Recursion Theory

Helmut Schwichtenberg

Notes for a lecture course, Wintersemester 2006/07.
Mathematisches Institut der Ludwig-Maximilians-Universität,
Theresienstraße 39, D-80333 München, Germany.
February 8, 2007.

Contents

CHAPTER 1

Computability

In this chapter we develop the basics of recursive function theory, or as it is more generally known, computability theory. Its history goes back to the seminal works of Turing, Kleene and others in the 1930's.

A computable function is one defined by a program whose operational semantics tell an idealized computer what to do to its storage locations as it proceeds deterministically from input to output, without any prior restrictions on storage space or computation time. We shall be concerned with various program-styles and the relationships between them, but the emphasis throughout will be on one underlying data-type, namely the natural numbers, since it is there that the most basic foundational connections between proof theory and computation are to be seen in their clearest light.

The two best-known models of machine computation are the Turing Machine and the (Unlimited) Register Machine of Shepherdson and Sturgis [1963]. We base our development on the latter since it affords the quickest route to the results we want to establish.

1.1. Register Machines

1.1.1. Programs. A *register machine* stores natural numbers in registers denoted u, v, w, x, y, z possibly with subscripts, and it responds step by step to a *program* consisting of an ordered list of basic instructions:

$$\begin{array}{l} I_0 \\ I_1 \\ \vdots \\ I_{k-1} \end{array}$$

Each instruction has one of the following three forms whose meanings are obvious:

Zero: $x := 0$,
Succ: $x := x + 1$,
Jump: $[\textbf{if } x = y \textbf{ then } I_m \textbf{ else } I_n]$.

The instructions are obeyed in order starting with I_0 except when a conditional jump instruction is encountered, in which case the next instruction will be either I_m or I_n according as the numerical contents of registers x and y are equal or not at that stage. The computation *terminates* when it runs out of instructions, that is when the next instruction called for is I_k. Thus if a program of length k contains a jump instruction as above then it must satisfy the condition $m, n \leq k$ and I_k means "halt". Notice of course that some programs do not terminate, for example the following one-liner:

$$[\textbf{if } x = x \textbf{ then } I_0 \textbf{ else } I_1]$$

1.1.2. Program constructs. We develop some shorthand for building up standard sorts of programs.

Transfer. "$x := y$" is the program

$$\begin{array}{l} x := 0 \\ [\textbf{if } x = y \textbf{ then } I_4 \textbf{ else } I_2] \\ x := x + 1 \\ [\textbf{if } x = x \textbf{ then } I_1 \textbf{ else } I_1], \end{array}$$

which copies the contents of register y into register x.

Predecessor. The program "$x := y \dot{-} 1$" copies the modified predecessor of y into x, and simultaneously copies y into z:

$$\begin{array}{l} x := 0 \\ z := 0 \\ [\textbf{if } x = y \textbf{ then } I_8 \textbf{ else } I_3] \\ z := z + 1 \\ [\textbf{if } z = y \textbf{ then } I_8 \textbf{ else } I_5] \\ z := z + 1 \\ x := x + 1 \\ [\textbf{if } z = y \textbf{ then } I_8 \textbf{ else } I_5]. \end{array}$$

Composition. "P **;** Q" is the program obtained by concatenating program P with program Q. However in order to ensure that jump instructions in Q of the form "$[\textbf{if } x = y \textbf{ then } I_m \textbf{ else } I_n]$" still operate properly within Q they need to be re-numbered by changing the addresses m, n to $k+m, k+n$ respectively where k is the length of program P. Thus the effect of this program is to do P until it halts (if ever) and then do Q.

Conditional. "**if** $x = y$ **then** P **else** Q **fi**" is the program

$$\begin{array}{l} [\textbf{if } x = y \textbf{ then } I_1 \textbf{ else } I_{k+2}] \\ \vdots P \\ [\textbf{if } x = x \textbf{ then } I_{k+2+l} \textbf{ else } I_2] \\ \vdots Q \end{array}$$

where k, l are the lengths of the programs P, Q respectively, and again their jump instructions must be appropriately renumbered by adding 1 to the addresses in P and $k + 2$ to the addresses in Q. Clearly if $x = y$ then program P is obeyed and the next jump instruction automatically bypasses Q and halts. If $x \neq y$ then program Q is performed.

