme importance as it can define exactly the primitive recursive functions.

The ordinal analysis of T is done mainly in two different ways. The first one is the method of reduction of infinite terms (to replace the recursors) and it was introduced by Tait [16] and studied by many since then (see for instance [12,8]). The second one is by a direct assignment of ordinals to the terms (the ordinals are decreased by reductions) and it was introduced by Howard [7] and studied by many since then (see for instance [11,18]).

Unfortunately, both methods are quite complicated. In order to show that an infinite term can be effectively reduced and then read off by an ε_0-recursive function requires non-trivial coding techniques involving Kleene's recursion theorem. The method of infinite terms has been recently greatly simplified by Schwichtenberg [13,14] based on an idea of Buchholz [2] but it is still quite complex.

The assignment of ordinals in the style of Howard suffers greatly in that it is very difficult to grasp the intuition behind it. Moreover, the original method does not assign the ordinals optimally for the fragments T_n and it requires a great technical effort to correct the situation. Witness the recent work of Weiermann [18] in this respect.

As can be seen from this brief overview, the ordinal analysis of Gödel's T attracted many researchers and we feel that it still lacks a relatively simple and intuitively obvious treatment. In this paper we present a contribution to this long list which is in the intuitively straightforward reduction style of Tait but avoids the need for the infinite terms. This is done by a trick which comes quite naturally to computer programmers where one uses auxiliary data structures to control a complicated computational process. We extend T with 'delayed' terms to steer the reduction process into certain subterms of a term to be reduced while temporarily stopping the process in others. We claim that our method is quite straightforward. Moreover, it yields an optimal assignment of ordinals to the fragments of T. How to achieve the last with infinite terms is not obvious at all as the normalization function is nested and the removal of the nested recursion increases exponentially the ordinals involved (see the treatment in Rose [9] pp. 101-108). Mints [8] used delaying of reductions but in the context of infinite terms.

2 Typed Terms

2.1 Typed terms. Typed terms of *Gödel's* T denote in the well-known way primitive recursive functionals of finite types (see [4]). The *types* are composed from the type of natural numbers 0 by $\alpha \mapsto \beta$ which groups to the right. The *level* of a type is defined as $Lev(0) := 0$ and $Lev(\alpha \mapsto \beta) := \max(Lev(\alpha) + 1, Lev(\beta))$. The set of (typed) *terms*, where we write $t : \alpha$ if the term t is of type α, is built up from the *zero numeral* $\bar{0} : 0$, the *successor functional* $S : 0 \mapsto 0$, and from the *variables* $v_i^\alpha : \alpha$, by *abstractions*

$\lambda v_i^\alpha t : \alpha \mapsto \beta$ where $t : \beta$, *recursors* $R^\alpha r : (0 \mapsto \alpha \mapsto \alpha) \mapsto \alpha \mapsto \alpha$ where the term $r : 0$ is called the *count* term, and by *applications* $ts : \beta$ where $t : \alpha \mapsto \beta$ and $s : \alpha$ for some α.

We write t, s, r for terms and x, y, z for variables. We drop the type superscripts in the variables and recursors when they can be recovered from the context or are immaterial. Application groups to the left, i.e. tsr abbreviates $(ts)r$. We let $\bar{t}$ stand for a possibly empty sequence of terms $t_1 t_2 \cdots t_n$. For a natural number n we denote by $\bar{n}$ the *numeral* term of type 0 which denotes n. Here $\overline{n+1} := S\bar{n}$. The term $R\bar{0}ts$ denotes the same functional as does s and the term $R\overline{n+1}ts$ the same functional as does $t\bar{n}(R\bar{n}ts)$.

We write $Tp(t)$ for the type of the term t. We abbreviate $Lev\ Tp(t)$ to $Lev(t)$. The *rank* $Rk(t)$ of a term t is the maximum of $Lev(\lambda x t_0)$ for all subterms $(\lambda x t_0)s$ of t. For terms t and $s : \alpha$, as well as a variable x^α, we denote by $t_x[s]$ the *substitution* function yielding a term obtained by replacing every free x in t by s. We assume that the substitution renames the bound variables if needed. We abbreviate $(t_x[s_1])_y[s_2]$ to $t_x[s_1]_y[s_2]$.

2.2 Reductions. For typed terms we consider two kinds of *reductions*. The usual *β-reduction* $(\lambda x t)s \Rightarrow t_x[s]$ and the *R-reduction*:

$$R\bar{n}ts \Rightarrow t\overline{n-1}(t\overline{n-2}(\cdots(t\bar{0}s)\cdots))$$

where we abbreviate the last term by $t^n s$ (note that $t^0 s$ is just s). The terms on the left-hand-side of reductions are called *redexes*. The relation $t \rhd s$ of t *reduces to* s is defined as usual by replacing finitely many times (including 0) and starting with t of a redex by its corresponding right-hand-side until we obtain s. A term t is *normal* if there are no redexes in it. Note that if $t \rhd s$ then $Tp(t) = Tp(s)$. It follows from the Church-Rosser property (see [3]) that terms possess at most one normal form. In Thm. 4.4 we will prove that every term has at least one normal form. It is still conceivable that for some term t there is an infinite reduction sequence. That this is not a case, i.e. that terms *strongly* normalize, is usually proved by the computability predicates of Tait [16,3].

2.3 Rank reduction. We can eliminate all β-redexes from a term t by the iteration $Red^{Rk(t)}(t)$ of the well-known reduction function $Red(t)$. This will normalize a term without recursors. The function Red was (implicitly) introduced by Tait ([16] see also [14]) and it lowers the rank of the term t.

It is defined by:

$$\begin{aligned}
Red(t) &:= t \qquad \text{if } t \text{ is } \bar{0},\ S, \text{ or } x \\
Red(\lambda x t) &:= \lambda x\, Red(t) \\
Red(Rr) &:= R\, Red(r) \\
Red(ts) &:= \begin{cases} (t_0)_x[Red(s)] \text{ if } Red(t) = \lambda x t_0 \\ Red(t)\, Red(s) \text{ otherwise} \end{cases}
\end{aligned}$$

As it is well known, we have $t \triangleright Red(t)$ and $Rk\; Red(t) \leq Rk(t) \dot{-} 1$.

