Program Extraction from Proofs

Helmut Schwichtenberg

Notes for a lecture course, Sommersemester 2007.
Mathematisches Institut der Ludwig-Maximilians-Universität,
Theresienstraße 39, D-80333 München, Germany.
July 20, 2007.

Contents

Introduction

The goal of this course is to study the computational content of proofs. We develop a natural deduction system for minimal logic in the language based on implication $\rightarrow$, conjunction $\wedge$, disjunction $\vee$ and the quantifiers $\forall$ and $\exists$.

CHAPTER 1

Arithmetic in Finite Types

1.1. ε_0-Recursion

We aim at showing that the provably recursive functions of Peano Arithmetic are exactly the ε_0-recursive functions, i.e., those definable from the primitive recursive functions by substitutions and (arbitrarily nested) recursions over "standard" well orderings of the natural numbers with order-types less than the ordinal

$$\varepsilon_0 = \sup\{\omega, \omega^\omega, \omega^{\omega^\omega}, \ldots\}.$$

As preliminaries, we must first develop some of the basic theory of these ordinals, and their standard codings as well-orderings on N. Then we define the hierarchies of fast-growing bounding functions naturally associated with them. These will provide an important complexity characterization through which we can more easily obtain the main result.

1.1.1. Ordinals below ε_0. Throughout the rest of this chapter, α, β, γ, δ, ... will denote ordinals less than ε_0. Every such ordinal is either 0 or can be represented uniquely in so-called Cantor Normal Form thus:

$$\alpha = \omega^{\gamma_1} \cdot c_1 + \omega^{\gamma_2} \cdot c_2 + \cdots + \omega^{\gamma_k} \cdot c_k$$

where $\gamma_k < \cdots < \gamma_2 < \gamma_1 < \alpha$ and the coefficients $c_1, c_2, \ldots, c_k$ are arbitrary positive integers. If $\gamma_k = 0$ then α is a successor ordinal, written $\mathrm{Succ}(\alpha)$, and its immediate predecessor $\alpha - 1$ has the same representation but with c_k reduced to $c_k - 1$. Otherwise α is a limit ordinal, written $\mathrm{Lim}(\alpha)$, and it has infinitely-many possible "fundamental sequences", i.e., increasing sequences of smaller ordinals whose supremum is α. However we shall pick out *one particular* fundamental sequence $\{\alpha(n)\}$ for each such limit ordinal α, as follows: first write α as $\delta + \omega^\gamma$ where $\delta = \omega^{\gamma_1} \cdot c_1 + \cdots + \omega^{\gamma_k} \cdot (c_k - 1)$ and $\gamma = \gamma_k$. Assume inductively that when γ is a limit, its fundamental sequence $\{\gamma(n)\}$ has already been specified. Then define, for each $n \in \mathbb{N}$,

$$\alpha(n) = \begin{cases} \delta + \omega^{\gamma-1} \cdot (n+1) & \text{if } \mathrm{Succ}(\gamma) \\ \delta + \omega^{\gamma(n)} & \text{if } \mathrm{Lim}(\gamma). \end{cases}$$

Clearly $\{\alpha(n)\}$ is an increasing sequence of ordinals with supremum α.

DEFINITION. With each $\alpha < \varepsilon_0$ and each natural number n, associate a finite set of ordinals $\alpha[n]$ as follows:

$$\alpha[n] = \begin{cases} \emptyset & \text{if } \alpha = 0 \\ (\alpha - 1)[n] \cup \{\alpha - 1\} & \text{if Succ}(\alpha) \\ \alpha(n)[n] & \text{if Lim}(\alpha). \end{cases}$$

LEMMA. *For each $\alpha = \delta + \omega^\gamma$ and all n,*

$$\alpha[n] = \delta[n] \cup \{\, \delta + \omega^{\gamma_1} \cdot c_1 + \cdots + \omega^{\gamma_k} \cdot c_k \mid \forall_i(\gamma_i \in \gamma[n] \wedge c_i \leq n) \,\}.$$

PROOF. By induction on γ. If $\gamma = 0$ then $\gamma[n]$ is empty and so the right hand side is just $\delta[n] \cup \{\delta\}$, which is the same as $\alpha[n] = (\delta + 1)[n]$ according to the definition above.

If γ is a limit then $\gamma[n] = \gamma(n)[n]$ so the set on the right hand side is the same as the one with $\gamma(n)[n]$ instead of $\gamma[n]$. By the induction hypothesis applied to $\alpha(n) = \delta + \omega^{\gamma(n)}$, this set equals $\alpha(n)[n]$, which is just $\alpha[n]$ again by definition.

Now suppose γ is a successor. Then α is a limit and $\alpha[n] = \alpha(n)[n]$ where $\alpha(n) = \delta + \omega^{\gamma-1} \cdot (n + 1)$. This we can write as $\alpha(n) = \alpha(n - 1) + \omega^{\gamma-1}$ where, in case $n = 0$, $\alpha(-1) = \delta$. By the induction hypothesis for $\gamma - 1$, the set $\alpha[n]$ is therefore equal to

$$\alpha(n-1)[n] \cup \{\, \alpha(n-1) + \omega^{\gamma_1} \cdot c_1 + \cdots + \omega^{\gamma_k} \cdot c_k \mid \forall_i(\gamma_i \in (\gamma-1)[n] \wedge c_i \leq n) \,\}$$

and similarly for each of $\alpha(n-1)[n]$, $\alpha(n-2)[n]$, ... , $\alpha(1)[n]$. Since for each $m \leq n$, $\alpha(m-1) = \delta + \omega^{\gamma-1} \cdot m$, this last set is the same as

$$\delta[n] \cup \{\, \delta + \omega^{\gamma-1} \cdot m + \omega^{\gamma_1} \cdot c_1 + \cdots + \omega^{\gamma_k} \cdot c_k \mid m \leq n \wedge \forall_i(\gamma_i \in (\gamma-1)[n] \wedge c_i \leq n) \,\}$$

and this is the set required because $\gamma[n] = (\gamma-1)[n] \cup \{\gamma-1\}$. This completes the proof. □

COROLLARY. *(i) For every limit ordinal $\alpha < \varepsilon_0$ and every n, $\alpha(n) \in \alpha[n+1]$. (ii) If $\beta \in \gamma[n]$ then $\omega^\beta \in \omega^\gamma[n]$ provided $n \neq 0$.*

DEFINITION. The *maximum coefficient* of $\beta = \omega^{\beta_1} \cdot b_1 + \cdots + \omega^{\beta_l} \cdot b_l$ is defined inductively to be the maximum of all the b_i and all the maximum coefficients of the exponents β_i.

LEMMA. *If $\beta < \alpha$ and the maximum coefficient of β is $\leq n$ then $\beta \in \alpha[n]$.*

PROOF. By induction on α. Let $\alpha = \delta + \omega^\gamma$. If $\beta < \delta$, then $\beta \in \delta[n]$ by IH and $\delta[n] \subseteq \alpha[n]$ by the lemma. Otherwise $\beta = \delta + \omega^{\beta_1} \cdot b_1 + \cdots + \omega^{\beta_k} \cdot b_k$ with $\alpha > \gamma > \beta_1 > \cdots > \beta_k$ and $b_i \leq n$. By IH $\beta_i \in \gamma[n]$. Hence $\beta \in \alpha[n]$ by the lemma. □

DEFINITION. Let $G_\alpha(n)$ denote the cardinality of the finite set $\alpha[n]$. Then immediately from the definition of $\alpha[n]$ we have

$$G_\alpha(n) = \begin{cases} 0 & \text{if } \alpha = 0 \\ G_{\alpha-1}(n) + 1 & \text{if Succ}(\alpha) \\ G_{\alpha(n)}(n) & \text{if Lim}(\alpha). \end{cases}$$

The hierarchy of functions G_α is called the "*slow growing*" hierarchy.

LEMMA. *If $\alpha = \delta + \omega^\gamma$ then for all n*

$$G_\alpha(n) = G_\delta(n) + (n+1)^{G_\gamma(n)}.$$

Therefore for each $\alpha < \varepsilon_0$, $G_\alpha(n)$ is the elementary function which results by substituting $n+1$ for every occurrence of ω in the Cantor Normal Form of α.

PROOF. By induction on γ. If $\gamma = 0$ then $\alpha = \delta + 1$, so $G_\alpha(n) = G_\delta(n) + 1 = G_\delta(n) + (n+1)^0$ as required. If γ is a successor then α is a limit and $\alpha(n) = \delta + \omega^{\gamma-1} \cdot (n+1)$, so by $n+1$ applications of the induction hypothesis for $\gamma - 1$ we have $G_\alpha(n) = G_{\alpha(n)}(n) = G_\delta(n) + (n+1)^{G_{\gamma-1}(n)} \cdot (n+1) = G_\delta(n) + (n+1)^{G_\gamma(n)}$ since $G_{\gamma-1}(n) + 1 = G_\gamma(n)$. Finally, if γ is a limit then $\alpha(n) = \delta + \omega^{\gamma(n)}$, so applying the induction hypothesis to $\gamma(n)$, we have $G_\alpha(n) = G_{\alpha(n)}(n) = G_\delta(n) + (n+1)^{G_{\gamma(n)}(n)}$ which immediately gives the desired result since $G_{\gamma(n)}(n) = G_\gamma(n)$ by definition. □

DEFINITION (Coding ordinals). Encode each ordinal $\beta = \omega^{\beta_1} \cdot b_1 + \omega^{\beta_2} \cdot b_2 + \cdots + \omega^{\beta_l} \cdot b_l$ by the sequence number $\bar{\beta}$ constructed recursively as follows:

$$\bar{\beta} = \langle \langle \bar{\beta}_1, b_1 \rangle, \langle \bar{\beta}_2, b_2 \rangle, \ldots, \langle \bar{\beta}_l, b_l \rangle \rangle.$$

The ordinal 0 is coded by the empty sequence number, also 0. Note that $\bar{\beta}$ is numerically greater than the maximum coefficient of β, and greater than the codes $\bar{\beta}_i$ of all its exponents, and their exponents etcetera.

LEMMA. (a) *There is an elementary function $h(m,n)$ such that, with $m = \bar{\beta}$,*

$$h(\bar{\beta}, n) = \begin{cases} 0 & \text{if } \beta = 0 \\ \overline{\beta - 1} & \text{if Succ}(\beta) \\ \overline{\beta(n)} & \text{if Lim}(\beta). \end{cases}$$

(b) *For each fixed $\alpha < \varepsilon_0$ there is an elementary well-ordering $\prec_\alpha \subset \mathbb{N}^2$ such that for all $b, c \in \mathbb{N}$, $b \prec_\alpha c$ if and only if $b = \bar{\beta}$ and $c = \bar{\gamma}$ for some $\beta < \gamma < \alpha$.*

PROOF. (a) Thinking of m as a $\bar{\beta}$, define $h(m,n)$ as follows: First set $h(0,n) = 0$. Then if m is a non-zero sequence number, see if its final (rightmost) component $\pi_2(m)$ is a pair $\langle m', n' \rangle$. If so, and $m' = 0$ but $n' \neq 0$, then β is a successor and the code of its predecessor, $h(m,n)$, is then defined to be the new sequence number obtained by reducing n' by one (or removing this final component altogether if $n' = 1$). Otherwise if $\pi_2(m) = \langle m', n' \rangle$ where m' and n' are both positive, then β is a limit of the form $\delta + \omega^\gamma \cdot n'$ where $m' = \bar{\gamma}$. Now let k be the code of $\delta + \omega^\gamma \cdot (n' - 1)$, obtained by reducing n' by one inside m (or if $n' = 1$, deleting the final component from m). Set k aside for the moment. At the "righthand end" of β we have a spare ω^γ which, in order to produce $\beta(n)$, must be reduced to $\omega^{\gamma-1} \cdot (n+1)$ if $\mathrm{Succ}(\gamma)$, or to $\omega^{\gamma(n)}$ if $\mathrm{Lim}(\gamma)$. Therefore the required code $h(m,n)$ of $\beta(n)$ will in this case be obtained by tagging onto the end of the sequence number k one extra pair coding this additional term. But if we assume inductively that $h(m',n)$ has already been defined for $m' < m$ then this additional component must be either $\langle h(m',n), n+1 \rangle$ if $\mathrm{Succ}(\gamma)$ or $\langle h(m',n), 1 \rangle$ if $\mathrm{Lim}(\gamma)$.

This defines $h(m,n)$, once we agree to set its value to zero in all extraneous cases where m is not a sequence number of the right form. However the definition so far given is a primitive recursion (depending on previous values for smaller m's). To make it elementary we need to check that $h(m,n)$ is also elementarily bounded, for then h is defined by "limited recursion" from elementary functions, and we know that the result will then be an elementary function. Now when m codes a successor then clearly, $h(m,n) < m$. In the limit case, $h(m,n)$ is obtained from the sequence number k (numerically smaller than m) by adding one new pair on the end. Recall that an extra item i is tagged onto the end of a sequence number k by the function $\pi(k,i)$ which is quadratic in k and i. If the item added is the pair $\langle h(m',n), n+1 \rangle$ where $\mathrm{Succ}(\gamma)$, then $h(m',n) < m$ and so $h(m,n)$ is numerically bounded by some fixed polynomial in m and n. In the other case, however, all we can say immediately is that $h(m,n)$ is numerically less than some fixed polynomial of m and $h(m',n)$. But since m' codes an exponent in the Cantor Normal Form coded by m, this second polynomial cannot be iterated more than d times, where d is the "exponential height" of the normal form. Therefore $h(m,n)$ is bounded by some d-times iterated polynomial of $m+n$. Since $d < m$ it is therefore bounded by the elementary function $2^{2^{c(m+n)}}$ for some constant c. Thus $h(m,n)$ is defined by limited recursion, so it is elementary.

(b) Fix $\alpha < \varepsilon_0$ and let d be the exponential height of its Cantor Normal Form. We use the function h just defined in part (a), and note that if we only apply it to codes for ordinals below α, they will all have exponential height $\leq d$, and so with this restriction we can consider h as being bounded by some

fixed polynomial of its two arguments. Define $g(0,n) = \bar{\alpha}$ and $g(i+1,n) = h(g(i,n),n)$, and notice that g is therefore bounded by an i-times iterated polynomial, so g is defined by an elementarily limited recursion from h, and hence is itself elementary.

Now define $b \prec_\alpha c$ if and only if $c \neq 0$ and there are i and j such that $0 < i < j \leq G_\alpha(\max(b,c)+1)$ and $g(i,\max(b,c)) = c$ and $g(j,\max(b,c)) = b$. Since the functions g and G_α are elementary, and since the quantifiers are bounded, the relation $\prec_\alpha$ is elementary. Furthermore by the properties of h it is clear that if $i < j$ then $g(i,\max(b,c))$ codes an ordinal greater than $g(j,\max(b,c))$ (provided the first is not zero). Hence if $b \prec_\alpha c$ then $b = \bar{\beta}$ and $c = \bar{\gamma}$ for some $\beta < \gamma < \alpha$.

We must show the converse, so suppose $b = \bar{\beta}$ and $c = \bar{\gamma}$ where $\beta < \gamma < \alpha$. Then since the code of an ordinal is greater than its maximum coefficient, we have $\beta \in \alpha[\max(b,c)]$ and $\gamma \in \alpha[\max(b,c)]$. This means that the sequence starting with α and at each stage descending from a δ to either $\delta - 1$ if $\mathrm{Succ}(\delta)$ or $\delta(\max(b,c))$ if $\mathrm{Lim}(\delta)$, must pass through first γ and later, β. In terms of codes it means that there is an i and a j such that $0 < i < j$ and $g(i,\max(b,c)) = c$ and $g(j,\max(b,c)) = b$. Thus $b \prec_\alpha c$ holds if we can show that $j \leq G_\alpha(\max(b,c)+1)$. In the descending sequence just described, only the successor stages actually contribute an element $\delta - 1$ to $\alpha[\max(b,c)]$. At the limit stages, $\delta(\max(b,c))$ does not get put in. However although $\delta(n)$ does not belong to $\delta[n]$, it does belong to $\delta[n+1]$. Therefore all the ordinals in the descending sequence lie in $\alpha[\max(b,c)+1]$. So j can be no bigger than the cardinality of this set, which is $G_\alpha(\max(b,c)+1)$. This completes the proof. □

Thus the principles of transfinite induction and transfinite recursion over initial segments of the ordinals below ε_0, can all be expressed in the language of elementary recursive arithmetic.

1.1.2. The fast growing hierarchy and ε_0-recursion.

Definition. The "Hardy Hierarchy" $\{H_\alpha\}_{\alpha<\varepsilon_0}$ is defined by recursion on α thus (cf. Hardy (1904)):

$$H_\alpha(n) = \begin{cases} n & \text{if } \alpha = 0 \\ H_{\alpha-1}(n+1) & \text{if } \mathrm{Succ}(\alpha) \\ H_{\alpha(n)}(n) & \text{if } \mathrm{Lim}(\alpha). \end{cases}$$

The "*Fast Growing Hierarchy*" $\{F_\alpha\}_{\alpha<\varepsilon_0}$ is defined by recursion on α thus:

$$F_\alpha(n) = \begin{cases} n+1 & \text{if } \alpha = 0 \\ F_{\alpha-1}^{n+1}(n) & \text{if } \mathrm{Succ}(\alpha) \\ F_{\alpha(n)}(n) & \text{if } \mathrm{Lim}(\alpha) \end{cases}$$

where $F_{\alpha-1}^{n+1}(n)$ is the $n+1$-times iterate of $F_{\alpha-1}$ on n.

NOTE. The H_α and F_α functions could equally well be defined purely number-theoretically, by working over the well-orderings $\prec_\alpha$ instead of directly over the ordinals themselves. Thus they are ε_0-recursive functions.

LEMMA. *For all α, β and all n,*
(a) $H_{\alpha+\beta}(n) = H_\alpha(H_\beta(n))$,
(b) $H_{\omega^\alpha}(n) = F_\alpha(n)$.

PROOF. The first part is proven by induction on β, the unstated assumption being that the Cantor Normal Form of $\alpha + \beta$ is just the result of concatenating their two separate Cantor Normal Forms, so that $(\alpha+\beta)(n) = \alpha+\beta(n)$. This of course requires that the leading exponent in the normal form of β is not greater than the final exponent in the normal form of α. We shall always make this assumption when writing $\alpha + \beta$.

If $\beta = 0$ the equation holds trivially because H_0 is the identity function. If $\mathrm{Succ}(\beta)$ then by the definition of the Hardy functions and the induction hypothesis for $\beta - 1$,

$$H_{\alpha+\beta}(n) = H_{\alpha+(\beta-1)}(n+1) = H_\alpha(H_{\beta-1}(n+1)) = H_\alpha(H_\beta(n)).$$

If $\mathrm{Lim}(\beta)$ then by the induction hypothesis for $\beta(n)$,

$$H_{\alpha+\beta}(n) = H_{\alpha+\beta(n)}(n) = H_\alpha(H_{\beta(n)}(n)) = H_\alpha(H_\beta(n)).$$

The second part is proved by induction on α. If $\alpha = 0$ then $H_{\omega^0}(n) = H_1(n) = n+1 = F_0(n)$. If $\mathrm{Succ}(\alpha)$ then by the limit case of the definition of H, the induction hypothesis, and the first part above,

$$H_{\omega^\alpha}(n) = H_{\omega^{\alpha-1}\cdot(n+1)}(n) = H_{\omega^{\alpha-1}}^{n+1}(n) = F_{\alpha-1}^{n+1}(n) = F_\alpha(n).$$

If $\mathrm{Lim}(\alpha)$ then the equation follows immediately by the induction hypothesis for $\alpha(n)$. This completes the proof. □

LEMMA. *For each $\alpha < \varepsilon_0$, H_α is strictly increasing and $H_\beta(n) < H_\alpha(n)$ whenever $\beta \in \alpha[n]$. The same holds for F_α, with the slight restriction that $n \neq 0$, for when $n = 0$ we have $F_\alpha(0) = 1$ for all α.*

PROOF. By induction on α. The case $\alpha = 0$ is trivial since H_0 is the identity function and $0[n]$ is empty. If $\mathrm{Succ}(\alpha)$ then H_α is $H_{\alpha-1}$ composed with the successor function, so it is strictly increasing by the induction hypothesis. Furthermore if $\beta \in \alpha[n]$ then either $\beta \in (\alpha-1)[n]$ or $\beta = \alpha - 1$ so, again by the induction hypothesis, $H_\beta(n) \leq H_{\alpha-1}(n) < H_{\alpha-1}(n+1) = H_\alpha(n)$. If $\mathrm{Lim}(\alpha)$ then $H_\alpha(n) = H_{\alpha(n)}(n) < H_{\alpha(n)}(n+1)$ by the induction hypothesis. But as noted previously, $\alpha(n) \in \alpha[n+1] = \alpha(n+1)[n+1]$, so by applying the induction hypothesis to $\alpha(n+1)$ we have $H_{\alpha(n)}(n+1) < H_{\alpha(n+1)}(n+1) = H_\alpha(n+1)$. Thus $H_\alpha(n) < H_\alpha(n+1)$. Furthermore if

$\beta \in \alpha[n]$ then $\beta \in \alpha(n)[n]$ so $H_\beta(n) < H_{\alpha(n)}(n) = H_\alpha(n)$ straightaway by the induction hypothesis for $\alpha(n)$.

The same holds for $F_\alpha = H_{\omega^\alpha}$ provided we restrict to $n \neq 0$ since if $\beta \in \alpha[n]$ we then have $\omega^\beta \in \omega^\alpha[n]$. This completes the proof. □

LEMMA. *If $\beta \in \alpha[n]$ then $F_{\beta+1}(m) \leq F_\alpha(m)$ for all $m \geq n$.*

PROOF. By induction on α, the zero case being trivial. If α is a successor then either $\beta \in (\alpha - 1)[n]$ in which case the result follows straight from the induction hypothesis, or $\beta = \alpha - 1$ in which case it's immediate. If α is a limit then we have $\beta \in \alpha(n)[n]$ and hence by the induction hypothesis, $F_{\beta+1}(m) \leq F_{\alpha(n)}(m)$. But $F_{\alpha(n)}(m) \leq F_\alpha(m)$ either by definition of F in case $m = n$, or by the last lemma when $m > n$ since then $\alpha(n) \in \alpha[m]$. □

DEFINITION (α-recursion).

(a) An *α-recursion* is a function-definition of the following form, defining $f\colon \mathbb{N}^{k+1} \to \mathbb{N}$ from given functions $g_0, g_1, \ldots, g_s$ by two clauses (in the second, $n \neq 0$):

$$f(0, \vec{m}) = g_0(\vec{m})$$
$$f(n, \vec{m}) = T(g_1, \ldots, g_s, f_{\prec n}, n, \vec{m})$$

where $T(g_1, \ldots, g_s, f_{\prec n}, n, \vec{m})$ is a fixed term built up from the number-variables $n, \vec{m}$ by applications of the functions $g_1, \ldots, g_s$ and the function $f_{\prec n}$ given by

$$f_{\prec n}(n', \vec{m}) = \begin{cases} f(n', \vec{m}) & \text{if } n' \prec_\alpha n \\ 0 & \text{otherwise.} \end{cases}$$

It is always assumed, when doing α-recursion, that $\alpha \neq 0$.

(b) An *unnested* α-recursion is one of the special form:

$$f(0, \vec{m}) = g_0(\vec{m})$$
$$f(n, \vec{m}) = g_1(n, \vec{m}, f(g_2(n, \vec{m}), \ldots, g_{k+2}(n, \vec{m})))$$

with just one recursive call on f where $g_2(n, \vec{m}) \prec_\alpha n$ for all n and all $\vec{m}$.

(c) Let $\varepsilon_0(0) = \omega$ and $\varepsilon_0(i+1) = \omega^{\varepsilon_0(i)}$. Then for each fixed i, a function is said to be *$\varepsilon_0(i)$-recursive* if it can be defined from primitive recursive functions by successive substitutions and α-recursions with $\alpha < \varepsilon_0(i)$. It is *unnested $\varepsilon_0(i)$-recursive* if all the α-recursions used in its definition are unnested. It is *ε_0-recursive* if it is $\varepsilon_0(i)$-recursive for some (any) i.

NOTE. The $\varepsilon_0(0)$-recursive functions are just the primitive recursive ones, since if $\alpha < \omega$ then α-recursion is just a finitely-iterated substitution. So the definition of $\varepsilon_0(0)$-recursion simply amounts to the closure of

the primitive recursive functions under substitution, which of course does not enlarge the primitive recursive class.

LEMMA (Bounds for α-recursion). *Suppose f is defined from $g_1, \ldots, g_s$ by an α-recursion:*

$$f(0, \vec{m}) = g_0(\vec{m})$$
$$f(n, \vec{m}) = T(g_1, \ldots, g_s, f_{\prec n}, n, \vec{m})$$

where for each $i \leq s$, $g_i(\vec{a}) < F_\beta(k + \max \vec{a})$ for all numerical arguments $\vec{a}$. (The β and k are arbitrary constants, but it is assumed that the last exponent in the Cantor Normal Form of β is $\geq$ the first exponent in the normal form of α, so that $\beta + \alpha$ is automatically in Cantor Normal Form). Then there is a constant d such that for all $n, \vec{m}$,

$$f(n, \vec{m}) < F_{\beta+\alpha}(k + 2d + \max(n, \vec{m})).$$

PROOF. The constant d will be the depth of nesting of the term T, where variables have depth of nesting 0 and each compositional term $g(T_1, \ldots, T_l)$ has depth of nesting one greater than the maximum depth of nesting of the subterms T_j.

First suppose n lies in the field of the well-ordering $\prec_\alpha$. Then $n = \bar{\gamma}$ for some $\gamma < \alpha$. We claim by induction on γ that

$$f(n, \vec{m}) < F_{\beta+\gamma+1}(k + 2d + \max(n, \vec{m})).$$

This holds immediately when $n = 0$, because $g_0(\vec{m}) < F_\beta(k + \max \vec{m})$ and F_β is strictly increasing and bounded by $F_{\beta+1}$. So suppose $n \neq 0$ and assume the claim for all $n' = \bar{\delta}$ where $\delta < \gamma$.

Let T' be any subterm of $T(g_1, \ldots, g_s, f_{\prec n}, n, \vec{m})$ with depth of nesting d', built up by application of one of the functions $g_1, \ldots, g_s$ or $f_{\prec n}$ to subterms $T_1, \ldots, T_l$. Now assume (for a sub-induction on d') that each of these T_j's has numerical value v_j less than $F^{2(d'-1)}_{\beta+\gamma}(k + 2d + \max(n, \vec{m}))$. If T' is obtained by application of one of the functions g_i then its numerical value will be

$$\begin{aligned} g_i(v_1, \ldots, v_l) &< F_\beta(k + F^{2(d'-1)}_{\beta+\gamma}(k + 2d + \max(n, \vec{m}))) \\ &< F^{2d'}_{\beta+\gamma}(k + 2d + \max(n, \vec{m})) \end{aligned}$$

since if $k < u$ then $F_\beta(k + u) < F_\beta(2u) < F^2_\beta(u)$ provided $\beta \neq 0$. On the other hand, if T' is obtained by application of the function $f_{\prec n}$, its value will be $f(v_1, \ldots, v_l)$ if $v_1 \prec_\alpha n$, or 0 otherwise. Suppose $v_1 = \bar{\delta} \prec_\alpha \bar{\gamma}$. Then by the induction hypothesis,

$$f(v_1, \ldots, v_l) < F_{\beta+\delta+1}(k + 2d + \max \vec{v}) \leq F_{\beta+\gamma}(k + 2d + \max \vec{v})$$

because v_1 is greater than the maximum coefficient of δ, so $\delta \in \gamma[v_1]$, so $\beta + \delta \in (\beta + \gamma)[v_1]$ and hence $F_{\beta+\delta+1}$ is bounded by $F_{\beta+\gamma}$ on arguments

$\geq v_1$. Therefore, inserting the assumed bounds for the v_j, we have

$$f(v_1,\ldots,v_l) < F_{\beta+\gamma}(k+2d+F^{2(d'-1)}_{\beta+\gamma}(k+2d+\max(n,\vec{m})))$$

and then by the same argument as before,

$$f(v_1,\ldots,v_l) < F^{2d'}_{\beta+\gamma}(k+2d+\max(n,\vec{m}))).$$

We have now shown that the value of every subterm of T with depth of nesting d' is less than $F^{2d'}_{\beta+\gamma}(k+2d+\max(n,\vec{m})))$. Applying this to T itself with depth of nesting d we thus obtain

$$f(n,\vec{m}) < F^{2d}_{\beta+\gamma}(k+2d+\max(n,\vec{m}))) < F_{\beta+\gamma+1}(k+2d+\max(n,\vec{m})))$$

as required. This proves the claim.

To derive the result of the lemma is now easy. If $n=\bar{\gamma}$ lies in the field of $\prec_\alpha$ then $\beta+\gamma \in (\beta+\alpha)[n]$ and so

$$f(n,\vec{m}) < F_{\beta+\gamma+1}(k+2d+\max(n,\vec{m}))) \leq F_{\beta+\alpha}(k+2d+\max(n,\vec{m}))).$$

If n does not lie in the field of $\prec_\alpha$ then the function $f_{\prec n}$ is the constant zero function, and so in evaluating $f(n,\vec{m})$ by the term T only applications of the g_i-functions come into play. Therefore a much simpler version of the above argument gives the desired

$$f(n,\vec{m}) < F^{2d}_{\beta}(k+2d+\max(n,\vec{m})) < F_{\beta+\alpha}(k+2d+\max(n,\vec{m}))$$

since $\alpha \neq 0$. This completes the proof. □

THEOREM. *For each i, a function is $\varepsilon_0(i)$-recursive if and only if it is register-machine computable in a number of steps bounded by F_α for some $\alpha < \varepsilon_0(i)$.*

PROOF. For the "if" part, recall that for every register-machine computable function g there is an elementary function U such that for all arguments $\vec{m}$, if $s(\vec{m})$ bounds the number of steps needed to compute $g(\vec{m})$ then $g(\vec{m}) = U(\vec{m}, s(\vec{m}))$. Thus if g is computable in a number of steps bounded by F_α, this means that g can be defined from F_α by the substitution

$$g(\vec{m}) = U(\vec{m}, F_\alpha(\max \vec{m})).$$

Hence g will be $\varepsilon_0(i)$-recursive if F_α is. We therefore need to show that if $\alpha < \varepsilon_0(i)$ then F_α is $\varepsilon_0(i)$-recursive. This is clearly true when $i=0$ since then α is finite, and the finite levels of the F hierarchy are all primitive recursive, and therefore $\varepsilon_0(0)$-recursive. Suppose then, that $i>0$, and that $\alpha = \omega^{\gamma_1}\cdot c_1 + \cdots + \omega^{\gamma_k}\cdot c_k$ is less than $\varepsilon_0(i)$. Adding one to each exponent, and inserting a successor term at the end, produces the ordinal $\beta = \alpha' + n$ where α' is the limit $\omega^{\gamma_1+1}\cdot c_1 + \cdots + \omega^{\gamma_k+1}\cdot c_k$. Since $i>0$ it is still the case that $\beta < \varepsilon_0(i)$. Obviously, from the code for α, here denoted a, we can elementarily compute the code for α', denoted a', and then $b = \pi(a', \langle 0, n\rangle)$

will be the code for β. Conversely from such a b we can elementarily decode a' and hence a, and also the n. Choosing a large enough $\delta < \varepsilon_0(i)$ so that $\beta < \delta$, we can now define a function $f(b, m)$ by δ-recursion, with the property that when b is the code for $\beta = \alpha' + n$, then $f(b, m) = F^n_\alpha(m)$. To explicate matters we shall expose the components from which b is constructed by writing $b = (a, n)$. Then the recursion defining $f(b, m) = f((a, n), m)$ has the following form, using the elementary function $h(a, n)$ defined earlier, which gives the code for $\alpha - 1$ if $\mathrm{Succ}(\alpha)$, or $\alpha(n)$ if $\mathrm{Lim}(\alpha)$:

$$f((a,n),m) = \begin{cases} m+n & \text{if } a = 0 \text{ or } n = 0 \\ f((h(a,m), m+1), m) & \text{if } \mathrm{Succ}(a) \text{ and } n = 1 \\ f((h(a,m), 1), m) & \text{if } \mathrm{Lim}(a) \text{ and } n = 1 \\ f((a,1), f((a,n-1),m)) & \text{if } n > 1 \\ 0 & \text{otherwise.} \end{cases}$$

Clearly then, f is $\varepsilon_0(i)$-recursive, and $F_\alpha(m) = f((\bar{\alpha}, 1), m)$, so F_α is $\varepsilon_0(i)$-recursive for every $\alpha < \varepsilon_0(i)$.

