# A Type Theory for Iterated Inductive Definitions

Anton Setzer
Department of Pure Mathematics, University of Leeds
Leeds LS2 9JT
email pmt6ans@leeds.ac.uk

9/iii/1994

**Abstract**

We introduce a type theory $FA_\omega$, which has at least the strength of finitely iterated inductive definitions $ID_{<\omega}$. This type theory has as ground types trees with finitely many branching degrees (so called free algebras). We introduce an equality in this theory, without the need for undecidable prime formulas. Then we give a direct well-ordering proof for this theory by representing a ordinal denotation system in the iteration of Kleene's $O$. This can be easily done, by introducing functions on the trees, which correspond to the functions in the ordinal denotation system. The proof shows, that $FA_\omega$ proofs transfinite induction up to $D_0 D_n 0$, which shows, that the strength of $FA_\omega$ is at least $ID_{<\omega}$. It seems to be obvious, that this bound is sharp.

## 1 Definition of the type theory $FA_\omega$

**Definition 1.1** *The type theory $FA_\omega$ is defined as follows:*

(a) *The ground types are defined inductively by:*

*If $n \geq 0$ and $\alpha_1, \ldots, \alpha_n$ are ground types, then $(\alpha_1, \ldots, \alpha_n)$ is a ground type.*

*The type $(\alpha_1, \ldots, \alpha_n)$ should be the type of well-founded trees with branching degrees $\alpha_1, \ldots, \alpha_n$.*

(b) *Ground types are types, and if $\alpha$, $\beta$ are types then $(\alpha \to \beta)$ is a type.*

*We will omit brackets, using the usual conventions.*

(c) *If $\alpha = (\alpha_1, \ldots, \alpha_n)$ is a ground type, then for $i = 1, \ldots, n$ $C_i^\alpha$ is a constant of type $(\alpha_i \to \alpha) \to \alpha$ ($C_i^\alpha$ are the constructors for this type) and if $\alpha$ is as before and $\sigma$ a type, then we have the recursion constant $R_{\alpha,\sigma}$ of type*

$$((\alpha_1 \to \alpha) \to (\alpha_1 \to \sigma) \to \sigma) \to \cdots \to ((\alpha_n \to \alpha) \to (\alpha_n \to \sigma) \to \sigma) \to \alpha \to \sigma$$

*We will write $(C_1 : \alpha_1, \ldots, C_n : \alpha_n)$ for $(\alpha_1, \ldots, \alpha_n)$ to indicate, that $C_i$ are names for $C_i^\alpha$.*

(d) *The terms are the typed lambda terms, built using the constants of (c). We write* $t^\sigma$ *for a term* $t$ *of type* $\sigma$.

(e) *Special groundtypes are the empty type* $\underline{0} := ()$ *with no constructors and ex falsum quodlibet* $EFQ_\sigma := R_{\underline{0},\sigma}$ *of type* $\underline{0} \to \sigma$, *(we will omit the index* $\sigma$, *if it is clear from the context) the type of booleans* $bool := \underline{2} := (\underline{0}, \underline{0})$ *with special terms* $true := C_1^{bool} EFQ_{bool}$ *and* $false := C_2^{bool} EFQ_{bool}$ *of type* $bool$.

(f) *We have as reduction rules* $\alpha$, $\beta$ *and* $\eta$*-reduction and the rules for the recursion-constants* $(\alpha = (\alpha_1, \ldots, \alpha_n))$ $R_{\alpha,\sigma} s_1 \cdots s_n (C_i f) \quad \to_{red} \quad s_i f(\lambda x. R_{\alpha,\sigma} s_1 \cdots s_n (f x))$, *where* $x$ *is a new variable, which will be applied to subterms as well.*

(g) *We will identify terms, that are equivalent using these reduction-rules and write* $s \simeq t$, *if we identify two terms. Therefore* $\simeq$ *is the transitive, reflexive and symmetric closure of the reduction-relation* $\to_{red}$.

*Note, that [Ber94] proofs strong normalization for an extension of this type theory.*

(h) *The formulas are* $atom(t^{bool})$ *where* $t$ *is a term of type* $bool$ *and if* $\phi, \psi$ *are formulas,* $x$ *a variable of type* $\sigma$, *then* $\phi \to \psi$, $\phi \wedge \psi$ *and* $\forall x^\sigma.\phi$ *are formulas.*

*We define* $\perp := atom(false)$, $\neg\phi := \phi \to \perp$.

(i) *We have the rules of intuitionistic logic. The axioms are* $atom(true)$ *and induction over ground types: If* $\alpha = (\alpha_1, \ldots, \alpha_n)$ *is a ground type,* $\phi(x)$ *a formula, then we have the axiom* $(\forall f^{\alpha_1 \to \alpha}.(\forall x^{\alpha_1} \phi(f x)) \to \phi(C_1 f)) \to \cdots \to (\forall f^{\alpha_n \to \alpha}.(\forall x^{\alpha_n} \phi(f x)) \to \phi(C_n f)) \to \forall x^\alpha \phi(x)$.

**Definition 1.2** *Further we have the following special types:*

(a) *The types with* $n$ *elements* $\underline{n} := (C_0^1 : \underline{0}, \ldots, C_{n-1}^1 : \underline{0})$.

$k_n := C_k^n EFQ_{\underline{n}}$.

(b) *For the booleans we define* $\underline{ifthenelse}_\sigma$: $\mathcal{B} \to \sigma \to \sigma \to \sigma$, *written* $\underline{if}_\sigma$ $r$ $\underline{then}$ $s$ $\underline{else}$ $t$ *for* $\underline{ifthenelse}_\sigma$ $rst$, *(we omit the* $\sigma$ *usually)* $\underline{ifthenelse}_\sigma := \lambda x, y, z. R_{\underline{2},\sigma}(\lambda u, v.y)(\lambda u, v.z)z$, *therefore* $\underline{if}$ $tt$ $\underline{else}$ $s$ $\underline{then}$ $t \to_{red} s$, $\underline{if}$ $ff$ $\underline{else}$ $s$ $\underline{then}$ $t \to_{red} t$. *This allows us to define a function on the booleans by case distinction on, wether a term is true or false.*

(c) *A boolean predicate* $P(x_1^{\alpha_1}, \ldots, x_n^{\alpha_n})$ *is a formula* $atom(f x_1 \cdots x_n)$ *where* $f : \alpha_1 \to \cdots \to \alpha_n \to bool$. *In this case* $(P(t_1, \ldots, t_n))_{bool} := f t_1 \cdots t_n$. *We will usually not mention* $f$ *in the definition of boolean predicates. Therefore, if we say: "we define a prediacte* $P(x^\alpha)$ *by induction on* $x^\alpha$*", and define further "*$P(C_i^\alpha g)$ *iff* $Q(g)$ *for some boolean property* $Q$*", we mean, the following: we define a function* $f : \alpha \to bool$ *by induction on* $\alpha$, *such that* $f(C_i^\alpha g) := (Q(g))_{bool}$, *and define* $P(t) := atom(f t)$.

*Note, that negation, disjunction and conjunction of boolean predicates is a boolean predicate, since we have boolean functions* $\neg_{\mathcal{B}} : bool \to bool$, $\vee_{\mathcal{B}} : bool \to bool \to bool$, $\wedge_{\mathcal{B}} : bool \to bool \to bool$.