3 Delayed Recursors

3.1 Rank reduction with infinite terms. The reduction function cannot be directly used for the elimination of R-redexes. This is because when eventually an R-redex $R^\alpha \bar{n} t s$ is formed there might not be sufficiently many iterations of *Red* pending to eliminate the β-redexes in the term $t^n s$ where new β-redexes of rank $Lev(t) = Lev(\alpha) + 1$ can be formed when $t = \lambda x t_0$. To overcome this difficulty Tait [16] introduced infinite terms where instead of $Rrts\bar{s} : 0$ one has $\{t^i s\bar{s}\}_i r : 0$ with reductions $\{t^i s\bar{s}\}_i \bar{n} \Rightarrow t^n s\bar{s}$. The reduction and substitution functions can be now extended to infinite terms by reducing and substituting in each term $t^i s\bar{s}$ independently. The problem is that despite recent improvements by Schwichtenberg [14,13] it is still not trivial to demonstrate that the reduction process operating on infinite objects is constructive, more precisely that the reduced infinite term can be evaluated by a $<\!\varepsilon_0$-recursive function.

3.2 Delayed recursors. We can manage the rank reduction process without infinite terms by delaying the reduction of a term $R^\alpha r\bar{t}$ until its count term r normalizes. For that we need a recursor capable of delaying the reduction and substitution functions. We therefore extend the class of typed terms into *extended-terms* (e-terms for short) with *delayed recursors*: $\langle r : \bar{t} : d\rangle^\alpha$. Here r is a term of type 0 playing the role of the count term. The sequence of terms $\bar{t}$ is such that $R^\alpha r\bar{t}$ is well-typed. For a technical definition of well-typed delayed recursor this means that $\bar{t}$ is either empty, or of the form t with $t : 0 \mapsto \alpha \mapsto \alpha$, or of the form ts with t as before and $s : \alpha$ or, of the form $tss_1 \cdots s_k$ with t, s as before and $s_i : \alpha_i$ where α is $\alpha_1 \mapsto \alpha_2 \mapsto \cdots \mapsto \alpha_k \mapsto \beta$. We leave to the reader the extension along these lines of the function Tp to be over e-terms and such that $Tp(\langle r : \bar{t} : d\rangle^\alpha) = Tp(R^\alpha r\bar{t})$ holds.

The last component of delayed recursors d is a possibly empty $(\emptyset)$ finite sequence of *markers* signifying the delayed operations of reduction and substitution. Without ambiguity we will use as markers the symbols Red and ${}_x[s]$ themselves.

We will define in Par. 3.3 a translation of e-terms into terms which can be viewed as assigning meaning to e-terms and we will extend in paragraphs 3.5 and 3.7 respectively the functions $Red(t)$ and $Rk(t)$ to e-terms The substitution function is extended to e-terms by the addition of the clause:

$$\langle r : \bar{t} : d\rangle_x[s] := \langle r_x[s] : \bar{t} : d_x[s]\rangle \ .$$

The reader will note how the substitution proceeds to the count term r while it is delayed by the addition of its marker to the sequence d.

The idea of normalization with delayed recursors can be demonstrated with an example e-term $\langle r : ts\bar{s} : Red_x[t_1]Red_y[t_2]\rangle$ whose translation is a term t_3. We will first normalize the translation of the count term r into a numeral $\bar{n}$. We will then apply the delayed reductions and substitutions to the e-term $t^n s\bar{s}$ by $Red(Red(t^n s\bar{s})_x[t_1])_y[t_2]$. Let the translation of the last e-term be t_4. We will have $t_3 \triangleright t_4$.

3.3 Translation of e-terms to terms. The operation t^* translating an e-term t into a term is defined by recursion on the construction of t simultaneously with the translation d^* of a sequence of markers to a composition of substitution operations: $\cdot_x[s]$ defined by recursion on the length of d to satisfy:

$$\begin{aligned}
t^* &:= t \qquad \text{if t is } 0,\ S, \text{ or } x \\
(\lambda x t)^* &:= \lambda x t^* \\
(R^\alpha r)^* &:= R^\alpha r^* \\
\langle r : t_1 \cdots t_k : d\rangle^{\alpha *} &:= R^\alpha r^* (t_1^* d^*) \cdots (t_k^* d^*) \\
(ts)^* &:= t^* s^*
\end{aligned}$$

$$\begin{aligned}
\emptyset^* &:= \emptyset \qquad \textit{identity} \\
(dRed)^* &:= d^* \\
(d_x[s])^* &:= d_x^*[s^*] \qquad \textit{composition}\ .
\end{aligned}$$

Note that for a term t we have $t^* = t$.

3.4 Translation lemma. *For all e-terms t and s we have*

$$(t_x[s])^* = t^*_x[s^*] \ . \tag{1}$$

Proof. By induction on the structure of t. This is straightforward if t is 0, S, or a variable (whether x or a different one). If t is $\lambda y t_0$ (we can always rename the bound variable to be different from x) then

$$\begin{aligned}((\lambda y t_0)_x[s])^* = (\lambda y (t_0)_x[s])^* = \lambda y ((t_0)_x[s])^* &\overset{IH}{=} \\ \lambda y (t^*_0)_x[s^*] = (\lambda y t^*_0)_x[s^*] = (\lambda y t_0)^*_x[s^*] \ .\end{aligned}$$

The cases when t is Rr or $t_1 t_2$ are similar. If t is $\langle r : t_1 \cdots t_k : d\rangle$ then

$$\begin{aligned}&(\langle r : t_1 \cdots t_k : d\rangle_x[s])^* = \langle r_x[s] : t_1 \cdots t_k : d_x[s]\rangle^* = \\ &\quad R(r_x[s])^*(t^*_1(d_x[s])^*) \cdots (t^*_k(d_x[s])^*) \overset{IH}{=} \\ &\quad R(r^*_x[s^*])(t^*_1 d^*_x[s^*]) \cdots (t^*_k d^*_x[s^*]) = \\ &\quad (Rr^*(t^*_1 d^*) \cdots (t^*_k d^*))_x[s^*] = \langle r : t_1 \cdots t_k : d\rangle^*_x[s^*] \ .\end{aligned}$$

□

3.5 Extension of the reduction function. The functions $Red(t)$ is extended to e-terms by

$$\begin{aligned}Red(t) &:= t \qquad \text{if } t \text{ is } \bar{0}, S, \text{ or } x \\ Red(\lambda x t) &:= \lambda x Red(t) \\ Red(Rr) &:= R Red(r) \\ Red(\langle r : \bar{t} : d\rangle) &:= \langle Red(r) : \bar{t} : d Red\rangle \\ Red(ts) &:= \begin{cases}(t_0)_x[Red(s)] & \text{if } Red(t) = \lambda x t_0 \\ \langle r : \bar{t}x : d_x[Red(s)]\rangle & \text{if } Red(t) = \langle r : \bar{t} : d\rangle \text{ and } x \text{ is new} \\ Red(t) Red(s) & \text{otherwise} \ .\end{cases}\end{aligned}$$

The reader will note how the reduction proceeds to the count term r of a delayed recursor while it is delayed by the addition of its marker to the sequence d.