For the "only if" part note first that the number of steps needed to compute a compositional term $g(T_1, \ldots, T_l)$ is the sum of the numbers of steps needed to compute all the subterms T_j, plus the number of steps needed to compute $g(v_1, \ldots, v_l)$ where v_j is the value of T_j. Furthermore, in a register-machine computation, these values v_j are bounded by the number of computation steps plus the maximum input. This means that we can compute a bound on the computation-steps for any such term, and we can do it elementarily from given bounds for the input data. Now suppose $f(n, \vec{m}) = T(g_1, \ldots, g_s, f_{\prec n}, n, \vec{m})$ is any recursion-step of an α-recursion. Then if we are given bounding functions on the numbers of steps to compute each of the g_i's, and we assume inductively that we already have a bound on the number of steps to compute $f(n', -)$ whenever $n' \prec_\alpha n$, it follows that we can elementarily estimate a bound on the number of steps to compute $f(n, \vec{m})$. In other words, for any function defined by an α-recursion from given functions $\vec{g}$, a bounding function (on the number of steps needed to compute f) is also definable by α-recursion from given bounding functions for the g's. Exactly the same thing holds for primitive recursions. But in the preceding lemma we showed that as we successively define functions by α-recursions, with $\alpha < \varepsilon_0(i)$, their values are bounded by functions $F_{\beta+\alpha}$ where also, $\beta < \varepsilon_0(i)$. But $\varepsilon_0(i)$ is closed under addition, so $\beta + \alpha < \varepsilon_0(i)$. Hence every $\varepsilon_0(i)$-recursive function is register-machine computable in a number of steps bounded by some F_γ where $\gamma < \varepsilon_0(i)$. This completes the proof. □

The following reduction of nested to unnested recursion is due to Tait (1961); see also Fairtlough and Wainer (1992) .

COROLLARY. *For each i, a function is $\varepsilon_0(i)$-recursive if and only if it is unnested $\varepsilon_0(i+1)$-recursive.*

PROOF. By the Theorem, every $\varepsilon_0(i)$-recursive function is computable in "time" bounded by $F_\alpha = H_{\omega^\alpha}$ where $\alpha < \varepsilon_0(i)$. It is therefore primitive recursively definable from H_{ω^α}. But H_{ω^α} is defined by an unnested ω^α-recursion, and clearly, $\omega^\alpha < \varepsilon_0(i+1)$. Hence arbitrarily nested $\varepsilon_0(i)$-recursions are reducible to unnested $\varepsilon_0(i+1)$-recursions.

Conversely, suppose f is defined from given functions $g_0, g_1, \ldots, g_{k+2}$ by an unnested α-recursion where $\alpha < \varepsilon_0(i+1)$:

$$f(0, \vec{m}) = g_0(\vec{m})$$
$$f(n, \vec{m}) = g_1(n, \vec{m}, f(g_2(n, \vec{m}), \ldots, g_{k+2}(n, \vec{m})))$$

with $g_2(n, \vec{m}) \prec_\alpha n$ for all n and $\vec{m}$. Then the number of recursion steps needed to compute $f(n, \vec{m})$ is $f'(n, \vec{m})$ where

$$f'(0, \vec{m}) = 0$$
$$f'(n, \vec{m}) = 1 + f'(g_2(n, \vec{m}), \ldots, g_{k+2}(n, \vec{m}))$$

and f is then primitive recursively definable from $g_2, \ldots, g_{k+2}$ and any bound for f'. Now assume that the given functions g_j are all primitive recursively definable from, and bounded by H_β where $\beta < \varepsilon_0(i+1)$. Then a similar, but easier, argument to that used in proving the lemma above providing bounds for α-recursion shows that $f'(n, \vec{m})$ is bounded by $H_{\beta \cdot \gamma}$ where $n = \bar{\gamma}$. This is simply because

$$H_{\beta\cdot(\gamma+1)}(x) = H_{\beta\cdot\gamma+\beta}(x) = H_{\beta\cdot\gamma}(H_\beta(x)).$$

Therefore f is primitive recursively definable from H_β and $H_{\beta\cdot\alpha}$. Clearly, since $\beta, \alpha < \varepsilon_0(i+1)$ we may choose $\beta = \omega^{\beta'}$ and $\alpha = \omega^{\alpha'}$ for appropriate $\beta', \alpha' < \varepsilon_0(i)$. Then $H_\beta = F_{\beta'}$ and $H_{\beta\cdot\alpha} = F_{\beta'+\alpha'}$ where of course, $\beta' + \alpha' < \varepsilon_0(i)$. Therefore f is $\varepsilon_0(i)$-recursive. □

1.1.3. Arithmetical theories. We aim at proving that for every $\alpha < \varepsilon_0(i)$, with $i > 0$, the function F_α is "provably recursive" in certain arithmetical theories. Here we define what these theories are, and also define the notion of provable recursiveness.

Let $\Delta_0(\exp)$ be the arithmetical theory based on the language

$$\{=, 0, S, P, +, \dot{-}, \cdot, \exp_2\}$$

where S, P denote the successor and predecessor functions. We shall generally use infix notations $x+1$, $x \dot{-} 1$, 2^x rather than the more formal $S(x)$,

$P(x)$, $\exp_2(x)$ etcetera. The axioms of $\mathrm{I}\Delta_0(\exp)$ are the usual axioms for equality, the following defining axioms for the constants:

$$\begin{array}{ll} x+1 \neq 0 & x+1 = y+1 \to x = y \\ 0 \dot{-} 1 = 0 & (x+1) \dot{-} 1 = x \\ x+0 = x & x+(y+1) = (x+y)+1 \\ x \dot{-} 0 = x & x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 \\ x \cdot 0 = 0 & x \cdot (y+1) = (x \cdot y) + x \\ 2^0 = 1\,(= 0+1) & 2^{x+1} = 2^x + 2^x \end{array}$$

and the axiom-scheme of "bounded induction":

$$B(0) \to \forall_x (B(x) \to B(x+1)) \to B(x)$$

for all "bounded" formulas B as defined below. Furthermore, we allow the "cases" scheme

$$B(0) \to \forall_x B(x+1) \to B(x)$$

for arbitrary formulas B.

Definition. We write $t_1 \leq t_2$ for $t_1 \dot{-} t_2 = 0$ and $t_1 < t_2$ for $t_1 + 1 \leq t_2$, where t_1, t_2 denote arbitrary terms of the language.

A Δ_0- or *bounded* formula is a formula in the langage of $\mathrm{I}\Delta_0(\exp)$, in which all quantifiers occur bounded, thus $\forall_{x<t} B(x)$ stands for $\forall x(x < t \to B(x))$ and $\exists_{x<t} B(x)$ stands for $\exists x(x < t \wedge B(x))$ (similarly with $\leq$ instead of $<$).

A Σ_1*-formula* is any formula of the form $\exists_{x_1} \exists_{x_2} \ldots \exists_{x_k} B$ where B is a bounded formula. The prefix of unbounded existential quantifiers is allowed to be empty, thus bounded formulas are Σ_1.

The first task in any axiomatic theory is to develop, from the axioms, those basic algebraic properties which are going to be used frequently without further reference. Thus, in the case of $\mathrm{I}\Delta_0(\exp)$ we need to establish the usual associativity, commutativity and distributivity laws for addition and multiplication, the laws of exponentiation, and rules governing the relations $\leq$ and $<$ just defined.

Lemma. *In* $\mathrm{I}\Delta_0(\exp)$ *one can prove (the universal closures of) case-distinction:*

$$x = 0 \vee x = (x \dot{-} 1) + 1$$

the associativity laws for addition and multiplication:

$$x + (y + z) = (x + y) + z \quad \textit{and} \quad x \cdot (y \cdot z) = (x \cdot y) \cdot z$$

the distributivity law:

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

the commutativity laws:

$$x + y = y + x \quad \textit{and} \quad x \cdot y = y \cdot x$$

the law:

$$x \dot{-} (y + z) = (x \dot{-} y) \dot{-} z$$

and the exponentiation law:

$$2^{x+y} = 2^x \cdot 2^y.$$

PROOF. Since $0 = 0$ and $x + 1 = ((x + 1) \dot{-} 1) + 1$ by axioms, a trivial induction on x gives the cases-distinction. A straightforward induction on z gives associativity for $+$, and distributivity follows from this by an equally straightforward induction, again on z. Associativity of multiplication is proven similarly, but requires distributivity. The commutativity of $+$ is done by induction on y (or x) using sub-inductions to first prove $0 + x = x$ and $(y + x) + 1 = (y + 1) + x$. Commutativity of $\cdot$ is done similarly using $0 \cdot x = 0$ and $y \cdot x + x = (y + 1) \cdot x$, this latter requiring both associativity and commutativity of $+$. That $x \dot{-} (y + z) = (x \dot{-} y) \dot{-} z$ follows easily by a direct induction on z. The base-case for the exponentiation law is $2^{x+0} = 2^x = 0 + 2^x = 2^x \cdot 0 + 2^x = 2^x \cdot (0 + 1) = 2^x \cdot 2^0$ and the induction step needs distributivity to give $2^{x+y+1} = 2^x \cdot 2^y + 2^x \cdot 2^y = 2^x \cdot 2^{y+1}$. □

LEMMA. *The following (and their universal closures) are provable in* $\mathrm{I}\Delta_0(\exp)$*:*

(1) $x \leq 0 \leftrightarrow x = 0 \quad \textit{and} \quad \neg\, x < 0$
(2) $0 \leq x \quad \textit{and} \quad x \leq x \quad \textit{and} \quad x < x + 1$
(3) $x < y + 1 \leftrightarrow x \leq y$
(4) $x \leq y \leftrightarrow x < y \vee x = y$
(5) $x \leq y \wedge y \leq z \rightarrow x \leq z \quad \textit{and} \quad x < y \wedge y < z \rightarrow x < z$
(6) $x \leq y \vee y < x$
(7) $x < y \rightarrow x + z < y + z$
(8) $x < y \rightarrow x \cdot (z + 1) < y \cdot (z + 1)$
(9) $x < 2^x \quad \textit{and} \quad x < y \rightarrow 2^x < 2^y$.

PROOF. (1) This is an immediate consequence of the axioms $x \dot{-} 0 = x$ and $x + 1 \neq 0$. (2) A simple induction proves $0 \dot{-} x = 0$, that is $0 \leq x$. Another induction on y gives $(x + 1) \dot{-} (y + 1) = x \dot{-} y$, and then a further induction proves $x \dot{-} x = 0$, which is $x \leq x$. Replacing x by $x + 1$ then gives $x < x{+}1$. (3) This follows straight from the equation $(x{+}1) \dot{-} (y{+}1) = x \dot{-} y$. (4) From $x \leq x$ we obtain $x = y \rightarrow x \leq y$, and from $x \dot{-} y = (x{+}1) \dot{-} (y{+}1)$ we obtain $x < y \rightarrow x \leq y$, hence $x < y \vee x = y \rightarrow x \leq y$. The converse $x \leq y \rightarrow x < y \vee x = y$ is proven by a case-distinction on y, the case $y = 0$ being immediate from part 1. In the other case $y = (y \dot{-} 1){+}1$ and one obtains $x \leq y \rightarrow x \dot{-} (y \dot{-} 1) = 0 \vee x \dot{-} (y \dot{-} 1) = 1$ by a case-distinction on $x \dot{-} (y \dot{-} 1)$.

Since $(x+1) \dot{-} y = x \dot{-} (y \dot{-} 1)$ this gives $x \leq y \to x < y \vee x \dot{-} (y \dot{-} 1) = 1$. It therefore remains only to prove $x \dot{-} (y \dot{-} 1) = 1 \to x = y$. But this follows immediately from $x \dot{-} z \neq 0 \to x = z + (x \dot{-} z)$, which is proven by induction on z using $(z+1) + (x \dot{-} (z+1)) = z + (x \dot{-} (z+1)) + 1 = z + ((x \dot{-} z) \dot{-} 1) + 1 = z + (x \dot{-} z)$. (5) Transitivity of $\leq$ is proven by induction on z using parts 1 for the basis and 4 for the induction step. Then, by replacing x by $x+1$ and y by $y+1$, the transitivity of $<$ follows. (6) can be proved by induction on x. The basis is immediate from $0 \leq y$. The induction step is straightforward since $y < x \to y < x+1$ by transitivity, and $x \leq y \to x < y \vee x = y \to x+1 \leq y \vee y < x+1$ by previous facts. (7) requires a simple induction on z, the induction step being $x+z < y+z \to x+z+1 \leq y+z < y+z+1$. (8) follows from part 7 and transitivity by another easy induction on z. (9) Using part 7 and transitivity again, one easily proves by induction, $2^x < 2^{x+1}$. Then $x < 2^x$ follows straightforwardly by another induction, as does $x < y \to 2^x < 2^y$ by induction on y, the induction step being $x < y+1 \to x \leq y \to 2^x \leq 2^y \to 2^x < 2^{y+1}$ by means of transitivity. □

NOTE. All of the inductions used in the lemmas above are inductions on "open", i.e., quantifier-free, formulas.

Of course in any theory many new functions and relations can be defined out of the given constants. What we are interested in are those which can not only be *defined* in the language of the theory, but also can be *proven to exist*. This gives rise to the following definition.

DEFINITION. We say that a function $f\colon \mathbb{N}^k \to \mathbb{N}$ is *provably recursive* in an arithmetical theory T if there is a Σ_1 formula $F(\vec{x}, y)$, called a "defining formula" for f, such that

- $f(\vec{n}) = m$ if and only if $F(\vec{n}, m)$ is true; (in the standard model);
- $T \vdash \exists_y F(\vec{x}, y)$;
- $T \vdash F(\vec{x}, y) \to F(\vec{x}, y') \to y = y'$.

The main result on concerning this notion and $\mathrm{I}\Delta_0(\exp)$ is:

THEOREM. *A number-theoretic function is elementary if and only if it is provably recursive in* $\mathrm{I}\Delta_0(\exp)$.

PROOF. Exercise. □

We now introduce stronger theories, which fewer restrictions on the formulas allowed in the recursion scheme. At this point it is important to distinguish the (constructive) existential quantifier $\exists$ from the weak (or classical) one, which we write as $\tilde{\exists}$. The reason is that – as we shall see later – a proof of $\exists_x A$ always has algorithmic content, whereas a proof of $\tilde{\exists}_x A$

in general only has one if A is a bounded formula (and even in this case a proof of $\exists_x A$ has a more direct algorithmic content).

Call a formula $\exists$-*free* if it does not contain $\exists$; in the literature such formulas are also called *negative*. A formula is calles *almost* $\exists$-*free* or *almost negative* if $\exists$ only occurs in contexts $\exists_{\vec{x}} A_0$ with A_0 bounded. It turns out that we only need to allow induction over almost negative formulas to derive that the F_α are provably recursive. To determine exactly which instances of induction are needed we introduce some subclasses of the almost negative formulas. For formulas involving the constructive existential quantifier $\exists$ there is no prenex normal form, and hence the usual notions of Σ_n- and Π_n-formulas do not make sense. Instead we define the *level* of an almost negative formula as follows. Let A_0, B_0 etc. range over bounded formulas. Then $\mathrm{lev}(A_0) := 0$, and for unbounded formulas A we define $\mathrm{lev}(A)$ by

$$\mathrm{lev}(\exists_{\vec{x}} A) := 1,$$

$$\mathrm{lev}(A \to B) := \begin{cases} 1 & \text{if } A \equiv \exists_{\vec{x}} A_0 \text{ and } B \equiv \forall_{\vec{y}} B_0 \\ 1 & \text{if } A \equiv \forall_{\vec{x}} A_0 \text{ and } B \text{ bounded} \\ \max(\mathrm{lev}(A)+1, \mathrm{lev}(B)) & \text{otherwise,} \end{cases}$$

$$\mathrm{lev}(\forall_x A) := \begin{cases} 1 & \text{if } A \text{ bounded} \\ 2 & \text{if } A \equiv \exists_{\vec{y}} A_0 \\ \mathrm{lev}(A) & \text{otherwise.} \end{cases}$$

We can now define the arithmetical theories we want to consider.

Definition. The theories below are all based on $\mathrm{I}\Delta_0(\exp)$. They always contain the cases scheme $B(0) \to \forall_x B(x+1) \to B(x)$.

(a) $\mathbf{Z}_i$ is the theory with induction restricted to almost negative formulas with $\mathrm{lev}(A) \leq i$.
(b) $\mathrm{I}\Sigma_i$ is the theory with induction restricted to (negative) Σ_i-formulas.

It doesn't matter whether one restricts to Σ_i or Π_i induction formulas since, in the presence of the subtraction function, induction on a Π_i formula A is reducible to induction on its Σ_i dual $\neg A$, and vice-versa.

For if one replaces $A(a)$ by $\neg A(t \dot{-} a)$ in the induction axiom, and then contraposes (using the equivalence of $A \to B \to C$ and $A \to \neg C \to \neg B$), one obtains

$$A(t \dot{-} t) \to \forall_a (A(t \dot{-} (a+1)) \to A(t \dot{-} a)) \to A(t \dot{-} 0)$$

from which follows the induction axiom for $A(a)$ itself, since $t \dot{-} t = 0$, $t \dot{-} 0 = t$, and $t \dot{-} a = (t \dot{-} (a+1)) + 1$ if $t \dot{-} a \neq 0$.

Proof. Exercise. For negative formulas, $A \to B \to C$ and $A \to \neg C \to \neg B$ are equivalent. □

The main result concerning provable recursive functions and the theories $\mathrm{I}\Sigma_1$ and $\mathbf{Z}_1$ is due to Parsons (1966).

THEOREM. *For a number-theoretic function f the following are equivalent.*

(a) *f is primitive recursive;*
(b) *f is provably recursive in $\mathrm{I}\Sigma_1$;*
(c) *f is provably recursive in $\mathbf{Z}_1$.*

PROOF. Exercise. □

DEFINITION. A function $f\colon \mathbb{N}^k \to \mathbb{N}$ is called *classically provably recursive* in an arithmetical theory T if there is a Σ_1 formula $F(\vec{x}, y)$ (i.e., one obtained by prefixing finitely many unbounded classical existential quantifiers to a $\Delta_0(\exp)$ formula) such that

- $f(\vec{n}) = m$ if and only if $F(\vec{n}, m)$ is true; (in the standard model)
- $T \vdash \tilde{\exists}_y F(\vec{x}, y)$;
- $T \vdash F(\vec{x}, y) \to F(\vec{x}, y') \to y = y'$.

Note that by the Gödel-Gentzen translation (also known as "double negation translation") every proof of A in $\mathbf{Z}_i$ yields a proof of $\tilde{A}$ in $\mathrm{I}\Sigma_i$, but not conversely. Hence it is more general to prove that F_α is provably recursive in the theory $\mathbf{Z}_i$.

1.1.4. Provable recursiveness of H_α and F_α. We now prove that for every $\alpha < \varepsilon_0(i)$, with $i > 0$, the function F_α is provably recursive in the theory $\mathbf{Z}_{i+1}$.

Since all of the machinery we have developed for coding ordinals below ε_0 is elementary, we can safely assume that it is available to us in an appropriate conservative extension of $\mathrm{I}\Delta_0(\exp)$, and can in fact be defined (with all relevant properties proven) in $\mathrm{I}\Delta_0(\exp)$ itself. In particular we shall again make use of the function h such that, if a codes a successor ordinal α then $h(a, n)$ codes $\alpha - 1$, and if a codes a limit ordinal α then $h(a, n)$ codes $\alpha(n)$. Note that we can decide whether a codes a succesor ordinal ($\mathrm{Succ}(a)$) or a limit ordinal ($\mathrm{Lim}(a)$), by asking whether $h(a, 0) = h(a, 1)$ or not. It is easiest to develop first the provable recursiveness of the Hardy functions H_α, since they have a simpler, unnested recursive definition. The fast growing functions are then easily obtained by the equation $F_\alpha = H_{\omega^\alpha}$.

DEFINITION. Let $H(a, x, y, z)$ denote the following $\Delta_0(\exp)$ formula:

$$(z)_0 = \langle 0, y\rangle \wedge \pi_2(z) = \langle a, x\rangle \wedge$$
$$\forall_{i<\mathrm{lh}(z)}(\mathrm{lh}((z)_i) = 2 \wedge (i > 0 \to (z)_{i,0} > 0)) \wedge$$
$$\forall_{0<i<\mathrm{lh}(z)}(\mathrm{Succ}((z)_{i,0}) \to (z)_{i-1,0} = h((z)_{i,0}, (z)_{i,1}) \wedge (z)_{i-1,1} = (z)_{i,1}{+}1) \wedge$$

$$\forall_{0<i<\mathrm{lh}(z)}(\mathrm{Lim}((z)_{i,0}) \to (z)_{i-1,0} = h((z)_{i,0},(z)_{i,1}) \wedge (z)_{i-1,1} = (z)_{i,1})$$

LEMMA (Definability of H_α). *$H_\alpha(n) = m$ if and only if $\exists_z H(\bar\alpha, n, m, z)$ is true. Furthermore, for each $\alpha < \varepsilon_0$ we can prove in $\mathbf{Z}_1$,*

$$\exists_z H(\bar\alpha, x, y, z) \to \exists_z H(\bar\alpha, x, y', z) \to y = y'.$$

PROOF. The meaning of the formula $\exists_z H(\bar\alpha, n, m, z)$ is that there is a finite sequence of pairs $\langle \alpha_i, n_i\rangle$, beginning with $\langle 0, m\rangle$ and ending with $\langle \alpha, n\rangle$, such that at each $i > 0$, if $\mathrm{Succ}(\alpha_i)$ then $\alpha_{i-1} = \alpha_i - 1$ and $n_{i-1} = n_i + 1$, and if $\mathrm{Lim}(\alpha_i)$ then $\alpha_{i-1} = \alpha_i(n_i)$ and $n_{i-1} = n_i$. Thus by induction up along the sequence, and using the original definition of H_α, we easily see that for each $i > 0$, $H_{\alpha_i}(n_i) = m$, and thus at the end, $H_\alpha(n) = m$. Conversely, if $H_\alpha(n) = m$ then there must exist such a computation-sequence, and this proves the first part of the lemma.

For the second part notice that, by induction on the length of the computation-sequence s, we can prove, for each n, m, m', s, s' that

$$H(\bar\alpha, n, m, s) \to H(\bar\alpha, n, m', s') \to s = s' \wedge m = m'.$$

This proof can be formalized directly in $\mathrm{I}\Delta_0(\exp)$ to give

$$H(\bar\alpha, x, y, z) \to H(\bar\alpha, x, y', z') \to z = z' \wedge y = y'$$

and hence

$$\exists_z H(\bar\alpha, x, y, z) \to \exists_z H(\bar\alpha, x, y', z) \to y = y'. \qquad \square$$

REMARK. Thus in order for H_α to be provably recursive it remains only to prove (in the required theory) $\exists_y \exists_z H(\bar\alpha, x, y, z)$.

LEMMA. *In $\mathrm{I}\Delta_0(\exp)$ we can prove*

$$\exists_z H(\omega^a, x, y, z) \to \exists_z H(\omega^a \cdot c, y, w, z) \to \exists_z H(\omega^a \cdot (c+1), x, w, z)$$

where ω^a is the elementary term $\langle\langle a, 1\rangle\rangle$ which constructs, from the code a of an ordinal α, the code for the ordinal ω^α, and $b\cdot 0 = 0$, $b\cdot(z+1) = b\cdot z \oplus b$, with $\oplus$ the elementary function which computes $\overline{\alpha+\beta}$ from $\bar\alpha$ and $\bar\beta$.

PROOF. By assumption we have sequences s, s' satisfying $H(\omega^a, n, m, s)$ and $H(\omega^a \cdot c, m, k, s')$. Add $\omega^a \cdot c$ (in the sense of $\oplus$) to the first component of each pair in s. Then the last pair in s' and the first pair in s become identical. By concatenating the two – taking this double pair only once – construct an elementary term $t(s, s')$ satisfying $H(\omega^a \cdot (c+1), n, k, t)$. We can then prove

$$H(\omega^a, x, y, z) \to H(\omega^a \cdot c, y, w, z') \to H(\omega^a \cdot (c+1), x, w, t)$$

in a conservative extension of $\mathrm{I}\Delta_0(\exp)$, and hence in $\mathrm{I}\Delta_0(\exp)$ derive

$$\exists_z H(\omega^a, x, y, z) \to \exists_z H(\omega^a \cdot c, y, w, z) \to \exists_z H(\omega^a \cdot (c+1), x, w, z). \qquad \square$$

LEMMA. *Let $H(a)$ be the Π_2 formula $\forall_x \exists_y \exists_z H(a,x,y,z)$. Then with Π_2-induction we can prove the following:*

(a) $H(\omega^0)$.
(b) $\mathrm{Succ}(a) \to H(\omega^{h(a,0)}) \to H(\omega^a)$.
(c) $\mathrm{Lim}(a) \to \forall_x H(\omega^{h(a,x)}) \to H(\omega^a)$.

PROOF. The term $t_0 = \langle\langle 0, x+1\rangle, \langle 1, x\rangle\rangle$ witnesses $H(\omega^0, x, x+1, t_0)$ in $\mathrm{I}\Delta_0(\exp)$, so $H(\omega^0)$ is immediate.

With the aid of the lemma just proven we can derive

$$H(\omega^{h(a,0)}) \to H(\omega^{h(a,0)} \cdot c) \to H(\omega^{h(a,0)}) \cdot (c+1)$$

Therefore by Π_2 induction we obtain

$$H(\omega^{h(a,0)}) \to H(\omega^{h(a,0)} \cdot (x+1))$$

and then

$$H(\omega^{h(a,0)}) \to \exists_y \exists_z H(\omega^{h(a,0)} \cdot (x+1), x, y, z).$$

But there is an elementary term t_1 with the property

$$\mathrm{Succ}(a) \to H(\omega^{h(a,0)} \cdot (x+1), x, y, z) \to H(\omega^a, x, y, t_1)$$

since t_1 only needs to tagg onto the end of the sequence z the new pair $\langle \omega^a, x\rangle$, thus $t_1 = \pi(z, \langle \omega^a, x\rangle)$. Hence by the quantifier rules,

$$\mathrm{Succ}(a) \to H(\omega^{h(a,0)}) \to H(\omega^a).$$

The final case is now straightforward, since the term t_1 just constructed also gives

$$\mathrm{Lim}(a) \to H(\omega^{h(a,x)}, x, y, z) \to H(\omega^a, x, y, t_1)$$

and so by quantifier rules again,

$$\mathrm{Lim}(a) \to \forall_x H(\omega^{h(a,x)}) \to H(\omega^a). \qquad \square$$

DEFINITION (Structural Transfinite Induction). The *structural progressiveness* of a formula $A(a)$ is expressed by $\mathrm{SProg}_a A$, which is the conjunction of the formulas $A(0)$, $\forall_a(\mathrm{Succ}(a) \to A(h(a,0)) \to A(a))$, and $\forall_a(\mathrm{Lim}(a) \to \forall_x A(h(a,x)) \to A(a))$. The principle of *structural transfinite induction* up to an ordinal α is then the following axiom-scheme, for all formulas A:

$$\mathrm{SProg}_a A \to \forall_{a \prec \bar{\alpha}} A(a)$$

where $a \prec \bar{\alpha}$ means a lies in the field of the well-ordering $\prec_\alpha$, in other words $a = 0 \vee 0 \prec_\alpha a$.

NOTE. The last lemma shows that the Π_2 formula $H(\omega^a)$ is structural progressive, and that this is provable with Π_2-induction.

We now make use of a famous result of Gentzen (1936), which says that transfinite induction is provable in arithmetic up to any $\alpha < \varepsilon_0$. For later use we prove this fact in a slightly more general form, where one can recur to *all* points strictly below the present one, and need not refer to distinguished fundamental sequences.

DEFINITION (Transfinite Induction). The (general) *progressiveness* of a formula $A(a)$ is

$$\mathrm{Prog}_a A(a) := \forall_a\big(\forall_{b\prec a} A(b) \to A(a)\big).$$

The principle of *transfinite induction* up to an ordinal α is the scheme

$$\mathrm{Prog}_a A(a) \to \forall_{a\prec\bar\alpha} A(a)$$

where again $a \prec \bar\alpha$ means a lies in the field of the well-ordering $\prec_\alpha$.

It is easy to see that structural transfinite induction up to α is derivable from transfinite induction up to α.

PROOF. Let A be an arbitrary formula and assume $\mathrm{SProg}_a A$; we must show $\forall_{a\prec\bar\alpha} A(a)$. Using transfinite induction for the formula $a \prec \bar\alpha \to A(a)$ it suffices to prove

$$\forall_a\big(\forall_{b\prec a; b\prec\bar\alpha} A(b) \to a \prec \bar\alpha \to A(a)\big)$$

which is equivalent to

$$\forall_{a\prec\bar\alpha}\big(\forall_{b\prec a} A(b) \to A(a)\big).$$

This is easily proved from $\mathrm{SProg}_a A$, using the properties of the h function, and distinguishing the cases $a = 0$, $\mathrm{Succ}(a)$ and $\mathrm{Lim}(a)$. □

REMARK. Induction over an arbitrary well-founded set is an easy consequence. Comparisons are made by means of a "measure function" μ, into an initial segment of the ordinals. The principle of "well-founded induction" up to an ordinal α is

$$\mathrm{Prog}^\mu_x A(x) \to \forall_{x;\mu x\prec\bar\alpha} A(x)$$

where $\mathrm{Prog}^\mu_x A(x)$ expresses "μ-progressiveness" w.r.t. the measure function μ and the ordering $\prec := \prec_\alpha$

$$\mathrm{Prog}^\mu_x A(x) := \forall_a\big(\forall_{y;\mu y\prec a} A(y) \to \forall_{x;\mu x=a} A(x)\big).$$

One proves easily that well-founded induction up to an ordinal α is provable from transfinite induction up to α.

PROOF. Assume $\mathrm{Prog}^\mu_x A(x)$; we must show $\forall_{x;\mu x\prec\bar\alpha} A(x)$. Consider

$$B(a) := \forall_{x;\mu x=a} A(x).$$

It suffices to prove $\forall_{a\prec\bar\alpha}B(a)$, which is $\forall_{a\prec\bar\alpha}\forall_{x;\mu x=a}A(x)$. By transfinite induction it suffices to prove $\mathrm{Prog}_a B$, which is

$$\forall_a\big(\forall_{b\prec a}\forall_{y;\mu y=b}A(y)\to\forall_{x;\mu x=a}A(x)\big).$$

But this follows from the assumption $\mathrm{Prog}^\mu_x A(x)$, since $\forall_{b\prec a}\forall_{y;\mu y=b}A(y)$ implies $\forall_{y;\mu y\prec a}A(y)$. □

We now come to Gentzen's theorem. In the proof we will need some properties of $\prec$ which can all be proved in $\mathrm{I}\Delta_0(\mathrm{exp})$: irreflexivity and transitivity for $\prec$, and also (following Schütte)

$$a\prec 0\to A, \tag{1.1}$$

$$c\prec b\oplus\omega^0\to(c\prec b\to A)\to(c=b\to A)\to A, \tag{1.2}$$

$$a\oplus 0=a, \tag{1.3}$$

$$a\oplus(b\oplus c)=(a\oplus b)\oplus c, \tag{1.4}$$

$$0\oplus a=a, \tag{1.5}$$

$$\omega^a 0=0, \tag{1.6}$$

$$\omega^a(x+1)=\omega^a x\oplus\omega^a, \tag{1.7}$$

$$a\neq 0\to c\prec b\oplus\omega^a\to c\prec b\oplus\omega^{\mathrm{e}(a,b,c)}\mathrm{m}(a,b,c), \tag{1.8}$$

$$a\neq 0\to c\prec b\oplus\omega^a\to \mathrm{e}(a,b,c)\prec a, \tag{1.9}$$

where e and m denote the appropriate function constants and A is any formula. (The reader should check that e, m can be taken to be elementary.)

Theorem (Gentzen (1936)). *For every formula F with $\mathrm{lev}(F)=2$ and each $i>0$ we can prove in $\mathbf{Z}_{i+1}$ the principle of transfinite induction up to α for all $\alpha<\varepsilon_0(i)$.*

Proof. Starting with any formula $A(a)$ of level j, we construct the formula

$$A^+(a):=\forall_b(\forall_{c\prec b}A(c)\to\forall_{c\prec b\oplus\omega^a}A(c))$$

where, as mentioned above, $\oplus$ is the elementary addition function on ordinal-codes thus: $\bar\alpha\oplus\bar\gamma=\overline{\alpha+\gamma}$. Note that $\mathrm{lev}(A)=j$ implies $\mathrm{lev}(A^+)=j+1$. The crucial point is that

$$\mathbf{Z}_j\vdash\mathrm{Prog}_a A(a)\to\mathrm{Prog}_a A^+(a).$$

So assume $\mathrm{Prog}_a A(a)$, that is, $\forall_a(\forall_{b\prec a}A(b)\to A(a))$ and

$$\forall_{b\prec a}A^+(b) \tag{1.10}$$

We have to show $A^+(a)$. So assume further

$$\forall_{c\prec b}A(c) \tag{1.11}$$

and $c\prec b\oplus\omega^a$. We have to show $A(c)$.

If $a = 0$, then $c \prec b \oplus \omega^0$. By (1.2) it suffices to derive $A(c)$ from $c \prec b$ as well as from $c = b$. If $c \prec b$, then $A(c)$ follows from (1.11), and if $c = b$, then $A(c)$ follows from (1.11) and $\mathrm{Prog}_a A$.