(d) $\underline{\omega} := (C_0^\omega : \underline{0}, C_1^\omega \underline{1})$. *We have the zero of type* $\underline{\omega}$ $0_\omega := C_1^\omega EFQ$, *an the successor function* $succ_\omega := \lambda x. C_1^\omega(\lambda y.x)$, *of type* $\underline{\omega} \to \underline{\omega}$.

We intruduce some basic predicates and functions:

**Definition 1.3** *Let* $\alpha = (C_1 : \alpha_1, \ldots, C_n\alpha_n)$.

(a) *We define the boolean predicate* $\tau_\alpha(t) = \alpha_i$ *by recursion on* $t : \alpha$. *(More precisely, if several of the types are identical, we have to use a name* $\widetilde{\alpha_i}$ *for each type* $\alpha_i$*, such that for* $i \neq j$ $\widetilde{\alpha_i} \neq \widetilde{\alpha_j}$*, and have to write* $\tau_\alpha(t) = \widetilde{\alpha_i}$*, but want to keep the notation simple):* $\tau_\alpha(C_i f) = \alpha_i$ *is true, and* $\tau_\alpha(C_j f) = \alpha_i$ *is false, if* $j \neq i$.

*We will usually omit the index* $\alpha$ *and write* $\tau(t) \neq \alpha_j$ *for* $\neg(\tau(t) = \alpha_j)$.

(b) *We define* $\cdot[\cdot]^{\alpha}_{\alpha_i} : \alpha \to \alpha_i$ *by recusion on* $\alpha$, $(C_i f)[s]^{\alpha}_{\alpha_i} := fs$, $(C_j f)[s]^{\alpha}_{\alpha_i} := C_j f$*, if* $j \neq i$. *If we have* $\tau(a) = \alpha_i$*, than by* $a[s]$ *stands for* $a[s]^{\alpha}_{\alpha_i}$*.par*

(c) *We define* $pred^{\alpha}_{\alpha_i} : \alpha \to (\alpha_i \to \alpha)$ *by recursion,* $pred^{\alpha}_{\alpha_i}(C_i f) := f$, $pred^{\alpha}_{\alpha_i}(C_j f) := \lambda x.(C_j f)$*, if* $j \neq i$.

We have to distinguish between empty and non empty types, in order to define dummy elements of non empty types. The empty types are those, which are built of non empty types.

**Definition 1.4** (a) *We define the Meta-predicate "*$\alpha$ *is empty" for ground-types* $\alpha$ *by induction on the definition of* $\alpha$*:*

*If* $\alpha = (\alpha_1, \ldots, \alpha_n)$*, then* $\alpha$ *is empty iff all* $\alpha_i$ *are not empty.*

(b) *We define for empty ground types* $\alpha$ *and arbitrary type* $\gamma$ *the function* $EFQ^{\alpha}_{\sigma} : \alpha \to \sigma$*, and for not empty ground types* $\alpha$ *an element* $dummy_\alpha : \alpha$*, simultaneously by Meta-induction on the types:*

*If* $\alpha = (C_1 : \alpha_1, \ldots, C_n\alpha_n)$ *is not empty,* $i$ *minimal such that* $\alpha_i$ *is empty then* $dummy_\alpha := C_i EFQ^{\alpha_i}\alpha$*, and if* $\alpha$ *is empty, we define by recursion* $EFQ^{\alpha}_{\sigma}t$ *for* $t : \alpha$*:* $EFQ^{\alpha}_{\sigma}(C_i f) := EFQ^{\alpha}_{\sigma}(f\ dummy_{\alpha_i})$*, which is possible, since* $\alpha_i$ *is not empty. The last arguments allows as well to prove* $\forall x^{\alpha}.\perp$*, by replacing the recursion by induction.*

**Definition 1.5** *We will introduce lists as functions* $\underline{\omega} \to \alpha$*. Assume* $\alpha$ *is a ground-type.*

(a) $list(\alpha) := \underline{\omega} \to \alpha$.

(b) $nil_\alpha := \lambda x.dummy_\alpha$*, defined only, if* $\alpha$ *is not empty.*

(c) $car_\alpha := \lambda x.(x0_\omega)$ *of type* $list(\alpha) \to \alpha$.

(d) $cdr_\alpha := \lambda x.\lambda y.x(succ\ y)$ *of type* $list(\alpha) \to list(\alpha)$.

(e) *We define* $cons_\alpha : \alpha \to list(\alpha) \to list(\alpha)$*,* $cons\ xyn$ *is defined by induction on* $n : \underline{\omega}$*:* $cons\ xy0 := x$*,* $cons\ xy(succ\ t) := yt$.

(f) $(\cdot)_{\cdot,\alpha} : list(\alpha) \to \underline{\omega} \to \alpha$*,* $(f)_{i,\alpha} := fi$.

**Remark 1.6** $car(cons\ s^{\alpha}t^{list(\alpha)}) \simeq s$*,* $cdr(cons\ s^{\alpha}t^{list(\alpha)}) \simeq t$.

**Proof**: $car(cons\ st) \simeq (cons\ st)0 \simeq s$*,* $cdr(cons\ st) \simeq \lambda x.(cons\ st)(succ\ x) \simeq \lambda x.tx \simeq t$.

## 2 Definition of the equality

The idea for the well-ordering proof is, to reduce it to the well-ordering on our trees. We want to show

$$(\forall x \in OT_n.(\forall y \prec \tau(x).y \in OT_n \to \phi(x[y])) \to \phi(x)) \to \forall x \in OT_n.\phi(x).$$

by defining

$$\psi(x^{\underline{\Omega}_{n+1}}) := \forall y \in OT_n.\iota_{OT,n} y = x \to \phi(x).$$

and using now transfinite induction over the type $\underline{\Omega}_{n+1}$ (which is the $n$th iteration of Kleene's $O$).
But in order to do this, we need an equality in our system. But we do not want any undecidable prime formulas in our theory (see [Sch92] for some discussion about it), therefore we have to work hard to do this.
An element of a ground type $\alpha$ is a tree, which has, if $\alpha = (\alpha_1, \ldots, \alpha_n)$, branching degrees $\alpha_1, \ldots, \alpha_n$. Now the idea is, that two terms are equal, if, whenever we go by a path in both trees to subtree, we have nodes, which are locally equal: they have the same constructor. Therefore we need to define the paths, which may consist of elements of the branching degrees. We will define a concept $s[l_1^{list(\alpha_1)}, \ldots, l_k^{list(\alpha_k)}; m^\omega]$, the intended meaning being, that we go $m$ times to the predecessor, starting at $s$, selecting the branch, which corresponds to the first element of the list of type $list(\alpha_i)$, if we have branching degree $\alpha_i$. If the type $\alpha_k$ is empty, we will stop with this process. Further we introduce local equality $=^{loc}_\alpha$.