The reduction and substitution functions delayed in a sequence d are applied to an e-term t by the function $Park(t, d)$ yielding e-terms. The function is defined by recursion on the length of d:

$$\begin{aligned}Park(t, \emptyset) &:= t \\ Park(t, d Red) &:= Red\, Park(t, d) \\ Park(t, d_x[s]) &:= Park(t, d)_x[s] \ .\end{aligned}$$

3.6 Reduction lemma. *For every e-term t and a sequence of markers d we have*

$$t^* \rhd Red(t)^* \tag{1}$$
$$t^* d^* \rhd Park(t, d)^* \; . \tag{2}$$

Proof. (1): By induction on the structure of t. Trivial if t is 0, S, or x. If t is $\lambda x t_0$ then

$$(\lambda x t_0)^* = \lambda x t_0^* \stackrel{IH}{\rhd} \lambda x \, Red(t_0)^* = (\lambda x \, Red(t_0))^* = Red(\lambda x t_0)^* \; .$$

The case when t is Rr is similar. If t is $\langle r : t_1 \cdots t_k : d\rangle$ then

$$\begin{aligned}\langle r : t_1 \cdots t_k : d\rangle^* = Rr^*(t_1^* d^*) \cdots (t_k^* d^*) &\stackrel{IH}{\rhd} \\ R\, Red(r)^*(t_1^*(d Red)^*) \cdots (t_k^*(d Red)^*) &= \\ \langle Red(r) : t_1 \cdots t_k : d Red\rangle^* &= Red(\langle r : t_1 \cdots t_k : d\rangle)^* \; .\end{aligned}$$

If t is $s_1 s_2$ then we consider three subcases. If $Red(s_1) = \lambda x s_0$ then

$$\begin{aligned}(s_1 s_2)^* = s_1^* s_2^* \stackrel{IH}{\rhd} Red(s_1)^* Red(s_2)^* = (\lambda x s_0^*) \, Red(s_2)^* \rhd \\ (s_0^*)_x[Red(s_2)^*] \stackrel{3.4(1)}{=} ((s_0)_x[Red(s_2)])^* = Red(s_1 s_2)^* \; .\end{aligned}$$

If $Red(s_1) = \langle r : t_1 \cdots t_k : d\rangle$ then

$$\begin{aligned}(s_1 s_2)^* = s_1^* s_2^* &\stackrel{IH}{\rhd} Red(s_1)^* Red(s_2)^* = \langle r : t_1 \cdots t_k : d\rangle^* Red(s_2)^* = \\ &Rr^*(t_1^* d^*) \cdots (t_k^* d^*) \; Red(s_2)^* = Rr^*(t_1^* d^*) \cdots (t_k^* d^*)(x_x[Red(s_2)^*]) = \\ &Rr^*(t_1^* d_x^*[Red(s_2)^*]) \cdots (t_k^* d_x^*[Red(s_2)^*])(x^* d_x^*[Red(s_2)^*]) = \\ &Rr^*(t_1^*(d_x[Red(s_2)])^*) \cdots (t_k^*(d_x[Red(s_2)])^*)(x^*(d_x[Red(s_2)])^*) = \\ &\langle r : t_1 \cdots t_k x : d_x[Red(s_2)]\rangle^* = Red(s_1 s_2)^* \; .\end{aligned}$$

If $Red(s_1)$ is neither a lambda nor a delayed recursor then

$$(s_1 s_2)^* = s_1^* s_2^* \stackrel{IH}{\rhd} Red(s_1)^* Red(s_2)^* = (Red(s_1) Red(s_2))^* = Red(s_1 s_2)^* \; .$$

(2): By induction on the length of d. Trivial if $d = \emptyset$. If $d = d_1 Red$ then

$$\begin{aligned}t^*(d_1 Red)^* = t^* d_1^* \stackrel{IH}{\rhd} Park(t, d_1)^* &\stackrel{(1)}{\rhd} \\ Red(Park(t, d_1))^* &= Park(t, d_1 Red)^* \; .\end{aligned}$$

if $d = (d_1)_x[s]$ then

$$\begin{aligned}t^*((d_1)_x[s])^* = t^*(d_1^*)_x[s^*] \stackrel{IH}{\rhd} Park(t, d_1)_x^*[s^*] &\stackrel{3.4(1)}{\rhd} \\ (Park(t, d_1)_x[s])^* &= Park(t, (d_1)_x[s])^* \; .\end{aligned}$$

$\square$

3.7 Extension of the rank function. We extend the rank function $Rk(t)$ to e-terms by

$$\begin{aligned}
Rk(t) &:= 0 \qquad \text{if } t \text{ is } 0,\, S, \text{ or } x\\
Rk(\lambda x t) &:= Rk(t)\\
Rk(Rr) &:= Rk(r)\\
Rk((\lambda x t)s) &:= \max(Lev(\lambda x t), Rk(t), Rk(s))\\
Rk(\langle r : \bar{t} : d\rangle^{\alpha} s) &:= \max(Lev(\langle r : \bar{t} : d\rangle^{\alpha}), Rk(\langle 0 : \bar{t} : d\rangle), Rk(s))\\
Rk(ts) &:= \max(Rk(t), Rk(s))\\
&\qquad \text{if } t \text{ is not a lambda or delayed recursor}\\
Rk(\langle 0 : t_1 \cdots t_k : \emptyset\rangle^{\alpha}) &:= \max(Lev(\alpha)+1, Rk(t_1), \cdots, Rk(t_k))\\
Rk(\langle 0 : \bar{t} : d Red\rangle^{\alpha}) &:= Rk(\langle 0 : \bar{t} : d\rangle^{\alpha}) \dot{-} 1\\
Rk(\langle 0 : \bar{t} : d_x[s]\rangle^{\alpha}) &:= \max(Lev(s), Rk(s), Rk(\langle 0 : \bar{t} : d\rangle^{\alpha}))\\
Rk(\langle r : \bar{t} : d\rangle^{\alpha}) &:= \max(Rk(r), Rk(\langle 0 : \bar{t} : d\rangle^{\alpha})) \qquad \text{if } r \neq 0\ .
\end{aligned}$$

Note that we always have

$$Rk(\langle r : \bar{t} : d\rangle) = \max(Rk(r), Rk(\langle 0 : \bar{t} : d\rangle))\ . \tag{1}$$

We know from Par. 2.3 that the application $Red(t)$ reduces the non-zero rank of the term t by at least one. We wish this to hold for e-terms as well (see Lemma 3.8) and this is the reason why each marker Red decreases the rank of a delayed recursor by one.