If $a \neq 0$, from $c \prec b \oplus \omega^a$ we obtain $c \prec b \oplus \omega^{\mathrm{e}(a,b,c)}\mathrm{m}(a,b,c)$ by (1.8) and $\mathrm{e}(a,b,c) \prec a$ by (1.9). From (1.10) we obtain $A^+(\mathrm{e}(a,b,c))$. By the definition of $A^+(x)$ we get

$$\forall_{u \prec b \oplus \omega^{\mathrm{e}(a,b,c)} x} A(u) \to \forall_{u \prec (b \oplus \omega^{\mathrm{e}(a,b,c)} x) \oplus \omega^{\mathrm{e}(a,b,c)}} A(u)$$

and hence, using (1.4) and (1.7)

$$\forall_{u \prec b \oplus \omega^{\mathrm{e}(a,b,c)} x} A(u) \to \forall_{u \prec b \oplus \omega^{\mathrm{e}(a,b,c)}(x+1)} A(u).$$

Also from (1.11) and (1.6), (1.3) we obtain

$$\forall_{u \prec b \oplus \omega^{\mathrm{e}(a,b,c)} 0} A(u).$$

Using an appropriate instance of the induction schema we can conclude

$$\forall_{u \prec b \oplus \omega^{\mathrm{e}(a,b,c)} \mathrm{m}(a,b,c)} A(u)$$

and hence $A(c)$.

Now fix $i > 0$ and (throughout the rest of this proof) let $\prec$ denote the well-ordering $\prec_{\varepsilon_0(i)}$. Given any formula $F(v)$ of level 2 define $A(a)$ to be the formula $\forall_{v \prec a} F(v)$. Then A is also of level 2 and furthermore it is easy to see that $\mathrm{Prog}_v F(v) \to \mathrm{Prog}_a A(a)$ is derivable in $\mathrm{I}\Delta_0(\exp)$. Therefore by iterating the above procedure i times starting with $j = 2$, we obtain successively the formulas A^+, A^{++}, ... $A^{(i)}$ where $A^{(i)}$ has level $i + 2$ and

$$\mathbf{Z}_{i+1} \vdash \mathrm{Prog}_v F(v) \to \mathrm{Prog}_u A^{(i)}(u).$$

Now fix any $\alpha < \varepsilon_0(i)$ and choose k so that $\alpha \leq \varepsilon_0(i)(k)$. By applying $k+1$ times the progressiveness of $A^{(i)}(u)$, one obtains $A^{(i)}(\overline{k+1})$ without need of any further induction, since k is fixed. Therefore

$$\mathbf{Z}_{i+1} \vdash \mathrm{Prog}_v F(v) \to A^{(i)}(\overline{k+1}).$$

But by instantiating the outermost universally quantified variable of $A^{(i)}$ to zero we have $A^{(i)}(\overline{k+1}) \to A^{(i-1)}(\omega^{\overline{k+1}})$. Again instantiating to zero the outermost universally quantified variable in $A^{(i-1)}$ we similarly obtain $A^{(i-1)}(\omega^{\overline{k+1}}) \to A^{(i-2)}(\omega^{\omega^{\overline{k+1}}})$. Continuing in this way, and noting that $\varepsilon_0(i)(k)$ consists of an exponential stack of i ω's with $k+1$ on the top, we finally get down (after i steps) to

$$\mathbf{Z}_{i+1} \vdash \mathrm{Prog}_v F(v) \to A(\overline{\varepsilon_0(i)(k)}).$$

Since $A(\overline{\varepsilon_0(i)(k)})$ is just $\forall_{v \prec \overline{\varepsilon_0(i)(k)}} F(v)$ we have therefore proved, in $\mathbf{Z}_{i+1}$, transfinite induction for F up to $\varepsilon_0(i)(k)$, and hence up to the given α. $\square$

THEOREM. *For each i and every $\alpha < \varepsilon_0(i)$, the fast growing function F_α is provably recursive in $\mathbf{Z}_{i+1}$.*

PROOF. If $i = 0$ then α is finite and F_α is therefore primitive recursive, so it is provably recursive in $\mathbf{Z}_1$.

Now suppose $i > 0$. Since $F_\alpha = H_{\omega^\alpha}$ we need only show, for every $\alpha < \varepsilon_0(i)$, that H_{ω^α} is provably recursive in $\mathbf{Z}_{i+1}$. But we have shown above that its defining Π_2 formula $H(\omega^a)$ is provably progressive in $\mathbf{Z}_2$, and therefore by Gentzen's result,

$$\mathbf{Z}_{i+1} \vdash \forall_{a \prec \bar{\alpha}} H(\omega^a).$$

One further application of progressiveness then gives

$$\mathbf{Z}_{i+1} \vdash H(\omega^{\bar{\alpha}})$$

which, together with the definability of H_α proved above, shows that H_{ω^α} is provably recursiveness of in $\mathbf{Z}_{i+1}$. □

COROLLARY. *Any $\varepsilon_0(i)$-recursive function is provably recursive in $\mathbf{Z}_{i+1}$.*

PROOF. We have seen already that each $\varepsilon_0(i)$-recursive function is register-machine computable in a number of steps bounded by some F_α with $\alpha < \varepsilon_0(i)$. Consequently, each such function is primitive recursively, and in fact elementarily, definable from an F_α which itself is provably recursive in $\mathbf{Z}_{i+1}$. But primitive recursions only need Σ_1-inductions to prove them defined. Thus in $\mathbf{Z}_{i+1}$ we can prove the Σ_1-definability of all $\varepsilon_0(i)$-recursive functions. □

1.2. Gödel's T

Gödel (1958) proposed to extend Hilbert's concept of "finitary methods" to include higher (but still finite) types. This makes it possible to consider definition schemes for functions (like primitive recursion or transfinite recursion) as higher type operators. Moreover, admittance of computable functionals of higher type is a must when we want to capture the computational content of proofs in arithmetic. In the present section we give the definition of a system of finitely typed terms known as Gödel's T based on higher type primitive recursion, and prove some of its basic properties:

- Every function F_α ($\alpha < \varepsilon_0$) of the fast growing hierarchy is definable in T, using iteration operators only.
- T has good closure properties, in the sense that it is closed under many forms of recursive definitions. This applies in particular to $\prec_\alpha$-recursion.
- Every term r in T can be "computed", that is, "converted" into a normal form. In particular, if r is closed and of the type $\mathbf{N}$ of natural numbers, then its normal form is a numeral.

1.2.1. Types. A free algebra is given by its *constructors*, for instance zero and successor for the natural numbers. We want to treat other data types as well, like lists and binary trees. When dealing with inductively defined sets, it will also be useful to explicitly refer to the generation tree. Such trees are quite often infinitely branching, and hence we allow infinitary free algebras.

The freeness of the constructors is expressed by requiring that their ranges are disjoint and that they are injective. To allow for partiality – which is mandatory when we want to deal with computable objects –, we have to embed our algebras into domains. Both requirements together imply that we need "lazy domains".

Our type system is defined by two type forming operations: arrow types $\rho \to \sigma$ and the formation of *inductively generated types* or *base types* $\mu_{\vec{\alpha}}\vec{\kappa}$, where $\vec{\alpha} = (\alpha_j)_{j<N}$ is a list of distinct "type variables", and $\vec{\kappa} = (\kappa_i)_{i<k}$ is a list of "constructor types", whose argument types contain $\alpha_0, \dots, \alpha_{N-1}$ in strictly positive positions only.

For instance, $\mu_\alpha(\alpha, \alpha \to \alpha)$ is the type of natural numbers; here the list $(\alpha, \alpha \to \alpha)$ stands for two generation principles: α for "there is a natural number" (the 0), and $\alpha \to \alpha$ for "for every natural number there is a next one" (its successor).

Definition. Let $\vec{\alpha} = (\alpha_j)_{j<N}$ be a list of distinct type variables. *Types* $\rho, \sigma, \tau, \mu \in \mathrm{Ty}$ and *constructor types* $\kappa \in \mathrm{KT}_{\vec{\alpha}}$ are defined inductively:

$$\frac{\vec{\rho}, \vec{\sigma}_0, \dots, \vec{\sigma}_{n-1} \in \mathrm{Ty}}{\vec{\rho} \to (\vec{\sigma}_\nu \to \alpha_{j_\nu})_{\nu<n} \to \alpha_j \in \mathrm{KT}_{\vec{\alpha}}} \quad (n \geq 0),$$

$$\frac{\kappa_0, \dots, \kappa_{k-1} \in \mathrm{KT}_{\vec{\alpha}}}{(\mu_{\vec{\alpha}}(\kappa_0, \dots, \kappa_{k-1}))_j \in \mathrm{Ty}} \quad (k \geq 1), \qquad \frac{\rho, \sigma \in \mathrm{Ty}}{\rho \to \sigma \in \mathrm{Ty}}.$$

Here $\vec{\rho} \to \sigma$ means $\rho_0 \to \ldots \to \rho_{n-1} \to \sigma$, associated to the right. For a constructor type $\vec{\rho} \to (\vec{\sigma}_\nu \to \alpha_{j_\nu})_{\nu<n} \to \alpha_j$ we call $\vec{\rho}$ the *parameter* argument types and the $\vec{\sigma}_\nu \to \alpha_{j_\nu}$ *recursive* argument types. We require that for every α_j $(j < N)$ there is a *nullary* constructor type κ_{i_j} with value type α_j, each of whose recursive argument types has a value type α_{j_ν} with $j_\nu < j$.

We reserve μ for base types, i.e., types of the form $(\mu_{\vec{\alpha}}(\kappa_0, \dots, \kappa_{k-1}))_j$. The *parameter types* of μ are all parameter argument types of its constructor types $\kappa_0, \dots, \kappa_{k-1}$.

Note. Types $\rho \in \mathrm{Ty}$ are closed in the sense that they do not contain free type parameters α. If one wants to allow say $\vec{\beta}$ as free type parameters, one must add rules $\vec{\beta} \in \mathrm{Ty}$.

Examples.

$$\mathbf{U} \qquad := \mu_\alpha \alpha,$$

$$\begin{aligned}
\mathbf{B} &:= \mu_\alpha(\alpha,\alpha),\\
\mathbf{N} &:= \mu_\alpha(\alpha,\alpha\to\alpha),\\
\mathbf{L}(\rho) &:= \mu_\alpha(\alpha,\rho\to\alpha\to\alpha),\\
\rho\otimes\sigma &:= \mu_\alpha(\rho\to\sigma\to\alpha),\\
\rho+\sigma &:= \mu_\alpha(\rho\to\alpha,\sigma\to\alpha),\\
(\text{tree},\text{tlist}) &:= \mu_{\alpha,\beta}(\mathbf{N}\to\alpha,\beta\to\alpha,\beta,\alpha\to\beta\to\beta),\\
\text{bin} &:= \mu_\alpha(\alpha,\alpha\to\alpha\to\alpha),\\
\mathcal{O} &:= \mu_\alpha(\alpha,\alpha\to\alpha,(\mathbf{N}\to\alpha)\to\alpha),\\
\mathcal{T}_0 &:= \mathbf{N},\\
\mathcal{T}_{n+1} &:= \mu_\alpha(\alpha,(\mathcal{T}_n\to\alpha)\to\alpha).
\end{aligned}$$

Note that there are many equivalent ways to define these types. For instance, we could take $\mathbf{U}+\mathbf{U}$ to be the type of booleans, and $\mathbf{L}(\mathbf{U})$ to be the type of natural numbers.

A type is *finitary* if it is a μ-type with all its parameter types $\vec{\rho}$ finitary, and all its constructor types have recursive argument types of the form α_{j_m} only (so the $\vec{\sigma}_m$ in the general definition are all empty). In the examples above $\mathbf{U}$, $\mathbf{B}$, $\mathbf{N}$, tree, tlist and bin are all finitary, but $\mathcal{O}$ and $\mathcal{T}_{n+1}$ are not. $\mathbf{L}(\rho)$, $\rho\otimes\sigma$ and $\rho+\sigma$ are finitary provided their parameter types are. An argument position in a type is called *finitary* if it is occupied by a finitary type.

In what follows we often restrict to finitary μ-types. Usually it even suffices to consider types built from the base types $\mathbf{N}$ and $\mathbf{B}$ by the formation of arrows $\rho\to\sigma$ and products $\rho\wedge\sigma$. For such types we define the *level* of a type by

$$\begin{aligned}
\mathrm{lev}(\mathbf{N}) &:= 0, \qquad & \mathrm{lev}(\rho\to\sigma) &:= \max(\mathrm{lev}(\rho)+1,\mathrm{lev}(\sigma)),\\
\mathrm{lev}(\mathbf{B}) &:= 0, \qquad & \mathrm{lev}(\rho\wedge\sigma) &:= \max(\mathrm{lev}(\rho),\mathrm{lev}(\sigma)).
\end{aligned}$$

1.2.2. The power of finite types. To demonstrate the usefulness and strength provided by finite types we show that all F_α ($\alpha<\varepsilon_0$) can be defined explicitely from higher type iteration operators alone. We define finite type extensions of the functions F_α, such that for $\alpha=0$ we obtain iteration operators. Define the *pure types* ρ_n, by $\rho_0:=\mathbf{N}$ and $\rho_{n+1}:=\rho_n\to\rho_n$. Let x_n be a variable of pure type ρ_n.

$$F_\alpha x_n\dots x_0 := \begin{cases} x_0+1 & \text{if } \alpha=0 \text{ and } n=0,\\ x_n^{x_0}x_{n-1}\dots x_0 & \text{if } \alpha=0 \text{ and } n>0,\\ F^{x_0}_{\alpha-1}x_n\dots x_0 & \text{if } \mathrm{Succ}(\alpha),\\ F_{\alpha(x_0)}x_n\dots x_0 & \text{if } \mathrm{Lim}(\alpha).\end{cases}$$

LEMMA. *$F_\alpha F_\beta = F_{\beta+\omega^\alpha}$, where it is assumed that α and β have Cantor Normal Forms which can simply be concatenated to form the normal form of $\alpha + \beta$.*

PROOF. By induction on α. If $\alpha = 0$,

$$F_0 F_\beta x_{n-1} \dots x_0 = F_\beta^{x_0} x_{n-1} \dots x_0 = F_{\beta+1} x_{n-1} \dots x_0.$$

If $\mathrm{Succ}(\alpha)$,

$$\begin{aligned} F_\alpha F_\beta x_{n-1} \dots x_0 &= F_{\alpha-1}^{x_0} F_\beta x_{n-1} \dots x_0 \\ &= F_{\beta+\omega^{\alpha-1}\cdot x_0} x_{n-1} \dots x_0 \quad \text{by IH} \\ &= F_{(\beta+\omega^\alpha)(x_0)} x_{n-1} \dots x_0 \\ &= F_{\beta+\omega^\alpha} x_{n-1} \dots x_0. \end{aligned}$$

If $\mathrm{Lim}(\alpha)$,

$$\begin{aligned} F_\alpha F_\beta x_{n-1} \dots x_0 &= F_{\alpha(x_0)}^{x_0} F_\beta x_{n-1} \dots x_0 \\ &= F_{\beta+\omega^{\alpha(x_0)}\cdot x_0} x_{n-1} \dots x_0 \quad \text{by IH} \\ &= F_{(\beta+\omega^\alpha)(x_0)} x_{n-1} \dots x_0 \\ &= F_{\beta+\omega^\alpha} x_{n-1} \dots x_0. \end{aligned}$$

This completes the proof. □

1.2.3. Recursion operators. The inductive structure of the types $\vec{\mu} = \mu_{\vec{\alpha}}\vec{\kappa}$ corresponds to two sorts of constants: with the *constructors* $\mathrm{C}_i^{\vec{\mu}} \colon \kappa_i(\vec{\mu})$ we can build elements of a type μ_j, and with the (structural) *recursion operators* $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ we can construct mappings recursion on the structure of $\vec{\mu}$.

In order to define the type of the recursion operators w.r.t. $\vec{\mu} = \mu_{\vec{\alpha}}\vec{\kappa}$ and result types $\vec{\tau}$, we first define for

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_\nu \to \alpha_{j_\nu})_{\nu<n} \to \alpha_j \in \mathrm{KT}_{\vec{\alpha}}$$

the *step type*

$$\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \to (\vec{\sigma}_\nu \to \mu_{j_\nu})_{\nu<n} \to (\vec{\sigma}_\nu \to \tau_{j_\nu})_{\nu<n} \to \tau_j.$$

Here $\vec{\rho}, (\vec{\sigma}_\nu \to \mu_{j_\nu})_{\nu<n}$ correspond to the *components* of the object of type μ_j under consideration, and $(\vec{\sigma}_\nu \to \tau_{j_\nu})_{\nu<N}$ to the previously defined values. The recursion operator $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ has type

$$\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}} \colon \mu_j \to \delta_0^{\vec{\mu},\vec{\tau}} \to \dots \to \delta_{k-1}^{\vec{\mu},\vec{\tau}} \to \tau_j$$

(recall that k is the total number of constructors for all types μ_j, $j < N$).

We will often write $\mathcal{R}_j^{\vec{\mu},\vec{\tau}}$ for $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$, and omit the upper indices $\vec{\mu},\vec{\tau}$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e., of type $\mu_\alpha\kappa$, for $\mathcal{R}_\mu^{\mu,\tau}$ we write $\mathcal{R}_\mu^\tau$.

For some common base types the constructors have standard names, as follows. We also spell out the type of the recursion operators:

$$\begin{aligned}
&\mathtt{tt}^{\mathbf{B}} := \mathrm{C}_1^{\mathbf{B}}, \quad \mathtt{ff}^{\mathbf{B}} := \mathrm{C}_2^{\mathbf{B}},\\
&\mathcal{R}_{\mathbf{B}}^\tau\colon \mathbf{B} \to \tau \to \tau \to \tau,\\
&0^{\mathbf{N}} := \mathrm{C}_1^{\mathbf{N}}, \quad \mathrm{S}^{\mathbf{N}\to\mathbf{N}} := \mathrm{C}_2^{\mathbf{N}},\\
&\mathcal{R}_{\mathbf{N}}^\tau\colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau,\\
&\mathrm{nil}^{\mathbf{L}(\rho)} := \mathrm{C}_1^{\mathbf{L}(\rho)}, \quad \mathrm{cons}^{\rho\to\mathbf{L}(\rho)\to\mathbf{L}(\rho)} := \mathrm{C}_2^{\mathbf{L}(\rho)},\\
&\mathcal{R}_{\mathbf{L}(\rho)}^\tau\colon \mathbf{L}(\rho) \to \tau \to (\rho \to \mathbf{L}(\rho) \to \tau \to \tau) \to \tau,\\
&\big(\mathrm{inl}_{\rho\sigma}\big)^{\rho\to\rho+\sigma} := \mathrm{C}_1^{\rho+\sigma}, \quad \big(\mathrm{inr}_{\rho\sigma}\big)^{\sigma\to\rho+\sigma} := \mathrm{C}_2^{\rho+\sigma},\\
&\mathcal{R}_{\rho+\sigma}^\tau\colon \rho+\sigma \to (\rho \to \tau) \to (\sigma \to \tau) \to \tau,\\
&\big(\otimes_{\rho\sigma}^+\big)^{\rho\to\sigma\to\rho\otimes\sigma} := \mathrm{C}_1^{\rho\otimes\sigma},\\
&\mathcal{R}_{\rho\otimes\sigma}^\tau\colon \rho\otimes\sigma \to (\rho \to \sigma \to \tau) \to \tau.
\end{aligned}$$

One often writes $x :: l$ as shorthand for $\mathrm{cons}\, x\, l$, and $\langle y, z\rangle$ for $\otimes^+ yz$.

Terms are inductively defined from typed variables x^ρ and the constants, that is, constructors $\mathrm{C}_i^{\vec{\mu}}$ and recursion operators $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$, by abstraction $\lambda_{x^\rho} M^\sigma$ and application $M^{\rho\to\sigma} N^\rho$.

Examples. We define the *canonical inhabitant* ε^ρ of a type $\rho \in \mathrm{Ty}$:

$$\varepsilon^{\mu_j} := \mathrm{C}_j^{\vec{\mu}} \varepsilon^{\vec{\rho}} (\lambda_{\vec{x}_1} \varepsilon^{\mu_{j_1}}) \dots (\lambda_{\vec{x}_n} \varepsilon^{\mu_{j_n}}), \quad \varepsilon^{\rho\to\sigma} := \lambda_x \varepsilon^\sigma.$$

The *projections* of a pair to its components can be defined easily:

$$M0 := \mathcal{R}_{\rho\otimes\sigma}^\rho M^{\rho\otimes\sigma}(\lambda_{x^\rho, y^\sigma} x^\rho), \quad M1 := \mathcal{R}_{\rho\otimes\sigma}^\rho M^{\rho\otimes\sigma}(\lambda_{x^\rho, y^\sigma} y^\sigma).$$

The *append*-function $:+:$ for lists is defined recursively by

$$\begin{aligned}
\mathrm{nil} :+: l_2 &:= l_2,\\
(x :: l_1) :+: l_2 &:= x :: (l_1 :+: l_2).
\end{aligned}$$

It can be defined as the term

$$l_1 :+: l_2 := \mathcal{R}_{\mathbf{L}(\alpha)}^{\mathbf{L}(\alpha)\to\mathbf{L}(\alpha)} l_1 (\lambda_{l_2} l_2)(\lambda_{x,l_1,p,l_2}(x :: (p l_2))) l_2.$$

Using the append function $:+:$ we can define *list reversal* Rev by

$$\mathrm{Rev}\ \mathrm{nil} := \mathrm{nil},$$

$$\mathrm{Rev}(x :: l) := (\mathrm{Rev}\ l) :+: (x :: \mathrm{nil});$$

the corresponding term is

$$\mathrm{Rev}\ l := \mathcal{R}^{\mathbf{L}(\alpha)}_{\mathbf{L}(\alpha)} l\ \mathrm{nil}(\lambda_{x,l,p}(p :+: (x :: \mathrm{nil}))).$$

Assume we want to define by simultaneous recursion two functions on $\mathbf{N}$, say $\mathrm{even}, \mathrm{odd}\colon \mathbf{N} \to \mathbf{B}$. We want

$$\begin{aligned} \mathrm{even}(0) &:= \mathtt{tt}, & \mathrm{odd}(0) &:= \mathtt{ff}, \\ \mathrm{even}(\mathrm{S}\, n) &:= \mathrm{odd}(n), & \mathrm{odd}(\mathrm{S}\, n) &:= \mathrm{even}(n). \end{aligned}$$

This can be achieved by using pair types: we recursively define the single function $\mathrm{evenodd}\colon \mathbf{N} \to \mathbf{B} \wedge \mathbf{B}$. The step types are

$$\delta_0 = \mathbf{B} \wedge \mathbf{B}, \quad \delta_1 = \mathbf{N} \to \mathbf{B} \wedge \mathbf{B} \to \mathbf{B} \wedge \mathbf{B},$$

and we can define $\mathrm{evenodd}\, m := \mathcal{R}^{\mathbf{B}\wedge\mathbf{B}}_{\mathbf{N}} m \langle \mathtt{tt}, \mathtt{ff} \rangle (\lambda_{n,p} \langle p1, p0 \rangle)$.

Another exmple concerns the simultaneously defined free algebras tree and tlist, whose constructors $\mathrm{C}^{(\mathrm{tree,tlist})}_i$ for $i \in \{0, \ldots, 3\}$ are

$$\mathrm{Leaf}^{\mathbf{N}\to\mathrm{tree}},\ \mathrm{Branch}^{\mathrm{tlist}\to\mathrm{tree}},\ \mathrm{Empty}^{\mathrm{tlist}},\ \mathrm{Tcons}^{\mathrm{tree}\to\mathrm{tlist}\to\mathrm{tlist}}.$$

Observe that the elements of the algebra tree are just the finitely branching trees, which carry natural numbers on their leaves.

Let us compute the types of the recursion operators w.r.t. the result types τ_0, τ_1, i.e., of $\mathcal{R}^{(\mathrm{tree,tlist}),(\tau_0,\tau_1)}_{\mathrm{tree}}$ and $\mathcal{R}^{(\mathrm{tree,tlist}),(\tau_0,\tau_1)}_{\mathrm{tlist}}$, or shortly $\mathcal{R}_{\mathrm{tree}}$ and $\mathcal{R}_{\mathrm{tlist}}$. The step types are

$$\begin{aligned} \delta_0 &:= \mathbf{N} \to \tau_0, & \delta_2 &:= \tau_1, \\ \delta_1 &:= \mathrm{tlist} \to \tau_1 \to \tau_0, & \delta_3 &:= \mathrm{tree} \to \mathrm{tlist} \to \tau_0 \to \tau_1 \to \tau_1. \end{aligned}$$

Hence the types of the recursion operators are

$$\begin{aligned} \mathcal{R}_{\mathrm{tree}}&\colon \mathrm{tree} \to \delta_0 \to \delta_1 \to \delta_2 \to \delta_3 \to \tau_0, \\ \mathcal{R}_{\mathrm{tlist}}&\colon \mathrm{tlist} \to \delta_0 \to \delta_1 \to \delta_2 \to \delta_3 \to \tau_1. \end{aligned}$$

To see a concrete example, let us recursively define addition $+\colon \mathrm{tree} \to \mathrm{tree} \to \mathrm{tree}$ and $\oplus\colon \mathrm{tlist} \to \mathrm{tree} \to \mathrm{tlist}$. The recursion equations to be satisfied are

$$\begin{aligned} +(\mathrm{Leaf}\, n) &= \lambda_a a, & \oplus \mathrm{Empty} &= \lambda_a \mathrm{Empty}, \\ +(\mathrm{Branch}\, bs) &= \lambda_a.\mathrm{Branch}(\oplus\, bs\, a), & \oplus(\mathrm{Tcons}\, b\, bs) &= \lambda_a.\mathrm{Tcons}(+\, b\, a)(\oplus\, bs\, a). \end{aligned}$$

We define $+$ and $\oplus$ by means of the recursion operators $\mathcal{R}_{\mathrm{tree}}$ and $\mathcal{R}_{\mathrm{tlist}}$ with result types

$$\tau_0 := \mathrm{tree} \to \mathrm{tree}, \quad \tau_1 := \mathrm{tree} \to \mathrm{tlist}.$$

The step terms are

$$M_0 := \lambda_{n,a} a, \qquad M_3 := \lambda_a \,\mathrm{Empty},$$
$$M_1 := \lambda_{bs,g^{\tau_1},a}.\mathrm{Branch}(g\,a), \quad M_4 := \lambda_{b,bs,f^{\tau_0},g^{\tau_1},a}.\mathrm{Tcons}(f\,a)(g\,a).$$

Then

$$a + b := \mathcal{R}_{\mathrm{tree}} a \vec{M} b, \quad bs \oplus a := \mathcal{R}_{\mathrm{tlist}}\, bs \vec{M} a.$$

Furthermore, for every finitary base type μ one can define a boolean-valued function for decidable *equality* $=_\mu \colon \mu \to \mu \to \mathbf{B}$, using recursion operators. The recursion equation for $\mathbf{N}$ are

$$(0 = 0) := \mathtt{tt}, \qquad (\mathrm{S}(m) = 0) := \mathtt{ff},$$
$$(0 = \mathrm{S}(n)) := \mathtt{ff}, \quad (\mathrm{S}(m) = \mathrm{S}(n)) := \ (m = n).$$

1.2.4. Special cases of structural recursion; general recursion.

Simplified simultaneous recursion. In a recursion on simultaneously defined algebras one may need to recur on some of those algebras only. Then we can simplify the type of the recursion operator accordingly, by

- omitting all step types $\delta_i^{\vec{\mu},\vec{\tau}}$ with irrelevant value type τ_j, and
- simplifying the remaining step types by omitting from the recursive argument types $(\vec{\sigma}_\nu \to \tau_{j_\nu})_{\nu<n}$ all those with irrelevant τ_{j_ν}.

In the tree, tlist-example, if we only want to recur on tlist, then the step types are

$$\delta_2 := \tau_1, \quad \delta_3 := \mathrm{tree} \to \mathrm{tlist} \to \tau_1 \to \tau_1.$$

Hence the type of the simplified recursion operator is

$$\mathcal{R}_{\mathrm{tlist}} \colon \mathrm{tlist} \to \delta_2 \to \delta_3 \to \tau_1.$$

An example is the recursive definition of the length of a tlist. The recursion equations are

$$\mathrm{lh}(\mathrm{Empty}) = 0, \quad \mathrm{lh}(\mathrm{Tcons}\, b\, bs) = \mathrm{lh}(bs) + 1.$$

The step terms are

$$M_2 := 0, \quad M_3 := \lambda_{b,bs,p}(p + 1).$$

Cases. There is an important variant of recursion, where no recursive calls occur. This variant is called the *cases operator*; it distinguishes cases according to the outer constructor form. Here all step types have the form

$$\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \to (\vec{\sigma}_\nu \to \mu_{j_\nu})_{\nu<n} \to \tau_j.$$

The intended meaning of the cases operator is given by the conversion rule

$$\mathcal{C}_j(\mathrm{C}_i^{\vec{\mu}} \vec{N})\vec{M} \mapsto M_i \vec{N}.$$

Notice that only those step terms are used whose value type is the present τ_j; this is due to the fact that there are no recursive calls. Therefore the type of the cases operator is

$$\mathcal{C}^{\vec{\mu}}_{\mu_j\to\tau_j}\colon \mu_j \to \delta_{i_0} \to \ldots \to \delta_{i_{q-1}} \to \tau_j,$$

where $\delta_{i_0},\ldots,\delta_{i_{q-1}}$ consists of all δ_i with value type τ_j. We write $\mathcal{C}^{\tau_j}_{\mu_j}$ or even $\mathcal{C}_j$ for $\mathcal{C}^{\vec{\mu}}_{\mu_j\to\tau_j}$.

The simplest example (for the type $\mathbf{B}$) is *if-then-else*. Another example is the predecessor function on $\mathbf{N}$, i.e., $\mathrm{P}(0) := 0$, $\mathrm{P}(\mathrm{S}(n)) := n$. It can formally be defined by the term

$$\mathrm{P}(m) := \mathcal{C}^{\mathbf{N}}_{\mathbf{N}} m 0(\lambda_n n).$$

In the tree, tlist-example we have

$$\mathcal{C}^{\tau_1}_{\mathrm{tlist}}\colon \mathrm{tlist} \to \tau_1 \to (\mathrm{tree} \to \mathrm{tlist} \to \tau_1) \to \tau_1.$$

When computing the value of a cases term, we do not want to (eagerly) evaluate all arguments, but rather compute the test argument first and depending on the result (lazily) evaluate at most one of the other arguments. This phenomenon is well known in functional languages; for instance, in SCHEME the `if`-construct is called a *special form* (as opposed to an operator). Therefore instead of taking the cases operator applied to a full list of arguments, one rather uses a `case`-construct to build this term; it differs from the former only in that it employs lazy evaluation. Hence the predecessor function is written in the form $[\mathbf{case}\ m\ \mathbf{of}\ 0,\ \lambda_n n]$. If there are exactly two cases, we also write $\lambda_m[\mathbf{if}\ m\ \mathbf{then}\ 0\ \mathbf{else}\ \lambda_n n]$ instead.

General recursion. In practice it often happens that one needs to recur to an argument which is not an immediate component of the present constructor object; this is not allowed in structural recursion. Of course, in order to ensure that the recursion terminates we have to assume that the recurrence is w.r.t. a given well-founded set; for simplicity we restrict ourselves to the base type $\mathbf{N}$. However, we do allow that the recurrence is with respect to a measure function μ, with values in $\mathbf{N}$. The operator $\mathcal{F}$ of *general recursion* then is defined by

$$\mathcal{F}\mu x G = Gx\big(\lambda_y[\mathbf{if}\ \mu y < \mu x\ \mathbf{then}\ \mathcal{F}\mu y G\ \mathbf{else}\ \varepsilon]\big), \tag{1.12}$$

where ε denotes a canonical inhabitant of the range. We leave it as an exercise to prove that $\mathcal{F}$ is definable from an appropriate structural recursion operator.

1.2.5. Conversion. To define the conversion relation, it will be helpful to use the following notation. Let $\vec{\mu} = \mu_{\vec{\alpha}}\vec{\kappa}$ and $\kappa_i =$

$$\rho_0 \to \ldots \to \rho_{m-1} \to (\vec{\sigma}_0 \to \alpha_{j_0}) \to \ldots \to (\vec{\sigma}_{n-1} \to \alpha_{j_{n-1}}) \to \alpha_j \in \mathrm{KT}_{\vec{\alpha}},$$

and consider $\mathrm{C}_i^{\vec{\mu}}\vec{N}$. We write $\vec{N}^P = N_0^P, \dots, N_{m-1}^P$ for the *parameter arguments* $N_0^{\rho_0}, \dots, N_{m-1}^{\rho_{m-1}}$ and $\vec{N}^R = N_0^R, \dots, N_{n-1}^R$ for the *recursive arguments* $N_m^{\vec{\sigma}_0 \to \mu_{j_0}}, \dots, N_{m+n-1}^{\vec{\sigma}_{n-1} \to \mu_{j_{n-1}}}$, and n^R for the number n of recursive arguments.

We define a *conversion relation* $\mapsto_\rho$ between terms of type ρ by

$$\begin{aligned}(\lambda_x M)N &\mapsto M[x := N],\\ \lambda_x(Mx) &\mapsto M \quad \text{if } x \notin \mathrm{FV}(M) \text{ } (M \text{ not an abstraction}),\\ \mathcal{R}_j(\mathrm{C}_i^{\vec{\mu}}\vec{N})\vec{M} &\mapsto M_i\vec{N}\big((\mathcal{R}_{j_0} \cdot \vec{M}) \circ N_0^R\big) \dots \big((\mathcal{R}_{j_{n-1}} \cdot \vec{M}) \circ N_{n-1}^R\big).\end{aligned}$$

Here we have written $\mathcal{R}_j \cdot \vec{M}$ for $\lambda_{x^{\mu_j}}(\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}} x^{\mu_j} \vec{M})$.