**Definition 2.1** *Assume $\alpha = (C_1 : \alpha_1, \ldots, C_n : \alpha_n)$. W.l.o.g. $\alpha_1, \ldots, \alpha_k$ are not empty, $\alpha_{k+1}, \ldots, \alpha_n$ are empty.*

*(a) We define $\cdot[\cdot, \ldots, \cdot; \cdot]_\alpha : \alpha \to list(\alpha_1) \to \cdots \to list(\alpha_k) \to \underline{\omega} \to \alpha$. We define $a[l_1, \ldots, l_k; m]$ by recursion on $m$, side-recursion on $a$:*

$$a[l_1, \ldots, l_k; C_0^\omega f] := a,$$

*if $i \le k$ we define*

$$(C_i f)[l_1, \ldots, l_k; C_1^\omega f] := (f\ car(l_i))[l_1, \ldots, l_{i-1}, cdr(l_i), l_{i+1}, \ldots, l_k; , f0_1]$$

*and if $k < i$ we have*

$$(C_i f)[l_1, \ldots, l_k; C_1^\omega f] := C_i f,$$

*We will write $a[\vec{l}; m]$ for $a[l_1, \ldots, l_k; m]$, and $\forall \vec{x}^{list(\alpha)}$ for $\forall x_1^{list(\alpha_1)}, \ldots, x_n^{list(\alpha_n)}$.*

*(b) We define the boolean predicate $s^\alpha =^{loc}_\alpha t^\alpha$ by recursion on $s$ and side-recursion on $t$: $C_i f =^{loc}_\alpha, C_j g$ iff $i = j$.*

The definition of the equality will be a little bit more complicated, than it seems at first sight. We will not be able to prove $\forall f^{\alpha \to \beta}.x^\alpha =_\alpha y^\alpha \to (fx) =_\beta (fy)$. All terms, we can construct, will fulfill this condition. We will call functions respectful, if they have this condition. Now this implies, that we need some notion of respectfulness on ground terms as well: We want, that in each occurrence of $C_i f$, we have $f$ is respectful. Now we want only to check equality, using respectful lists. Therefore we have to define simultaneously for all ground types $\alpha$ the predicate $Resp_\alpha(t^\alpha)$ for $t$ is respectful, and the equality $s^\alpha =_\alpha t^\alpha$ for $s$ and $t$ are equal elements of type $\alpha$.

**Definition 2.2** *We define simultaneously for all ground types the predicate $Resp_\alpha(t^\alpha)$, the predicate $s^\alpha =_\alpha t^\alpha$ and $Resplist_\alpha(l^{list(\alpha)})$ (an auxiliary definition for $l$ is a respectful list) and $Locresp_\alpha(t^\alpha)$ (for $t$ is locally respectful).*
*Assume $\alpha = (C_1 : \alpha_1, \ldots, C_n : \alpha_n)$, w.l.o.g. $\alpha_1, \ldots, \alpha_k$ not empty, $\alpha_{k+1}, \ldots, \alpha_n$ empty.*
*We write $Resplist_\alpha(\vec{x})$ for $Resplist_{\alpha_1}(x_1) \wedge \cdots \wedge Resplist_{\alpha_k}(x_k)$.*

*(a) $s^\alpha =_\alpha t^\alpha$ is defined as $\forall \vec{x}^{list(\alpha)}, m^\omega.Resplist_\alpha(\vec{x}) \to s[\vec{x}; m] =^{loc}_\alpha t[\vec{x}; m].par$*

*(b) $Locresp_\alpha(t) := \bigwedge_{i=1,\ldots,n}(\tau(t) = \alpha_i \to \forall x^{\alpha_i}, y^{\alpha_i}.Resp_{\alpha_i}(x) \to x =_{\alpha_i} y \to t[x] =_\alpha t[y])$.*

*(c) $Resp_\alpha(t) := \forall \vec{x}^{list(\alpha)}, m^\omega(Resplist_\alpha(\vec{x}) \to Locresp_\alpha(t[\vec{x}; m])$.*

*(d) $Resplist_\alpha(l) := \forall n^{\underline{\omega}}.Resp_\alpha((l)_n))$.*

We will now characterize the equality:

**Lemma 2.3** *Assume $\alpha$ is a ground type.*

*(a) $\forall x^\alpha, y^{list(\alpha)}.Resp(x) \to Resplist(y) \to Resplist(cons\ xy))$.*

*(b) $\forall x^{list(\alpha)}.Resplist(x) \to (Resp(car\ x) \wedge Resplist(cdr\ x))$.*

**Proof**: trivial.

**Lemma 2.4** *(a) $=_\alpha$ and $=^{loc}_\alpha$ are equivalence relations (for ground-types $\alpha$).*

*(b) If $\alpha$ is not empty, then $Resp(dummy_\alpha)$, and $Resplist(nil_\alpha)$.*

**Proof**:(a): For $=^{loc}_\alpha$ this follows by induction on the type $\alpha$, for $=_\alpha$ by Meta-induction on the definition of the ground-types.
(b): We have $Locresp_\alpha(dummy_\alpha)$, since $\tau(dummy_\alpha) \neq \alpha_j$ for not empty types $\alpha_j$.
Now follows by induction on $n^\omega$, $Locresp(dummy[\vec{x}; n])$: In both cases $i = 0, 1$ we have $dummy[\vec{x}; C_i f] \simeq dummy$, therefore $Locresp(dummy[\vec{x}; C_i f]$.
$Resplist(nil_\alpha)$ follows now directly.

We charactrize equality and $Resp(t)$ as follows:

**Lemma 2.5** *Assume $\alpha = (C_1 : \alpha_1, \ldots, C_n : \alpha_n)$, $f, g : \alpha_i \to \alpha$, $h : \alpha_j \to \alpha$, $r, s : \alpha$. W.l.o.g. $\alpha_1, \ldots, \alpha_k$ are not empty, $\alpha_{k+1}, \ldots, \alpha_n$ are empty.*

*(a) $\neg(C_i f =_\alpha C_j h)$, if $i \neq j$.*

*(b) $C_i f =_\alpha C_i g \leftrightarrow \forall x^{\alpha_i}.Resp(x) \to fx =_\alpha gx$.*

*(c) $Locresp_\alpha(C_i f) \leftrightarrow all x^{\alpha_i}, y^{\alpha_i}.Resp(x) \to x = y \to fx =_\alpha fy$.*

*(d) $Resp_\alpha(C_i f) \leftrightarrow (Locresp_\alpha(C_i f) \wedge \forall x^{\alpha_i}.Resp(x) \to Resp(fx))$.*

*(e) If $\alpha_i$ is empty, then $C_i f =_\alpha C_i g$, $Locresp(C_i f)$, $Resp(C_i f)$.*

*(f) $s =_\alpha t \to ((Locresp(s) \leftrightarrow Locresp(t)) \wedge (Resp(s) \leftrightarrow Resp(t)))$.*

**Proof**:
Let $\vec{nil} := nil_{\alpha_1}, \ldots, nil_{\alpha_k}$. Then we have $Resplist(\vec{nil})$.
(a): If $C_i f =_\alpha C_j g$, then, $C_i f[\vec{nil}; 0] =^{loc}_\alpha C_j g[\vec{nil}; 0]$, $C_i f[\vec{nil}; 0] \simeq C_i f$, $C_i g[\vec{nil}; 0] \simeq C_i g$, therefore $i = j$.
(b) Case $\alpha_i$ is not empty:
Assume $C_i f = C_i g$ and $x^{\alpha_i}$, $Resp(x)$, and $\vec{l}^{list(\alpha)}$, $m^\omega$, $Resplist(\vec{l})$.
Then $Resplist(cons\ x l_i)$, therefore

$$(fx)[\vec{l}; m] \simeq (f(car\ (cons\ x l_i)))[l_1, \ldots, cdr\ (cons\ x l_i), \ldots, l_k; m]$$

$$\simeq (C_i f)[l_1 \ldots, cons\ x l_i, \ldots, l_k; succ\ m]$$
$$=^{loc}_\alpha (C_i g)[l_1 \ldots, cons\ x l_i, \ldots, l_k; succ\ m]$$
$$\simeq (gx)[\vec{l}; m],$$

therefore $fx =_\alpha gx$.
If we have the r.h.s. $\vec{l}^{list(\alpha)}$, $Resplist(\vec{l})$, then we have:
If $m = C_0 g$, $(C_i f)[l_1 \ldots l_k; m] \simeq C_i f =^{loc}_\alpha C_i g \simeq (C_i g)[l_1 \ldots l_k; m]$.
If $m = C_1 g$, $Resp(car\ l_i)$, $Resplist(cdr\ l_i)$, therefore

$$(C_i f)[l_1 \ldots l_k; m] \simeq (f(car\ l_i))[l_1, \ldots, cdr\ l_i, \ldots, l_k; g0]$$

$$=^{loc}_\alpha (g(car\ l_i))[l_1, \ldots, cdr\ l_i, \ldots, l_k; g0]$$
$$\simeq (C_i g)[l_1 \ldots l_k; m]$$