3.8 Rank lemma. *For all e-terms t, s, $\bar{t}$. and sequence markers d we have*

$$Rk(t_x[s]) \leq \max(Lev(s), Rk(t), Rk(s)) \tag{1}$$

$$Rk\ Red(t) \leq Rk(t) \dot{-} 1 \tag{2}$$

$$\begin{aligned}
&Rk(t) \leq \max(Lev(\alpha)+1, Rk(t_1), \cdots, Rk(t_k)) \rightarrow\\
&\qquad Rk\ Park(t,d) \leq Rk(\langle 0 : t_1 \cdots t_k : d\rangle^{\alpha})\ .
\end{aligned} \tag{3}$$

Proof. (1): Intuitively, $Rk(t_x[s])$ can be increased from $\max(Rk(t), Rk(s))$ to $Lev(s)$ when the substitution reaches within t either a delayed recursor or else a subterm xt_1 and s is a lambda or a delayed recursor. This is technically proved by induction on the structure of t.

We prove by induction on the structure of e-terms t an auxiliary property:

$$\begin{aligned}
&\text{if } Red(t) \text{ is a lambda (delayed recursor) and } t \text{ is not then}\\
&\qquad Lev(t) \leq Rk(t)\ .
\end{aligned} \tag{4}$$

Clearly, we must have $t = t_1 t_2$ and we consider two cases. If t_1 is a lambda (delayed recursor) then $Lev(t) \leq Lev(t_1) \leq Rk(t)$. If t is not a lambda or a delayed recursor then, since $Red(t_1)$ must be a lambda (delayed recursor), we have $Lev(t) \leq Lev(t_1) \stackrel{IH}{\leq} Rk(t_1) \leq Rk(t)$.

(2): By induction on the structure of t. This is trivial when t is 0, S, or x and follows directly from IH when t is either $\lambda x t_0$ or Rr. If t is $\langle r : \bar{x} : d\rangle$ then

$$
\begin{aligned}
&Rk\,Red(\langle r : \bar{t} : d\rangle) = Rk(\langle Red(r) : \bar{x} : dRed\rangle) \stackrel{3.7(1)}{=} \\
&\quad \max(Rk\,Red(r), Rk(\langle 0 : \bar{t} : dRed\rangle)) \stackrel{IH}{\leq} \\
&\quad \max(Rk(r) \dot{-} 1, Rk(\langle 0 : \bar{t} : d\rangle) \dot{-} 1) = \max(Rk(r), Rk(\langle 0 : \bar{t} : d\rangle)) \dot{-} 1 \stackrel{3.7(1)}{=} \\
&\quad Rk(\langle r : \bar{t} : d\rangle) \dot{-} 1 \ .
\end{aligned}
$$

If t is $t_1 t_2$ then $Lev\,Red(t_2) = Lev(t_2) < Lev(t_1) = Lev\,Red(t_1)$ and we consider three subcases. If $Red(t_1) = \lambda x t_0$ then

$$
\begin{aligned}
Rk\,Red(t_1 t_2) = Rk((t_0)_x[Red(t_2)]) &\stackrel{(1)}{\leq} \\
\max(Lev(t_2), Rk(t_0), Rk\,Red(t_2)) &\leq \\
\max(Lev(t_1) \dot{-} 1, Rk\,Red(t_1), Rk\,Red(t_2)) &\stackrel{IH}{\leq} \\
\max(Lev(t_1) \dot{-} 1, Rk(t_1) \dot{-} 1, Rk(t_2) \dot{-} 1) &= \\
\max(Lev(t_1), Rk(t_1), Rk(t_2)) \dot{-} 1 \ .&
\end{aligned}
$$

If t_1 is a lambda then $\max(Lev(t_1), Rk(t_1), Rk(t_2)) = Rk(t_1 t_2)$. Otherwise we have

$$
\max(Lev(t_1), Rk(t_1), Rk(t_2)) \stackrel{(4)}{=} \max(Rk(t_1), Rk(t_2)) = Rk(t_1 t_2) \ .
$$

If $Red(t_1) = \langle r : \bar{t} : d\rangle$ then

$$
\begin{aligned}
Rk\,Red(t_1 t_2) = Rk(\langle r : \bar{t}x : d_x[Red(t_2)]\rangle) &\stackrel{3.7(1)}{=} \\
\max(Rk(r), Rk(\langle 0 : \bar{t}x : d_x[Red(t_2)]\rangle)) &= \\
\max(Rk(r), Rk(\langle 0 : \bar{t} : d\rangle), Rk\,Red(t_2), Lev\,Red(t_2)) &\stackrel{3.7(1)}{=} \\
\max(Rk(\langle r : \bar{t} : d\rangle), Rk\,Red(t_2), Lev\,Red(t_2)) &= \\
\max(Rk\,Red(t_1), Rk\,Red(t_2), Lev(t_2)) &\stackrel{IH}{\leq} \\
\max(Lev(t_1), Rk(t_1), Rk(t_2)) \dot{-} 1 &
\end{aligned}
$$

and by a similar reasoning as in the previous case we get

$$\max(Lev(t_1), Rk(t_1), Rk(t_2)) = Rk(t_1 t_2)\ .$$

If $Red(t_1)$ is neither a lambda nor a delayed recursor then

$$Rk\ Red(t_1 t_2) = Rk(Red(t_1) Red(t_2)) = \max(Rk\ Red(t_1), Rk\ Red(t_2)) \overset{IH}{\leq} \max(Rk(t_1), Rk(t_2)) \dot{-} 1 \leq Rk(t_1 t_2) \dot{-} 1\ .$$

(3): By induction on the length of d. The case $d = \emptyset$ is trivial. If $d = d_1 Red$ then

$$Rk\ Park(t, d_1 Red) = Rk\ Red\ Park(t, d_1) \overset{(2)}{\leq} Rk\ Park(t, d_1) \dot{-} 1 \overset{IH}{\leq} Rk(\langle 0 : \bar{t} : d_1 \rangle) \dot{-} 1 = Rk(\langle 0 : \bar{t} : d_1 Red \rangle)\ .$$

If $d = (d_1)_x[s]$ then

$$Rk\ Park(t, (d_1)_x[s]) = Rk(Park(t, d_1)_x[s]) \overset{(1)}{\leq} \max(Lev(s), Rk\ Park(t, d_1), Rk(s)) \overset{IH}{\leq} \max(Lev(s), Rk(\langle 0 : \bar{t} : d_1 \rangle), Rk(s)) = Rk(\langle 0 : \bar{t} : (d_1)_x[s] \rangle)\ .$$ □

4 Normalization of E-terms

The normalization function $N(t)$ defined in Par. 4.3 will repeatedly remove from an e-term t of rank 0 all delayed recursors whose count terms are normal terms. The function N is defined by transfinite recursion with the ordinal measure $|t|$ which is an estimate of the height of the e-term t visualized as a tree.