The *one step reduction relation* $\to$ can now be defined as follows. $M \to N$ if N is obtained from M by replacing a subterm M' in M by N', where $M' \mapsto N'$. The reduction relations $\to^+$ and $\to^*$ are the transitive and the reflexive transitive closure of $\to$, respectively. For $\vec{M} = M_1, \dots, M_n$ we write $\vec{M} \to \vec{M}'$ if $M_i \to M_i'$ for some $i \in \{1, \dots, n\}$ and $M_j = M_j'$ for all $i \neq j \in \{1, \dots, n\}$. A term M is *normal* (or in *normal form*) if there is no term N such that $M \to N$.

Clearly normal closed terms are of the form $\mathrm{C}_i^{\vec{\mu}}\vec{N}$.

THEOREM. *Every term can be reduced to a normal form.*

1.3. HA^ω

We define Heyting Arithmetic HA and its extension HA^ω to a finitely typed language.

1.3.1. Derivations as terms. Recall that we have a decidable equality $=_\mu \colon \mu \to \mu \to \mathbf{B}$, for finitary base types μ. Every *atomic formula* has the form $\mathrm{atom}(r^{\mathbf{B}})$, i.e., is built from a boolean term $r^{\mathbf{B}}$. In particular, there is no need for (logical) falsity $\bot$, since we can take the atomic formula $F := \mathrm{atom}(\mathsf{ff})$ – called *arithmetical falsity* – built from the boolean constant ff instead.

The *formulas* of HA^ω are built from atomic ones by the connectives $\to$ and $\forall$. We define *negation* $\neg A$ by $A \to F$. We use natural deduction rules: $\to^+$, $\to^-$, $\forall^+$ and $\forall^-$.

It will be convenient to write derivations as terms, where the derived formula is viewed as the type of the term. This representation is known under the name *Curry-Howard correspondence*. From now on we use M, N etc. to range over derivation terms, and r, s etc. for object terms.

We give an inductive definition of derivation terms in Table 1, where for clarity we have written the corresponding derivations to the left. For the universal quantifier $\forall$ there is an introduction rule $\forall^+ x$ and an elimination rule $\forall^-$, whose right premise is the term r to be substituted. The rule $\forall^+ x$

derivation	term
$u\colon A$	u^A
$\dfrac{\begin{array}{c}[u\colon A]\\ \mid M\\ B\end{array}}{A \to B} \to^+ u$	$(\lambda_{u^A} M^B)^{A\to B}$
$\dfrac{\begin{array}{cc}\mid M & \mid N\\ A \to B & A\end{array}}{B} \to^-$	$(M^{A\to B} N^A)^B$
$\dfrac{\begin{array}{c}\mid M\\ A\end{array}}{\forall_x A} \forall^+ x$ (with var.cond.)	$(\lambda_x M^A)^{\forall_x A}$ (with var.cond.)
$\dfrac{\begin{array}{cc}\mid M & \\ \forall_x A(x) & r\end{array}}{A(r)} \forall^-$	$(M^{\forall_x A(x)} r)^{A(r)}$

TABLE 1. Derivation terms for $\to$ and $\forall$

is subject to the standard *(Eigen-) variable condition*: The derivation term M of the premise A should not contain any open assumption with x as a free variable.

We require the *truth axiom* $\mathrm{Ax}_{\mathtt{tt}}\colon \mathrm{atom}(\mathtt{tt})$.

1.3.2. Structural induction. The general form of (structural) *induction* over simultaneous free algebras $\vec{\mu} = \mu_{\vec{\alpha}}\,\vec{\kappa}$, with goal formulas $A_j(x_j^{\mu_j})$ is as follows (cf. 1.2.3). For the constructor type

$$\kappa_i = \vec{\rho} \to (\vec{\sigma}_\nu \to \alpha_{j_\nu})_{\nu<n} \to \alpha_j \in \mathrm{KT}_{\vec{\alpha}}$$

we have the *step formula*

(1.13)

$$D_i := \forall_{y_0^{\rho_0},\dots,y_{m-1}^{\rho_{m-1}},y_m^{\vec\sigma_0\to\mu_{j_0}},\dots,y_{m+n-1}^{\vec\sigma_{n-1}\to\mu_{j_{n-1}}}} \big(\forall_{\vec x^{\vec\sigma_0}} A_{j_0}(y_{m+1}\vec x) \to \cdots \to \forall_{\vec x^{\vec\sigma_{n-1}}} A_{j_{n-1}}(y_{m+n-1}\vec x) \to A_j(\mathrm{C}_i^{\vec\mu}\vec y)\big).$$

Here $\vec y = y_0^{\rho_0},\dots,y_{m-1}^{\rho_{m-1}},y_m^{\vec\sigma_0\to\mu_{j_0}},\dots,y_{m+n-1}^{\vec\sigma_{n-1}\to\mu_{j_{n-1}}}$ are the *components* of the object $\mathrm{C}_i^{\vec\mu}\vec y$ of type μ_j under consideration, and

$$\forall_{\vec x^{\vec\sigma_0}} A_{j_0}(y_m\vec x),\dots,\forall_{\vec x^{\vec\sigma_{n-1}}} A_{j_{n-1}}(y_{m+n-1}\vec x)$$

are the hypotheses available when proving the induction step. The induction axiom $\mathrm{Ind}_{\mu_j}^{\vec x,\vec A}$ then proves the universal closure of the formula

$$\forall_{x_j}\big(D_0 \to \cdots \to D_{k-1} \to A_j(x_j^{\mu_j})\big).$$

We will often write $\mathrm{Ind}_j^{\vec x,\vec A}$ for $\mathrm{Ind}_{\mu_j}^{\vec x,\vec A}$, and omit the upper indices $\vec x,\vec A$ when they are clear from the context. In case of a non-simultaneous free algebra, i.e., of type $\mu_\alpha\vec\kappa$, for $\mathrm{Ind}_\mu^{x,A}$ we normally write $\mathrm{Ind}_{x,A}$.

Examples.

$$\begin{aligned}
&\mathrm{Ind}_{p,A}\colon \forall_p\big(A(\mathtt{tt}) \to A(\mathtt{ff}) \to A(p^{\mathbf{B}})\big),\\
&\mathrm{Ind}_{n,A}\colon \forall_m\big(A(0) \to \forall_n(A(n) \to A(\mathrm{S}n)) \to A(m^{\mathbf{N}})\big),\\
&\mathrm{Ind}_{l,A}\colon \forall_l\big(A(\mathrm{nil}) \to \forall_{x,l'}(A(l') \to A(x :: l')) \to A(l^{\mathbf{L}(\rho)})\big),\\
&\mathrm{Ind}_{x,A}\colon \forall_x\big(\forall_{y_1} A(\mathrm{inl}\, y_1) \to \forall_{y_2} A(\mathrm{inr}\, y_2) \to A(x^{\rho_1+\rho_2})\big),\\
&\mathrm{Ind}_{x,A}\colon \forall_x\big(\forall_{y^\rho,z^\sigma} A(\langle y,z\rangle) \to A(x^{\rho\wedge\sigma})\big),
\end{aligned}$$

where $x :: l$ is shorthand for $\mathrm{cons}\, x\, l$ and $\langle y,z\rangle$ for $\otimes^+ yz$.

Let HA^ω be the theory based on the axioms above including the induction axioms, and ML^ω be the (many-sorted) minimal logic, where the induction axioms are left out.

Lemma (Ex falso quodlibet). $\mathrm{HA}^\omega \vdash F \to A$.

Proof. Induction on A, using $\mathrm{Ind}_{p,\mathrm{atom}(p)}$ in the prime formula case. The details are left as an exercise. □

Lemma (Stability). $\mathrm{HA}^\omega \vdash \neg\neg A \to A$.

Proof. Induction on A (exercise). □

Lemma (Compatibility). *For finitary μ,*

$$\mathrm{HA}^\omega \vdash x_1 =_\mu x_2 \to A(x_1) \to A(x_2).$$

PROOF. Induction on x_1 with a side induction on x_2, using ex falso quodlibet. The details are left as an exercise. □

1.3.3. Special cases of structural induction; general induction. As for structural recursion in 1.2.4, we can single out some special cases of (structural) induction: simplified simultaneous induction, and a "cases" variant of induction, where no inductive calls occur.

Parallel to general recursion, one can also consider a more general form of induction, called "general induction", which allows recurrence to *all* points "strictly below" the present one. For applications it is best to make the necessary comparisons w.r.t. a "measure function" μ. Then it suffices to use an initial segment of the ordinals instead of a well-founded set. For simplicity we here restrict ourselves to the segment given by ω, so the ordering we refer to is just the standard $<$-relation on the natural numbers. The principle of general induction then is

$$\forall_{\mu,x}\big(\mathrm{Prog}^{\mu}_{x}A(x) \to A(x)\big) \tag{1.14}$$

where $\mathrm{Prog}^{\mu}_{x}A(x)$ expresses "progressiveness" w.r.t. the measure function μ and the ordering $<$:

$$\mathrm{Prog}^{\mu}_{x}A(x) := \forall_x\big(\forall_{y;\mu y<\mu x}A(y) \to A(x)\big).$$

It is easy to see that in our special case of the $<$-relation we can *prove* (1.14) from structural induction. However, using the general induction as a primitive axiom has an advantage when we consider its computational content, which is general recursion.

CHAPTER 2

Realizability Interpretation

We now study the concept of "computational content" of a proof. This only makes sense after we have introduced inductively defined predicates to our "negative" language of HA^ω involving $\forall$ and $\to$ only. The resulting system will be called *arithmetic with inductively defined predicates* ID^ω.

The intended meaning of an inductively defined predicate I is quite clear: the clauses correspond to constructors of an appropriate algebra μ (or better μ_I). We associate to I a new predicate $I^{\mathbf{r}}$, of arity $(\mu, \vec{\rho})$, where the first argument r of type μ represents a "generation tree", witnessing how the other arguments $\vec{r}$ were put into I. This object r of type μ is called a "realizer" of the prime formula $I(\vec{r})$.

Moreover, we want to be able to select relevant parts of the complete computational content of a proof. This will be possible if some "uniformities" hold; we express this fact by using a uniform variant $\forall^{\mathsf{U}}$ of the universal quantifier $\forall$ (as done by Berger (2005)) and in addition a uniform variant $\to^{\mathsf{U}}$ of implication $\to$. Both are governed by the same rules as the non-uniform ones. However, we will have to put some uniformity conditions on a proof to ensure that the extracted computational content will be correct.

2.1. Inductively Defined Predicates and Uniformity

As we have seen, type variables allow for a general treatment of inductively generated types $\mu_{\vec{\alpha}}\vec{\kappa}$. Similarly, we can use predicate variables to inductively generate predicates $\mu_{\vec{X}}\vec{K}$.

More precisely, we allow the formation of inductively generated predicates $\mu_{\vec{X}}\vec{K}$, where $\vec{X} = (X_j)_{j<N}$ is a list of distinct predicate variables, and $\vec{K} = (K_i)_{i<k}$ is a list of constructor formulas (or "clauses") whoses premises contain $X_0, \dots, X_{N-1}$ in strictly positive positions only.

2.1.1. Introduction and elimination axioms.

Definition (Inductively defined predicates). Let $\vec{X} = (X_j)_{j<N}$ be a list of distinct predicate variables. *Formulas* $A, B, C, D \in \mathrm{F}$, *predicates* $P, Q, I \in \mathrm{Preds}$ and *constructor formulas* (or *clauses*) $K \in \mathrm{KF}_{\vec{X}}$ are defined inductively as follows. Let $\breve{\forall}$ denote either $\forall$ or $\forall^{\mathsf{U}}$, and $\breve{\to}$ denote either $\to$

or $\to^{\mathsf{U}}$.

$$\frac{\vec{A}, \vec{B}_0, \dots, \vec{B}_{n-1} \in \mathrm{F}}{\breve{\forall}_{\vec{x}}\big(\vec{A} \mathrel{\breve{\to}} \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} X_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to X_j(\vec{t}\,)\big) \in \mathrm{KF}_{\vec{X}}} \quad (n \geq 0)$$

$$\frac{K_0, \dots, K_{k-1} \in \mathrm{KF}_{\vec{X}} \quad (k \geq 1)}{(\mu_{\vec{X}}(K_0, \dots, K_{k-1}))_j \in \mathrm{Preds}} \qquad \frac{P \in \mathrm{Preds}}{P(\vec{r}\,) \in \mathrm{F}} \qquad \frac{C \in \mathrm{F}}{\{\, \vec{x} \mid C \,\} \in \mathrm{Preds}}$$

$$\frac{A, B \in \mathrm{F}}{A \to B \in \mathrm{F}} \quad \frac{A \in \mathrm{F}}{\forall_{x^\rho} A \in \mathrm{F}} \quad \frac{A, B \in \mathrm{F}}{A \to^{\mathsf{U}} B \in \mathrm{F}} \quad \frac{A \in \mathrm{F}}{\forall^{\mathsf{U}}_{x^\rho} A \in \mathrm{F}} \quad \mathrm{atom}(r) \in \mathrm{F}.$$

Here $\vec{A} \mathrel{\breve{\to}} B$ means $A_0 \mathrel{\breve{\to}} \dots \mathrel{\breve{\to}} A_{n-1} \mathrel{\breve{\to}} B$, associated to the right. For a constructor formula $\breve{\forall}_{\vec{x}}(\vec{A} \mathrel{\breve{\to}} (\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} X_{j_\nu}(\vec{s}_\nu)))_{\nu<n} \mathrel{\breve{\to}} X_j(\vec{t}\,))$ we call $\vec{A}$ the *parameter* premises and the $\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} X_{j_\nu}(\vec{s}_\nu))$ *recursive* premises. We require that for every X_j $(j < N)$ there is a clause K_{i_j} with final conclusion $X_j(\vec{t}\,)$, amongst whose premises there is either a parameter premise or else a recursive premise with final conclusion $X_{j_\nu}(\vec{s}_\nu)$ with $j_\nu < j$. (The presence of such clauses guarantees that we can derive ex-falso-quodlibet for every inductively defined predicate I). A clause of the form $\forall^{\mathsf{U}}_{\vec{x}}(F \to X_j(\vec{x}\,))$ is called an *efq-clause*.

A predicate of the form $\{\, \vec{x} \mid C \,\}$ is called a *comprehension term*. We identify $\{\, \vec{x} \mid C(\vec{x}\,) \,\}(\vec{r}\,)$ with $C(\vec{r}\,)$. I will be used for predicates of the form $(\mu_{\vec{X}}\,(K_0, \dots, K_{k-1}))_j$.

Consider inductively defined predicates $\vec{I} := \mu_{\vec{X}}(K_0, \dots K_{k-1})$. For each of the k clauses we have an introduction axiom, as follows. Let the i-th clause for I_j be

$$K(\vec{x}\,) := \breve{\forall}_{\vec{x}}\big(\vec{A} \mathrel{\breve{\to}} \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} X_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to X_j(\vec{t}\,)\big).$$

The corresponding *introduction axiom* then is $K(\vec{I}\,)$, that is

$$(I_j)^+_i : \breve{\forall}_{\vec{x}}\big(\vec{A} \mathrel{\breve{\to}} \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} I_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to I_j(\vec{t}\,)\big). \tag{2.1}$$

Also for every I_j we have the *elimination axiom*

$$\forall^{\mathsf{U}}_{\vec{x}}\big(I_j(\vec{x}\,) \to K_0(\vec{P}) \to \dots \to K_{k-1}(\vec{P}) \to P_j(\vec{x}\,)\big).$$

However, in applications one often wants to use a strengthened form of the elimination axioms. For their formulation it is useful to introduce the notation

$$K(\vec{Q}, \vec{P}) := \breve{\forall}_{\vec{x}}\big(\vec{A} \mathrel{\breve{\to}} \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} Q_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} P_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to P_j(\vec{t}\,)\big).$$

Then the *strengthened elimination axioms* are

$$I^-_j : \forall^{\mathsf{U}}_{\vec{x}}\big(I_j(\vec{x}\,) \to K_0(\vec{I}, \vec{P}) \to \dots \to K_{k-1}(\vec{I}, \vec{P}) \to P_j(\vec{x}\,)\big). \tag{2.2}$$

They are indeed stronger (and hence easier to use), since each premise $K_i(\vec{I},\vec{P})$ is weaker than $K_i(\vec{P})$ (because $K_i(\vec{I},\vec{P})$ has more premises than $K_i(\vec{P})$). However, there is no essential difference, because from the (ordinary) elimination axiom

$$\forall^{\mathsf{U}}_{\vec{x}}\big(I_j(\vec{x}\,) \to K_0(\vec{I}\wedge\vec{P}) \to \cdots \to K_{k-1}(\vec{I}\wedge\vec{P}) \to I_j(\vec{x}\,)\wedge P_j(\vec{x}\,)\big)$$

(with $\vec{I}\wedge\vec{P}$ denoting the list of predicates $\{\,\vec{x} \mid I_j(\vec{x}\,)\wedge P_j(\vec{x}\,)\,\}$, and conjunction $\wedge$ as defined below)) we can derive the strengthened one

$$\forall^{\mathsf{U}}_{\vec{x}}\big(I_j(\vec{x}\,) \to K_0(\vec{I},\vec{P}) \to \cdots \to K_{k-1}(\vec{I},\vec{P}) \to P_j(\vec{x}\,)\big).$$

To see this, assume $\vec{x}$ and $I_j(\vec{x}\,)$, $K_0(\vec{I},\vec{A})$ … $K_{k-1}(\vec{I},\vec{A})$. We must show $A_j(\vec{x}\,)$. To this end we use the ordinary elimination axiom above. Hence it suffices to prove each $K_i(\vec{I}\wedge\vec{A})$. Assume $\vec{x}$, $\vec{A}$, $\forall_{\vec{y}_\nu}(\vec{B}_\nu \to I_{j_\nu}(\vec{s}_\nu)\wedge A_{j_\nu}(\vec{s}_\nu))$ for $\nu<n$. We must show $I_j(\vec{t}\,)\wedge A_j(\vec{t}\,)$. Now $I_j(\vec{t}\,)$ follows from the introduction axioms, and $A_j(\vec{t}\,)$ follows from $K_i(\vec{I},\vec{A})$.

We shall exclusively use the strengthened elimination axioms. Moreover, we often omit all premises $K_i(\vec{I},\vec{P})$ stemming from efq-clauses, since they are derivable. However, when we later in 2.4.2 prove the soundness theorem for the elimination axioms, we must take their "official" form, with the premises from efq-clauses. This is necessary to ensure that the recursion operator has the right number of arguments.

As is to be expected from the terminology, we have conversion rules, converting a (strengthened) elimination immediately following an introduction:

$$\begin{aligned}&I^-_j\,\vec{p}\vec{q}\vec{t}_i[\vec{x}_i:=\vec{r}_i]((I_j)^+_i\,\vec{p}\vec{r}_i\vec{N}^P_i\vec{N}^R_i)(M_l^{K_l(\vec{I}(\vec{p}),\vec{P}(\vec{p},\vec{q}))})_{l<k} \mapsto\\ &M_i\vec{r}_i\vec{N}^P_i\vec{N}^R_i\big(\lambda_{\vec{y}_{i\nu},\vec{u}_{i\nu}}(I^-_{j_{i\nu}}\vec{p}\vec{q}\vec{s}_{i\nu}[\vec{x}_i{:=}\vec{r}_i](N^R_{i\nu}\vec{y}_{i\nu}\vec{u}_{i\nu})(M_l)_{l<k})\big)_{\nu<n_i},\end{aligned}$$

for $j<N$. Here $\vec{p}$ are to be substituted for the free variables of $\vec{I}$, and $\vec{q}$ for those of $\vec{P}$ which are not in $\vec{I}$. Moreover, the $\vec{N}^P_i$ are the parameter arguments for the i-th introduction axiom $K_i(\vec{I}\,)$ (i.e., proofs of its premises $\vec{A}_i$), and $\vec{N}^R_i$ are the recursive arguments (i.e., proofs of its further premises $\forall\vec{y}_{i\nu}(\vec{B}_{i\nu}\to I_{j_{i\nu}}(\vec{s}_{i\nu}))$).

2.1.2. Examples. The following inductive definitions of the existential quantifier, conjunction, falsity, equality and disjunction have been used by Martin-Löf (1971).

Existential quantifier. Let α be a type variable, y an object variable of type α, and $\hat{Q}$ a predicate variable of arity (α). We have four variants, depending on where we require uniformity.

$$\mathrm{Ex}(\alpha,\hat{Q}) \quad := \mu_X\big(\forall_y(\hat{Q}(y)\to X)\big),$$

$$\begin{aligned}
\mathrm{ExL}(\alpha,\hat{Q}) &:= \mu_X\big(\forall_y(\hat{Q}(y) \to^{\mathsf{U}} X)\big),\\
\mathrm{ExR}(\alpha,\hat{Q}) &:= \mu_X\big(\forall^{\mathsf{U}}_y(\hat{Q}(y) \to X)\big),\\
\mathrm{ExU}(\alpha,\hat{Q}) &:= \mu_X\big(\forall^{\mathsf{U}}_y(\hat{Q}(y) \to^{\mathsf{U}} X)\big).
\end{aligned}$$

The introduction axioms are

$$\begin{aligned}
\exists^+\colon&\quad \forall_x(A \to \exists_x A),\\
(\exists^{\mathsf{L}})^+\colon&\ \forall_x(A \to^{\mathsf{U}} \exists^{\mathsf{L}}_x A),\\
(\exists^{\mathsf{R}})^+\colon&\ \forall^{\mathsf{U}}_x(A \to \exists^{\mathsf{R}}_x A),\\
(\exists^{\mathsf{U}})^+\colon&\ \forall^{\mathsf{U}}_x(A \to^{\mathsf{U}} \exists^{\mathsf{U}}_x A),
\end{aligned}$$

where $\exists_x A$ abbreviates $\mathrm{Ex}(\rho,\{\, x^\rho \mid A\,\})$ (and similarly for the other ones), and the elimination axioms are (with $x \notin \mathrm{FV}(C)$)

$$\begin{aligned}
\exists^-\colon&\quad \exists_x A \to \forall_x(A \to C) \to C,\\
(\exists^{\mathsf{L}})^-\colon&\ \exists^{\mathsf{L}}_x A \to \forall_x(A \to^{\mathsf{U}} C) \to C,\\
(\exists^{\mathsf{R}})^-\colon&\ \exists^{\mathsf{R}}_x A \to \forall^{\mathsf{U}}_x(A \to C) \to C,\\
(\exists^{\mathsf{U}})^-\colon&\ \exists^{\mathsf{U}}_x A \to \forall^{\mathsf{U}}_x(A \to^{\mathsf{U}} C) \to C.
\end{aligned}$$

Conversion:

$$\exists^-\vec{p}\vec{q}\big(\exists^+\vec{p}r^\rho N^{A(\vec{p},r)}\big)M^{\forall_n(A(\vec{p})\to Q(\vec{p},\vec{q}))} \mapsto MrN.$$

Conjunction. Let $\hat{P},\hat{Q}$ be nullary predicate variables. We define

$$\begin{aligned}
\mathrm{And}(\hat{P},\hat{Q}) &:= \mu_X\big(\hat{P} \to \hat{Q} \to X\big).\\
\mathrm{AndL}(\hat{P},\hat{Q}) &:= \mu_X\big(\hat{P} \to \hat{Q} \to^{\mathsf{U}} X\big).\\
\mathrm{AndR}(\hat{P},\hat{Q}) &:= \mu_X\big(\hat{P} \to^{\mathsf{U}} \hat{Q} \to X\big).\\
\mathrm{AndU}(\hat{P},\hat{Q}) &:= \mu_X\big(\hat{P} \to^{\mathsf{U}} \hat{Q} \to^{\mathsf{U}} X\big).
\end{aligned}$$

The introduction axioms are

$$\begin{aligned}
\wedge^+\colon&\quad A \to B \to A \wedge B,\\
(\wedge^{\mathsf{L}})^+\colon&\ A \to B \to^{\mathsf{U}} A \wedge^{\mathsf{L}} B,\\
(\wedge^{\mathsf{R}})^+\colon&\ A \to^{\mathsf{U}} B \to A \wedge^{\mathsf{R}} B,\\
(\wedge^{\mathsf{U}})^+\colon&\ A \to^{\mathsf{U}} B \to^{\mathsf{U}} A \wedge^{\mathsf{U}} B
\end{aligned}$$

where $A \wedge B$ abbreviates $\mathrm{And}(\{\mid A\,\},\{\mid B\,\})$ (and similarly for the other ones), and elimination axioms

$$\wedge^-\colon\quad A \wedge B \to (A \to B \to C) \to C,$$

$$(\wedge^{\mathsf{L}})^{-}\colon A \wedge^{\mathsf{L}} B \to (A \to B \to^{\mathsf{U}} C) \to C,$$
$$(\wedge^{\mathsf{R}})^{-}\colon A \wedge^{\mathsf{R}} B \to (A \to^{\mathsf{U}} B \to C) \to C,$$
$$(\wedge^{\mathsf{U}})^{-}\colon A \wedge^{\mathsf{U}} B \to (A \to^{\mathsf{U}} B \to^{\mathsf{U}} C) \to C.$$

Conversion (assuming that there are no free variables in A, B and C):

$$\wedge^{-}(\wedge^{+}N_0^A N_1^B)M^{A\to B\to C} \mapsto MN_0N_1.$$

Falsity. $\bot := \mu_X(F \to X)$. This example is somewhat extreme, since the list $\vec{K}$ in the general form $\mu_{\vec{X}}\vec{K}$ is almost empty here: it only consists of an efq-clause. The only introduction axiom is

$$\bot^{+}\colon F \to \bot$$

and the elimination axiom

$$\bot^{-}\colon \bot \to C.$$

Conversion (assuming that there are no free variables in C): Let $N_0\colon F \to C$.

$$(\bot^{-})^{\bot\to C}\big((\bot^{+})^{F\to\bot}M^F\big) \mapsto N_0M.$$

Equality. Let α be a type variable, x, y object variables of type α, and X a predicate variable of arity (α, α). We define *Leibniz equality* by

$$\mathrm{Eq}(\alpha) := \mu_X\big(\forall^{\mathsf{U}}_{x,y}(F \to X(x,y)), \forall^{\mathsf{U}}_{x}X(x,x)\big).$$

The introduction axioms are

$$\mathrm{Eq}_0^{+}\colon \forall^{\mathsf{U}}_{n,m}(F \to \mathrm{Eq}(n,m)), \quad \mathrm{Eq}_1^{+}\colon \forall^{\mathsf{U}}_{n}\mathrm{Eq}(n,n)$$

where $\mathrm{Eq}(n,m)$ abbreviates $\mathrm{Eq}(\rho)(n^\rho, m^\rho)$, and the elimination axiom is

$$\mathrm{Eq}^{-}\colon \forall^{\mathsf{U}}_{n,m}\big(\mathrm{Eq}(n,m) \to \forall^{\mathsf{U}}_{n}Q(n,n) \to Q(n,m)\big).$$

One easily proves symmetry, transitivity and also *compatibility* of Eq:

LEMMA (CompatEq). $\forall^{\mathsf{U}}_{n_1,n_2}\big(\mathrm{Eq}(n_1,n_2) \to Q(n_1) \to Q(n_2)\big)$.

PROOF. Use Eq^{-}; the details are left as an exercise. □

Conversions:

$$(\mathrm{Eq}_0^{+})^{\forall_{n,m}(F\to\mathrm{Eq}(n,m))}rsM^F\{N_0^{\forall_{n,m}(F\to Q(n,m))}, N_1^{\forall_n Q(n,n)}\} \mapsto N_0rsM,$$
$$(\mathrm{Eq}_1^{+})^{\forall_n\mathrm{Eq}(n,n)}t\{N_0^{\forall_{n,m}(F\to Q(n,m))}, N_1^{\forall_n Q(n,n)}\} \mapsto N_1t.$$

Disjunction. Let $\hat{P}, \hat{Q}$ be nullary predicate variables. We define

$$\mathrm{Or}(\hat{P}, \hat{Q}) := \mu_X\big(\hat{P} \to X, \hat{Q} \to X\big).$$

The introduction axioms are

$$\vee_0^+ \colon A \to A \vee B, \quad \vee_1^+ \colon B \to A \vee B$$

where $A \vee B$ abbreviates $\mathrm{Or}(\{\,|\,A\,\}, \{\,|\,B\,\})$, and the elimination axiom is

$$\vee^- \colon A \vee B \to (A \to C) \to (B \to C) \to C.$$

Conversions:

$$\vee^-\big(\vee_0^+ N^A\big) M_0^{A\to C} M_1^{B\to C} \mapsto M_0 N,$$
$$\vee^-\big(\vee_1^+ N^B\big) M_0^{A\to C} M_1^{B\to C} \mapsto M_1 N,$$

and in case $A \vee B$ has free variables to be substituted by $\vec{p}$ and the remaining free variables of C are to be substituted by $\vec{q}$

$$\vee^- \vec{p}\vec{q}\big(\vee_0^+ \vec{p} N^{A(\vec{p})}\big) M_0^{A(\vec{p})\to Q(\vec{p},\vec{q})} M_1^{B(\vec{p})\to Q(\vec{p},\vec{q})} \mapsto M_0 N,$$
$$\vee^- \vec{p}\vec{q}\big(\vee_1^+ \vec{p} N^{B(\vec{p})}\big) M_0^{A(\vec{p})\to Q(\vec{p},\vec{q})} M_1^{B(\vec{p})\to Q(\vec{p},\vec{q})} \mapsto M_1 N.$$

The even numbers. The introduction axioms are

$$\mathrm{Even}_0^+ \colon \forall_n^{\mathsf{U}}(F \to \mathrm{Even}(n)),$$
$$\mathrm{Even}_1^+ \colon \mathrm{Even}(0),$$
$$\mathrm{Even}_2^+ \colon \forall_n^{\mathsf{U}}(\mathrm{Even}(n) \to \mathrm{Even}(\mathrm{S}(\mathrm{S}n)))$$

and the (strengthened) elimination axiom is Even^-:

$$\forall_m^{\mathsf{U}}\big(\mathrm{Even}(m) \to P(0) \to \forall_n^{\mathsf{U}}(\mathrm{Even}(n) \to P(n) \to P(\mathrm{S}(\mathrm{S}n))) \to P(m)\big).$$

Notice that Even when defined inductively "requires witnesses" (as defined in 2.2.3 below), which intuitively means that it has computational content. However, for the boolean-valued function even: $\mathbf{N} \to \mathbf{B}$ we can prove

$$\forall_n\big(\mathrm{Even}(n) \to \mathrm{atom}(\mathrm{even}(n))\big),$$
$$\forall_n\big(\mathrm{atom}(\mathrm{even}(n)) \to \mathrm{Even}(n)\big).$$

The first proof uses Even^- and the fact that $\mathrm{atom}(\mathrm{even}(n))$ has the right closure properties. The second one employs induction on n, using Even_1^+ and Even_2^+.

The even and the odd numbers. As an easy example of a simultaneous inductive definition we take the sets Ev and Od of the even and odd numbers. Again we choose the natural numbers as the underlying algebra. The introduction axioms are

$$\begin{aligned}&\mathrm{Ev}_0^+ : \forall_n^{\mathsf{U}}(F \to \mathrm{Ev}(n)),\\ &\mathrm{Ev}_1^+ : \mathrm{Ev}(0),\\ &\mathrm{Ev}_2^+ : \forall_n^{\mathsf{U}}\big(\mathrm{Od}(n) \to \mathrm{Ev}(\mathrm{S}n)\big),\end{aligned}\qquad\begin{aligned}&\mathrm{Od}_0^+ : \forall_n^{\mathsf{U}}(F \to \mathrm{Od}(n)),\\ &\mathrm{Od}_1^+ : \forall_n^{\mathsf{U}}\big(\mathrm{Ev}(n) \to \mathrm{Od}(\mathrm{S}n)\big)\end{aligned}$$

and the (strengthened) elimination axioms are

$$\begin{aligned}\mathrm{Ev}^- : \forall_n^{\mathsf{U}}\big(&P_0(0) \to \forall_n^{\mathsf{U}}(\mathrm{Od}(n) \to P_1(n) \to P_0(\mathrm{S}n)) \to\\ &\forall_n^{\mathsf{U}}(\mathrm{Ev}(n) \to P_0(n) \to P_1(\mathrm{S}n)) \to \mathrm{Ev}(n) \to P_0(n)\big),\end{aligned}$$

$$\begin{aligned}\mathrm{Od}^- : \forall_n^{\mathsf{U}}\big(&P_0(0) \to \forall_n^{\mathsf{U}}(\mathrm{Od}(n) \to P_1(n) \to P_0(\mathrm{S}n)) \to\\ &\forall_n^{\mathsf{U}}(\mathrm{Ev}(n) \to P_0(n) \to P_1(\mathrm{S}n)) \to \mathrm{Od}(n) \to P_1(n)\big).\end{aligned}$$

The accessible part of an ordering. Let $\prec$ be a binary relation. Assume that we have decidable sets M of its minimal elements and I of its interior elements. Then the *accessible part* of $\prec$ is inductively defined as follows. The introduction axioms are

$$\begin{aligned}&\mathrm{Acc}_0^+ : \forall_x(M(x) \to^{\mathsf{U}} \mathrm{Acc}(x)),\\ &\mathrm{Acc}_1^+ : \forall_x(I(x) \to^{\mathsf{U}} \forall_{y \prec x}\mathrm{Acc}(y) \to \mathrm{Acc}(x)),\end{aligned}$$

and the (strengthened) elimination axiom is as expected.