Case $\alpha_i$ is empty: Assume $C_i f = C_i g\ \vec{l}^{list(\alpha)}$, $m^\omega$, $Resplist(\vec{l})$. Then we have, in both cases $r = 0, 1$ and $m = C_r g$, $(C_i f)[l_1 \ldots l_k; m] \simeq C_i f =^{loc}_\alpha C_i g \simeq (C_i g)[l_1 \ldots l_k; m]$. Therefore $C_i f =_\alpha C_i g$, and $\forall x_i^\alpha. \perp$, therefore the r.h.s. as well.
(c): Follows, in case $\alpha_i$ is not empty, since $(C_i f)[x] \simeq fx$ by definition.
If $\alpha_i$ is empty, then we have $Locresp(C_i f)$, and the r.h.s. follows from ex falsum quodlibet.
(d): Follows as (a), (b).
(e): Follows from the proofs above.
(f): Follows from (a), (b), (c), (d) by induction on $s$ and $t$.

We can then define the equality on arbitrary types as follows:
**Definition 2.6** *We define $=_\alpha$ and $Resp_\alpha$ for all types by induction on the definition. For ground types we have already defined it. Let $\alpha = \beta \to \gamma$.*

*(a) $f^\alpha =_\alpha g^\alpha$ iff $\forall x^\beta, Resp_\beta(x) \to fx =_\gamma gx$.*

*(b) $Resp_\alpha(f^\alpha)$ iff $\forall x^\beta, y^\beta. Resp_\beta(x) \to x =_\beta y \to (fx =_\gamma fy \wedge Resp_\gamma(fx)$.*

We have now:

**Lemma 2.7** *(a) $=_\alpha$ is an equivalence relation on types.*

*(b) $s =_\alpha t \to (Resp(s) \leftrightarrow Resp(t)))$.*

*(c) For every constructor $C$ and every recursion constant $R$ we have $Resp(C)$, $Resp(R)$ and if $s^{\alpha \to \beta}$, $t^\alpha$ are terms, then $Resp(s) \to Resp(t) \to Resp(st)$.*

*(d) For every term $t$ with free varibles among $x_1^{\alpha_1}, \dots, x_n^{\alpha_n}$ we have*

$$\forall x_1, \dots, x_n, x_1', \dots, x_n'.Resp(x_1) \to \cdots \to Resp(x_n) \to x_1 = x_1' \to \cdots \to x_n = x_n' \to$$

$$Resp(t) \wedge t(x_1, \dots, x_n) = t(x_1', \dots, x_n).$$

**Proof**:
(a): Easy.
(b): Follows by Meta-induction on the definition of types.
(c): If $C : (\beta \to \alpha) \to \alpha$, and $f, g : \beta \to \alpha$, $Resp(f)$, $f =_\alpha g$, then $\forall x.Resp(x) \to fx = gx$, therefore $Cf = Cg$, and we have $Resp(Cf)$.
If $R = R_{\alpha,\sigma}$, $\alpha = (C_1 : \alpha_1, \dots, C_n : \alpha_n)$, and we have $s_i, s_i' : ((\alpha_i \to \alpha) \to (\alpha_i \to \sigma) \to \sigma)$, $Resp(s_i)$, $s_i = s_i'$, we have to show $\forall x^\alpha, x'^\alpha.Resp(x) \to x = x' \to (Rs_1 \cdots s_n x = Rs_1 \cdots s_n x' \wedge Resp(Rs_1 \cdots s_n x))$. We show this by induction on $x, x'$. Assume $x = C_i f$, $x' = C_j f'$. If $i \neq j$, then $x \neq x'$, and if $i = j$, then by $Resp(x)$ and $x = x'$ follows $Resp(f)$, $Resp(x')$, $Resp(f')$, and by IH (since

$$\forall y^{\alpha_i}, y'^{\alpha_i}.Resp(y) \to y = y' \to (Resp(fy) \wedge fy = fy' \wedge fy = f'y)$$

for all $y^{\alpha_i}$, $y'^{\alpha_i}$, with $Resp(y)$, $y = y'$, we have $Resp(Rs_1 \cdots s_n(fy))$,

$$Rs_1 \cdots s_n(fy) = Rs_1 \cdots s_n(fy'),$$

$$Rs_1 \cdots s_n(fy) = Rs_1' \cdots s_n'(f'y),$$

therefore $Resp(\lambda y.Rs_1 \cdots s_n(fy))$,

$$\lambda y.Rs_1 \cdots s_n(fy) = \lambda y.Rs_1' \cdots s_n'(f'y).$$

Now follows $Resp(s_i f(\lambda y.Rs_1 \cdots s_n(fy)))$, and

$$s_i f(\lambda y.Rs_1 \cdots s_n(fy)) = s_i f'(\lambda y.Rs_1' \cdots s_n'(f'y)) = s_i' f'(\lambda y.Rs_1' \cdots s_n'(f'y)).$$

since

$$Rs_1, \dots, s_n(C_i f) \simeq s_i f(\lambda y.Rs_1 \cdots s_n(fy))$$
$$Rs_1', \dots, s_n'(C_i f') \simeq s_i' f'(\lambda y.Rs_1' \cdots s_n'(f'y))$$

follows the assertion.
$Resp(st)$ is trivial.
(d) Follows immediately from (c) by induction on the defintion of terms.

**Notation 2.8** *We write $\forall x^\sigma; \phi$ for "$\forall x^\sigma.Resp(x) \to \phi$". If we write "assume $Resp(x^\sigma)$" we mean, "assume $x^\sigma$, assume $Resp(x)$" and if we write "for all respectful $x^\sigma$" we mean "for all $x^\sigma$ such that $Resp(x)$".*

**Definition 2.9** *(a) We define $flat : \underline{\omega} \to \underline{\omega}$ by recursion on $\underline{\omega}$:*

*$flat(C_0 f) := zero$, $flat(C_1 f) := succ(flat(f0))$.*

*(b) We define the boolean predicate $eqnat(x^{\underline{\omega}}, y^{\underline{\omega}})$ by recursion on $\underline{\omega}$:*

*$eqnat(C_0 f, C_0 g)$ is true, $eqnat(C_0 f, C_1 g)$ is false, $eqnat(C_1 f, C_1 g)$ iff $eqnat(f0, g0)$.*

**Lemma 2.10** $\forall x^{\underline{\omega}}, y^{\underline{\omega}}.Resp(flat(x)) \wedge (eqnat(x, y) \to flat(x) = flat(y))$.

**Proof** by induction on $\underline{\omega}$, using, that $Resp(0)$, $Resp(succ)$ (since both are terms), therefore $x = y \to succ(x) = succ(y)$, $Resp(x) \to Resp(succx)$.