4.1 Depth of e-terms. For every e-term t we define an ordinal $|t|$, called its *depth* (height), by

$$\begin{aligned}
|t| &:= 0 \qquad \text{if } t \text{ is } \bar{0},\ S, \text{ or } x \\
|\lambda x t| &:= |t| + 1 \\
|Rr| &:= |r| + 1 \\
|\langle r : t_1 \cdots t_n : d \rangle| &:= \max(|r| + 1, W(\max(|t_1|, \cdots, |t_n|) + \omega, d)) \\
|ts| &:= \max(|t|, |s|) + 1\ .
\end{aligned}$$

Here the function $W(\alpha, d)$ takes an ordinal α and a sequence d of delayed markers to yield an ordinal such that:

$$
\begin{aligned}
W(\alpha, \emptyset) &:= \alpha \\
W(\alpha, d\mathit{Red}) &:= 2^{W(\alpha,d)} \\
W(\alpha, d_x[s]) &:= |s| + W(\alpha, d) \ .
\end{aligned}
$$

The ordinal ω is used in the clause for delayed recursors because we have

$$|t^n s s_1 \cdots s_k| < \max(|t|, |s|, |s_1|, \cdots, |s_k|) + \omega \ . \tag{1}$$

It is well known that over terms we have $|t_x[s]| \leq |s|+|t|$ and $|\mathit{Red}(t)| \leq 2^{|t|}$. This holds also for e-terms (see Lemma 4.2) and it is the reason for the assignment of depth to the delayed reductions and substitutions by the function W.

For a term t we clearly have $|t| < \omega$ whereas for an e-term t we have $|t| < \varepsilon_0$.

4.2 Depth lemma. *For all e-terms t, s and sequences of markers d we have:*

$$|t_x[s]| \leq |s| + |t| \tag{1}$$

$$|\mathit{Red}(t)| \leq 2^{|t|} \tag{2}$$

$$|\mathit{Park}(t, d)| \leq W(|t|, d) \ . \tag{3}$$

Proof. (1): by induction on the structure of t. Trivial if t is 0, S, or a variable (whether x or a different one). If t is $\lambda y t_0$ (we can always rename the bound variable to be different from x) then

$$|(\lambda y t_0)_x[s]| = |\lambda y (t_0)_x[s]| = |(t_0)_x[s]| + 1 \overset{IH}{\leq} |s| + |t_0| + 1 = |s| + |\lambda y t_0| \ .$$

The case when t is Rr is similar. If t is $\langle r : t_1 \cdots t_k : d\rangle$ then

$$
\begin{aligned}
&|\langle r : t_1 \cdots t_k : d\rangle_x[s]| = |\langle r_x[s] : t_1 \cdots t_k : d_x[s]\rangle| = \\
&\quad \max(|r_x[s]| + 1, W(\max(|t_1|, \cdots, |t_k|) + \omega, d_x[s])) \overset{IH}{\leq} \\
&\quad \max(|s| + |r| + 1, |s| + W(\max(|t_1|, \cdots, |t_k|) + \omega, d)) = \\
&\quad |s| + \max(|r| + 1, W(\max(|t_1|, \cdots, |t_k|) + \omega, d)) = |s| + |t| \ .
\end{aligned}
$$

If t is $t_1 t_2$ then

$$
\begin{aligned}
|(t_1 t_2)_x[s]| &= |(t_1)_x[s](t_2)_x[s]| = \max(|(t_1)_x[s]|, |(t_2)_x[s]|) + 1 \overset{IH}{\leq} \\
&\max(|s| + |t_1|, |s| + |t_2|) + 1 = |s| + \max(|t_1|, |t_2|) + 1 = \\
&|s| + |t_1 t_2| \ .
\end{aligned}
$$

(2): by induction on the structure of t. If t is 0, S, or x we have $|Red(t)| = |t| = 0 \leq 2^0 = 2^{|t|}$. If t is $\lambda x t_0$ then

$$|Red(\lambda x t_0)| = |Red(t_0)| + 1 \overset{IH}{\leq} 2^{|t_0|} + 1 \leq 2^{|t_0|+1} = 2^{|\lambda x t_0|} \ .$$

The case when t is Rr is similar. If t is $\langle r : t_1 \cdots t_k : d\rangle$ then

$$\begin{aligned}
|Red(\langle r : t_1 \cdots t_k : d\rangle)| &= |\langle Red(r) : t_1 \cdots t_k : dRed\rangle| = \\
&\max(|Red(r)| + 1, W(\max(t_1, \cdots, t_k) + \omega, dRed)) \overset{IH}{\leq} \\
&\max(2^{|r|+1}, 2^{W(\max(t_1,\cdots,t_k)+\omega,d)}) = \\
&2^{\max(|r|+1, W(\max(t_1,\cdots,t_k)+\omega,d))} = 2^{|\langle r : t_1 \cdots t_k : d\rangle|} \ .
\end{aligned}$$

If t is $s_1 s_2$ then we consider three subcases. If $Red(s_1) = \lambda x s_0$ then

$$\begin{aligned}
|Red(s_1 s_2)| = |(s_0)_x[Red(s_2)]| &\overset{(1)}{\leq} |Red(s_2)| + |s_0| \leq \\
&|Red(s_2)| + |Red(s_1)| \overset{IH}{\leq} 2^{|s_2|} + 2^{|s_1|} \leq \\
&2^{\max(|s_1|,|s_2|)+1} = 2^{|s_1 s_2|} \ .
\end{aligned}$$

If $Red(s_1) = \langle r : t_1 \cdots t_k : d\rangle$ then

$$\begin{aligned}
|Red(s_1 s_2)| &= |\langle r : t_1 \cdots t_k x : d_x[Red(s_2)]\rangle| = \\
&\max(|r| + 1, W(\max(|t_1|, \cdots, |t_k|, |x|) + \omega, d_x[Red(s_2)])) = \\
&\max(|r| + 1, |Red(s_2)| + W(\max(|t_1|, \cdots, |t_k|, 0) + \omega, d)) \leq \\
&|Red(s_2)| + \max(|r| + 1, W(\max(|t_1|, \cdots, |t_k|) + \omega, d)) = \\
&|Red(s_2)| + |Red(s_1)| \overset{IH}{\leq} 2^{|s_2|} + 2^{|s_1|} \leq 2^{\max(|s_1|,|s_2|)+1} = 2^{|s_1 s_2|} \ .
\end{aligned}$$