The bar predicate. Call a sequence $w_0, \ldots, w_{n-1}$ of words *good* if there are indices $i < j < n$ and an embedding f of w_i into w_j. The introduction axioms are

$$\begin{aligned}&\forall_{ws,i,j,f}^{\mathsf{U}}(\mathrm{Good}(ws, i, j, f) \to \mathrm{Bar}(ws)),\\ &\forall_{ws}^{\mathsf{U}}(\forall_w \mathrm{Bar}(ws * w) \to \mathrm{Bar}(ws))\end{aligned}$$

and the (strengthened) elimination axiom is

$$\begin{aligned}\forall_{ws}^{\mathsf{U}}\big(\mathrm{Bar}(ws) \to \forall_{ws,i,j,f}^{\mathsf{U}}&(\mathrm{Good}(ws, i, j, f) \to P(ws)) \to\\ &\forall_{ws}^{\mathsf{U}}(\forall_w \mathrm{Bar}(ws * w) \to \forall_w P(ws * w) \to P(ws)) \to\\ &P(ws)\big).\end{aligned}$$

The transitive closure of a relation $\prec$. The introduction axioms are

$$\begin{aligned}&\forall_{x,y}(x \prec y \to^{\mathsf{U}} \mathrm{TrCl}(x, y)),\\ &\forall_x \forall_{y,z}^{\mathsf{U}}(x \prec y \to^{\mathsf{U}} \mathrm{TrCl}(y, z) \to \mathrm{TrCl}(x, z))\end{aligned}$$

and the (strengthened) elimination axiom is

$$\forall^{\mathsf{U}}_{x_1,y_1}\big(\mathrm{TrCl}(x_1,y_1) \to \forall_{x,y}(x \prec y \to^{\mathsf{U}} P(x,y)) \to \forall_x\forall^{\mathsf{U}}_{y,z}(x \prec y \to^{\mathsf{U}} \mathrm{TrCl}(y,z) \to P(y,z) \to P(x,z)) \to P(x_1,y_1)\big).$$

Pointwise equality. For a type ρ let $\mathrm{fvt}(\rho)$ be the set of its *final value types*, consisting of base types. We define $\mathrm{fvt}(\rho)$ by induction on ρ: $\mathrm{fvt}(\rho \to \sigma) := \mathrm{fvt}(\sigma)$, and for a base type $\mu = (\mu_{\vec{\alpha}}(\kappa_0,\dots,\kappa_{k-1}))_j$, $\mathrm{fvt}(\mu)$ consists of all the (base) types $\mu_{\vec{\alpha}}(\kappa_0,\dots,\kappa_{k-1})$, plus the final value types of all parameter types of μ.

For every type ρ we inductively define *pointwise equality* $=_\rho$. The introduction axioms are, for every base type μ without parameter types,

$$\forall^{\mathsf{U}}_{x_1,x_2}(F \to x_1 =_\mu x_2).$$

For every constructor C_i of a base type μ_j we have an introduction axiom

$$\forall^{\mathsf{U}}_{\vec{y},\vec{z}}\big(\vec{y}^P =_{\vec{\rho}} \vec{z}^P \to \big(\forall_{\vec{x}_\nu}(y^R_{m+\nu}\vec{x}_\nu =_{\mu_{j_\nu}} z^R_{m+\nu}\vec{x}_\nu)\big)_{\nu<n} \to \mathrm{C}_i\vec{y} =_{\mu_j} \mathrm{C}_i\vec{z}\big).$$

For every arrow type $\rho \to \sigma$ we have the introduction axiom

$$\forall^{\mathsf{U}}_{x_1,x_2}\big(\forall_y(x_1y =_\sigma x_2y) \to x_1 =_{\rho\to\sigma} x_2\big).$$

For example, $=_{\mathbf{N}}$ is inductively defined by

$$\forall^{\mathsf{U}}_{n_1,n_2}(F \to n_1 =_{\mathbf{N}} n_2),$$
$$0 =_{\mathbf{N}} 0,$$
$$\forall^{\mathsf{U}}_{n_1,n_2}(n_1 =_{\mathbf{N}} n_2 \to \mathrm{S}n_1 =_{\mathbf{N}} \mathrm{S}n_2),$$

and the elimination axiom is

$$\forall^{\mathsf{U}}_{m_1,m_2}\big(m_1 =_{\mathbf{N}} m_2 \to P(0,0) \to \forall^{\mathsf{U}}_{n_1,n_2}\big(n_1 =_{\mathbf{N}} n_2 \to P(n_1,n_2) \to P(\mathrm{S}n_1,\mathrm{S}n_2)\big) \to P(m_1,m_2)\big).$$

An example with a non-finitary base type is $=_{\mathbf{T}}$ with $\mathbf{T} := \mathcal{T}_1$ (cf. 1.2.1):

$$\forall^{\mathsf{U}}_{x_1,x_2}(F \to x_1 =_{\mathbf{T}} x_2),$$
$$0 =_{\mathbf{T}} 0,$$
$$\forall^{\mathsf{U}}_{f_1,f_2}(\forall_n(f_1n =_{\mathbf{T}} f_2n) \to \mathrm{Sup}f_1 =_{\mathbf{T}} \mathrm{Sup}f_2),$$

and the elimination axiom is

$$=^{-}_{\mathbf{T}}\colon \forall^{\mathsf{U}}_{x_1,x_2}\big(x_1 =_{\mathbf{T}} x_2 \to P(0,0) \to \forall^{\mathsf{U}}_{f_1,f_2}\big(\forall_n(f_1 n =_{\mathbf{T}} f_2 n) \to \forall_n P(f_1 n, f_2 n) \to P(\mathrm{Sup} f_1, \mathrm{Sup} f_2)\big) \to P(x_1,x_2)\big).$$

Recall that in 1.2.3 we have defined, for a finitary base type μ, (decidable) equality as a boolean-valued function $=_\mu\colon \mu \to \mu \to \mathbf{B}$. It is easy to see that we can derive

$$\forall_{x_1,x_2}\big(x_1 =_\mu x_2 \to \mathrm{atom}(=_\mu (x_1,x_2))\big),$$
$$\forall_{x_1,x_2}\big(\mathrm{atom}(=_\mu (x_1,x_2)) \to x_1 =_\mu x_2\big).$$

The proof uses the elimination axiom in the first case, and in the second one a double induction on x_1 and x_2, and the clause $\forall^{\mathsf{U}}_{x_1,x_2}(F \to x_1 =_\mu x_2)$. The details are left as an exercise. However, for a non-finitary base type like $\mathbf{T}$ this alternative definition is not available.

One can prove *reflexivity* of $=_\rho$, using meta-induction on ρ, and induction on the types in $\mathrm{fvt}(\rho)$.

Lemma (ReflPtEq). $\forall_n(n =_\rho n)$.

A consequence is that Leibniz equality implies pointwise equality:

Lemma (EqToPtEq). $\forall_{n_1,n_2}\big(\mathrm{Eq}(n_1,n_2) \to n_1 =_\rho n_2\big)$.

Proof. Use CompatEq and ReflPtEq. □

2.1.3. Further axioms and their consequences. We express *extensionality* of our intended model by stipulating that pointwise equality implies Leibniz equality:

Axiom (PtEqToEq). $\forall_{n_1,n_2}\big(n_1 =_\rho n_2 \to \mathrm{Eq}(n_1,n_2)\big)$.

Notice that this implies the following proposition, which is sometimes called extensionality as well:

Lemma (CompatPtEqFct). $\forall_f \forall^{\mathsf{U}}_{n_1,n_2}(n_1 =_\rho n_2 \to f n_1 =_\sigma f n_2)$.

Proof. We obtain $\mathrm{Eq}(n_1,n_2)$ by PtEqToEq. By ReflPtEq we have $f n_1 =_\sigma f n_1$, hence $f n_1 =_\sigma f n_2$ by CompatEq. □

A consequence of the extensionality axioms is that compatibility holds for pointwise equality as well:

Lemma (CompatPtEq). $\forall^{\mathsf{U}}_{n_1,n_2}\big(n_1 =_\rho n_2 \to P(n_1) \to P(n_2)\big)$.

Proof. Use PtEqToEq and CompatEq. □

We write E-ID$^\omega$ when the extensionality axioms PtEqToEq are present. In E-ID$^\omega$ we can prove properties of the constructors of our free algebras: that they are *injective*, and have *disjoint ranges*. For finitary algebras this can be seen easily, using boolean-valued equality. However, for non-finitary algebras we need extensionality. Since extensionality implies that pointwise and Leibniz equality are equivalent, it suffices to consider pointwise equality. Rather than dealing with the general case, we confine ourselves with the algebra $\mathbf{T}$.

The proof uses some recursive functions: $\mathrm{TreeSup}\colon \mathbf{T} \to \mathbf{B}$ defined by

$$\mathrm{TreeSup}(0) := \mathsf{ff}, \quad \mathrm{TreeSup}(\mathrm{Sup} f) := \mathsf{tt}$$

and a predecessor function $\mathrm{TreePred}\colon \mathbf{T} \to \mathbf{N} \to \mathbf{B}$ defined by

$$\mathrm{TreePred}(0, n) := 0, \quad \mathrm{TreePred}(\mathrm{Sup} f, n) := fn.$$

LEMMA. *$0 =_{\mathbf{T}} \mathrm{Sup} f \to F$, and $\forall^{\mathsf{U}}_{f_1,f_2}(\mathrm{Sup} f_1 =_{\mathbf{T}} \mathrm{Sup} f_2 \to f_1 =_{\mathbf{N}\to\mathbf{T}} f_2)$.*

PROOF. The proof uses the various compatibilities, and the fact that conversions give rise to Leibniz equalities. □

We now list some further axioms, which will be mentioned when we use them. All of them involve (inductively defined) existentially quantified formulas, wich come in four versions, with $\exists, \exists^{\mathsf{R}}, \exists^{\mathsf{L}}, \exists^{\mathsf{U}}$. Let $\breve{\exists}$ denote any of these. When $\breve{\exists}$ appears more than once in an axiom below, it is understood that it denotes the same quantifier each time.

The *axiom of choice* (AC) is the scheme

AXIOM (AC). $\forall_{x^\rho} \breve{\exists}_{y^\sigma} A(x, y) \to \breve{\exists}_{f^{\rho\to\sigma}} \forall_{x^\rho} A(x, f(x))$.

The *independence* axioms express the intended meaning of uniformities. The *independence of premise* axiom (IP) is

AXIOM (IP). $(A \to^{\mathsf{U}} \breve{\exists}_x B) \to \breve{\exists}_x (A \to^{\mathsf{U}} B)$ *($x \notin \mathrm{FV}(A)$).*

Similarly we have an *independence of quantifier* axiom (IQ) axiom

AXIOM (IQ). $\forall^{\mathsf{U}}_x \breve{\exists}_y A \to \breve{\exists}_y \forall^{\mathsf{U}}_x A$ *($x \notin \mathrm{FV}(A)$).*

We will also consider the (constructively doubtful) *Markov principle*, for a higher type variable x^ρ and decidable atoms A_0, B_0:

AXIOM (MP). $(\forall_{x^\rho} A_0 \to B_0) \to \exists_{x^\rho}(A_0 \to B_0)$ *($x^\rho \notin \mathrm{FV}(B_0)$).*

2.2. Computational Content

Along the inductive definition of formulas, predicates and constructor formulas (or clauses) in 2.1.1, we define simultaneously

- the *type* $\tau(A)$ of a formula A;

- when a formula is *computationally relevant*;
- the formula z *realizes* A, written z $\mathbf{r}$ A, for a variable z of type $\tau(A)$;
- when a formula is *negative*;
- when an inductively defined predicate requires *witnesses*;
- for an inductively defined I requiring witnesses, its base type μ_I, and – if I has an efq-clause – when an object of type μ_I is *efq-free*;
- for an inductively defined predicate I of arity $\vec{\rho}$ requiring witnesses, a *witnessing* predicate $I^{\mathbf{r}}$ of arity $(\mu_I, \vec{\rho})$, and a predicate $I^{\mathbf{ef}}$ of arity (μ_I) expressing *efq-free*ness.

2.2.1. The type of a formula. Every formula A possibly containing inductively defined predicates can be seen as a "computational problem". We define $\tau(A)$ as the type of a potential realizer of A, i.e., the type of the term (or "program") to be extracted from a proof of A.

More precisely, we assign to every formula A an object $\tau(A)$ (a type or the "nulltype" symbol ε). In case $\tau(A) = \varepsilon$ proofs of A have no computational content; such formulas A are called *Harrop formulas*, or computationally *irrelevant* (c.i.). Non-Harrop formulas are also called computationally *relevant* (c.r.).

The definition can be conveniently written if we extend the use of $\rho \to \sigma$ to the nulltype symbol ε:

$$(\rho \to \varepsilon) := \varepsilon, \quad (\varepsilon \to \sigma) := \sigma, \quad (\varepsilon \to \varepsilon) := \varepsilon.$$

With this understanding of $\rho \to \sigma$ we can simply write

$$\tau(\mathrm{atom}(r)) := \varepsilon, \quad \tau(I(\vec{r})) := \begin{cases} \varepsilon & \text{if } I \text{ does not require witnesses} \\ \mu_I & \text{otherwise,} \end{cases}$$
$$\tau(A \to B) := (\tau(A) \to \tau(B)), \quad \tau(\forall_{x^\rho} A) := (\rho \to \tau(A)),$$
$$\tau(A \to^{\mathsf{U}} B) := \tau(B), \quad \tau(\forall^{\mathsf{U}}_{x^\rho} A) := \tau(A).$$

2.2.2. Realizability. Let A be a formula and z either a variable of type $\tau(A)$ if the latter is a type, or the nullterm symbol ε if $\tau(A) = \varepsilon$. For a convenient definition we extend the use of term application to the nullterm symbol ε:

$$\varepsilon x := \varepsilon, \quad z\varepsilon := z, \quad \varepsilon\varepsilon := \varepsilon.$$

We define the formula z $\mathbf{r}$ A, to be read z *realizes* A. The definition uses the predicates $I^{\mathbf{r}}$ and $I^{\mathbf{ef}}$ introduced below.

$$z\ \mathbf{r}\ \mathrm{atom}(s) \quad := \mathrm{atom}(s),$$

$$z\ \mathbf{r}\ I(\vec{s}) := \begin{cases} I(\vec{s}) & \text{if } I \text{ does not require witnesses} \\ I^{\mathbf{r}}(z,\vec{s}) & \text{if not, and } I \text{ has no efq-clause} \\ I^{\mathbf{ef}}(z) \wedge I^{\mathbf{r}}(z,\vec{s}) & \text{otherwise,} \end{cases}$$

$$z\ \mathbf{r}\ (A \to B) := \forall_x(x\ \mathbf{r}\ A\ \to\ zx\ \mathbf{r}\ B),$$

$$z\ \mathbf{r}\ (\forall_x A) := \forall_x\, zx\ \mathbf{r}\ A,$$

$$z\ \mathbf{r}\ (A \to^{\mathsf{U}} B) := (A \to z\ \mathbf{r}\ B),$$

$$z\ \mathbf{r}\ (\forall_x^{\mathsf{U}} A) := \forall_x\, z\ \mathbf{r}\ A.$$

Formulas which do not contain inductively defined predicates requiring witnesses play a special role in this context; we call them *negative*. Their crucial property is $(\varepsilon\ \mathbf{r}\ A) = A$. Notice also that every formula $z\ \mathbf{r}\ A$ is negative.

2.2.3. Witnesses. Consider a particularly simple inductively defined predicate, where

- there is at most one clause apart from an efq-clause, and
- this clause is uniform, i.e., contains no $\forall$ but $\forall^{\mathsf{U}}$ only, and its premises are either negative or followed by $\to^{\mathsf{U}}$.

Examples are $\exists^{\mathsf{U}}$, $\wedge^{\mathsf{U}}$, $\bot$, Eq. We call those predicates "uniform one-clause" defined. An inductively defined predicate *requires witnesses* if it is not one of those, and not one of the predicates $I^{\mathbf{r}}$ and $I^{\mathbf{ef}}$ introduced below.

For an inductively defined predicate I requiring witnesses, we define μ_I to be the corresponding component of the types $\vec{\mu} = \mu_{\vec{\alpha}}\vec{\kappa}$ generated from constructor types $\kappa_i := \tau(K_i)$ for all constructor formulas $K_0, \dots K_{k-1}$ from $\vec{I} = \mu_{\vec{X}}(K_0, \dots K_{k-1})$. An object of type μ_I is called *efq-free* it it does not contain a constructor of μ_I corresponding to an efq-clause.

The witnessing predicate $I^{\mathbf{r}}$ of arity $(\mu_I, \vec{\rho})$ can now be defined as follows. For every constructor formula

$$K = \breve{\forall}_{\vec{x}}\big(\vec{A} \mathrel{\breve{\to}} \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} X_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to X_j(\vec{t}\,)\big)$$

of the original inductive definition of $\vec{I}$ we build the new constructor formula

$$K^{\mathbf{r}} := \breve{\forall}_{\vec{x}}\forall_{\vec{u},\vec{f}}\big(\vec{u}\ \mathbf{r}\ \vec{A} \to \big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to Y_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu, \vec{s}_\nu))\big)_{\nu<n} \to Y_j(\mathrm{C}\vec{x}\vec{u}\vec{f}, \vec{t}\,)\big),$$

with the understanding that

- only those x_i with a non-uniform $\forall_{x_i}$ occur as arguments in $\mathrm{C}\vec{x}\vec{u}\vec{f}$,
- only those u_i with A_i a non-uniform premise and $\tau(A_i) \neq \varepsilon$ actually appear (for the other A_i we take either A_i or $\varepsilon\ \mathbf{r}\ A_i$),

and similarly for $y_{\nu,i}$, $v_{\nu,i}$ and $f_\nu\vec{y}_\nu\vec{v}_\nu$. Here C is the constructor of the algebra $\vec{\mu} = \mu_{\vec{\alpha}}\vec{\kappa}$ generated from our constructor types $\kappa_i := \tau(K_i)$ (i.e., for

K_i we have $\mathrm{C} := \mathrm{C}_i$). Then $\vec{I}^{\mathbf{r}} := \mu_{\vec{Y}}(\vec{K}^{\mathbf{r}})$. The corresponding introduction axiom then is $K^{\mathbf{r}}(\vec{I}^{\mathbf{r}})$, that is

$$(I_j^{\mathbf{r}})_i^+ \colon \forall_{\vec{x},\vec{u},\vec{f}}\big(\vec{u}\ \mathbf{r}\ \vec{A} \to \big(\forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n} \to I_j^{\mathbf{r}}(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)\big) \tag{2.3}$$

and the (strengthened) elimination axiom is

$$(I_j^{\mathbf{r}})^- \colon \forall_w \forall^{\mathsf{U}}_{\vec{x}}\big(I_j^{\mathbf{r}}(w,\vec{x}\,) \to \big(K_i^{\mathbf{r}}(\vec{I}^{\mathbf{r}},\vec{P})\big)_{i<k} \to P_j(w,\vec{x}\,)\big) \tag{2.4}$$

with

$$\begin{aligned} K^{\mathbf{r}}(\vec{I}^{\mathbf{r}},\vec{P}) := \breve{\forall}_{\vec{x}}\forall_{\vec{u},\vec{f}}\big(\vec{u}\ \mathbf{r}\ \vec{A} \to\ & \big(\forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n} \to \\ & \big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to P_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n} \to \\ & P_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)\big). \end{aligned}$$

Notice that each of the clauses $(I_j^{\mathbf{r}})_i^+$ has a conclusion $I_j^{\mathbf{r}}(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)$ with its own constructor C. Therefore it is to be expected that the following *inversion* properties for $I^{\mathbf{r}}$ hold:

Lemma (Inversion).

$(I_j^{\mathbf{r}})_i^{\mathrm{invEq}} \colon I_j^{\mathbf{r}}(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{z}\,) \to \exists_{\vec{y}}\mathrm{Eq}(\vec{z},\vec{t}\,)$ *($\vec{y}$ the uniform variables)*,

$(I_j^{\mathbf{r}})_i^{\mathrm{invP}} \colon I_j^{\mathbf{r}}(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,) \to \vec{u}\ \mathbf{r}\ \vec{A}$,

$(I_j^{\mathbf{r}})_i^{\mathrm{invR},\nu} \colon I_j^{\mathbf{r}}(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,) \to \forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))$.

Proof. $(I_j^{\mathbf{r}})_i^{\mathrm{invEq}}$. We use $(I_j^{\mathbf{r}})^-$ with $P(w,\vec{z}\,) :=$

$$\forall_{\vec{x},\vec{x}',\vec{u},\vec{f}}\big(w = \mathrm{C}\vec{x}'\vec{u}\vec{f} \to \exists_{\vec{y}}\mathrm{Eq}(\vec{z},\vec{t}\,)\big)$$

with $\vec{x}$ the non-uniform and $\vec{y}$ the uniform outside universally quantified variables of the clause. It suffices to prove all $K_i^{\mathbf{r}}(\vec{I}^{\mathbf{r}},\vec{P})$ for these $\vec{P}$. Since constructors have disjoint ranges, we only need to consider the C-clause. Its conclusion follows from the injectivity of C.

$(I_j^{\mathbf{r}})_i^{\mathrm{invP}}$. We use $(I_j^{\mathbf{r}})^-$ with

$$P(w,\vec{z}\,) := \forall_{\vec{x},\vec{x}',\vec{u},\vec{f}}\big(w = \mathrm{C}\vec{x}'\vec{u}\vec{f} \to \vec{z} = \vec{t} \to \vec{u}\ \mathbf{r}\ \vec{A}\,\big).$$

It suffices to prove all $K_i^{\mathbf{r}}(\vec{I}^{\mathbf{r}},\vec{P})$ for these $\vec{P}$. The conclusion of its C-clause follows from its first premise, using the injectivity of C.

$(I_j^{\mathbf{r}})_i^{\mathrm{invR},\nu}$. We use $(I_j^{\mathbf{r}})^-$ with

$$\begin{aligned} P(w,\vec{z}\,) := \forall_{\vec{x},\vec{x}',\vec{u},\vec{f}}\big(& w = \mathrm{C}\vec{x}'\vec{u}\vec{f} \to \vec{z} = \vec{t} \to \\ & \forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big). \end{aligned}$$

It suffices to prove all $K_i^{\mathbf{r}}(\vec{I}^{\mathbf{r}}, \vec{P})$ for these $\vec{P}$. The conclusion of its C-clause follows from its second premise, again using the injectivity of C. □

For an inductively defined predicate I requiring witnesses and with an efq-clause we define the predicate $I^{\mathbf{ef}}$ of arity (μ_I) expressing efq-freeness as follows. For every constructor formula

$$K = \breve{\forall}_{\vec{x}}\big(\vec{A} \mathrel{\breve{\to}} \big(\breve{\forall}_{\vec{y}_\nu}(\vec{B}_\nu \mathrel{\breve{\to}} X_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \mathrel{\breve{\to}} X_j(\vec{t}\,)\big)$$

of the original inductive definition of $\vec{I}$ *except* the efq-clause the corresponding introduction axiom is $(I_j^{\mathbf{ef}})_i^+$:

$$\breve{\forall}_{\vec{x},\vec{u},\vec{f}}\big(\vec{u}\ \mathbf{r}\ \vec{A} \to \big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{ef}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu))\big)_{\nu<n} \to I_j^{\mathbf{ef}}(\mathrm{C}\vec{x}\vec{u}\vec{f}\,)\big) \tag{2.5}$$

and the elimination axiom is

$$(I_j^{\mathbf{ef}})^-\colon \forall_w\big(I_j^{\mathbf{ef}}(w) \to \big(K_i^{\mathbf{ef}}(\vec{I}^{\mathbf{ef}}, \vec{P})\big)_{i<k} \to P_j(w)\big). \tag{2.6}$$

As before we can prove

Lemma (Inversion).

$$(I_j^{\mathbf{ef}})_i^{\mathrm{invP}}\colon I_j^{\mathbf{ef}}(\mathrm{C}\vec{x}\vec{u}\vec{f}\,) \to \vec{u}\ \mathbf{r}\ \vec{A},$$
$$(I_j^{\mathbf{ef}})_i^{\mathrm{invR},\nu}\colon I_j^{\mathbf{ef}}(\mathrm{C}\vec{x}\vec{u}\vec{f}\,) \to \forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{ef}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu)).$$

As an example of an inductive definition requiring non-finitary witnesses consider pointwise equality $=_{\mathbf{T}}$ for the algebra $\mathbf{T}$ (cf. 2.1.2). Recall that the constructors of $\mathbf{T}$ are 0 and Sup. Let $\mathrm{C}_0, \mathrm{C}_1, \mathrm{C}_2$ denote the constructors of the witnessing algebra $\mu_{=_{\mathbf{T}}}$. The introduction axioms for the inductively defined witnessing predicate $=^{\mathbf{r}}_{\mathbf{T}}$ are

$$(=^{\mathbf{r}}_{\mathbf{T}})_0^+\colon \forall_{x_1,x_2}\big(F \to =^{\mathbf{r}}_{\mathbf{T}}(\mathrm{C}_0, x_1, x_2)\big),$$
$$(=^{\mathbf{r}}_{\mathbf{T}})_1^+\colon =^{\mathbf{r}}_{\mathbf{T}}(\mathrm{C}_1, 0, 0),$$
$$(=^{\mathbf{r}}_{\mathbf{T}})_2^+\colon \forall_{f,f_1,f_2}\big(\forall_n =^{\mathbf{r}}_{\mathbf{T}}(fn, f_1n, f_2n) \to =^{\mathbf{r}}_{\mathbf{T}}(\mathrm{C}_2f, \mathrm{Sup}f_1, \mathrm{Sup}f_2)\big)$$

and the elimination axiom is

$$\begin{aligned}(=^{\mathbf{r}}_{\mathbf{T}})^-\colon \forall_w\forall^{\mathsf{U}}_{x_1,x_2}\big(&=^{\mathbf{r}}_{\mathbf{T}}(w, x_1, x_2) \to\\ &P(\mathrm{C}_1, 0, 0) \to\\ &\forall_f\forall^{\mathsf{U}}_{f_1,f_2}\big(\forall_n =^{\mathbf{r}}_{\mathbf{T}}(fn, f_1n, f_2n) \to \forall_n P(fn, f_1n, f_2n) \to\\ &\qquad P(\mathrm{C}_2f, \mathrm{Sup}f_1, \mathrm{Sup}f_2)\big) \to\\ &P(w, x_1, x_2)\big).\end{aligned}$$

The inversion lemma is

LEMMA (Inversion).

$$(=^{\mathbf{r}}_{\mathbf{T}})^{\mathrm{invEq}}_2 \colon =^{\mathbf{r}}_{\mathbf{T}}(\mathrm{C}_2 f, z_1, z_2) \to \exists_{f_i}\mathrm{Eq}(z_i, \mathrm{Sup} f_i) \quad (i = 1, 2),$$
$$(=^{\mathbf{r}}_{\mathbf{T}})^{\mathrm{invR}}_2 \colon =^{\mathbf{r}}_{\mathbf{T}}(\mathrm{C}_2 f, \mathrm{Sup} f_1, \mathrm{Sup} f_2) \to \forall_n =^{\mathbf{r}}_{\mathbf{T}}(fn, f_1 n, f_2 n).$$

PROOF. $(=^{\mathbf{r}}_{\mathbf{T}})^{\mathrm{invEq}}_2$. We use $(=^{\mathbf{r}}_{\mathbf{T}})^-$ with

$$P_i(w, z_1, z_2) := \forall_f\big(w = \mathrm{C}_2 f \to \exists_{f_i}\mathrm{Eq}(z_i, \mathrm{Sup} f_i)\big) \quad (i = 1, 2).$$

It suffices to prove all clauses for P_i. Let $i = 1$. The first clause is $P_1(\mathrm{C}_1, 0, 0)$, i.e.,

$$\forall_f\big(\mathrm{C}_1 = \mathrm{C}_2 f \to \exists_{f_1}\mathrm{Eq}(0, \mathrm{Sup} f_1)\big).$$

This holds since constructors have disjoint ranges. Now consider the second one. Its conclusion is $P_1(\mathrm{C}_2 f, \mathrm{Sup} f_1, \mathrm{Sup} f_2)$, i.e.,

$$\forall_{f'}\big(\mathrm{C}_2 f = \mathrm{C}_2 f' \to \exists_{f'_1}\mathrm{Eq}(\mathrm{Sup} f_1, \mathrm{Sup} f'_1)\big).$$

This follows from reflexivity of Eq.

$(=^{\mathbf{r}}_{\mathbf{T}})^{\mathrm{invR}}_2$. We use $(=^{\mathbf{r}}_{\mathbf{T}})^-$ with

$$P(w, z_1, z_2) := \forall_{f_1, f_2, f}\big(w = \mathrm{C}_2 f \to z_1 = \mathrm{Sup} f_1 \to z_2 = \mathrm{Sup} f_2 \to \forall_n =^{\mathbf{r}}_{\mathbf{T}}(fn, f_1 n, f_2 n)\big).$$

It suffices to prove all clauses for P. The first clause is $P(\mathrm{C}_1, 0, 0)$, which holds since constructors have disjoint ranges. Now consider the second one. Its conclusion is $P(\mathrm{C}_2 f, \mathrm{Sup} f_1, \mathrm{Sup} f_2)$, i.e.,

$$\forall_{f'_1, f'_2, f'}\big(\mathrm{C}_2 f = \mathrm{C}_2 f' \to \mathrm{Sup} f_1 = \mathrm{Sup} f'_1 \to \mathrm{Sup} f_2 = \mathrm{Sup} f'_2 \to \forall_n =^{\mathbf{r}}_{\mathbf{T}}(f' n, f'_1 n, f'_2 n)\big).$$

Using the injectivity of constructors, this follows from its first premise. □

The predicate $=^{\mathbf{ef}}_{\mathbf{T}}$ expressing efq-freeness has the introduction axioms

$$(=^{\mathbf{ef}}_{\mathbf{T}})^+_1 \colon =^{\mathbf{ef}}_{\mathbf{T}}(\mathrm{C}_1),$$
$$(=^{\mathbf{ef}}_{\mathbf{T}})^+_2 \colon \forall_f\big(\forall_n =^{\mathbf{ef}}_{\mathbf{T}}(fn) \to =^{\mathbf{ef}}_{\mathbf{T}}(\mathrm{C}_2 f)\big)$$

and the elimination axiom $(=^{\mathbf{ef}}_{\mathbf{T}})^-$:

$$\forall_w\big(=^{\mathbf{ef}}_{\mathbf{T}}(w) \to P(\mathrm{C}_1) \to \forall_f\big(\forall_n =^{\mathbf{ef}}_{\mathbf{T}}(fn) \to \forall_n P(fn) \to P(\mathrm{C}_2 f)\big) \to P(w)\big).$$

One part of the inversion lemma is

$$=^{\mathbf{ef}}_{\mathbf{T}}(\mathrm{C}_2 f) \to \forall_n =^{\mathbf{ef}}_{\mathbf{T}}(fn).$$

2.3. Extracted Terms and Uniform Derivations

We define the extracted term of a derivation, and (using this concept) the notion of a uniform proof, which gives a special treatment to uniform implication $\to^{\mathsf{U}}$ and the uniform universal quantifier $\forall^{\mathsf{U}}$.

2.3.1. Extracted terms. For a derivation M in $\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP}_\varepsilon + \mathrm{Ax}_\varepsilon$, we simultaneously define

- its *extracted term* $[\![M]\!]$, of type $\tau(A)$, and
- when M is *uniform*.

For derivations M^A where $\tau(A) = \varepsilon$ (i.e., A is a Harrop formula) let $[\![M]\!] := \varepsilon$ (the *nullterm* symbol); every such derivation is uniform. Now assume that M derives a formula A with $\tau(A) \neq \varepsilon$. Recall our extended use of term application to the nullterm symbol ε: $\varepsilon x := \varepsilon$, $z\varepsilon := z$, $\varepsilon\varepsilon := \varepsilon$. We also understand that in case $\tau(A) = \varepsilon$, $\lambda_{x_u^{\tau(A)}}[\![M]\!]$ means just $[\![M]\!]$. Then

$$\begin{aligned}
[\![u^A]\!] &:= x_u^{\tau(A)} \quad (x_u^{\tau(A)} \text{ uniquely associated with } u^A),\\
[\![\lambda_{u^A} M]\!] &:= \lambda_{x_u^{\tau(A)}}[\![M]\!],\\
[\![M^{A\to B} N]\!] &:= [\![M]\!][\![N]\!],\\
[\![(\lambda_{x^\rho} M)^{\forall_x A}]\!] &:= \lambda_{x^\rho}[\![M]\!],\\
[\![M^{\forall_x A} r]\!] &:= [\![M]\!] r.\\
[\![\lambda^{\mathsf{U}}_{u^A} M]\!] &:= [\![M^{A\to^{\mathsf{U}} B} N]\!] := [\![(\lambda^{\mathsf{U}}_{x^\rho} M)^{\forall^{\mathsf{U}}_x A}]\!] := [\![M^{\forall^{\mathsf{U}}_x A} r]\!] := [\![M]\!].
\end{aligned}$$

In all these cases uniformity is preserved, except possibly in those involving λ^{U}: $\lambda^{\mathsf{U}}_{u^A} M$ is uniform if M is and $x_u \notin \mathrm{FV}([\![M]\!])$, and $\lambda^{\mathsf{U}}_{x^\rho} M$ is uniform if M is and – in addition to the usual variable condition – $x \notin \mathrm{FV}([\![M]\!])$.