**Definition 2.11** *We interpret $n$ ary primitive recursive functions in $FA_\omega$ as functions $\underbrace{\underline{\omega} \to \cdots \to \underline{\omega}}_{n times}$ as follows: the function with constant value $n$ is interpreted as $succ^{\underline{\omega}}$, the identity as $\lambda x.x$, the composition of $f$ with $n$-ary functions $g_i$ as*

$$\lambda x_1, \ldots, x_n.f(g_1 x_1 \cdots x_n) \cdots (g_m x_1 \cdots x_n),$$

*and the primitive recursive function $f0\vec{x} := g\vec{x}$, $f(succn)\vec{x} := hn(fn\vec{x})\vec{x}$ as*

$$R_{\underline{\omega}}(\lambda u, v.g)(\lambda u, v.h(u0)(v0)).$$

.

**Lemma 2.12** *If $f$ is a $n$-ary primitive recursive function, then*

$$\forall x_1, \ldots, x_n.flat(fx_1 \cdots x_n) = f(flat\ x_1) \cdots (flat\ x_n)$$

.

**Proof**: Induction on the definition of the primitive recursive functions.

# 3 The ordinal denotation system $OT$

We introduce a primitive recursive denotation system, omitting the proofs (see [Buc86], [BS88] for proofs and details, for this particular system it is worked out in [Set90]).

**Definition 3.1** *Let $0_{OT}, D_0, D_1, \ldots$ be a sequence of formal symbols (We will write $0$ for $0_{OT}$).*
*Then for $n \in \omega$ the set $T_n$ of terms is defined by*
*$0 \in T_n$ is a term,*
*if $a \in T_n$, $k \leq n$, then $D_k a \in T_n$,*
*and if $k \geq 1$, $a_0, \ldots, a_k \in T_n$, then $(a_0, \ldots, a_k) \in T_n$.*
*We assume some coding of $T := \bigcup_{n\in\omega} T_n$ in $\omega$, s.t. we have some boolean predicate $s \in T_n$ (depending on $s^{\underline{\omega}}$ and $n^{\underline{\omega}}$), and a function $length : \omega \to \omega$ for the length of the term.*
*The terms $D_k a$ are called principal terms.*

If $a$ is a principal term, then $(a) := a$.
$\Omega_{n+1} := D_n 0$, $1 := D_0 0$, $\omega := D_0 1$.
We will write $\forall x \in T.\phi(x)$ for $\forall x^{\underline{\omega}}.x \in T \to \phi(x)$, where $x \in T$ is the boolean predicate correponding to $T$, similar for other sets, which are introduced as boolean predicates.

**Definition 3.2** *We define $a \prec b$ for $a, b \in T$ by*
*$a \prec b$ if $b \neq 0$,*
*$D_u a \prec D_v b$ if $u < v$ or ($u = v$ and $a \prec b$)*
*and if $n \geq 1$ or $m \geq 1$, then for $a = (a_0, \ldots, a_n), b = (b_0, \ldots b_m)$,*
*$a \prec b \Leftrightarrow (n < m \wedge \forall i \leq n.a_i = b_i) \vee \exists k \leq min\{n, m\}.a_k \prec b_k \wedge \forall i < n.a_i = b_i$.*
*Let $a \preceq b :\leftrightarrow (a \prec b \vee a = b)$.*
*$\prec$, $\preceq$ can be defined as boolean predicates on $\underline{\omega} \times \underline{\omega}$.*

We define for $a \in T$ and $M, M' \subset T$:
$M \preceq M' :\Leftrightarrow \forall x \in M \exists y \in M'(x \preceq y)$,
$M \prec a :\Leftrightarrow \forall x \in M(x \prec a)$,
$a \preceq M :\Leftrightarrow \exists x \in M(a \preceq x)$.

**Definition 3.3** *We define $G_u a \subset T$ for $a \in T$ by*
$G_u 0 := \emptyset$,
$G_u(a_0, \dots, a_k) := G_u a_0 \cup \dots \cup G_u a_k$ *(if $k \geq 1$)*,
$$G_u D_v b := \begin{cases} \{b\} \cup G_u b, \text{ if } u \leq v, \\ \emptyset, \text{ if } v < u. \end{cases}$$

**Definition 3.4** *We define the set of terms $OT (OT \subset T)$: by*
$0 \in OT$,
*if $a_0, \dots, a_k \in OT \quad (k \geq 1)$, $a_k \preceq \dots \preceq a_0$, then $(a_0, \dots, a_k) \in OT$*
*and if $b \in OT$, $G_v b \prec b$, then $D_v b \in OT$.*
$OT_n := OT \cap T_n$.
*In the following $\forall x \prec t.\phi(x) := \forall x \in OT.x \prec t \rightarrow \phi(x)$, similar for $\preceq$.*
*Further for $a \in OT$ a considered as a set will be the set $\{x \in OT | x \prec a\}$.*

**Definition 3.5** *Definition of $a + b$ for $a, b \in T$:*
$a + 0 := 0 + a := a$, $(a_0, \dots, a_k) + (b_0, \dots, b_m) := (a_0, \dots, a_k, b_0, \dots, b_m)$ *(if $(a_0, \dots, a_k)$ and $(b_0, \dots, b_m) \neq 0$).*
*Definition of $a \cdot n$ for $a \in T$, $n \in OT$, $n \prec \omega$:*
$a \cdot 0 := 0$, $a \cdot (n + 1) := a \cdot n + a$.

**Definition 3.6** *(Definition of $\tau(a)$ and $a[z]$ for $a \in OT$, $z \in \tau(a)$).*

- *([ ].0)* $\tau(0) := 0$.
- *([ ].1)* $\tau(1) := 1$. $1[0] := 0$.
- *([ ].2)* $\tau(D_{u+1}0) := \Omega_{u+1}$. $(D_{u+1}0)[z] := z$.
- *([ ].3)* *Assume $a = D_v b$ s.t. $b \neq 0$:*
  - *(i)* *If $\tau(b) = 1$: $\tau(a) := \omega$,*
    $a[n] := (D_v b[0]) \cdot (n + 1)$.
  - *(ii)* *If $\tau(b) = \Omega_{u+1}$ $(v \leq u < \omega)$: $\tau(a) := \omega$, $a[n] := D_v b[\zeta_n]$,*
    *wobei $\zeta_0 := D_u 0$, $\zeta_{n+1} := D_u b[\zeta_n]$.*
  - *(iii)* $\tau(b) \in \{\omega\} \cup \{\Omega_{u+1} \mid u < v\}$: $\tau(a) := \tau(b)$,
    $a[z] := D_v b[z]$.
- *([ ].5)* $a = (a_0, \dots, a_k)(k \geq 1)$: $\tau(a) := \tau(a_k)$,
  $a[z] := (a_0, \dots, a_{k-1}) + a_k[z]$.

*Wir define $0[n] := 0$, $a[n] := a[0]$ for $a \in T$ with $\tau(a) = 1$.*

**Lemma 3.7** *(a) $\prec$ is a linear ordering on $OT$.*

*(b) $\{x \in OT | x \prec \omega\}$ is the least set $X$ such that $0 \in X$ and if $x \in X$, $x + 1 \in X$.*

*(c) $\forall a \in OT.\forall x \prec \tau(a).a[x] \in OT \wedge a[x] \prec a$.*

*(d) $\forall a \in OT.\forall b \in OT.b \prec a \rightarrow \exists x \in OT.b \prec a[x]$.*

*(e) $\forall a \in OT.a \prec \omega \rightarrow (a = 0 \vee \exists x \in OT.length(x) < a \wedge a = x + 1)$.*

*(f) $\forall a \in OT.a \prec D_0 D_{n+1} 0 \rightarrow a \in OT_n$.*

**Proof**: Is done in the cited literature.

# 4 The $n$th numberclasses in the type theory $FA$

We introduce now the iteration of Kleene's $O$.