If $Red(s_1)$ is neither a lambda nor a delayed recursor then

$$\begin{aligned}
|Red(s_1 s_2)| = |Red(s_1) Red(s_2)| &= \max(|Red(s_1)|, |Red(s_2)|) + 1 \overset{IH}{\leq} \\
&\max(2^{|s_1|}, 2^{|s_2|}) + 1 = 2^{\max(|s_1|,|s_2|)} + 1 \leq \\
&2^{\max(|s_1|,|s_2|)+1} = 2^{|s_1 s_2|} \ .
\end{aligned}$$

(3): from (1) and (2) by a straightforward induction on the length of d. $\square$

4.3 Normalization of e-terms. We will normalize terms by embedding them into e-terms with the help of an *embedding* function $E(t)$ taking a term t into an e-term by replacing in t every of its maximal applications $R^\alpha r\bar{t}$ by $\langle r : \bar{t} : \emptyset\rangle$. An occurrence of $\bar{t}$ in t is *maximal* if the occurrence is not a part of an application $\bar{t}s$. We clearly have $E(t)^* = t$ and $|E(t)| < \omega^2$.

Let us denote by $Rk_R(t)$ the *R-rank* of a term t which is the maximum of $Lev(\alpha)+1$ for all recursors $R^\alpha r$ occurring in t.

Given a term t we first reduce its rank by $t_1 := Red^{Rk(t) \dot{-} Rk_R(t)}(t)$. We have $Rk(t_1) \overset{3.8(2)}{\leq} Rk_R(t)$ and $t \overset{3.6(1)}{\triangleright} t_1$. We then embed t_1 into an e-term $t_2 := E(t_1)$. We have $Rk(t_2) = Rk_R(t)$ and $t_1 = E(t_1)^* = t_2^*$. We then park $Rk_R(t)$ reductions in t_2 by $t_3 := Park(t_2, \overbrace{Red \cdots Red}^{Rk_R(t)})$ whereby we get $Rk(t_3) \overset{3.8(2)}{=} 0$ and

$$t_2^* = t_2^*(\overbrace{Red \cdots Red}^{Rk_R(t)})^* \overset{3.6(2)}{\triangleright} t_3^* \ .$$

We thus have $t \triangleright t_3^*$. We plan to remove the delayed recursors from t_3 by $N(t_3)$ where the function N is defined by transfinite recursion up to ε_0 with the measure $|t|$:

$$\begin{aligned}
N(t) &:= t \qquad \text{if } t \text{ is } \bar{0},\ S, \text{ or } x \\
N(\lambda x t) &:= \lambda x\, N(t) \\
N(Rr) &:= R\, N(r) \\
N(\langle r : t_1 \cdots t_k : d\rangle^\alpha) &:= \begin{cases} N\, Park(t_1^n t_2 t_3 \cdots t_k, d) \\ \quad \text{if } N(r) = \bar{n} \text{ and } k \geq 2 \\ R^\alpha N(r)(N\, Park(t_1, d)) \cdots (N\, Park(t_k, d)) \\ \quad \text{otherwise} \end{cases} \\
N(ts) &:= N(t)N(s) \ .
\end{aligned}$$

That the depth of the argument to N goes down should be obvious for all recursive applications of N except perhaps for the ones applied to the results of *Park*. For the first such application we have:

$$\begin{aligned}
|\langle r : t_1 t_2 t_3 \cdots t_k : d\rangle| &= \max(|r|+1, W(\max(|t_1|, |t_2|, |t_3|, \cdots, |t_k|) + \omega, d)) \\
&\geq W(\max(|t_1|, |t_2|, |t_3|, \cdots, |t_k|) + \omega, d) \\
&\overset{4.1(1)}{>} W(|t_1^n t_2 t_3 \cdots t_k|, d) \overset{4.2(3)}{\geq} |Park(t_1^n t_2 t_3 \cdots t_k, d)| \ .
\end{aligned}$$

The reasoning for the remaining such applications is similar and follows from $|t_i| < \max(|t_1|, \cdots, |t_i|, \cdots, |t_k|) + \omega$.

We note that the above term t_2 does not contain any recursors $R^\alpha r$. Inspection of the function *Red* reveals that it cannot introduce recursors $R^\alpha r$ into terms without them. This holds also for $Park(t, d)$ if neither t nor d contain recursors.

We now prove by transfinite induction on $|t|$ an auxiliary property:

> if t is an e-term of rank 0 and without recursors $R^\alpha r$ then $t^* \triangleright N(t)$ with $N(t)$ a normal term.

Take any e-term t of rank 0 and without recursors and consider the following cases. If t is 0, S, or x the property holds trivially. If t is $\lambda x t_0$ the property follows directly from IH. If t is $s_1 s_2$ then $Rk(s_1) = Rk(s_2) = 0$ and $Lev(s_1) > 0$. Thus s_1 cannot be a lambda or a delayed recursor and it must be S. We have

$$t^* = S s_2^* \overset{IH}{\triangleright} SN(s_2) = N(S)N(s_2) = N(Ss_2) = N(t)$$

where $N(s_2)$ is a normal term by IH and so is $SN(s_2)$.

If t is $\langle r : t_1 \cdots t_k : d\rangle$ then $Rk(r) = 0$ and $r^* \triangleright N(r)$ with the last a normal term by IH. We consider two subcases. If $N(r) = \bar{n}$ and $k \geq 2$ then we have

$$\begin{aligned}
t^* = \langle r : t_1 \cdots t_k : d\rangle^* = Rr^*(t_1^* d^*) \cdots (t_k^* d^*) \overset{IH}{\triangleright} \\
R\bar{n}(t_1^* d^*) \cdots (t_k^* d^*) \triangleright (t_1^* d^*)^n (t_2^* d^*) \cdots (t_k^* d^*) = \\
(t_1^n t_2 \cdots t_k)^* d^* \overset{3.6(2)}{\triangleright} Park(t_1^n t_2 \cdots t_k, d)^* \overset{IH}{\triangleright} \\
N\, Park(t_1^n t_2 \cdots t_k, d) = N(t)
\end{aligned}$$

and the last is a normal term by IH which applies because $Park(t_1^n t t_2 \cdots t_k, d)$ is of rank 0 and without recursors by 3.8(3).