It remains to define extracted terms for the axioms: structural and general induction, introduction and elimation axioms for inductively defined predicates, (AC) and (IP_ε).

The extracted term $[\![\mathrm{Ind}_j]\!]$ of an induction axiom is defined to be the recursion operator $\mathcal{R}^{\vec{\mu},\vec{\tau}}_{\mu_j}$. For example, in case of an induction scheme

$$\mathrm{Ind}_{n,A} \colon \forall_m\big(A(0) \to \forall_n(A(n) \to A(\mathrm{S}n)) \to A(m^{\mathbf{N}})\big)$$

we have

$$[\![\mathrm{Ind}_{n,A}]\!] := \mathcal{R}^\tau_{\mathbf{N}} \colon \mathbf{N} \to \tau \to (\mathbf{N} \to \tau \to \tau) \to \tau \quad (\tau := \tau(A) \neq \varepsilon).$$

Generally, the $\vec{\mu}, \vec{\tau}$ in $\mathcal{R}^{\vec{\mu},\vec{\tau}}_{\mu_j}$ list only the types μ_j, τ_j with $\tau_j := \tau(A_j) \neq \varepsilon$, i.e., the recursion operator is simplified accordingly.

For general induction, we take general recursion as its extracted term. For the introduction axiom (2.1) and the (strengthened) elimination axiom (2.2) of an inductively defined predicate I we define

$$[\![(I_j)^+_i]\!] := \mathrm{C}, \quad [\![I^-_j]\!] := \mathcal{R}_j,$$

and similary for the introduction and elimination axioms for $I^{\mathbf{r}}$ and $I^{\mathbf{ef}}$: (2.3), (2.4) and (2.5), (2.6), respectively.

As extracted terms of (AC), (IP) and (IQ) we take identities of the appropriate types.

2.3.2. Uniform derivations. Here we collect some general remarks on uniform derivations.

LEMMA. *There are purely logical uniform derivations of*

(a) $A \to B$ *from* $\mathrm{A} \to^{\mathsf{U}} B$;
(b) $A \to^{\mathsf{U}} B$ *from* $\mathrm{A} \to B$, *provided* $\tau(A) = \varepsilon$ *or* $\tau(B) = \varepsilon$;
(c) $\forall_x A$ *from* $\forall_x^{\mathsf{U}} A$;
(d) $\forall_x^{\mathsf{U}} A$ *from* $\forall_x A$, *provided* $\tau(A) = \varepsilon$.

PROOF. (a). $\lambda_v(u^{A\to^{\mathsf{U}}B} v^A)$ is uniform (there are no conditions on λ_v). (b). If $\tau(B) = \varepsilon$, then $\lambda_v^{\mathsf{U}}(u^{A\to B} v^A)$ is unform because its conclusion is a Harrop formula. Now assume $\tau(A) = \varepsilon$. Then for $\lambda_v^{\mathsf{U}}(u^{A\to B} v^A)$ to be uniform we need to know that $x_v \notin \mathrm{FV}([\![uv]\!])$. But $[\![uv]\!] = [\![u]\!]$ because of $\tau(A) = \varepsilon$. (c). $\lambda_x(u^{\forall_x^{\mathsf{U}} A} x)$ is uniform (there is only the usual variable condition on λ_x). (d). $\lambda_x^{\mathsf{U}}(u^{\forall_x A} x)$ is uniform because its conclusion is a Harrop formula. □

We certainly want to know that in formulas involving $\to^{\mathsf{U}}$ and $\forall^{\mathsf{U}}$ we can replace a subformula by an equivalent one.

LEMMA. *There are purely logical uniform derivations of*

(a) $(A \to^{\mathsf{U}} B) \to (B \to B') \to A \to^{\mathsf{U}} B'$;
(b) $(A' \to A) \to^{\mathsf{U}} (A \to^{\mathsf{U}} B) \to A' \to^{\mathsf{U}} B$;
(c) $\forall_x^{\mathsf{U}} A \to (A \to A') \to \forall_x^{\mathsf{U}} A'$.

PROOF. (a). $\lambda_{u,v}\lambda_w^{\mathsf{U}}(v^{B\to B'}(u^{A\to^{\mathsf{U}}B} w^A))$ is uniform because $[\![v(uw)]\!] = x_v x_u$ does not contain x_w free. (b). $\lambda_u^{\mathsf{U}}\lambda_v\lambda_w^{\mathsf{U}}(v^{A\to^{\mathsf{U}}B}(u^{A'\to A} w^{A'}))$ is uniform since $[\![v(uw)]\!] = x_v$ does not contain x_u, x_w free. (c). $\lambda_{u,v}\lambda_x^{\mathsf{U}}(v^{A\to A'}(u^{\forall_x^{\mathsf{U}} A} x))$ is uniform because $[\![v(ux)]\!] = x_v x_u$ does not contain x free. □

For the (inductively defined) existential quantifiers $\exists, \exists^{\mathsf{R}}, \exists^{\mathsf{L}}, \exists^{\mathsf{U}}$ we observe the following. Let $\breve{\exists}$ denote any of these.

LEMMA. *There are uniform derivations using* $\exists$*-axioms only of*

(a) $\exists_x A \to \breve{\exists}_x A$;
(b) $\breve{\exists}_x A \to \exists_x^{\mathsf{U}} A$;
(c) $\exists_x^{\mathsf{L}} A \to \exists_x A$, *provided* $\tau(A) = \varepsilon$.

PROOF. (a) Use $\exists^-\colon \exists_x A \to \forall_x(A \to \breve{\exists}_x A) \to \breve{\exists}_x A$. We derive the second premise using an introduction axiom. An example is

$$\dfrac{\dfrac{(\exists^{\mathsf{L}})^+\colon \forall_x(A \to^{\mathsf{U}} \exists^{\mathsf{L}}_x A) \qquad x \qquad u\colon A}{\exists^{\mathsf{L}}_x A}}{\dfrac{A \to \exists^{\mathsf{L}}_x A}{\forall_x(A \to \exists^{\mathsf{L}}_x A)}\,\forall^+ x}\,{\to^+}u$$

(b). Assume $\breve{\exists}$ is $\exists^{\mathsf{L}}$. Use $(\exists^{\mathsf{L}})^-\colon \exists^{\mathsf{L}}_x A \to \forall_x(A \to^{\mathsf{U}} \exists^{\mathsf{U}}_x A) \to \exists^{\mathsf{U}}_x A$. We can prove the second premise uniformly from $(\exists^{\mathsf{U}})^+\colon \forall^{\mathsf{U}}_x(A \to^{\mathsf{U}} \exists^{\mathsf{U}}_x A)$, by $\lambda_x\lambda^{\mathsf{U}}_u((\exists^{\mathsf{U}})^+ x u^A)$. (c). Use $(\exists^{\mathsf{L}})^-\colon \exists^{\mathsf{L}}_x A \to \forall_x(A \to^{\mathsf{U}} \exists_x A) \to \exists_x A$. This time the second premise is proven uniformly from $\exists^+\colon \forall_x(A \to \exists_x A)$ by $\lambda_x\lambda^{\mathsf{U}}_u(\exists^+ x u^A)$, because $\tau(A) = \varepsilon$ implies $[\![\exists^+ x u^A]\!] = x$. □

2.3.3. Characterization. We consider the question when a formula A and its modified realizability interpretation $\exists_x\, x\ \mathbf{r}\ A$ are equivalent.

Let us first look at some examples. First take $\exists_x A$, which abbreviates $\mathrm{Ex}(\rho, \{\, x^\rho \mid A \,\})$. Recall that $\exists_x A$ is an inductively defined predicate requiring witnesses, and that it has no efq-clause. We want to compare $\exists_x A$ with $\exists_w\,(w\ \mathbf{r}\ \exists_x A)$, which is $\exists_w(\exists_x A)^{\mathbf{r}}(w)$. One direction

$$\exists_w(\exists_x A)^{\mathbf{r}}(w) \to \exists_x A$$

is proved using

$$((\exists_x A)^{\mathbf{r}})^-\colon \forall_w\big((\exists_x A)^{\mathbf{r}}(w) \to \forall_{x,u}\big(u\ \mathbf{r}\ A \to P(\mathrm{C}xu)\big) \to P(w)\big),$$

with $P(w) := \exists_x A$. We only need to show $\forall_{x,u}(u\ \mathbf{r}\ A \to \exists_x A)$. Fix x, u and assume $u\ \mathbf{r}\ A$. Then A by IH (we need to do an induction along the inductive definition of formulas, predicates and constructor formulas (or clauses) in 2.1.1), and $\exists_x A$ by $\exists^+$. For the other direction

$$\exists_x A \to \exists_w(\exists_x A)^{\mathbf{r}}(w)$$

we use $\exists^-$. Then we have so show $\forall_x(A \to \exists_w(\exists_x A)^{\mathbf{r}}(w))$. Fix x and assume A. Then $u\ \mathbf{r}\ A$ for some u by IH. The claim follows from $\exists^+$ and

$$((\exists_x A)^{\mathbf{r}})^+\colon \forall_{x,u}\big(u\ \mathbf{r}\ A \to (\exists_x A)^{\mathbf{r}}(\mathrm{C}xu)\big).$$

As our second example we take A to be $x_1 =_{\mathbf{T}} x_2$. Recall that $=_{\mathbf{T}}$ is an inductively defined predicate requiring witnesses, and that it has an efq-clause. We want to compare $x_1 =_{\mathbf{T}} x_2$ with $\exists_w\, w\ \mathbf{r}\ (x_1 =_{\mathbf{T}} x_2)$, which is $\exists_w(=^{\mathbf{ef}}_{\mathbf{T}}(w) \land =^{\mathbf{r}}_{\mathbf{T}}(w, x_1, x_2))$. One direction

$$\forall_w\big(=^{\mathbf{ef}}_{\mathbf{T}}(w) \to =^{\mathbf{r}}_{\mathbf{T}}(w, x_1, x_2) \to x_1 =_{\mathbf{T}} x_2\big)$$

is proved using $(=^{\mathbf{ef}}_{\mathbf{T}})^-$, with $P(w) := \forall^{\mathsf{U}}_{x_1,x_2}(=^{\mathbf{r}}_{\mathbf{T}}(w,x_1,x_2) \to x_1 =_{\mathbf{T}} x_2)$ (exercise). For the other direction

$$x_1 =_{\mathbf{T}} x_2 \to \exists_w\big(=^{\mathbf{ef}}_{\mathbf{T}}(w) \land =^{\mathbf{r}}_{\mathbf{T}}(w,x_1,x_2)\big)$$

we use $=^-_{\mathbf{T}}$ (cf. 2.1.2), for

$$P(x_1,x_2) := \exists_w\big(=^{\mathbf{ef}}_{\mathbf{T}}(w) \land =^{\mathbf{r}}_{\mathbf{T}}(w,x_1,x_2)\big).$$

Since $P(0,0)$ clearly holds with $w := \mathrm{C}_1$, it suffices to show

$$\forall^{\mathsf{U}}_{f_1,f_2}\big(\forall_n(f_1 n =_{\mathbf{T}} f_2 n) \to \forall_n P(f_1 n, f_2 n) \to P(\mathrm{Sup} f_1, \mathrm{Sup} f_2)\big).$$

Assume f_1, f_2 and the premises. By (AC) we have an f such that

$$\forall_n =^{\mathbf{ef}}_{\mathbf{T}}(fn) \land \forall_n =^{\mathbf{r}}_{\mathbf{T}}(fn, f_1 n, f_2 n)$$

We can now take $w := \mathrm{C}_2 f$, since $=^{\mathbf{ef}}_{\mathbf{T}}(\mathrm{C}_2 f)$ and $=^{\mathbf{r}}_{\mathbf{T}}(\mathrm{C}_2 f, \mathrm{Sup} f_1, \mathrm{Sup} f_2)$ follow from $(=^{\mathbf{ef}}_{\mathbf{T}})^+_2$ and $(=^{\mathbf{r}}_{\mathbf{T}})^+_2$, respectively.

THEOREM (Characterization; cf. Troelstra (1973, 3.4.8)).

$$\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ} \vdash A \leftrightarrow \exists_x\, x\ \mathbf{r}\ A.$$

PROOF. Induction on A, along the inductive definition of formulas, predicates and constructor formulas (or clauses) in 2.1.1. The case of an inductively defined predicate is similar to the examples above. *Case* $A \to B$.

$$\begin{aligned}
(A \to B) &\leftrightarrow (\exists_x\, x\ \mathbf{r}\ A \to \exists_z\, z\ \mathbf{r}\ B) && \text{by IH}\\
&\leftrightarrow \forall_x(x\ \mathbf{r}\ A \to \exists_z\, z\ \mathbf{r}\ B)\\
&\leftrightarrow \forall_x \exists_z(x\ \mathbf{r}\ A \to z\ \mathbf{r}\ B) && \text{by (IP)}\\
&\leftrightarrow \exists_f \forall_x(x\ \mathbf{r}\ A \to f(x)\ \mathbf{r}\ B) && \text{by (AC)}\\
&\leftrightarrow \exists_f f\ \mathbf{r}\ (A \to B).
\end{aligned}$$

Case $\forall_x A$.

$$\begin{aligned}
\forall_x A &\leftrightarrow \forall_x \exists_z\, z\ \mathbf{r}\ A && \text{by IH}\\
&\leftrightarrow \exists_f \forall_x\, fx\ \mathbf{r}\ A && \text{by (AC)}\\
&\leftrightarrow \exists_f f\ \mathbf{r}\ \forall_x A.
\end{aligned}$$

Case $A \to^{\mathsf{U}} B$.

$$\begin{aligned}
(A \to^{\mathsf{U}} B) &\leftrightarrow (A \to^{\mathsf{U}} \exists_z\, z\ \mathbf{r}\ B) && \text{by IH}\\
&\leftrightarrow \exists_z(A \to^{\mathsf{U}} z\ \mathbf{r}\ B) && \text{by (IP)}\\
&\leftrightarrow \exists_z\, z\ \mathbf{r}\ (A \to^{\mathsf{U}} B).
\end{aligned}$$

Case $\forall^{\mathsf{U}}_x A$.

$$\forall^{\mathsf{U}}_x A \leftrightarrow \forall^{\mathsf{U}}_x \exists_z\, z\ \mathbf{r}\ A \quad \text{by IH}$$

$$\leftrightarrow \exists_z \forall_x^{\mathsf{U}}\, z\ \mathbf{r}\ A \quad \text{by (IQ)}$$
$$\leftrightarrow \exists_z\, z\ \mathbf{r}\ \forall_x^{\mathsf{U}} A.$$

This concludes the proof. □

2.4. Soundness

We prove that every theorem in E-ID$^\omega$+AC+IP+IQ+Ax$_\varepsilon$ has a realizer. Here (Ax$_\varepsilon$) is an arbitrary set of Harrop formulas viewed as axioms.

2.4.1. Soundness for induction and general induction. We show that general recursion provides a realizer for general induction. For structural induction the argument is similar (and simpler). Recall that according to (1.14) general induction is the scheme

$$\forall_{\mu,x}\big(\mathrm{Prog}_x^\mu A(x) \to A(x)\big)$$

where $\mathrm{Prog}_x^\mu A(x)$ expresses "progressiveness" w.r.t. the measure function μ and the ordering $<$:

$$\mathrm{Prog}_x^\mu A(x) := \forall_x\big(\forall_{y;\mu y<\mu x} A(y) \to A(x)\big).$$

Lemma (Realizability Interpretation of General Induction).

$$\mathcal{F}\ \mathbf{r}\ \forall_{\mu,x}\big(\mathrm{Prog}_x^\mu A(x) \to A(x)\big).$$

Proof. We must show

$$\forall_{\mu,x,G}\big(G\ \mathbf{r}\ \forall_x(\forall_{y;\mu y<\mu x} A(y) \to A(x)) \to \mathcal{F}\mu xG\ \mathbf{r}\ A(x)\big).$$

Fix μ, x, G and assume the premise, which unfolds into

$$\forall_{x,f}\big((\forall_{y;\mu y<\mu x} fy\ \mathbf{r}\ A(y)) \to Gxf\ \mathbf{r}\ A(x)\big). \tag{2.7}$$

We have to show $\mathcal{F}\mu xG\ \mathbf{r}\ A(x)$. To this end we use an instance of general induction with the formula $\mathcal{F}\mu xG\ \mathbf{r}\ A(x)$, that is

$$\forall_x\big(\forall_{y;\mu y<\mu x}\mathcal{F}\mu yG\ \mathbf{r}\ A(y) \to \mathcal{F}\mu xG\ \mathbf{r}\ A(x)\big) \to \mathcal{F}\mu xG\ \mathbf{r}\ A(x).$$

It suffices to prove the premise. Fix x and assume $\forall_{y;\mu y<\mu x}\mathcal{F}\mu yG\ \mathbf{r}\ A(y)$. We must show $\mathcal{F}\mu xG\ \mathbf{r}\ A(x)$. Recall that by definition (1.12)

$$\mathcal{F}\mu xG = Gxf_0 \quad \text{with } f_0 := \lambda_y[\mathbf{if}\ \mu y < \mu x\ \mathbf{then}\ \mathcal{F}\mu yG\ \mathbf{else}\ \varepsilon].$$

Hence we can apply (2.7) to x, f_0, and it remains to show

$$\forall_{y;\mu y<\mu x} f_0 y\ \mathbf{r}\ A(y).$$

Fix y with $\mu y < \mu x$. Then $f_0 y = \mathcal{F}\mu yG$, and by the last assumption we have $\mathcal{F}\mu yG\ \mathbf{r}\ A(y)$. □

2.4.2. Soundness for introduction and elimination axioms. We first treat inductively defined predicates requiring witnesses. Recall the introduction axioms (2.3) and the (strengthened) elimination axiom (2.4) for $I^{\mathbf{r}}_j$.

By $K^{\mathbf{r}}(\vec{I^{\mathbf{r}}})$ we clearly have $\mathrm{C}\ \mathbf{r}\ (I_j)^+_i$.

LEMMA. $\mathcal{R}_j\ \mathbf{r}\ I^-_j$.

PROOF. For to prove

$$\mathcal{R}_j\ \mathbf{r}\ \forall^{\mathsf{U}}_{\vec{x}}\big(I_j(\vec{x}\,) \to \big(K_i(\vec{I},\vec{P})\big)_{i<k} \to P_j(\vec{x}\,)\big),$$

let $\vec{x}, w$ be given and assume $w\ \mathbf{r}\ I_j(\vec{x}\,)$. Let further $w_0,\dots,w_{k-1}$ be such that $w_i\ \mathbf{r}\ K_i(\vec{I},\vec{P})$, i.e.,

$$\begin{aligned}\breve{\forall}_{\vec{x}}\forall_{\vec{u},\vec{f},\vec{g}}\big(\vec{u}\ \mathbf{r}\ \vec{A} \to \big(\forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to f_\nu\vec{y}_\nu\vec{v}_\nu\ \mathbf{r}\ I_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to\\ \big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to g_\nu\vec{y}_\nu\vec{v}_\nu\ \mathbf{r}\ P_{j_\nu}(\vec{s}_\nu))\big)_{\nu<n} \to\\ w_i\vec{x}\vec{u}\vec{f}\vec{g}\ \mathbf{r}\ P_j(\vec{t}\,)\big).\end{aligned} \tag{2.8}$$

Our goal is

$$\mathcal{R}_j w\vec{w}\ \mathbf{r}\ P_j(\vec{x}\,) =: Q_j(w,\vec{x}\,).$$

We use the (strengthened) elimination axiom for $I^{\mathbf{r}}_j$ with $Q_j(w,\vec{x}\,)$, i.e.,

$$\forall_w\forall^{\mathsf{U}}_{\vec{x}}\big(I^{\mathbf{r}}_j(w,\vec{x}\,) \to \big(K^{\mathbf{r}}_i(\vec{I^{\mathbf{r}}},\vec{Q})\big)_{i<k} \to Q_j(w,\vec{x}\,)\big).$$

Hence it suffices to prove $K^{\mathbf{r}}(\vec{I^{\mathbf{r}}},\vec{Q})$ for every constructor formula K, i.e.,

$$\begin{aligned}\breve{\forall}_{\vec{x}}\forall_{\vec{u},\vec{f}}\big(\vec{u}\ \mathbf{r}\ \vec{A} \to \big(\forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n} \to\\ \big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu \to Q_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n} \to\\ Q_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)\big).\end{aligned} \tag{2.9}$$

So assume $\vec{x},\vec{u},\vec{f}$ and the premises of (2.9). We show $Q_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)$, i.e.,

$$\mathcal{R}_j(\mathrm{C}\vec{x}\vec{u}\vec{f})\vec{w}\ \mathbf{r}\ P_j(\vec{t}\,).$$

Since $\mathrm{C} = \mathrm{C}_i$, by the conversion rules for $\mathcal{R}$ (cf. 1.2.5) this is the same as

$$w_i\vec{x}\vec{u}\vec{f}\big(\lambda_{\vec{y}_\nu,\vec{v}_\nu}\mathcal{R}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu)\vec{w}\big)_{\nu<n}\ \mathbf{r}\ P_j(\vec{t}\,).$$

To this end we use (2.8) with $\vec{x},\vec{u},\vec{f},(\lambda_{\vec{y}_\nu,\vec{v}_\nu}\mathcal{R}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu)\vec{w})_{\nu<n}$. Its conclusion is what we want, and its premises follow from the premises of (2.9). □

We now treat inductively defined predicates not requiring witnesses. Realizability of the introduction axioms follows from the very same axioms, since all the formulas involved are negative. For a uniform one-clause inductively defined predicates the elimination axiom is realized by the identity.

For $I^{\mathbf{r}}$ realizability of the elimination axiom by the recursion operator can be shown as above:

LEMMA. $\mathcal{R}_j\ \mathbf{r}\ (I^{\mathbf{r}}_j)^-$.

PROOF. For to prove

$$\mathcal{R}_j\ \mathbf{r}\ \forall_w\forall^{\mathsf{U}}_{\vec{x}}\big(I^{\mathbf{r}}_j(w,\vec{x}\,)\to\big(K^{\mathbf{r}}_i(\vec{I}^{\mathbf{r}},\vec{P})\big)_{i<k}\to P_j(w,\vec{x}\,)\big),$$

let $w,\vec{x}$ be given and assume $I^{\mathbf{r}}_j(w,\vec{x}\,)$. Let further $w_0,\dots,w_{k-1}$ be such that $w_i\ \mathbf{r}\ K^{\mathbf{r}}_i(\vec{I}^{\mathbf{r}},\vec{P})$, i.e.,

$$\begin{aligned}&\breve{\forall}_{\vec{x}}\forall_{\vec{u},\vec{f},\vec{g}}\big(\vec{u}\ \mathbf{r}\ \vec{A}\to\\&\qquad\big(\forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu\to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n}\to\\&\qquad\big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu\to g_\nu\vec{y}_\nu\vec{v}_\nu\ \mathbf{r}\ P_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n}\to\\&\qquad w_i\vec{x}\vec{u}\vec{f}\vec{g}\ \mathbf{r}\ P_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)\big).\end{aligned}\tag{2.10}$$

Our goal is

$$\mathcal{R}_jw\vec{w}\ \mathbf{r}\ P_j(w,\vec{x}\,)=:Q_j(w,\vec{x}\,).$$

We use the (strengthened) elimination axiom for $I^{\mathbf{r}}_j$ with $Q_j(w,\vec{x}\,)$, i.e.,

$$\forall_{w,\vec{x}}\big(I^{\mathbf{r}}_j(w,\vec{x}\,)\to\big(K^{\mathbf{r}}_i(\vec{I}^{\mathbf{r}},\vec{Q})\big)_{i<k}\to Q_j(w,\vec{x}\,)\big).$$

Hence it suffices to prove $K^{\mathbf{r}}(\vec{I}^{\mathbf{r}},\vec{Q})$ for every constructor formula K, i.e.,

$$\begin{aligned}&\breve{\forall}_{\vec{x}}\forall_{\vec{u},\vec{f}}\big(\vec{u}\ \mathbf{r}\ \vec{A}\to\big(\forall_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu\to I^{\mathbf{r}}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n}\to\\&\qquad\big(\breve{\forall}_{\vec{y}_\nu,\vec{v}_\nu}(\vec{v}_\nu\ \mathbf{r}\ \vec{B}_\nu\to Q_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu,\vec{s}_\nu))\big)_{\nu<n}\to\\&\qquad Q_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)\big).\end{aligned}\tag{2.11}$$

So assume $\vec{x},\vec{u},\vec{f}$ and the premises of (2.11). We show $Q_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,)$, i.e.,

$$\mathcal{R}_j(\mathrm{C}\vec{x}\vec{u}\vec{f}\,)\vec{w}\ \mathbf{r}\ P_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,).$$

Since $\mathrm{C}=\mathrm{C}_i$, by the conversion rules for $\mathcal{R}$ (cf. 1.2.5) this is the same as

$$w_i\vec{x}\vec{u}\vec{f}\big(\lambda_{\vec{y}_\nu,\vec{v}_\nu}\mathcal{R}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu)\vec{w}\big)_{\nu<n}\ \mathbf{r}\ P_j(\mathrm{C}\vec{x}\vec{u}\vec{f},\vec{t}\,).$$

To this end we use (2.10) with $\vec{x},\vec{u},\vec{f},(\lambda_{\vec{y}_\nu,\vec{v}_\nu}\mathcal{R}_{j_\nu}(f_\nu\vec{y}_\nu\vec{v}_\nu)\vec{w})_{\nu<n}$. Its conclusion is what we want, and its premises follow from the premises of (2.11). □

For $I^{\mathbf{ef}}$ the elimination axiom (2.6) again is realized by a recursion operator; this can be proved as above, but somewhat simpler.

2.4.3. Soundness theorem. We work in $\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ}$.

THEOREM (Soundness). *Let M be a derivation of A from assumptions $u_i\colon C_i$ $(i < n)$. Then we can find a derivation $\sigma(M)$ of $[\![M]\!]\ \mathbf{r}\ A$ from assumptions $\bar{u}_i\colon x_{u_i}\ \mathbf{r}\ C_i$ for a non-uniform u_i (i.e., $x_{u_i} \in \mathrm{FV}([\![M]\!])$), and $\bar{u}_i\colon C_i$ for the other ones.*

PROOF. Induction on M. *Case* $u\colon A$. Let $\sigma(u) := \bar{u}$.

Case $c\colon A$, c an axiom. These cases have been treated above.

Case $\lambda_{u^A} M^B$. We must find a derivation $\sigma(\lambda_u M)$ of

$$[\![\lambda_u M]\!]\ \mathbf{r}\ (A \to B), \quad \text{which is} \quad \forall_x(x\ \mathbf{r}\ A\ \to\ [\![\lambda_u M]\!]x\ \mathbf{r}\ B).$$

Recall that $[\![\lambda_u M]\!] = \lambda_{x_u}[\![M]\!]$.

Subcase $x_u \in \mathrm{FV}([\![M]\!])$. Renaming x into x_u, our goal is to find a derivation of

$$\forall_{x_u}(x_u\ \mathbf{r}\ A\ \to\ [\![M]\!]\ \mathbf{r}\ B),$$

since we identify terms with the same β-normal form. By IH we have a derivation $\sigma(M)$ of $[\![M]\!]\ \mathbf{r}\ B$ from $\bar{u}\colon x_u\ \mathbf{r}\ A$. Hence we can define $\sigma(\lambda_u M) := \lambda_{x_u}\lambda_{\bar{u}}\sigma(M)$.

Subcase $x_u \notin \mathrm{FV}([\![M]\!])$. By IH we have a derivation $\sigma(M)$ of $[\![M]\!]\ \mathbf{r}\ B$ from $\bar{u}\colon A$. By the Characterization Theorem we obtain a derivation

$$K_A\colon \forall_x(x\ \mathbf{r}\ A \to A).$$

Then we have a derivation

$$\lambda_x\lambda_v\big((\lambda_u\sigma(M))(K_A x v^{x\mathbf{r}A})\big)\colon \forall_x(x\ \mathbf{r}\ A\ \to\ [\![M]\!]\ \mathbf{r}\ B).$$

The claim follows from $[\![M]\!] = (\lambda_{x_u}[\![M]\!])x$, since $x_u \notin \mathrm{FV}([\![M]\!])$.

Case $M^{A\to B}N^A$. We must find a derivation $\sigma(MN)$ of $[\![MN]\!]\ \mathbf{r}\ B$. Recall that $[\![MN]\!] = [\![M]\!][\![N]\!]$. By IH we have derivations $\sigma(M)$ of

$$[\![M]\!]\ \mathbf{r}\ (A \to B), \quad \text{which is} \quad \forall_x(x\ \mathbf{r}\ A\ \to\ [\![M]\!]x\ \mathbf{r}\ B)$$

and $\sigma(N)$ of $[\![N]\!]\ \mathbf{r}\ A$; hence we can define $\sigma(MN) := \sigma(M)[\![N]\!]\sigma(N)$.

Case $\lambda_x M^A$. We must find a derivation $\sigma(\lambda_x M)$ of $[\![\lambda_x M]\!]\ \mathbf{r}\ \forall_x A$. By definition $[\![\lambda_x M]\!] = \lambda_x[\![M]\!]$. Recall that

$$\lambda_x[\![M]\!]\ \mathbf{r}\ \forall_x A, \quad \text{which is} \quad \forall_x(\lambda_x[\![M]\!])x\ \mathbf{r}\ A.$$

Since we identify terms with the same β-normal form, by IH we can define $\sigma(\lambda_x M) := \lambda_x\sigma(M)$. It is easy to see that the variable condition is satisfied.

Case $M^{\forall_x A(x)}t$. We must find a derivation $\sigma(Mt)$ of $[\![Mt]\!]\ \mathbf{r}\ A(t)$. By definition $[\![Mt]\!] = [\![M]\!]t$, and by IH we have a derivation $\sigma(M)$ of

$$[\![M]\!]\ \mathbf{r}\ \forall_x A(x), \quad \text{which is} \quad \forall_x([\![M]\!]x\ \mathbf{r}\ A(x)).$$

Hence we can define $\sigma(Mt) := \sigma(M)t$.

Case $\lambda^{\mathsf{U}}_{u^A} M^B$. We must find a derivation $\sigma(\lambda^{\mathsf{U}}_u M)$ of

$$[\![\lambda^{\mathsf{U}}_u M]\!]\ \mathbf{r}\ (A \to^{\mathsf{U}} B), \quad \text{which is} \quad A \ \to\ [\![M]\!]\ \mathbf{r}\ B.$$

Because of $x_u \notin \mathrm{FV}([\![M]\!])$, by IH we have $[\![M]\!]\ \mathbf{r}\ B$ from $\bar{u}\colon A$. Hence we can define $\sigma(\lambda^{\mathsf{U}}_u M) := \lambda_{\bar{u}}\sigma(M)$.

Case $M^{A\to^{\mathsf{U}}B} N^A$. We must find a derivation $\sigma(MN)$ of $[\![MN]\!]\ \mathbf{r}\ B$. Recall that $[\![MN]\!] = [\![M]\!]$. By IH we have derivations $\sigma(M)$ of

$$[\![M]\!]\ \mathbf{r}\ (A \to^{\mathsf{U}} B), \quad \text{which is} \quad A \to [\![M]\!]\ \mathbf{r}\ B$$

and $\sigma(N)$ of $[\![N]\!]\ \mathbf{r}\ A$. The Characterization Theorem gives a derivation $K_A[\![N]\!]\sigma(N)$ of A. Hence we can define $\sigma(MN) := \sigma(M)(K_A[\![N]\!]\sigma(N))$.

Case $\lambda^{\mathsf{U}}_x M^A$. We must find a derivation $\sigma(\lambda^{\mathsf{U}}_x M)$ of

$$[\![\lambda^{\mathsf{U}}_x M]\!]\ \mathbf{r}\ \forall^{\mathsf{U}}_x A, \quad \text{which is} \quad \forall_x\, [\![M]\!]\ \mathbf{r}\ A.$$

By IH we have a derivation of $[\![M]\!]\ \mathbf{r}\ A$. Let $\sigma(\lambda^{\mathsf{U}}_x M) := \lambda_x \sigma(M)$.

Case $M^{\forall^{\mathsf{U}}_x A(x)} t$. We must find a derivation $\sigma(Mt)$ of $[\![Mt]\!]\ \mathbf{r}\ A(t)$. By definition $[\![Mt]\!] = [\![M]\!]$, and by IH we have a derivation $\sigma(M)$ of

$$[\![M]\!]\ \mathbf{r}\ \forall^{\mathsf{U}}_x A(x), \quad \text{which is} \quad \forall_x\, [\![M]\!]\ \mathbf{r}\ A(x).$$

Hence we can define $\sigma(Mt) := \sigma(M)t$. □

As a corollary to the Soundness Theorem we obtain the following. Let M be a closed derivation (in $\mathrm{ID}^\omega + \mathrm{AC} + \mathrm{IP} + \mathrm{IQ}$) of $\forall_x \exists_y A$, with A an arbitrary formula. Then $[\![M]\!]\ \mathbf{r}\ \forall_x \exists_y A$, i.e., $\forall_x\, [\![M]\!]x\ \mathbf{r}\ \exists_y A$, which by definition is $\forall_x (\exists_y A)^{\mathbf{r}}([\![M]\!]x)$. Hence $[\![M]\!]x$ is a witness for $\exists_y A$.