**Definition 4.1** *(a) We define the n-th numberclass $\underline{\Omega}_n$ ($\underline{\Omega}_n = O_n$, where $O_n$ is the iteration of Kleene's O) by Meta-recursion on n for $n \geq 1$: $\underline{\Omega}_n := (C_0^{\Omega_n} : \underline{0}, C_1^{\Omega_n} : \underline{1}, C_\omega^{\Omega_n} : \underline{\omega}, C_{\Omega_1}^{\Omega_n} : \underline{\Omega}_1, \ldots, C_{\Omega_{n-1}}^{\Omega_n} : \underline{\Omega}_{n-1})$.*

*We define $0_{\Omega_n} := C_0^{\Omega_n} EFQ$, $succ_{\Omega_n} : \lambda x.C_1^{\Omega_n}(\lambda u.x)$ of type $\Omega_n \to \Omega_n$.*

*(b) The branch ordinals are the ordinals $0, 1, \omega, \Omega_1, \Omega_2, \Omega_3, \ldots$.*

*The ordinal types are $\underline{\alpha}$ for $\alpha$ a branch ordinal.*

*(c) We define for $\alpha, \beta$ branch ordinals $\underline{\alpha} \subset_{type} \underline{\beta} :\Leftrightarrow \alpha \prec \beta$, and $\underline{\alpha} \subseteq_{type} \underline{\beta} :\Leftrightarrow \alpha \preceq \beta$.*

*(d) For an letter $\tau = 0, 1, \omega, \Omega_1, \ldots$, $\underline{\tau}$ is the corresponding type $\underline{0}, \underline{1}, \underline{\omega}, \underline{\Omega}, \ldots$.*

*(e) If $\sigma \preceq \tau$, then we define the embedding $\iota_{\sigma,\tau} : \underline{\sigma} \to \underline{\tau}$, by recursion on $\underline{\sigma}$: For $\alpha \prec \sigma$, $\iota_{\sigma,\tau}(C_\alpha^\sigma f) := C_\alpha^\tau \lambda x.\iota_{\sigma,\tau}(fx)$.*

*Further the projection is defined, $proj_{\tau,\sigma} : \underline{\tau} \to \underline{\sigma}$, ($\sigma \preceq \tau$) by recursion on $\underline{\tau}$: For $\alpha \prec \sigma$, $proj_{\tau,\sigma}(C_\alpha^\tau f) := C_\alpha^\sigma \lambda x.proj_{\tau,\sigma}(fx)$.*

*and if $\sigma \preceq \alpha \prec \tau$, then $proj_{\tau,\sigma}(C_\alpha^\tau f) := 0_{\underline{\sigma}}$.*

**Lemma 4.2** *Assume $\alpha \preceq \beta \preceq \gamma$ are branch ordinals.*

*(a) $\forall x^{\underline{\alpha}}; proj_{\beta,\alpha}\iota_{\alpha,\beta}x = x$.*

*(b) $\forall x^{\underline{\alpha}}; \iota_{\beta,\gamma}\iota_{\alpha,\beta}x = \iota_{\alpha,\gamma}x$.*

*(c) $\forall x^{\underline{\gamma}}; proj_{\beta,\alpha}proj_{\gamma,\beta}x = proj_{\gamma,\alpha}x$.*

**Proof**:
(a): Induction on $x^{\underline{\alpha}}$. If $f : \underline{\sigma} \to \underline{\alpha}$, and by IH $\forall y^{\underline{\sigma}}; proj(\iota(fy)) = fy$, then $proj(\iota(C_\alpha^\sigma f)) \simeq C_\alpha^\sigma \lambda y.proj(\iota(fy)) = C_\alpha^\sigma \lambda y.fy \simeq C_\alpha^\sigma f$.

# 5 Representation of $OT$ in $\underline{\Omega}_n$

**Definition 5.1** *We define the functions, representing the usual ordinal functions on OT as follows*

*(a) We define addition $+_{\underline{\sigma}} : \underline{\sigma} \to \underline{\sigma} \to \underline{\sigma}$ for an ordinaltype $\underline{\sigma}$ by recursion on the second argument:*

*$s + C_0 f := s$, $s + C_1 f := succ\ (s + f(0_1))$, $s + C_{\underline{\sigma}} f := C_{\underline{\sigma}} \lambda x.(s + x)$.*

*(b) We define multiplication $\cdot_{\underline{\sigma}} : \underline{\sigma} \to \underline{\omega} \to \underline{\sigma}$ for an ordinaltype $\underline{\sigma}$ by recursion on the second argument:*

*$s \cdot C_0 f := 0$, $s + C_1 f := (s \cdot (f0_1)) + s$.*

*(c) $\Omega_0^\sigma := 1^\sigma := succ\ 0_\sigma$ (if $\sigma \neq 0, 1$).*

(d) *For $\Omega_1 \preceq \sigma$, we define* $\omega^\sigma := C_\omega \lambda x.1 \cdot (succ\ x)$.

(e) *For $\Omega_k \prec \sigma$, we define* $\Omega_k^\sigma := C^\sigma_{\Omega_k} \lambda x.\iota_{\Omega_k,\sigma} x$.

(f) *We define for natural numbers $n \geq 1$, $\widetilde{D}_n : \underline{\Omega}_n \to (\underline{n} \to \underline{\Omega}_n)$. ($\widetilde{D}_n xk$ is the representation of the collapsing function $Dk$ in $n$ (which we will call $D_k^{\Omega_n}$. This is a way, to define simultaneously the functions $D_0^{\Omega_n}, \ldots, D_{n-1}^{\Omega_n}$).*

*We define $\widetilde{D}_n$ by recursion on $\underline{\Omega}$:*

$\widetilde{D}_n(C_0 f)k := \Omega_{k,\Omega_n}$.

$\widetilde{D}_n(C_1 f)k := C_\omega \lambda x.(\widetilde{D}_n(f0)k) \cdot x$.

*If $\Omega_k \prec \Omega_{l+1} = \alpha$, then*

$$\widetilde{D}_n(C_\alpha f)k := C_\omega \lambda x.\widetilde{D}_n(f(proj_{\Omega_{n+1},\alpha}\zeta_{succ\ x}))k,$$

*where $\zeta. : \underline{\omega} \to \underline{\Omega}_{n+1}$ is defined by recursion on $\underline{\omega}$,*

$$\zeta_{C_0 f} := \Omega_l,\ \zeta_{C_1 f} := \widetilde{D}_n(f(proj_{\Omega_n,\Omega_{l+1}}\zeta_{f0_1})).$$

*If $1 \prec \alpha \preceq \Omega_k$, then $\widetilde{D}_n(C_\alpha f)k := C_\alpha \lambda x.\widetilde{D}_n(fx)k$,*

(g) $D_k^{\Omega_n} := \lambda x.\widetilde{D}_n x(k+1)$, $D_k^{\Omega_n} : \underline{\Omega}_n \to \underline{\Omega}_n$.