For Loop. "**for** $i = 1 \ldots x$ **do** P **od**" is the program

$$\begin{array}{l} i := 0 \\ [\textbf{if } x = i \textbf{ then } I_{k+4} \textbf{ else } I_2] \\ i := i + 1 \\ \vdots P \\ [\textbf{if } x = i \textbf{ then } I_{k+4} \textbf{ else } I_2] \end{array}$$

where again, k is the length of program P and the jump instructions in P must be appropriately re-addressed by adding 3. The intention of this new program is that it should iterate the program P x times (do nothing if $x = 0$). This requires the restriction that the register x and the "local" counting-register i are not re-assigned new values inside P.

While Loop. "**while** $x \neq 0$ **do** P **od**" is the program

$$\begin{array}{l} y := 0 \\ [\textbf{if } x = y \textbf{ then } I_{k+3} \textbf{ else } I_2] \\ \vdots P \\ [\textbf{if } x = y \textbf{ then } I_{k+3} \textbf{ else } I_2] \end{array}$$

where again, k is the length of program P and the jump instructions in P must be re-addressed by adding 2. This program keeps on doing P until (if ever) the register x becomes 0; it requires the restriction that the auxiliary register y is not re-assigned new values inside P.

1.1.3. Register machine computable functions. A register machine program P may have certain distinguished "input registers" and "output registers". It may also use other "working registers" for scratchwork and these will initially be set to zero. We write $P(x_1, \ldots, x_k; y)$ to signify that program P has input registers $x_1, \ldots, x_k$ and one output register y, which are distinct.

DEFINITION. The program $P(x_1, \ldots, x_k; y)$ is said to *compute* the k-ary partial function $\varphi\colon \mathbb{N}^k \to \mathbb{N}$ if, starting with any numerical values $n_1, \ldots, n_k$ in the input registers, the program terminates with the number m in the output register if and only if $\varphi(n_1, \ldots, n_k)$ is defined with value m. In this case, the input registers hold their original values.

A function is *register machine computable* if there is some program which computes it.

Here are some examples.

Addition. "Add$(x, y; z)$" is the program

$$z := x \textbf{ ; for } i = 1, \ldots, y \textbf{ do } z := z + 1 \textbf{ od}$$

which adds the contents of registers x and y into register z.

Subtraction. "Subt$(x, y; z)$" is the program

$$z := x \textbf{ ; for } i = 1, \ldots, y \textbf{ do } w := z \dot{-} 1 \textbf{ ; } z := w \textbf{ od}$$

which computes the modified subtraction function $x \dot{-} y$.

Bounded Sum. If $P(x_1, \ldots, x_k, w; y)$ computes the $k+1$-ary function φ then the program $Q(x_1, \ldots, x_k, z; x)$:

$x := 0$ **;**
for $i = 1, \ldots, z$ **do** $w := i \dot{-} 1$ **;** $P(\vec{x}, w; y)$ **;** $v := x$ **;** Add$(v, y; x)$ **od**

computes the function

$$\psi(x_1, \ldots, x_k, z) = \sum_{w<z} \varphi(x_1, \ldots, x_k, w)$$

which will be undefined if for some $w < z$, $\varphi(x_1, \ldots, x_k, w)$ is undefined.

Multiplication. Deleting "$w := i \dot{-} 1$ **;** P" from the last example gives a program Mult$(z, y; x)$ which places the product of y and z into x.