CHAPTER 3

Complexity

We now focus much of the technical/logical work of the previous chapter onto theories with limited (more feasible) computational strength. The initial motivation is the surprising result of Bellantoni and Cook (1992) characterizing the polynomial-time functions by the primitive recursion schemes, but with a judicially placed semicolon first used by Simmons (1988), separating the variables into two kinds (or sorts). The first "normal" kind controls the length of recursions, and the second "safe" kind marks the places where substitutions are allowed. (Various alternative names have arisen for the two sorts of variables, which will play a fundamental role throughout this chapter, thus "normal"/"input"/"complete" and "safe"/"output"/"incomplete". The important distinction here is that complete and incomplete variables will not just be of ground type, but may be of arbitrary higher type.)

We aim at developing a basic version of arithmetic which incorporates this variable separation. This theory EA(;) or simply A(;) will have elementary recursive strength (hence the prefix E) and sub-elementary (polynomially bounded) strength when restricted to its Σ_1-inductive fragment. We first extend the Bellantoni and Cook variable separation to also incorporate higher types. This produces a two-sorted version T(;) of Gödel's T, which will give a functional interpretation for A(;). We then go a stage further in formulating a theory LA(;) all of whose provable recursions, not just the Σ_1-inductive fragment, are polynomially bounded; but to achieve this, an important additional aspect now comes into play. We need the logic to be linear (hence the prefix L) and the corresponding term system LT(;) to have a linearity restriction on higher type "incomplete" (or "safe") variables in order to ensure that the computational content remains polynomial-time computable.

Our goal is to develop the restricted versions T(;) and LT(;) of Gödel's T so that the following relationships hold between the theories and their corresponding functional interpretations:

$$\frac{\text{Arithmetic}}{\text{Gödel's T}} = \frac{\text{A(;)}}{\text{T(;)}} = \frac{\text{LA(;)}}{\text{LT(;)}}.$$

The leading intuition is of course that one should use the Curry-Howard correspondence between terms in lambda-calculus and derivations in arithmetic. However, in the two-sorted versions we are about to develop, care must be taken to arrive at flexible and easy-to-use systems which can be understood in their own right.

3.1. A Two-Sorted Variant T(;) of Gödel's T

We define a two-sorted variant T(;) of Gödel's T, by lifting the approach of Simmons (1988) and Bellantoni and Cook (1992) to higher types. It is shown that the functions definable in T(;) are exactly the elementary functions. The proof is based on the observation that β-normalization of terms of rank $\leq k$ has elementary complexity, and that the two-sortedness restriction allows to unfold $\mathcal{R}$ in a controlled way.

3.1.1. Higher order terms with input/output restrictions. We shall work with two forms of arrow types and abstraction terms:

$$\begin{cases} \mathbf{N} \to \sigma \\ \lambda_n r \end{cases} \quad \text{as well as} \quad \begin{cases} \rho \multimap \sigma \\ \lambda_z r \end{cases}$$

and a corresponding syntactic distinction between input and output (typed) variables. Formally we proceed as follows. The *types* are

$$\rho, \sigma, \tau ::= \mathbf{N} \mid \mathbf{N} \to \rho \mid \rho \multimap \sigma,$$

and the *level* of a type is defined by

$$l(\mathbf{N}) := 0,$$
$$l(\rho \to \sigma) := l(\rho \multimap \sigma) := \max\{l(\sigma), 1 + l(\rho)\}.$$

Ground types are the types of level 0, and a *higher* type is any type of level at least 1. The $\to$-free types are called *safe*. In particular, every ground type is safe.

The *constants* are $0\colon \mathbf{N}$ and, for safe τ,

$$\mathrm{S}\colon \mathbf{N} \multimap \mathbf{N},$$
$$\mathcal{C}_\tau\colon \mathbf{N} \multimap \tau \multimap (\mathbf{N} \multimap \tau) \multimap \tau,$$
$$\mathcal{R}_\tau\colon \mathbf{N} \to \tau \multimap (\mathbf{N} \to \tau \multimap \tau) \multimap \tau.$$

The restriction to safe types τ is needed in the proof of the Normalization Theorem below. $\mathcal{C}_\tau$ is used for definition by cases, and $\mathcal{R}_\tau$ as a recursion operator. Generally, the typing of $\mathcal{R}_\tau$ with its peculiar choices of $\to$ and $\multimap$ deserves some comments. The first argument is the one that is recursed on and hence must be an input argument, so the type starts with $\mathbf{N} \to \cdots$. The third argument is the step argument; here we have used the type $\mathbf{N} \to \tau \multimap \tau$

rather than $\mathbf{N} \multimap \tau \multimap \tau$, because then we can construct a step term in the form $\lambda_{n,p}t$ rather than $\lambda_{a,p}t$, which is more flexible.

We shall work with typed variables. A variable of type $\mathbf{N}$ is either an *input* or an *output* variable, and variables of a type different from $\mathbf{N}$ are always output variables. We use the following conventions:

x	(input or output) variable;
z	output variable;
n, m	input variable of type $\mathbf{N}$;
a	output variable of type $\mathbf{N}$.

T(;)-*terms* (terms for short) are

$$r, s, t ::= x \mid C \mid (\lambda_n r)^{\mathbf{N}\to\sigma} \mid r^{\mathbf{N}\to\sigma} s^{\mathbf{N}} \ (s \text{ input term}) \mid (\lambda_z r)^{\rho\multimap\sigma} \mid r^{\rho\multimap\sigma} s^{\rho}.$$

We call s an *input term* if all its free variables are input variables. C is a constant.

The *conversion* rules are

$$\begin{aligned}
&\big(\lambda_{\vec{x},x} r(\vec{x}, x)\big)\vec{s}s &&\mapsto \big(\lambda_{\vec{x}} r(\vec{x}, s)\big)\vec{s},\\
&\mathcal{C}_\tau 0ts &&\mapsto t,\\
&\mathcal{C}_\tau(\mathrm{S}r)ts &&\mapsto sr,\\
&\mathcal{R}_\tau 0ts &&\mapsto t,\\
&\mathcal{R}_\tau(\mathrm{S}r)ts &&\mapsto sr(\mathcal{R}_\tau rts).
\end{aligned}$$

Note that converting $(\lambda_{\vec{x},x} r(\vec{x}, x))\vec{s}s$ into $(\lambda_{\vec{x}} r(\vec{x}, s))\vec{s}$ may be viewed as first converting $(\lambda_{\vec{x},x} r(\vec{x}, x))\vec{s}s$ "permutatively" into $(\lambda_{\vec{x}}((\lambda_x r(\vec{x}, x))s))\vec{s}$ and then performing the inner conversion to obtain $(\lambda_{\vec{x}} r(\vec{x}, s))\vec{s}$. One may ask why we take this conversion relation as our basis and not the more common $(\lambda_x r(x))s \mapsto r(s)$. The reason is that our notion of level is defined with the clause $l(\rho \to \sigma) := l(\rho \multimap \sigma) := \max\{l(\sigma), 1 + l(\rho)\}$ and not $:= \max\{l(\sigma), l(\rho)\} + 1$; this in turn seems reasonable since then the level of $\rho_1, \ldots, \rho_m \to \sigma$ (i.e., of $(\rho_1 \to (\rho_2 \to \ldots (\rho_m \to \sigma) \ldots))$) is 1 and hence independent of m. But given this definition of level, and given the need in some arguments (e.g., in the proof of the β-normalization theorem below) to perform conversions of highest level first, we must be able to convert $(\lambda_{\vec{x},x} r(\vec{x}, x))\vec{s}s$ with $\vec{x}$ of a low and x of a high level into $(\lambda_{\vec{x}} r(\vec{x}, s))\vec{s}$.

Redexes are subterms shown on the left side of the conversion rules above. We write $r \to r'$ ($r \to^* r'$) if r can be reduced into r' by one (an arbitrary number of) conversion of a subterm. A term is in *normal form* if it does not contain a redex as a subterm.

A function f is called *definable in* T(;) if there is a closed T(;)-term $t_f\colon \mathbf{N} \twoheadrightarrow \dots \mathbf{N} \twoheadrightarrow \mathbf{N}$ ($\twoheadrightarrow \in \{\to, \multimap\}$) in T(;) denoting this function. Notice that it is always desirable to have more $\multimap$ in the type of t_f, because then there are less restrictions on its argument terms.

3.1.2. Examples. *Addition* can be defined by a term t_+ of type $\mathbf{N} \multimap \mathbf{N} \to \mathbf{N}$. The recursion equations are

$$a + 0 := a, \quad a + (\mathrm{S}n) := \mathrm{S}(a + n),$$

and the representing term is

$$t_+ := \lambda_{a,n}.\mathcal{R}_{\mathbf{N}} n a(\lambda_{n,p}.\mathrm{S}p).$$

The *predecessor* function P can be defined by a term t_P of type $\mathbf{N} \multimap \mathbf{N}$ if we use the cases operator $\mathcal{C}$:

$$t_P := \lambda_a.\mathcal{C}_{\mathbf{N}} a 0(\lambda_b b).$$

From the predecessor function we can define *modified subtraction* $\dot{-}$:

$$a \dot{-} 0 := a, \quad a \dot{-} (\mathrm{S}n) := P(a \dot{-} n)$$

by the term

$$t_{\dot{-}} := \lambda_{a,n}.\mathcal{R}_{\mathbf{N}} n a(\lambda_{n,p}.Pp).$$

If f is defined from g by *bounded summation* $f(\vec{n}, n) := \sum_{i<n} g(\vec{n}, i)$, i.e.,

$$f(\vec{n}, 0) := 0, \quad f(\vec{n}, \mathrm{S}n) := f(\vec{n}, n) + g(\vec{n}, \mathrm{S}n)$$

and we have a term t_g of type $\mathbf{N}^{(k+1)} \to \mathbf{N}$ defining g, then we can build a term t_f of type $\mathbf{N}^{(k+1)} \to \mathbf{N}$ defining f by

$$t_f := \lambda_{\vec{n},n}.\mathcal{R}_{\mathbf{N}} n 0(\lambda_{n,p}.p + (t_g \vec{n} n)).$$

We now show that in spite of our restrictions on the formation of types and terms we can define functions of exponential growth.

Probably the easiest function of exponential growth is $B(n, a) = a + 2^n$ of type $B\colon \mathbf{N} \to \mathbf{N} \multimap \mathbf{N}$, with the defining equations

$$B(0, a) = a + 1,$$
$$B(n + 1, a) = B(n, B(n, a)).$$

We formally define B as a term in T(;) by

$$B := \lambda_n\big(\mathcal{R}_{\mathbf{N}\multimap\mathbf{N}} n \mathrm{S}\big(\lambda_{m,p,a}(p^{\mathbf{N}\multimap\mathbf{N}}(pa))\big)\big).$$

From B we can define the exponential function $E := \lambda_n(Bn0)$ of type $E\colon \mathbf{N} \to \mathbf{N}$, and also iterated exponential functions like $\lambda_n(E(En))$.

Now consider iteration $I(n, f) = f^n$, with f a variable of type $\mathbf{N} \multimap \mathbf{N}$.

$$\begin{array}{rcl} I(0, f, a) := a, & & I(0, f) := \mathrm{id}, \\ I(n + 1, f, a) := I(n, f, f(a)), & \text{or} & I(n + 1, f) := I(n, f) \circ f. \end{array}$$

Formally, for every variable f of type $\mathbf{N} \multimap \mathbf{N}$ we have the term

$$I_f := \lambda_n\big(\mathcal{R}_{\mathbf{N}\multimap\mathbf{N}} n(\lambda_a a)\big(\lambda_{m,p,a}(p^{\mathbf{N}\multimap\mathbf{N}}(fa))\big)\big).$$

For the general definition we need the *pure safe types* ρ_k, defined by $\rho_0 := \mathbf{N}$ and $\rho_{k+1} := \rho_k \multimap \rho_k$. Then within T(;) we can define

$$Ina_k \dots a_0 := a_k^n a_{k-1} \dots a_0,$$

with a_k of type ρ_k. However, a definition $F_0 a_k \dots a_0 := Ia_0 a_k \dots a_0$ is *not* possible: Ia_0 is not allowed.

We now discuss the necessity of the restrictions on the type of $\mathcal{R}$. We must require that the value type is a safe type, for otherwise we could define

$$I_E := \lambda_n\big(\mathcal{R}_{\mathbf{N}\to\mathbf{N}} n(\lambda_m m)\big(\lambda_{n,p,m}(p^{\mathbf{N}\to\mathbf{N}}(Em))\big)\big),$$

and $I_E(n,m) = E^n(m)$, a function of superelementary growth.

We also need to require that the "previous"-variable is an output variable, because otherwise we could define

$$S := \lambda_n\big(\mathcal{R}_{\mathbf{N}} n 0\big(\lambda_{n,m}(Em)\big)\big) \quad \text{(superelementary)}.$$

Then $S(n) = E^n(0)$.

3.1.3. Normalization. The *size* (or *length*) $|r|$ of a term r is the number of occurrences of constructors, variables and constants in r: $|x| = |C| = 1$, $|\lambda_n r| = |\lambda_z r| = |r| + 1$, and $|rs| = |r| + |s| + 1$.

In this section, the distinction between input and output variables and our two type formers $\to$ and $\multimap$ plays no role.

We first deal with the (generalized) *β-conversion* rule above:

$$\big(\lambda_{\vec{x},x} r(\vec{x}, x)\big)\vec{s}s \mapsto \big(\lambda_{\vec{x}} r(\vec{x}, s)\big)\vec{s}.$$

β-redexes are instances of the left side of the β-conversion rule. A term is said to be in *β-normal form* if it does not contain a β-redex.

We want to show that every term reduces to a β-normal form. This can be seen easily if we follow a certain order in our conversions. To define this order we have to make use of the fact that all our terms have types.

A β-convertible term $\big(\lambda_{\vec{x},x} r(\vec{x}^{\vec{\rho}}, x^\rho)\big)\vec{s}s$ is also called a *cut* with *cut-type* ρ. By the level of a cut we mean the level of its cut-type. The *cut-rank* of a term r is the least number bigger than the levels of all cuts in r. Now let t be a term of cut-rank $k+1$. Pick a cut of the maximal level k in t, such that s does not contain another cut of level k. (E.g., pick the rightmost cut of level k.) Then it is easy to see that replacing the picked occurrence of $(\lambda_{\vec{x},x} r(\vec{x}^{\vec{\rho}}, x^\rho))\vec{s}s$ in t by $(\lambda_{\vec{x}} r(\vec{x}, s))\vec{s}$ reduces the number of cuts of the maximal level k in t by 1. Hence

THEOREM (β-Normalization). *We have an algorithm which reduces any given term into a β-normal form.*

We now want to give an estimate of the number of conversion steps our algorithm takes until it reaches the normal form. The key observation for this estimate is the obvious fact that replacing one occurrence of

$$\big(\lambda_{\vec{x},x} r(\vec{x},x)\big)\vec{s}s \quad \text{by} \quad \big(\lambda_{\vec{x}} r(\vec{x},s)\big)\vec{s}.$$

in a given term t at most squares the size of t.

An elementary bound $E_k(l)$ for the number of steps our algorithm takes to reduce the rank of a given term of size l by k can be derived inductively, as follows. Let $E_0(l) := 0$. To obtain $E_{k+1}(l)$, first note that by induction hypothesis it takes $\leq E_k(l)$ steps to reduce the rank by k. The size of the resulting term is $\leq l^{2^n}$ where $n := E_k(l)$ since any step (i.e., β-conversion) at most squares the size. Now to reduce the rank by one more, we convert – as described above – one by one all cuts of the present rank, where each such conversion does not produce new cuts of this rank. Therefore the number of additional steps is bounded by the size n. Hence the total number of steps to reduce the rank by $k+1$ is bounded by

$$E_k(l) + l^{2^{E_k(l)}} =: E_{k+1}(l).$$

THEOREM (Upper bound for the complexity of β-normalization). *The β-normalization algorithm given in the proof above takes at most $E_k(l)$ steps to reduce a given term of cut-rank k and size l to normal form, where*

$$E_0(l) := 0 \quad \textit{and} \quad E_{k+1}(l) := E_k(l) + l^{2^{E_k(l)}}.$$

We now show that we can also eliminate the recursion operator, and still have an elementary estimate on the time needed.

LEMMA ($\mathcal{R}$ Elimination). *Let $t(\vec{x}\,)$ be a β-normal term of safe type. There is an elementary function E_t such that: if $\vec{m}$ are safe type $\mathcal{R}$-free terms and the free variables of $t(\vec{m})$ are output variables of safe type, then in time $E_t(|\vec{m}|)$ (with $|\vec{m}| := \sum_i |m_i|$) one can compute an $\mathcal{R}$-free term $\mathrm{rf}(t;\vec{x};\vec{m})$ such that $t(\vec{m}) \to^* \mathrm{rf}(t;\vec{x};\vec{m})$.*

PROOF. Induction on $|t|$.

If $t(\vec{x}\,)$ has the form $\lambda_x u_1$, then x is an output variable and x, u_1 have safe type because t has safe type. If $t(\vec{x}\,)$ is of the form $D\vec{u}$ with D a variable or a constant different from $\mathcal{R}$, then each u_i is a safe type term. Here (in case D is a variable) we need that $\vec{x}$ and the free variables of $t(\vec{m})$ are of safe type.

In all of the preceding cases, the free variables of each $u_i(\vec{m})$ are output variables of safe type. Apply the IH to obtain $u_i^* := \mathrm{rf}(u_i;\vec{x};\vec{m})$. Let t^* be obtained from t by replacing each u_i by u_i^*. Then t^* is $\mathcal{R}$-free. The result is obtained in linear time from $\vec{u}^*$. This finishes the lemma in all of these cases.

The only remaining case is if t is an $\mathcal{R}$ clause. Then it is of the form $\mathcal{R}rus\vec{t}$, because the term has safe type. One obtains $\mathrm{rf}(r;\vec{x};\vec{m})$ in time $E_r(|\vec{m}|)$ by the IH. By assumption $t(\vec{m})$ has free output variables only. Hence $r(\vec{m})$ is closed, because the type of $\mathcal{R}$ requires $r(\vec{m})$ to be an input term. By β-normalization one obtains the number $N := \mathrm{nf}(\mathrm{rf}(r;\vec{x};\vec{m}))$ in a further elementary time, $E'_r(|\vec{m}|)$.

Now consider sn with a new variable n, and let s' be its β-normal form. Since s is β-normal, $|s'| \leq |s| + 1 < |t|$. Applying the IH to s' one obtains a monotone elementary bounding function E_{sn}. One computes all $s_i := \mathrm{rf}(s';\vec{x},n;\vec{m},i)$ $(i < N)$ in a total time of at most

$$\sum_{i<N} E_{sn}(|\vec{m}| + i) \leq E'_r(|\vec{m}|) \cdot E_{sn}(|\vec{m}| + E'_r(|\vec{m}|)).$$

Consider u, $\vec{t}$. The IH gives $\hat{u} := \mathrm{rf}(u;\vec{x};\vec{m})$ in time $E_u(|\vec{m}|)$, and all $\hat{t}_i := \mathrm{rf}(t_i;\vec{x};\vec{m})$ in time $\sum_i E_{t_i}(|\vec{m}|)$. These terms are also $\mathcal{R}$-free by IH.

Using additional time bounded by a polynomial P in the lengths of these computed values, one constructs the $\mathcal{R}$-free term

$$\mathrm{rf}(\mathcal{R}rus\vec{t};\vec{x};\vec{m}) := \big(s_{N-1}\ldots(s_1(s_0\hat{u}))\ldots\big)\vec{\hat{t}}.$$

Defining $E_t(l) := P(E_u(l) + \sum_i E_{t_i}(l) + E'_r(l) \cdot E_{sn}(l + E'_r(l)))$, the total time used in this case is at most $E_t(|\vec{m}|)$. □

Let the *$\mathcal{R}$-rank* of a term t be the least number bigger than the level of all value types τ of recursion operators $\mathcal{R}_\tau$ in t. By the *rank* of a term we mean the maximum of its cut-rank and its $\mathcal{R}$-rank. Combining last two lemmas gives the following.

Lemma. *For every k there is an elementary function N_k such that every* T(;)*-term t of rank $\leq k$ can be reduced in time $N_k(|t|)$ to $\beta\mathcal{R}$ normal form.*

It remains to remove the occurrences of the cases operator $\mathcal{C}$. We may assume that only $\mathcal{C}_{\mathbf{N}}$ occurs.

Lemma ($\mathcal{C}$ Elimination). *Let t be an $\mathcal{R}$-free closed β-normal term of ground type* $\mathbf{N}$. *Then in time linear in $|t|$ one can reduce t to a numeral.*

Proof. If the term does not contain $\mathcal{C}$ we are done. Otherwise remove all occurrences of $\mathcal{C}$, as follows. The term has the form $\mathrm{S}r$ or $\mathcal{C}rts$. Proceed with r and iterate until we reach $\mathcal{C}rts$ where r does not contain $\mathcal{C}$. Then r is 0 or $\mathrm{S}r_0$. In the first case, convert $\mathcal{C}0ts$ to t. In the second case, notice that s has the form $\lambda_a s_0(a)$. Convert $\mathcal{C}(\mathrm{S}r_0)t(\lambda_a s_0(a))$ first into $(\lambda_a s_0(a))r_0$ and then into $s_0(r_0)$. Each time we have removed one occurrence of $\mathcal{C}$. □

We can now combine our results and state the final theorem.

THEOREM (Normalization). *Let t be a closed* T(;)*-term of type* $\mathbf{N} \twoheadrightarrow \ldots \mathbf{N} \twoheadrightarrow \mathbf{N}$ *(*$\twoheadrightarrow \in \{\to, \multimap\}$*). Then t denotes an elementary function.*

PROOF. We produce an elementary function E_t such that for all numerals $\vec{n}$ with $t\vec{n}$ is of type $\mathbf{N}$ one can compute $\mathrm{nf}(t\vec{n})$ in time $E_t(|\vec{n}|)$. Let $\vec{x}$ be new variables such that $t\vec{x}$ is of type $\mathbf{N}$. The β normal form $\beta\text{-nf}(t\vec{x})$ of $t\vec{x}$ is computed in an amount of time that may be large, but it is still only a constant with respect to $\vec{n}$.

By $\mathcal{R}$ Elimination one reduces to an $\mathcal{R}$-free term $\mathrm{rf}(\beta\text{-nf}(t\vec{x}); \vec{x}; \vec{n})$ in time $F_t(|\vec{n}|)$ with F_t elementary. Since the running time bounds the size of the produced term, $|\mathrm{rf}(\beta\text{-nf}(t\vec{x}); \vec{x}; \vec{n})| \leq F_t(|\vec{n}|)$. By a further β-normalization one can compute

$$\beta\mathcal{R}\text{-nf}(t\vec{n}) = \beta\text{-nf}(\mathrm{rf}(\beta\text{-nf}(t\vec{x}); \vec{x}; \vec{n}))$$

in time elementary in $|\vec{n}|$. Finally in time linear in the result we can remove all occurrences of $\mathcal{C}$ and arrive at a numeral. □

3.1.4. Sufficiency. Conversely, it is not hard to see that every elementary function is definable in T(;). More precisely, we show that for every elementary function f there is a term t_f of type $\mathbf{N}^{(k)} \to \mathbf{N}$ defining f.

For the proof it is easiest to use the following characterization of elementary functions: The class $\mathcal{E}$ consists of those number theoretic functions which can be defined from the initial functions: constant 0, successor S, projections (onto the ith coordinate), addition $+$, modified subtraction $\dot{-}$, multiplication $\cdot$ and exponentiation 2^x, by applications of composition and bounded minimization.

Recall that bounded minimization

$$f(\vec{n}, m) = \mu_{k<m}(g(\vec{n}, k) = 0)$$

is definable from bounded summation and $\dot{-}$:

$$f(\vec{n}, m) = \sum_{i<m} \big(1 \dot{-} \sum_{k \leq i} (1 \dot{-} g(\vec{n}, k))\big).$$

Now the claim follows from the first examples in 3.1.2 above.

3.2. A Linear Two-Sorted Variant LT(;) of Gödel's T

We now add some linearity restrictions, which will allow us to characterize the polynomially computable functions as those definable in a certain fragment of Gödel's T. Recall that in the first example above of a recursion producing exponential growth, the definition of $B(n, a) = a + 2^n$, we had the defining term

$$B := \lambda_n\big(\mathcal{R}_{\mathbf{N} \multimap \mathbf{N}} n \mathrm{S}\big(\lambda_{m,p,a}(p^{\mathbf{N} \multimap \mathbf{N}}(pa))\big)\big)$$

with the higher type variable p for the "previous" value appearing twice in the step term. The linearity restriction will forbid this.

When discussing polynomial time, it is appropriate to work with a *binary* (rather than unary) representation of the natural numbers, with two successors $S_0(a) = 2a$ and $S_1(a) = 2a + 1$.

We essentially keep the definitions of types, safe types, input/output variables from 3.1.1. However, the term definition will be different: it now involves a linearity constraint. Moreover, the typing of the recursion operator $\mathcal{R}$ needs to be changed: its (higher type) step argument will be used many times, and hence we need a $\to$ after it. As a consequence, we now allow higher types as argument types for $\to$. Therefore we change the names of input/output variables into normal/safe variables.

3.2.1. Feasible computation with higher types. We shall work with two forms of arrow types and abstraction terms:

$$\begin{cases} \rho \to \sigma \\ \lambda_{\bar{x}^\rho} r \end{cases} \quad \text{as well as} \quad \begin{cases} \rho \multimap \sigma \\ \lambda_{x^\rho} r \end{cases}$$

and a corresponding syntactic distinction between normal and safe (typed) variables, $\bar{x}$ and x. The intuition is that a function of type $\rho \to \sigma$ may recurse on its argument (if it is of ground type) or use it many times (if it is of higher type), whereas a function of type $\rho \multimap \sigma$ is not allowed to recurse on its argument (if it is of ground type) or can use it only once (if it is of higher type). As is well known, we then need a corresponding distinction for product types: the ordinary product $\wedge$ for $\to$, and the tensor product $\otimes$ for the linear arrow $\multimap$. Formally we proceed as follows. The *types* are

$$\rho, \sigma, \tau ::= \mathbf{U} \mid \mathbf{B} \mid \mathbf{L}(\rho) \mid \rho \to \sigma \mid \rho \multimap \sigma \mid \rho \wedge \sigma \mid \rho \otimes \sigma,$$

and the *level* of a type is defined by

$$\begin{aligned} l(\mathbf{U}) &:= 0, \\ l(\mathbf{B}) &:= 0, \\ l(\mathbf{L}(\rho)) &:= l(\rho), \end{aligned} \qquad \begin{aligned} l(\rho \to \sigma) := l(\rho \multimap \sigma) &:= \max\{l(\sigma), 1 + l(\rho)\}, \\ l(\rho \wedge \sigma) := l(\rho \otimes \sigma) &:= \max\{l(\rho), l(\sigma)\}. \end{aligned}$$

Ground types are the types of level 0, and a *higher* type is any type of level at least 1. The $\to$-free types are also called *safe*. In particular, every ground type is safe.

The *constants* are $\mathbf{u}\colon \mathbf{U}$, $\mathtt{tt}, \mathtt{ff}\colon \mathbf{B}$, $\mathrm{nil}_\rho\colon \mathbf{L}(\rho)$ and, for safe ρ, τ,

$$\begin{aligned} &::_\rho\colon \rho \multimap \mathbf{L}(\rho) \multimap \mathbf{L}(\rho), \\ &\mathrm{if}_\tau\colon \mathbf{B} \multimap \tau \wedge \tau \multimap \tau, \\ &\mathcal{C}^\rho_\tau\colon \mathbf{L}(\rho) \multimap \tau \wedge (\rho \multimap \mathbf{L}(\rho) \multimap \tau) \multimap \tau, \\ &\mathcal{R}^\rho_\tau\colon \mathbf{L}(\rho) \to \tau \multimap (\rho \to \mathbf{L}(\rho) \to \tau \multimap \tau) \to \tau \quad (\rho \text{ ground}). \end{aligned}$$

The restriction to safe types τ is needed in the proof of the Normalization Theorem below (in 3.2.4). $\mathcal{C}^\rho_\tau$ is used for definition by cases (on the constructor form of a list), and $\mathcal{R}^\rho_\tau$ as a recursion operator. Note that a single recursion operator (over lists) is used here to cover both, numeric and word recursion.

The typing of $\mathcal{R}^\rho_\tau$ with its peculiar choices of $\to$ and $\multimap$ deserves some comments. The first argument is the one that is recursed on and hence must be normal, so the type starts with $\mathbf{L}(\rho) \to \cdots$. The third argument is for the step term, which is of a higher type and will be used many times (when the recursion operator is unfolded), so it must be normal as well. Hence we need a $\to$ after the step type. We will crucially need this typing when we prove (in the Sufficiency Lemma below) that the functions definable in LT(;) are closed under "safe recursion".

Further constants are, for safe ρ, σ, τ,

$$\begin{aligned}
&\otimes^+_{\rho\sigma} \ : \rho \multimap \sigma \multimap \rho \otimes \sigma,\\
&\otimes^-_{\rho\sigma\tau} : \rho \otimes \sigma \multimap (\rho \multimap \sigma \multimap \tau) \multimap \tau,\\
&\wedge^+_{\rho\sigma} \ : \rho \multimap \sigma \multimap \rho \wedge \sigma,\\
&\mathrm{fst}_{\rho\sigma} : \rho \wedge \sigma \multimap \rho, \quad \mathrm{snd}_{\rho\sigma} : \rho \wedge \sigma \multimap \sigma.
\end{aligned}$$

The restriction to safe types ρ, σ, τ again will be needed in the proof of the Normalization Theorem.

LT(;)-*terms* (terms for short) are built from these constants and typed variables $\bar{x}^\sigma$ (normal variables) and x^σ (safe variables) by introduction and elimination rules for the two type forms $\rho \to \sigma$ and $\rho \multimap \sigma$, i.e.,

$$\begin{aligned}
&\bar{x}^\rho \mid x^\rho \mid C^\rho \quad \text{(constant)} \mid\\
&(\lambda_{\bar{x}^\rho} r^\sigma)^{\rho\to\sigma} \mid (r^{\rho\to\sigma} s^\rho)^\sigma \quad (s \text{ "normal"}) \mid\\
&(\lambda_{x^\rho} r^\sigma)^{\rho\multimap\sigma} \mid (r^{\rho\multimap\sigma} s^\rho)^\sigma \quad \text{(higher type safe variables in } r, s \text{ distinct)},
\end{aligned}$$

where a term s is called *normal* if all its free variables are normal. By the restriction on safe variables in the formation of an application $r^{\rho\multimap\sigma} s$, every higher type safe variable can occur at most once in a given term.

The *conversion* rules are as expected: β-conversion (for normal and safe variables) plus

$$\begin{aligned}
&\mathrm{if}_\tau \mathtt{tt} s \mapsto \mathrm{fst}_{\tau\tau} s, \quad \mathrm{if}_\tau \mathtt{ff} s \mapsto \mathrm{snd}_{\tau\tau} s,\\
&\mathcal{C}^\rho_\tau \mathrm{nil}_\rho s \mapsto \mathrm{fst}_{\tau\sigma} s, \quad \mathcal{C}^\rho_\tau (r ::_\rho l) s \mapsto \mathrm{snd}_{\tau\sigma} s r l \quad (\sigma := \rho \multimap \mathbf{L}(\rho) \multimap \tau),\\
&\mathcal{R}^\rho_\tau \mathrm{nil}_\rho t s \mapsto t, \quad \mathcal{R}^\rho_\tau (r ::_\rho l) t s \mapsto s r l (\mathcal{R}^\rho_\tau l t s),\\
&\otimes^-_{\rho\sigma\tau} (\otimes^+_{\rho\sigma} r s) t \mapsto t r s,\\
&\mathrm{fst}_{\rho\sigma}(\wedge^+_{\rho\sigma} r s) \mapsto r, \quad \mathrm{snd}_{\rho\sigma}(\wedge^+_{\rho\sigma} r s) \mapsto s.
\end{aligned}$$

Redexes are subterms shown on the left side of the conversion rules above. We write $r \to r'$ ($r \to^* r'$) if r can be reduced into r' by one (an arbitrary number of) conversion of a subterm.