(h) *We define for branch ordinals $\alpha \prec \Omega_{n+1}$ the interpretation of $OT_n \cap \alpha$ in $\underline{\Omega}_{k+1}$ $\iota'_{OT,n} : \underline{\omega} \to \underline{\alpha}$ ($\iota' := \iota_{OT,n}$): $\iota'(0) := 0_\alpha$, $\iota'(D_k a) := D_k^{\Omega_n}\iota_{OT,n} a$, $\iota'(a_0, \ldots, a_n) := \iota'(a_0) + \cdots \iota'(a_n)$.*

*To avoit problems with equality, we define:*

$\iota := \lambda x.\iota'(flat(x))$.

**Lemma 5.2** (a) $\forall x^{\underline{\omega}}.Resp(\iota(x))$.

(b) $\forall x^{\underline{\omega}}, y^{\underline{\omega}}.eqnat(x, y) \to \iota(x) = \iota(y)$.

**Lemma 5.3** *If $\alpha, \beta$ are branch ordinals, $\alpha \preceq \beta$, $\iota := \iota_{\alpha,\beta}$, $proj := proj_{\beta,\alpha}$, then*

(a) $\forall x^{\underline{\alpha}}, y^{\underline{\alpha}}; \iota(x + y) = \iota(x) + \iota(y)$.

(b) $\forall x^{\underline{\alpha}}, y^{\underline{\omega}}; \iota(x \cdot y) = \iota(x) \cdot y$.

(c) *For $x$ branch ordinal or $x = \Omega_0$ s.t. $x \prec \alpha$. we have $\iota(x_\alpha) = x_\beta$, $proj(x_\beta) = x_\alpha$,*

(d) *If $\alpha = \Omega_n$, $\beta = \Omega_m$, then $\forall x^{\underline{\alpha}}; \iota(\widetilde{D}_n xk) = \widetilde{D}_m \iota(x)k$, $\forall x^{\underline{\alpha}}; \iota(\widetilde{D}_{n,k} x) = \widetilde{D}_{m,k}\iota(x)$.*

The **proof** follows immediately by induction on the definition of the respective functions (using naturally the implicit predicates *Resp*.

**Definition 5.4** *We define for $\alpha \preceq \beta$ branch ordinals $t^{\underline{\beta}} \prec_\beta \alpha$ for $\iota_{\alpha,\beta}(proj_{\beta,\alpha} t) = t$. The meaning is, that $t$ is the embedding of term of type $\underline{\beta}$ in $\underline{\alpha}$.*

**Remark 5.5** *If $\alpha \preceq \beta$ are branch ordinals, $\gamma \prec \beta$, then*

(a) *If $\alpha \preceq \gamma$, $\neg(C^\beta_\gamma f \prec_\beta \alpha)$.*

(b) *If* $\gamma \prec \alpha$*, then* $C^{\beta}_{\gamma} f \prec_{\beta} \alpha \leftrightarrow \forall x^{\underline{\gamma}}. fx \prec_{\beta} \alpha$.

(c) *If* $\alpha \preceq \beta \preceq \gamma$*, then* $\forall x^{\underline{\gamma}}; x \prec \alpha \to x \prec \beta$.

**Proof**:
(a) $\iota_{\alpha,\beta} proj_{\alpha,\beta} C^{\beta}_{\gamma} f = C^{\beta}_{0} g$ for some $g$, therefore not equal to $C^{\beta}_{\gamma} f$.
(b) $\iota proj(C_{\gamma} f) = C_{\gamma} \lambda x. \iota(proj(fx)) = C_{\gamma} f$, iff $\forall x^{\underline{\gamma}}; \iota(proj(fx) = fx$. (Note, that we have implicitly always the respectfulness predicate.
(c) follows by induction on $\underline{\gamma}$, using (a) and (b).

**Lemma 5.6** *Assume* $\alpha \preceq \beta$ *branch ordinals.*

(a) $\forall x^{\underline{\beta}}, y^{\underline{\beta}}; x \prec_{\beta} \alpha \to y \prec_{\beta} \alpha \to x + y \prec_{\beta} \alpha$.

(b) $\forall x^{\underline{\beta}}, y^{\underline{\omega}}; x \prec_{\beta} \alpha \to\to x \cdot y \prec_{\beta} \alpha$.

(c) *If* $x = 0, 1, \omega, \Omega_0, \Omega_1, \ldots$*, then* $x_{\beta} \prec \alpha$ *iff* $x \prec \alpha$.

(d) *If* $\beta = \Omega_{n+1}$*,* $\alpha \prec \beta$*, then* $\forall x^{\alpha}; D^{\Omega_n}_{k} x \prec_{\beta} \Omega_{k+1}$.

(e) $D^{\Omega_n}_{0} 0 \prec_{\beta} \underline{\omega}$.

(f) *If* $a \in OT_n$*,* $a \prec \beta \preceq \gamma \prec \Omega_{n+1}$*, then* $\iota_{OT,n}(x) \prec \beta$.

**Proof**: (a) - (e) follow by induction on the types.
(f) follows by (a) - (e) by induction on the length of $a$.

**Lemma 5.7** *Let* $\alpha$ *be a branch ordinal.*

(a) $\tau(0_{\alpha}) = \underline{0} \wedge \forall x^{\underline{0}}. \perp$. $(\alpha \neq 0)$

(b) $\tau(1_{\alpha}) = \underline{1} \wedge \forall x^{\underline{1}}; 1_{\alpha}[x] = \underline{0}$ $(\alpha \neq 0, 1)$.

(c) $\tau(\Omega_{u+1,\alpha}) = \underline{\Omega}_{u+1} \wedge \forall x^{\underline{\Omega}_{u+1}}; \Omega^{\alpha}_{u+1}[x] = \iota_{\Omega^{\alpha}_{u+1}} x. (\Omega_{u+1} \prec \alpha)$.

(d) $\forall x^{\underline{\Omega}_{n+1}}; \tau(x) = \underline{0} \to \tau(D^{\Omega_n}_{u+1} x) = \underline{\Omega}_{u+1} \wedge \forall y^{\underline{\Omega}_{u+1}}; (D^{\Omega_n}_{u+1} x)[y] = \iota_{\Omega_{u+1},\alpha} y$.

(e) $\forall x^{\underline{\Omega}_{n+1}}; \tau(x^{\Omega_{n+1}}) = \underline{1} \to \tau(D^{\Omega_n}_{u} x) = \underline{\omega} \wedge \forall x^{\underline{\omega}}; (D^{\Omega_n}_{u} x)[n] = (D^{\Omega_n}_{u}(x[0])) \cdot succ\ n$.

(f) *If* $v \leq u < n$*, then with* $\zeta'_{n,u,v} x : \underline{\omega} \to \underline{\Omega}_{n+1}$*, s.t.*

$$\zeta'_{n,u,v} x(C_0 f) := D^{\Omega_n}_{u} 0,\ \zeta'_{n,u,v} x(C_1 f) := D^{\Omega_n}_{u}(x[proj_{\Omega_{n+1},\Omega_{u+1}}(\zeta'_{n,u,v} x(f0))]),$$

*we have*

$$\forall x^{\underline{\Omega}_{n+1}}; \tau(x) = \underline{\Omega}_{u+1} \to \tau(D^{\Omega_n}_{v} x) = \underline{\omega} \wedge$$
$$\forall y^{\underline{\omega}}; (D^{\Omega_n}_{v} x)[y] = D^{\Omega_n}_{v}(x[proj_{\Omega_{n+1},\Omega_{u+1}}(\zeta'_{n,u,v} x(succ\ y))]).$$

(g) *If* $\beta = \omega$ *or* $\beta = \Omega_{u+1}$*,* $u < v$*, then*

$$\forall x^{\underline{\Omega}_n}; \tau(x) = \underline{\beta} \to \tau(D^{\Omega_n}_{v} x) = \underline{\beta} \wedge \forall y^{\underline{\beta}}; (D^{\Omega_n}_{v} x)[y] = D^{\Omega_n}_{v}(x[y])$$

*(h) If $\beta \neq 0$, then*

$$\forall x^{\underline{\alpha}}, y^{\underline{\alpha}}; \tau(x) = \underline{\beta} \rightarrow \tau(y+x) = \underline{\beta} \wedge \forall z^{\underline{\beta}}; (y+x)[z] = y + (x[z])$$

**Proof**: Either immediate, oder transfinite induction on the type of $x$ (where we actually do not need really induction, but just that we have we have $x \equiv Cf$ for some constructor.