If $N(r)$ is not a numeral or $k < 2$ then for $1 \leq i \leq k$ we have

$$t_i^* d^* \overset{3.6(2)}{\triangleright} Park(t_i, d)^* \overset{IH}{\triangleright} N\, Park(t_i, d)$$

and the last is a normal term by IH which applies because $Park(t_i, d)$ is of rank 0 and without recursors by 3.8(3). Hence,

$$\begin{aligned}
t^* = \langle r : t_1 \cdots t_k : d\rangle^* = Rr^*(t_1^* d^*) \cdots (t_k^* d^*) \overset{\text{IH and above}}{\triangleright} \\
RN(r)(N\, Park(t_1, d)) \cdots (N\, Park(t_k, d)) = N(t)
\end{aligned}$$

with the last a normal term. This ends the proof of the auxiliary property. Instantiating it with the above t_3 which is of rank 0 and without recursors we get $t \triangleright t_3^* \triangleright N(t_3)$ and the last is a normal term. We have just proved the following theorem.

4.4 Normalization theorem. *The function explicitly defined by:*

$$Norm(t) := N\,Park(E\,Red^{Rk(t) \dot{-} Rk_R(t)}(t), \overbrace{Red \cdots Red}^{Rk_R(t)}) \ .$$

yields a normal form of the term t. □

5 Fragments of T

5.1 Terms as natural numbers. We did not want to clutter the presentation in previous sections but a natural approach to the normalization process is to treat it by recursive functions. For that it suffices to treat terms not as sequences of symbols but as natural numbers by selecting a suitable coding and identifying the terms with their codes. Similarly for the types and the markers in the delayed recursors (see [6]). For our purposes it suffices to know that the functions involved are elementary, i.e. from the class $\mathcal{E}^3$ of the Grzegorczyk hierarchy (see [5,9]). Thus the classes of types, terms, and e-terms are elementary predicates. The functions $Tp(t)$, $Lev(\alpha)$, $t^n s$, $\bar{n}$, $\overbrace{Red \cdots Red}^{k}$, $Rk(t)$, $Rk_R(t)$, $Red(t)$, $t_x[s]$, and E are elementary. However, the function $Park$, being an iteration of Red, is in the class $\mathcal{E}^4$ of the Grzegorczyk hierarchy.

Ordinals less than ε_0 can be coded into natural numbers in many canonical ways such that the usual ordinal operations are elementary (see [9]). Then the functions $|t|$ and $W(\alpha, d)$ will be in most codings in the class $\mathcal{E}^4$ because W iterates ordinal exponentiation. The normalization functions N and $Norm$ will become ε_0-recursive functions because the function N is defined by nested recursion up to ε_0 under which ε_0-recursive functions are closed. Inspection of the definition of N will reveal that it uses an elementary function Dc such that $Dc(\bar{n}) = n$.

5.2 Functions represented by typed terms. Closed terms (without free variables) of level 1 represent functions over natural numbers in a natural way. Let t be a closed term of type $\overbrace{0 \mapsto \cdots \mapsto 0}^{n} \mapsto 0$. We say that

t *represents* the n-ary function f if

$$f(x_1, \cdots, x_n) = Dc\, Norm(t\overline{x_1}\,\overline{x_2} \cdots \overline{x_n}) \ .$$

Every such function is thus $<\varepsilon_0$-recursive because for a concrete term t the function N can be restricted to an ordinal $<\varepsilon_0$. The converse also holds and every $<\varepsilon_0$-recursive function is representable in T (see [9]).

5.3 Fragments of T. A *fragment* T_n of Gödel's T consists of terms where at most the recursors $R^\alpha r$ with $Lev(\alpha) \leq n$ are allowed. Fragments T_n are important because the class of functions representable in T_n is identical to the class of functions provably recursive in the fragment $I\Sigma_{n+1}$ of Peano arithmetic (with induction restricted to Σ_{n+1} formulas). The last class of functions is identical to the functions definable by $< \omega_{n+2}$-recursion, which is identical to functions defined by $< \omega \cdot \omega_{n+1}$ nested recursion, which is identical to the union of classes $\mathcal{E}^\alpha$ for $\alpha < \omega_{n+1}$ of the extended Grzegorczyk hierarchy (see [9]). Here $\omega_0 := 1$ and $\omega_{n+1} := \omega^{\omega_n}$.

As $<\omega_2$-recursion is reducible to primitive recursion, the functions representable in T_0 are primitive recursive. The fragment T_0 is thus a natural candidate for the semantics of modern programming languages because already the functions in $\mathcal{E}^4$, or even the elementary functions not computable in polynomial time, are computationally unfeasible and so there is no need to go to the vastly longer reduction sequences afforded by T_{n+1}.

5.4 Suitable recursors. Unfortunately, our normalization function N cannot be used to show that the functions representable in T_n are $< \omega_{n+2}$-recursive. This is because for a term t of T_n we have $k := Rk_R(t) \leq n+1$. E-terms $t_2 = E(t_1)$ from Par. 4.3 are such that $|t_2| < \omega^2$ and so

$$|t_3| \overset{4.2(3)}{\leq} W(|t_2|, \overbrace{Red \cdots Red}^{k}) = 2_k(|t_2|) < 2_{n+1}(\omega^2) = \omega_{n+2}$$

where $2_0(\alpha) := \alpha$ and $2_{n+1}(\alpha) := 2^{2_n(\alpha)}$. Even if we restricted the function N to e-terms with the depth $< \omega_{n+2}$, the removal of nested recursion by the theorem of Tait ([15,9]) yields a $< \omega_{n+3}$-recursive definition of N.

The 'tower' of omegas can be lowered by the following observation. Every delayed recursor of rank 0 has the form

$$\langle r : t_1 \cdots t_k : dRed_{x_1}[s_1] \cdots_{x_m} [s_m] \rangle$$

where the e-terms r, s_1, ..., s_m are of rank 0. We call the delayed recursor *suitable* if these e-terms are normal terms. Instead of N we can manage the elimination of delayed recursors by the iteration of a function $Dec(t)$ which finds and reduces in t a suitable delayed recursor. For that we note that if the above delayed recursor is suitable we have

$$
\begin{aligned}
&|\langle r : t_1 \cdots t_k : dRed_{x_1}[s_1] \cdots_{x_m}[s_m]\rangle| = \\
&\quad \max(|r|+1, W(\max(|t_1|, \cdots, |t_k|) + \omega, dRed_{x_1}[s_1] \cdots_{x_n}[s_m])) \overset{|r|<\omega}{=} \\
&\quad |s_m| + \cdots + |s_1| + 2^{W(\max(|t_1|,\cdots,|t_k|)+\omega,d)} \overset{|s_i|<\omega}{=} \\
&\quad 2^{W(\max(|t_1|,\cdots,|t_k|)+\omega,d)} = \omega^\alpha
\end{aligned}
$$

for some α. This is because $\max(|t_1|, \cdots, |t_k|) + \omega = w \cdot (p+1)$ for some natural number p and hence $W(\max(|t_1|, \cdots, |t_k|) + \omega, d)$ is a limit ordinal.