Note that projections w.r.t. $\rho \otimes \sigma$ can be defined easily: For a term t of type $\rho \otimes \sigma$ let

$$t0 := \otimes^-_{\rho\sigma\rho} t(\lambda_{x^\rho, y^\sigma} x) \quad \text{and} \quad t1 := \otimes^-_{\rho\sigma\sigma} t(\lambda_{x^\rho, y^\sigma} y).$$

Then clearly

$$(\otimes^+_{\rho\sigma} rs)0 = \otimes^-_{\rho\sigma\rho}(\otimes^+_{\rho\sigma} rs)(\lambda_{x^\rho, y^\sigma} x) \mapsto (\lambda_{x^\rho, y^\sigma} x)rs \to^* r,$$
$$(\otimes^+_{\rho\sigma} rs)1 = \otimes^-_{\rho\sigma\sigma}(\otimes^+_{\rho\sigma} rs)(\lambda_{x^\rho, y^\sigma} y) \mapsto (\lambda_{x^\rho, y^\sigma} y)rs \to^* s.$$

A function f is called *definable in* LT(;) if there is a closed LT(;)-term $r\colon \mathbf{W} \twoheadrightarrow \ldots \mathbf{W} \twoheadrightarrow \mathbf{W}$ ($\twoheadrightarrow \in \{\to, \multimap\}$) in LT(;) denoting this function.

3.2.2. Examples. We now look at some examples intended to explain how our restrictions on the formation of types and terms make it impossible obtain exponential growth. However, for definiteness we first have to say precisely what we mean by a *numeral*.

Terms of the form $r^\rho_1 ::_\rho (r^\rho_2 ::_\rho \ldots (r^\rho_n ::_\rho \mathrm{nil}_\rho)\ldots)$ are called *lists*. We abbreviate $\mathbf{N} := \mathbf{L}(\mathbf{U})$ and $\mathbf{W} := \mathbf{L}(\mathbf{B})$.

$$0 := \mathrm{nil}_{\mathbf{U}}, \qquad 1 := \mathrm{nil}_{\mathbf{B}},$$
$$\mathrm{S} := \lambda_l.\mathbf{u} :: l^{\mathbf{N}}, \qquad S_0 := \lambda_l.\mathtt{ff} :: l^{\mathbf{W}},$$
$$S_1 := \lambda_l.\mathtt{tt} :: l^{\mathbf{W}}.$$

Particular lists are $\mathrm{S}(\ldots(\mathrm{S}0)\ldots)$ and $S_{i_1}(\ldots(S_{i_n}1)\ldots)$. The former are called *unary numerals*, and the latter *binary numerals* (or *numerals of type* $\mathbf{W}$). We denote binary numerals by ν.

Two recursions. Consider

$$D(1) := S_0(1), \qquad E(1) := 1,$$
$$D(S_i(x)) := S_0(S_0(D(x))), \qquad E(S_i(x)) := D(E(x)).$$

The corresponding terms are

$$D := \lambda_{\bar{x}}.\mathcal{R}_{\mathbf{W}}\bar{x}(S_0 1)(\lambda_{\bar{z},\bar{l},p}.S_0(S_0 p)),$$
$$E := \lambda_{\bar{x}}.\mathcal{R}_{\mathbf{W}}\bar{x}1(\lambda_{\bar{z},\bar{l},p}.Dp).$$

Here D is legal, but E is not: the application Dp is not allowed.

Recursion with parameter substitution. Consider

$$E(1, y) := S_0(y), \qquad E(S_i(x), y) := E(x, E(x, y)),$$

or

$$E(1) := S_0, \qquad E(S_i(x)) := E(x) \circ E(x).$$

The corresponding term

$$\lambda_{\bar{x}}.\mathcal{R}_{\mathbf{W}\multimap\mathbf{W}}\bar{x}S_0(\lambda_{\bar{z},\bar{l},p,y}.p^{\mathbf{W}\multimap\mathbf{W}}(py))$$

does not satisfy the linearity condition: the higher type variable p occurs twice, and the typing of $\mathcal{R}$ requires p to be safe.

A different form of recursion with parameter substitution is

$$\begin{aligned} E(1,y) &:= y, \\ E(S_i(x),y) &:= E(x,D(y)), \end{aligned} \quad \text{or} \quad \begin{aligned} E(1) &:= \mathrm{id}, \\ E(S_i(x)) &:= E(x) \circ D. \end{aligned}$$

The corresponding term would be

$$\lambda_{\bar{x}}.\mathcal{R}_{\mathbf{W}\to\mathbf{W}}\bar{x}(\lambda_y y)(\lambda_{\bar{z},\bar{l},p,\bar{x}}.p^{\mathbf{W}\to\mathbf{W}}(D\bar{x})),$$

but it is not legal, since the result type is not safe.

Higher argument types: iteration. Consider

$$\begin{aligned} I(1,f,y) &:= y, \\ I(S_i(x),f,y) &:= f(I(x,f,y)), \end{aligned} \quad \text{or} \quad \begin{aligned} I(1,f) &:= \mathrm{id}, \\ I(S_i(x),f) &:= f \circ I(x,f) \end{aligned}$$

with the corresponding term

$$\begin{aligned} I_f &:= \lambda_{\bar{x}}.\mathcal{R}_{\mathbf{W}\multimap\mathbf{W}}\bar{x}(\lambda_y y)(\lambda_{\bar{z},\bar{l},p,y}.f(p^{\mathbf{W}\multimap\mathbf{W}}y)), \\ E &:= \lambda_x.I_D x 1. \end{aligned}$$

Here I_f is legal, but E is not: the type of D prohibits iteration, i.e., formation of I_D. – Note that in PV^ω (see Cook and Kapron (1990), Cook (1992)) I is *not* definable, for otherwise we could define $\lambda_z.IDz$.

A related phenomenon occurs in

$$\begin{aligned} E(1) &:= \mathrm{id}, \\ E(S_i(x)) &:= E(x) \circ D. \end{aligned}$$

Now the term is

$$E := \lambda_{\bar{x}}.\mathcal{R}_{\mathbf{W}\multimap\mathbf{W}}\bar{x}S_0(\lambda_{\bar{z},\bar{l},q^{\mathbf{W}\multimap\mathbf{W}}}.I_q(S_0(S_0 1))).$$

Again E is not legal, this time because the free parameter f in the step term of I_f is substituted with the *safe* variable q. This variable needs to be normal because of the typing of the recursion operator.

3.2.3. Polynomials. It is high time that we give some examples of what *can* de done in our term system. It is easy to define $\oplus\colon \mathbf{W} \to \mathbf{W} \multimap \mathbf{W}$ such that $x \oplus y$ concatenates $|x|$ bits onto y:

$$\begin{aligned} 1 \oplus y &= S_0 y, \\ (S_i x) \oplus y &= S_0(x \oplus y). \end{aligned}$$

The representing term is

$$\bar{x} \oplus y := \mathcal{R}_{\mathbf{W}\multimap\mathbf{W}}\bar{x}S_0(\lambda_{\bar{z},\bar{l},p,y}.S_0(p^{\mathbf{W}\multimap\mathbf{W}}y))y.$$

Similarly we define $\odot\colon \mathbf{W} \to \mathbf{W} \to \mathbf{W}$ such that $x \odot y$ has output length $|x| \cdot |y|$:

$$x \odot 1 = x,$$
$$x \odot (S_i y) = x \oplus (x \odot y).$$

The representing term is $\bar{x} \odot \bar{y} := \mathcal{R}_{\mathbf{W}} \bar{y} \bar{x} (\lambda_{\bar{z},\bar{l},p}.\bar{x} \oplus p)$.

Note that the typing $\oplus\colon \mathbf{W} \to \mathbf{W} \multimap \mathbf{W}$ is crucial: it allows using the safe variable p in the definition of $\odot$. If we try to go on and define exponentiation from multiplication $\odot$ just as $\odot$ was defined from $\oplus$, we find that we cannot go ahead, because of the different typing $\odot\colon \mathbf{W} \to \mathbf{W} \to \mathbf{W}$.

3.2.4. Normalization. A *dag* is a directed acyclic graph. A *parse dag* is a structure like a parse tree but admitting in-degree greater than one. For example, a parse dag for $\lambda_x r$ has a node containing λ_x and a pointer to a parse dag for r. A parse dag for (rs) has a node containing a pair of pointers, one to a parse dag for r and the other to a parse dag for s. Terminal nodes are labeled by constants and variables.

The *size* $|d|$ of a parse dag d is the number of nodes in it, but counting 3 for $\mathcal{C}_\tau$ nodes. Starting at any given node in the parse dag, one obtains a term by a depth-first traversal; it is the term *represented* by that node. We may refer to a node as if it were the term it represents.

A parse dag is *conformal* if (i) every node having in-degree greater than 1 is of ground type, and (ii) every maximal path to a bound variable x passes through the same binding λ_x node.

A parse dag is *h-affine* if every higher-type variable occurs at most once in the dag.

We adopt a model of computation over parse dags in which operations such as the following can be performed in unit time: creation of a node given its label and pointers to the sub-dags; deletion of a node; obtaining a pointer to one of the subsidiary nodes given a pointer to an interior node; conditional test on the type of node or on the constant or variable in the node. Concerning computation over terms (including numerals), we use the same model and identify each term with its parse tree. Although not all parse dags are conformal, every term is conformal (assuming a relabeling of bound variables).

A term is called *simple* if it contains no higher type normal variables. Obviously simple terms are closed under reductions, taking of subterms, and applications. Every simple term is h-affine, due to the linearity of safe higher-type variables.

LEMMA (Simplicity). *Let t be a ground type term whose free variables are of ground type. Then* $\mathrm{nf}(t)$ *contains no higher type normal variables.*

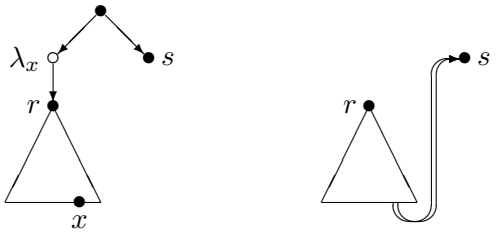

FIGURE 1. Redex $(\lambda_x r)s$ with r of ground type.

PROOF. Suppose a variable $\bar{x}^\sigma$ with $l(\sigma) > 0$ occurs in $\mathrm{nf}(t)$. It must be bound in a subterm $(\lambda_{\bar{x}^\sigma} r)^{\sigma\to\tau}$ of $\mathrm{nf}(t)$. By the well known subtype property of normal terms, the type $\sigma \to \tau$ either occurs positively in the type of $\mathrm{nf}(t)$, or else negatively in the type of one of the constants or free variables of $\mathrm{nf}(t)$. The former is impossible since t is of ground type, and the latter by inspection of the types of the constants. □

LEMMA (Sharing Normalization). *Let t be an $\mathcal{R}$-free simple term. Then a parse dag for $\mathrm{nf}(t)$, of size at most $|t|$, can be computed from t in time $O(|t|^2)$.*

PROOF. Under our model of computation, the input t is a parse tree. Since t is simple, it is an h-affine conformal parse dag of size at most $|t|$. If there are no nodes which represent a redex, then we are done. Otherwise, locate a node representing a redex; this takes time at most $O(|t|)$. We show how to update the dag in time $O(|t|)$ so that the size of the dag has strictly decreased and the redex has been eliminated, while preserving conformality. Thus, after at most $|t|$ iterations the resulting dag represents the normal-form term $\mathrm{nf}(t)$. The total time therefore is $O(|t|^2)$.

Assume first that the redex in t is $(\lambda_x r)s$ with x of ground type (see Figure 1); the argument is similar for a normal variable $\bar{x}$. Replace pointers to x in r by pointers to s. Since s does not contain x, no cycles are created. Delete the λ_x node and the root node for $(\lambda_x r)s$ which points to it. By conformality (i) no other node points to the λ_x node. Update any node which pointed to the deleted node for $(\lambda_x r)s$, so that it now points to the revised r subdag. This completes the β reduction on the dag (one may also delete the x nodes). Conformality (ii) gives that the updated dag represents a term t' such that $t \to t'$.

One can verify that the resulting parse dag is conformal and h-affine, with conformality (i) following from the fact that s has ground type.

If the redex in t is $(\lambda_x r)s$ with x of higher type (see Figure 2), then x occurs at most once in r because the parse dag is h-affine. By conformality

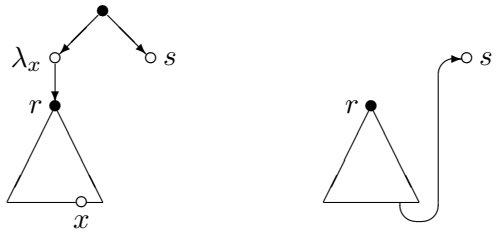

FIGURE 2. Redex $(\lambda_x r)s$ with r of higher type.

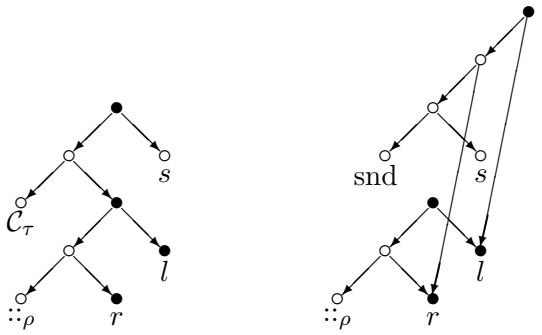

FIGURE 3. $\mathcal{C}_\tau(r ::_\rho l)s \mapsto \mathrm{snd}\, srl$ with ρ a ground type.

(i) there is at most one pointer to that occurrence of x. Update it to point to s instead, deleting the x node. As in the preceding case, delete the λ_x and the $(\lambda_x r)s$ node pointing to it, and update other nodes to point to the revised r. Again by conformality (ii) the updated dag represents t' such that $t \to t'$. Conformality and acyclicity are preserved, observing this time that conformality (i) follows because there is at most one pointer to s.

The remaining reductions are for the constant symbols.

Case $\mathrm{if}_\tau \mathtt{tt} s \mapsto \mathrm{fst}_{\tau\tau} s$. Easy; similar for $\mathtt{ff}$.

Case $\mathcal{C}_\tau \mathrm{nil}_\rho s \mapsto \mathrm{fst}\, s$. Easy.

Case $\mathcal{C}_\tau(r ::_\rho l)s \mapsto \mathrm{snd}\, srl$ with ρ a ground type (see Figure 3). Note that the new dag has one node more than the original one, but one $\mathcal{C}_\tau$-node less. Since we count the $\mathcal{C}_\tau$-nodes 3-fold, the total number of nodes decreases.

The remaining cases are treated in the Figures 4 – 7 below. □

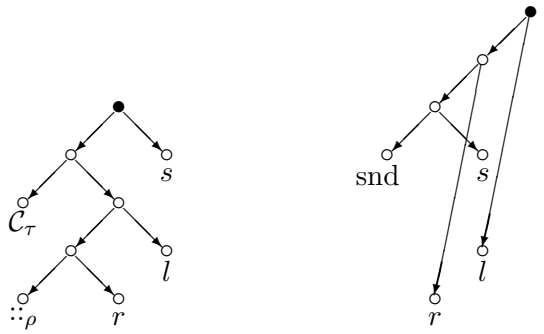

FIGURE 4. $\mathcal{C}_\tau(r ::_\rho l)s \mapsto \mathrm{snd}\, srl$ with ρ not a ground type.

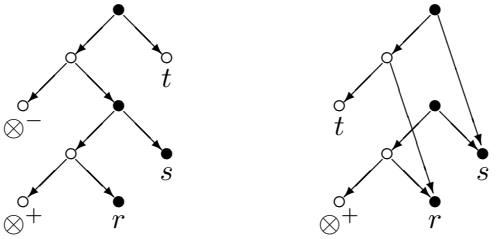

FIGURE 5. $\otimes^-_{\rho\sigma\tau}(\otimes^+_{\rho\sigma} rs)t \mapsto trs$ with $\rho \otimes \sigma$ a ground type.

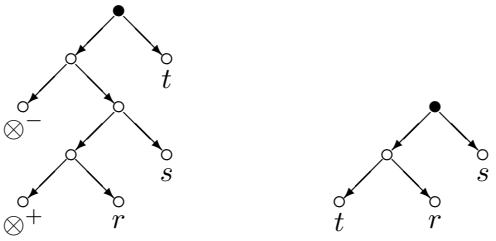

FIGURE 6. $\otimes^-_{\rho\sigma\tau}(\otimes^+_{\rho\sigma} rs)t \mapsto trs$ with $\rho \otimes \sigma$ not a ground type.

COROLLARY (Base Normalization). *Let t be a closed $\mathcal{R}$-free simple term of type* $\mathbf{W}$. *Then the binary numeral* $\mathrm{nf}(t)$ *can be computed from t in time* $O(|t|^2)$, *and* $|\mathrm{nf}(t)| \leq |t|$.

PROOF. By the Sharing Normalization Lemma we obtain a parse dag for $\mathrm{nf}(t)$ of size at most $|t|$, in time $O(|t|^2)$. Since $\mathrm{nf}(t)$ is a binary numeral, there

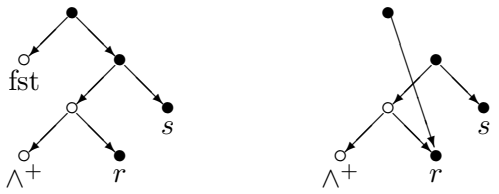

FIGURE 7. $\mathrm{fst}_{\rho\sigma}(\wedge^+_{\rho\sigma} rs) \mapsto r$ with $\rho \wedge \sigma$ a ground type.

is only one possible parse dag for it – namely, the parse tree of the numeral. This is identified with the numeral itself in our model of computation. □

LEMMA ($\mathcal{R}$ Elimination). *Let $t(\vec{x})$ be a simple term of safe type. There is a polynomial P_t such that: if $\vec{m}$ are safe type $\mathcal{R}$-free closed simple terms and the free variables of $t(\vec{m})$ are safe and of safe type, then in time $P_t(|\vec{m}|)$ one can compute an $\mathcal{R}$-free simple term $\mathrm{rf}(t;\vec{x};\vec{m})$ such that $t(\vec{m}) \to^* \mathrm{rf}(t;\vec{x};\vec{m})$.*

PROOF. By induction on $|t|$.

If $t(\vec{x})$ has the form $\lambda_z u_1$, then z is safe and z, u_1 have safe type because t has safe type. If $t(\vec{x})$ is of the form $D\vec{u}$ with D a variable or one of the constants $\mathbf{u}$, $\mathtt{tt}$, $\mathtt{ff}$, nil_ρ, $::_\rho$, if_τ, $\mathcal{C}_\tau$, $\otimes^+_{\rho\sigma}$, $\otimes^-_{\rho\sigma\tau}$, $\wedge^+_{\rho\sigma\tau}$, $\mathrm{fst}_{\rho\sigma}$ or $\mathrm{snd}_{\rho\sigma}$, then each u_i is a safe type term. Here (in case D is a variable x_i) we need that x_i is of safe type.

In all of the preceding cases, each $u_i(\vec{m})$ has only safe free variables of safe type. Apply the IH as required to simple terms u_i to obtain $u_i^* := \mathrm{rf}(u_i;\vec{x};\vec{m})$; so each u_i^* is $\mathcal{R}$-free. Let t^* be obtained from t by replacing each u_i by u_i^*. Then t^* is an $\mathcal{R}$-free simple term; here we need that $\vec{m}$ are closed, to avoid duplication of variables. The result is obtained in linear time from $\vec{u}^*$. This finishes the lemma in all of these cases.

If t is $(\lambda_y r)s\vec{u}$ with a safe variable y of ground type, apply the IH to yield $(r\vec{u})^* := \mathrm{rf}(r\vec{u};\vec{x};\vec{m})$ and $s^* := \mathrm{rf}(s;\vec{x};\vec{m})$. Redirect the pointers to y in $(r\vec{u})^*$ to point to s^* instead. If t is $(\lambda_{\bar{y}} r)s\vec{u}$ with a normal variable $\bar{y}$ of ground type, apply the IH to yield $s^* := \mathrm{rf}(s;\vec{x};\vec{m})$. Note that s^* is closed, since it is normal and the free variables of $s(\vec{m})$ are safe. Then apply the IH again to obtain $\mathrm{rf}(r\vec{u};\vec{x},\bar{y};\vec{m},s^*)$. The total time is at most $Q(|t|) + P_s(|\vec{m}|) + P_r(|\vec{m}| + P_s(|\vec{m}|))$, as it takes at most linear time to construct $r\vec{u}$ from $(\lambda_y r)s\vec{u}$.

If t is $(\lambda_y r(y))s\vec{u}$ with y of higher type, then y can occur at most once in r, because t is simple. Thus $|r(s)\vec{u}| < |(\lambda_y r)s\vec{u}|$. Apply the IH to obtain $\mathrm{rf}(r(s)\vec{u};\vec{x};\vec{m})$. Note that the time is bounded by $Q(|t|) + P_{r(s)\vec{u}}(|\vec{m}|)$ for a degree one polynomial q, since it takes at most linear time to make the at-most-one substitution in the parse tree.

The only remaining case is if the term is an $\mathcal{R}$ clause. Then it is of the form $\mathcal{R}lus\vec{t}$, because the term has safe type.

Since l is normal, all free variables of l are normal – they must be in $\vec{x}$ since free variables of $(\mathcal{R}lus\vec{t})[\vec{x} := \vec{m}]$ are safe. Then $l(\vec{m})$ is closed, implying $\mathrm{nf}(l(\vec{m}))$ is a list. One obtains $\mathrm{rf}(l; \vec{x}; \vec{m})$ in time $P_l(|\vec{m}|)$ by the IH. Then by Base Normalization one obtains the list $\hat{l} := \mathrm{nf}(\mathrm{rf}(l; \vec{x}; \vec{m}))$ in a further polynomial time. Let $\hat{l} = r_0 ::_\rho (r_1 ::_\rho \ldots (r_{N-1} ::_\rho \mathrm{nil}_\rho) \ldots)$ and let l_i, $i < N$ be obtained from $\hat{l}$ by omitting the initial elements $r_0, \ldots, r_i$. Thus all $\{ r_i, l_i \mid i < N \}$ are obtained in a total time bounded by $P'_l(|\vec{m}|)$ for a polynomial P'_l.

Now consider $s\bar{z}\bar{y}$ with new variables $\bar{z}^\rho$ and $\bar{y}^{\mathbf{L}(\rho)}$. Applying the IH to $s\bar{z}\bar{y}$ one obtains a monotone bounding polynomial $P_{s\bar{z}\bar{y}}$. One computes all $s_i := \mathrm{rf}(s\bar{z}\bar{y}; \vec{x}, \bar{z}, \bar{y}; \vec{m}, r_i, l_i)$ in a total time of at most

$$\sum_{i<N} P_{s\bar{z}\bar{y}}(|r_i| + |l_i| + |\vec{m}|) \leq P'_l(|\vec{m}|) \cdot P_{s\bar{z}\bar{y}}(2P'_l(|\vec{m}|) + |\vec{m}|).$$

Each s_i is $\mathcal{R}$-free by the IH. Furthermore, no s_i has a free safe variable: any such variable would also be free in s contradicting that s is normal.

Consider u, $\vec{t}$. The IH gives $\hat{u} := \mathrm{rf}(u; \vec{x}; \vec{m})$ in time $P_u(|\vec{m}|)$, and all $\hat{t}_i := \mathrm{rf}(t_i; \vec{x}; \vec{m})$ in time $\sum_i P_{t_i}(|\vec{m}|)$. These terms are also $\mathcal{R}$-free by IH. Clearly u and the t_i do not have any free (or bound) higher type safe variables in common. The same is true of $\hat{u}$ and all $\hat{t}_i$.

Using additional time bounded by a polynomial p in the lengths of these computed values, one constructs the $\mathcal{R}$-free term

$$\big(\lambda_x.s_0(s_1 \ldots (s_{N-1}x) \ldots)\big)\hat{u}\vec{\hat{t}}.$$

Defining $P_t(n) := P(\sum_i P_{t_i}(n) + P'_l(n) \cdot P_{s\bar{z}\bar{y}}(2P'_l(n) + n))$, the total time used in this case is at most $P_t(|\vec{m}|)$. The result is a term because $\hat{u}$ and the $\hat{t}_i$ are terms which do not have any free higher-type safe variable in common and because s_i does not have any free higher-type safe variables at all. □

Theorem (Normalization). *Let r be a closed* LT(;)*-term of type* $\mathbf{W} \twoheadrightarrow \ldots \mathbf{W} \twoheadrightarrow \mathbf{W}$ *($\twoheadrightarrow \in \{\to, \multimap\}$). Then r denotes a polytime function.*

Proof. One must find a polynomial Q_t such that for all $\mathcal{R}$-free simple closed terms $\vec{n}$ of types $\vec{\rho}$ one can compute $\mathrm{nf}(t\vec{n})$ in time $Q_t(|\vec{n}|)$. Let $\vec{x}$ be new variables of types $\vec{\rho}$. The normal form of $t\vec{x}$ is computed in an amount of time that may be large, but it is still only a constant with respect to $\vec{n}$.

$\mathrm{nf}(t\vec{x})$ is simple by the Simplicity Lemma. By $\mathcal{R}$ Elimination one reduces to an $\mathcal{R}$-free simple term $\mathrm{rf}(\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n})$ in time $P_t(|\vec{n}|)$. Since the running time bounds the size of the produced term, $|\mathrm{rf}(\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n})| \leq P_t(|\vec{n}|)$.

By Sharing Normalization one can compute

$$\mathrm{nf}(t\vec{n}) = \mathrm{nf}(\mathrm{rf}(\mathrm{nf}(t\vec{x}); \vec{x}; \vec{n}))$$

in time $O(P_t(|\vec{n}|)^2)$. Let Q_t be the polynomial referred to by the big-O notation. □

3.2.5. Sufficiency. The converse holds as well. The proof uses a characterization of the polynomial-time computable functions given by Bellantoni and Cook (1992). There the polynomial time computable functions are characterized by a function algebra B based on *safe recursion* and *safe composition*. There every function is written in the form $f(\vec{x}; \vec{y})$ where $\vec{x}; \vec{y}$ denotes a bookkeeping of those variables $\vec{x}$ that are used in a recursion defining f, and those variables $\vec{y}$ that are not recursed on. We proceed by induction on the definition of $f(x_1, \ldots, x_k; y_1, \ldots, y_l)$ in B, associating to f a closed term t_f of type $\mathbf{W}^{(k)} \to \mathbf{W}^{(l)} \multimap \mathbf{W}$, such that t denotes f.

The functions in B were defined over the non-negative integers rather than the positive ones, but this clearly is a minor point.

Lemma (Sufficiency). *Let f be a polynomial-time computable function. Then f is denoted by a closed* LT(;)*-term t_f.*

Proof. If f in B is an initial function 1, S_0, S_1, P, conditional C or projection $\pi_i^{m,n}$, then t_f is easily defined. For example, the predecessor function P of type $\mathbf{W} \multimap \mathbf{W}$ with the recursion equations $P(;1) := 1$ and $P(; S_i n) := n$ is denoted by $t_P := \lambda_n.\mathcal{C}_{\mathbf{W}} n s$ with $s := \wedge^{+}_{\mathbf{W},\mathbf{B}\multimap\mathbf{W}\multimap\mathbf{W}} 1(\lambda_{z,n} n)$.

If f is defined by safe composition, then

$$f(\vec{x}; \vec{y}) := g(r_1(\vec{x};), \ldots, r_m(\vec{x};); s_1(\vec{x}; \vec{y}), \ldots, s_n(\vec{x}; \vec{y})).$$

Using the IH to obtain t_g, $t_{\vec{r}}$ and $t_{\vec{s}}$, define

$$t_f := \lambda_{\vec{x},\vec{y}}.t_g(t_{r_1}\vec{\vec{x}}) \ldots (t_{r_m}\vec{\vec{x}})\,(t_{s_1}\vec{\vec{x}}\vec{y}) \ldots (t_{r_m}\vec{\vec{x}}\vec{y}).$$

Finally consider f defined by safe recursion,

$$\begin{aligned} f(1, \vec{x}; \vec{y}) &:= g(\vec{x}; \vec{y}), \\ f(S_i n, \vec{x}; \vec{y}) &:= h_i(n, \vec{x}; \vec{y}, f(n, \vec{x}; \vec{y})). \end{aligned}$$

One has t_g, t_{h_0} and t_{h_1} by IH. Let p be a variable of type $\tau := \mathbf{W}^{(\#(\vec{y}))} \multimap \mathbf{W}$; this is the safe type used in the recursion. Then define a step term by

$$s := \lambda_{\bar{x},\bar{l},p,\vec{y}}.\mathrm{if}_{\mathbf{W}\multimap\mathbf{W}}\bar{x}\big(\wedge^{+}(\lambda_z.t_{h_0}\bar{l}\vec{\vec{x}}\vec{y}z)(\lambda_z.t_{h_1}\bar{l}\vec{\vec{x}}\vec{y}z)\big)(p\vec{y}).$$

Note p is used only once. Let $t_f := \lambda_{\bar{n},\vec{\vec{x}}}.\mathcal{R}_\tau \bar{n}(t_g\vec{\vec{x}})s$. □

3.3. Towards Curry-Howard Extensions to Arithmetic

Curry and Howard observed that types correspond to formulas, and terms to proofs, when the logic is formulated in Gentzen's natural deduction calculus. Therefore it is tempting the transfer our restricted term systems to arithmetical theories, which then by construction have limited computational power: elementary arithmetic A(;) for T(;), and polynomial-time arithmetic LA(;) for LT(;). Initial attempts in this direction have already been carried out: by Ostrin and Wainer (2005) for the elementary case, and by Schwichtenberg (2006) for the polynomial-time case. There is also related work by Bellantoni and Hofmann (2002), which however uses a different approach based on the Hilbert calculus.

It remains to be seen whether such attempts to obtain feasible programs become feasible in practice. In any case, since such programs are automatically generated by extraction from checkable proofs, by their very construction the meet the highest possible security demands.

3.4. Notes

The elementary variant T(;) of Gödel's T developed in 3.1 has many relatives in the literature.

Beckmann and Weiermann (1996) characterize the elementary functions by means of a restriction of the combinatory logic version of Gödel's T. The restriction consists in allowing occurrences of the iteration operator only when immediately applied to a type $\mathbf{N}$ argument. For the proof they use an ordinal assignment due to Howard (1970) and Schütte (1977). The authors remark (on p. 477) that the methods of their paper can also be applied to a λ-formulation of T: the restriction on terms then consists in allowing only iterators of the form $\mathcal{I}_\rho t^{\mathbf{N}}$ and in disallowing λ-abstraction of the form $\lambda_x \dots \mathcal{I}_\rho t^{\mathbf{N}} \dots$ where x occurs in $t^{\mathbf{N}}$; however, no details are given. Moreover, our restrictions are slightly more liberal (input variables in t *can* be abstracted), and also the proof method is very different.

Aehlig and Johannsen (2005) characterize the elementary functions by means of a fragment of Girard's system F. They make essential use of the Church style representation of numbers in F. A somewhat different approach for characterizing the elementary functions based on a "predicative" setting has been developed by Leivant (1994).

Bibliography

K. Aehlig and J. Johannsen. An elementary fragment of second-order lambda calculus. *ACM Transactions on Computational Logic*, 6(2):468–480, Apr. 2005.

A. Beckmann and A. Weiermann. A term rewriting characterization of the polytime functions and related complexity classes. *Archive for Mathematical Logic*, 36:11–30, 1996.

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

S. Bellantoni and M. Hofmann. A new "feasible" arithmetic. *The Journal of Symbolic Logic*, 67(1):104–116, 2002.

U. Berger. Uniform Heyting Arithmetic. *Annals Pure Applied Logic*, 133: 125–148, 2005.

S. A. Cook. Computability and complexity of higher type functions. In Y. Moschovakis, editor, *Logic from Computer Science, Proceedings of a Workshop held November 13–17, 1989*, number 21 in MSRI Publications, pages 51–72. Springer Verlag, Berlin, Heidelberg, New York, 1992.

S. A. Cook and B. M. Kapron. Characterizations of the basic feasible functionals of finite type. In S. Buss and P. Scott, editors, *Feasible Mathematics*, pages 71–96. Birkhäuser, 1990.

M. V. Fairtlough and S. S. Wainer. Ordinal complexity of recursive definitions. *Information and Computation*, 99:123–153, 1992.

G. Gentzen. Die Widerspruchsfreiheit der reinen Zehlentheorie. *Mathematische Annalen*, 112:493–565, 1936.

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

G. H. Hardy. A theorem concerning the infinite cardinal numbers. *Quaterly Journal of Mathematics*, 35:87–94, 1904.

W. A. Howard. Assignment of ordinals to terms for primitive recursive functionals of finite type. In J. M. A. Kino and R. Vesley, editors, *Intuitionism and Proof Theory, Proceedings of the summer conference at Buffalo N.Y. 1968*, Studies in logic and the foundations of mathematics, pages 443–458. North–Holland, Amsterdam, 1970.

D. Leivant. Predicative recurrence in finite type. In A. Nerode and Y. Matiyasevich, editors, *Logical Foundations of Computer Science*, volume 813 of *LNCS*, pages 227–239, 1994.

P. Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216. North–Holland, Amsterdam, 1971.

G. Ostrin and S. S. Wainer. Elementary arithmetic. *Annals of Pure and Applied Logic*, 133:275–292, 2005.

C. Parsons. Ordinal recursion in partial systems of number theory (abstract). *Notices of the American Mathematical Society*, 13:857–858, 1966.

K. Schütte. *Proof Theory*. Springer Verlag, Berlin, Heidelberg, New York, 1977.

H. Schwichtenberg. An arithmetic for polynomial-time computation. *Theoretical Computer Science*, 357:202–214, 2006.

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

W. W. Tait. Nested recursion. *Math. Annalen*, 143:236–250, 1961.

A. S. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer Verlag, Berlin, Heidelberg, New York, 1973.

Index