**Lemma 5.8** *(a) $\forall x \in OT_n.\iota_{OT,n}(x) = \iota_{OT,n}(flat(x))$.*

*$\forall x, y \in OT_n.eqnat(x, y) \rightarrow \iota_{OT,n}(x) = \iota_{OT,n}(y)$.*

*(b) $\forall y \in OT_n.\iota_{OT,n}(y+1) = succ\ \iota_{OT,n}(y)$.*

*(c) $\forall x, y \in OT_n.\iota_{OT,n}(x+y) = (\iota_{OT,n}x) + (\iota_{OT,n}y)$.*

*(d) $\forall x, y \in OT_n.y \prec \omega \rightarrow (\iota_{OT,n}x) \cdot (\iota_{OT,n}y) = \iota_{OT,n}(x \cdot y)$.*

**Proof**: Induction on the length of $y$.

**Lemma 5.9** *If $\alpha = \Omega_{n+1}$, then*

$$\forall x \in OT_n.\tau(\iota_{OT,n}x) = \underline{x} \wedge \forall y \in OT_n \cap \tau(x).\iota_{OT,n}(x[y]) = (\iota_{OT,n})[proj_{\Omega_{n+1},\tau(x)}\iota_{OT,n}y].$$

*(Where this stands more precisely for the formula:*

$$\forall x \in OT_n. \bigwedge_{\{\alpha|\alpha\prec\Omega_{n+1}\wedge\alpha\, branch\ ordinal\}} \tau(x) = \alpha \rightarrow \tau(\iota_{OT,n}x) = \underline{\alpha} \wedge \forall y \in OT_n.y \prec \alpha \rightarrow$$

$$\iota_{OT,n}(x[y]) = (\iota_{OT,n})[proj_{\Omega_{n+1},\alpha}\iota_{OT,n}y].$$

**Proof**: Induction on the definition of $OT_n$, using $eqnat(flat(x[y]), flat(x)[flat(y)])$, (by $\cdot[\cdot]$ is primitive recursive and 2.12), Lemmata 5.6, 5.7, 5.8. In the case $x = D_v b$, $\tau(b) = \Omega_{u+1}$, we prove by induction on the length of $y$

$$\forall y \in OT.\iota_{OT,n}\zeta_y = \zeta'_{n,u,v}(\iota_{OT,n}b)(proj_{\Omega_{n+1},\omega}\iota_{OT,n}y),$$

using 3.7 (e).

# 6 Proof of transfinite induction

**Lemma 6.1** *If $\phi(x^{\underline{\omega}})$ is a formula, then*

$$(\forall x \in OT_n.(\forall y \prec \tau(x).y \in OT_n \rightarrow \phi(x[y])) \rightarrow \phi(x)) \rightarrow \forall x \in OT_n.\phi(x).$$

**Proof**:
Assume $\forall x \in OT_n.(\forall y \prec \tau(x).y \in OT_n \rightarrow \phi(x[y])) \rightarrow \phi(x)$.
Let $\psi(x^{\underline{\Omega}_{n+1}}) := \forall y \in OT_n.\iota_{OT,n}y = x \rightarrow \phi(x)$. Then, we have for all branch ordinals $\alpha \prec \Omega_{n+1}$ respectful $f^{\underline{\alpha}\rightarrow\underline{\Omega}_n}$ $(\forall z^{\underline{\alpha}}.\psi(fz)) \rightarrow \psi(C_\alpha f)$:
Assume $\forall z^{\underline{\alpha}}; \psi(fz)$. If $\iota_{OT,n}(y) = C_\alpha f$, then $\tau(y) = \alpha$ and for $z \in OT_n$, $z \prec \alpha$, $\iota_{OT,n}(y[z]) = (C_\alpha f)[proj_{\Omega_{n+1},\underline{\alpha}}(\iota_{OT,n}z)] = f(proj_{\Omega_{n+1},\underline{\alpha}}(\iota_{OT,n}z))$, by IH therefore $\phi(y[z])$ and therefore by assumption $\phi(y)$ and we get $\psi(C_\alpha f)$. By transfinite induction on $\underline{\Omega}_{n+1}$ follows therefore $\forall x^{\underline{\Omega}_{n+1}}; \psi(x)$, therefore $\forall y \in OT_n.\phi(x)$.

**Theorem 6.2** *If $\phi(x^{\underline{\omega}})$ is a formula (where after every quantifier we have the predicate $Resp(x)$), then*

$$(\forall x \in OT \cap \Omega_1.(\forall y \prec \tau(x).\phi(y)) \rightarrow \phi(x)) \rightarrow \forall x \prec D0D_n0.\phi(x).$$

**Proof**: Let $\psi(x) := x \prec D_0D_n0 \rightarrow \forall y \prec x.\phi(x)$. Then $\forall x \in OT_n.(\forall y \prec \tau(x).y \in OT_n \rightarrow \phi(x[y])) \rightarrow \phi(x)$: $\tau(x) = 0, 1, \omega$ therefore $OT_n \cap \tau(x) = \tau(x)$. From $\forall y \in OT_n \cap \tau(x).\phi(x[y])$ follows therefore using 3.7 (d) $\forall y \prec \tau(x).\phi(y)$ and therefore $\phi(x)$.
By 6.1 follows $\forall x \in OT_n.\psi(x)$, therefore $\forall x \in OT_n.\psi(x+1)$ and we are done.

# References


[Ber94] U. Berger. Strong normalization for the typed lambda-calculus with recursion over wellfounded trees. abstract, submitted to PPC'94, 1994.

[BFPS81] W. Buchholz, S. Feferman, W. Pohlers, and W. Sieg. *Iterated Inductive Definitions. Recent Prooftheoretical Studies*, volume 897 of *Springer Lecture Notes in Mathematics*. Springer, Berlin, Heidelberg, New York, 1981.

[BS88] W. Buchholz and K. Schütte. *Proof Theory of Impredicative Subsystems of Analysis.* Bibliopolis, Naples, 1988.

[Buc86] W. Buchholz. A new system of proof-theoretic ordinal functions. *Ann. Pure a. Appl. Logic*, 32:195 – 207, 1986.

[Sch92] H. Schwichtenberg. Proofs as programs. In P. Aczel, H. Simmons, and S. S. Wainer, editors, *Proof Theory. A selection of papers from the Leeds Proof Theory Programme 1990*, pages 79 – 113, Cambridge, 1992. Cambridge University Press.

[Set90] A. Setzer. Ordinalzahlanalyse prädikativer Theorien unter Verwendung von Fundamentalfolgen. Master's thesis, Universität München, January 1990.

[Set93] A. Setzer. *Proof theoretical strength of Martin-Löf Type Theory with W-type and one universe.* PhD thesis, Universität München, 1993.