Thus the depth of a suitable recursor is given by a principal additive ordinal for which we have $\beta \# \gamma < \omega^\alpha$ provided $\beta, \gamma < \omega^\alpha$ holds. Here # denotes the natural sum of ordinals (see [11]). This fact is crucial in the lowering of the tower of omegas in the next paragraph.

5.5 Normalization in T_n. An application of $Dec(t)$ will decrease the depth of a suitable delayed recursor but this will not necessarily decrease the depth of t. We are thus lead to measure an e-term t by the *size* function $\|t\|$ which sums up the depth of all of its delayed recursors of rank 0. The size function is defined in $\mathcal{E}^4$ by:

$$
\begin{aligned}
\|t\| &:= 0 \qquad \text{if } t \text{ is } 0,\, S, \text{ or } x \\
\|\lambda x t\| &:= \|t\| \\
\|Rr\| &:= \|r\| \\
\|\langle r : \bar{t} : dRed_{x_1}[s_1] \cdots_{x_m}[s_m]\rangle\| &:= |\langle r : \bar{t} : dRed_{x_1}[s_1] \cdots_{x_m}[s_m]\rangle| \,\#\\
&\qquad \|r\| \# \|s_1\| \# \cdots \# \|s_m\| \\
\|ts\| &:= \|t\| \# \|s\| \ .
\end{aligned}
$$

We can easily prove by induction that $\|t\| = 0$ holds for terms t and that $|t| < \omega^\alpha \to \|t\| < \omega^\alpha$ holds for e-terms t.

The function $Dec(t)$ finds a suitable delayed recursor $\langle r : t_1 \cdots t_k : d\rangle$ in a non-normal e-term t of rank 0 (there must be always at least one) and replaces it by an e-term s which is $Park(t_1^n t_2 \cdots t_k, d)$ if $r = \bar{n}$ and $k \geq 2$ and $Rr\, Park(t_1, d) \cdots Park(t_k, d)$ otherwise. Thus $Dec \in \mathcal{E}^4$ and we have

$$
|s| < |\langle r : \bar{t} : d\rangle| \overset{\|r\|,\|s_i\|=0}{=} \|\langle r : \bar{t} : d\rangle\| = \omega^\alpha
$$

for some α. Hence, also $\|s\| < \|\langle r : \bar{t} : d\rangle\|$. This propagates upwards in the e-term t yielding $\|Dec(t)\| < \|t\|$. Thus we can normalize a given term t of

T_n by forming an e-term

$$Park(E\,Red^{Rk(t)\,\dot{-}\,(n+1)}(t), \overbrace{Red \cdots Red}^{n+1})$$

with depth, and thus also size, $< \omega_{n+2}$ and then iterating Dec starting with this e-term by a $< \omega_{n+2}$-recursive function. This proves that functions representable in T_n are $< \omega_{n+2}$ recursive.

6 Conclusion: An Application in Computer Science Teaching

We have already mentioned above that the fragment T_0 is of extreme importance from the point of view of computer programming languages because it can represent exactly the primitive recursive functions. The author and his colleague J. Komara have implemented a programming and proof checking system called CL (Clausal Language) admitting exactly the primitive recursive functions. The formal theory of CL can be interpreted in $I\Sigma_1$ arithmetic. CL uses pairing in the style of LISP (see [17]) and it is similar to the current functional programming languages. In addition, one can also reason about the defined functions within the embedded proof system which is a major simplification of Computational Logic of Boyer-Moore [1] because we work with the domain of natural numbers. We currently use CL for the teaching of three CS courses: Introduction to declarative programming, Program specification and verification, and Introduction to theory of computability. We have carried out the research into the ordinal analysis of Gödel's T by means of delayed recursors because we intend to extend CL with intensional functionals. The functionals will be coded by indices into natural numbers (something similar is done in LISP where the functionals are just S-expressions). As a result CL will be like any functional programming language but the underlying formal theory will still be one of the simplest possible: $I\Sigma_1$ fragment of Peano arithmetic.

References

1. R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, 1979.
2. W. Buchholz: *Notation systems for infinitary derivations.* Archive for Mathematical Logic, (30) pp. 277-296, 1991.
3. J. Y. Girard: *Proofs and types*, Cambridge University Press, 1989,
4. K. Gödel: *Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes.* Pp. 280-287, Dialectica 12 (1958).

5. A. Grzegorczyk: *Some classes of recursive functions.* Rozprawy Mate. No. IV, Warsaw 1953.
6. P. Hájek, P. Pudlák: *Mathematics of first-order arithmetic.* Springer Verlag 1993.
7. W. A. Howard: *Assignment of ordinals to terms for primitive recursive functionals of finite type.* In Intuitionism and Proof Theory, pp. 443-458, Amsterdamm 1970.
8. G. E. Mints: *Finite investigations of transfinite derivations.* Journal of Soviet Mathematics (10), pp. 548-596, 1978.
9. H. E. Rose: *Subrecursion, Functions and Hierarchies.* Clarendon Press, Oxford 1984.
10. J. R. Shoenfield: *Mathematical logic.* Addison Wesley, 1967.
11. K. Schütte: *Proof Theory.* Springer Verlag 1977.
12. H. Schwichtenberg: *Elimination of higher type levels in definitions of primitive recursive functionals by means of transfinite recursion.* In Logic Colloquium 1973 (eds. H. E. Rose and J. C. Shepherdson), North Holland, 1975.
13. H. Schwichtenberg: *Classifying recursive functions.* Handbook of Recursion Theory, to appear, 1996.
14. H. Schwichtenberg: *Finite notations for infinite terms*, to appear in APAL. Recursion Theory 96.
15. W. W. Tait: *Nested recursion.* Math. Ann. 143, pp. 236-250, 1961.
16. W. W. Tait: *Infinitely long terms of transfinite type.* In Formal systems and recursive functions (ed. J. N. Crossley and M. Dummett), pp. 176-185, North Holland 1967.
17. P. J. Voda: *Subrecursion as a basis for a feasible programming language.* In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, number 933 in LNCS, pages 324–338. Springer Verlag, 1994.
18. A. Weiermann: *A proof of strongly uniform termination for Gödel's T by methods from local predicativity*, to appear in Archives for Math. Logic, 1997.