Bounded Product. If in the bounded sum example, the instruction $x := x + 1$ is inserted immediately after $x := 0$, and if $\mathrm{Add}(v, y; x)$ is replaced by $\mathrm{Mult}(v, y; x)$, then the resulting program computes the function

$$\psi(x_1, \ldots, x_k, z) = \prod_{w<z} \varphi(x_1, \ldots, x_k, w)\,.$$

Composition. If $P_j(x_1, \ldots, x_k; y_j)$ computes φ_j for each $j = i, \ldots, m$ and if $P_0(y_1, \ldots, y_m; y_0)$ computes φ_0, then the program $Q(x_1, \ldots, x_k; y_0)$:

$$P_1(x_1, \ldots, x_k; y_1) \ ; \ \ldots \ ; \ P_m(x_1, \ldots, x_k; y_m) \ ; \ P_0(y_1, \ldots, y_m; y_0)$$

computes the function

$$\psi(x_1, \ldots, x_k) = \varphi_0(\varphi_1(x_1, \ldots, x_k), \ldots, \varphi_m(x_1, \ldots, x_k))$$

which will be undefined if any of the φ-subterms on the right hand side is undefined.

Unbounded Minimization. If $P(x_1, \ldots, x_k, y; z)$ computes φ then the program $Q(x_1, \ldots, x_k; z)$:

$y := 0$; $z := 0$; $z := z + 1$;
while $z \neq 0$ **do** $P(x_1, \ldots, x_k, y; z)$; $y := y + 1$ **od** ;
$z := y \dot{-} 1$

computes the function

$$\psi(x_1, \ldots, x_k) \ = \ \mu_y \left(\varphi(x_1, \ldots, x_k, y) = 0\right)$$

that is, the *least number* y such that $\varphi(x_1, \ldots, x_k, y')$ is defined for every $y' \leq y$ and $\varphi(x_1, \ldots, x_k, y) = 0$.

1.2. Elementary Functions

1.2.1. Definition and simple properties. The *elementary functions* of Kalmár (1943) are those number-theoretic functions which can be defined explicitly by compositional terms built up from variables and the constants $0, 1$ by repeated applications of addition $+$, modified subtraction $\dot{-}$, bounded sums and bounded products.

By omitting bounded products, one obtains the *subelementary* functions.

The examples in the previous section show that all elementary functions are computable and totally defined. Multiplication and exponentiation are elementary since

$$m \cdot n = \sum_{i<n} m \ \text{ and } \ m^n = \prod_{i<n} m$$

and hence by repeated composition, all exponential polynomials are elementary.

In addition the elementary functions are closed under

Definitions by Cases.

$$f(\vec{n}) = \begin{cases} g_0(\vec{n}) & \text{if } h(\vec{n}) = 0 \\ g_1(\vec{n}) & \text{otherwise} \end{cases}$$

since f can be defined from g_0, g_1 and h by

$$f(\vec{n}) = g_0(\vec{n}) \cdot (1 \dot{-} h(\vec{n})) + g_1(\vec{n}) \cdot (1 \dot{-} (1 \dot{-} h(\vec{n}))).$$

Bounded Minimization.

$$f(\vec{n}, m) = \mu_{k<m}(g(\vec{n}, k) = 0)$$

since f can be defined from g by

$$f(\vec{n}, m) = \sum_{i<m} \big(1 \dot{-} \sum_{k\leq i}(1 \dot{-} g(\vec{n}, k))\big).$$

Note: this definition gives value m if there is no $k < m$ such that $g(\vec{n}, k) = 0$. It shows that not only the elementary, but in fact the subelementary functions are closed under bounded minimization. Furthermore, we define $\mu_{k\leq m}\,(g(\vec{n}, k) = 0)$ as $\mu_{k<m+1}\,(g(\vec{n}, k) = 0)$. Another notational convention will be that we shall often replace the brackets in $\mu_{k<m}\,(g(\vec{n}, k) = 0)$ by a dot, thus: $\mu_{k<m}.\, g(\vec{n}, k) = 0$.

LEMMA.

(a) *For every elementary function* $f\colon \mathbb{N}^r \to \mathbb{N}$ *there is a number* k *such that for all* $\vec{n} = n_1, \ldots, n_r$,

$$f(\vec{n}) \;<\; 2_k(\max(\vec{n}))$$

where $2_0(m) = m$ *and* $2_{k+1}(m) = 2^{2_k(m)}$.

(b) *Hence the function* $n \mapsto 2_n(1)$ *is not elementary.*

PROOF. (a). By induction on the build-up of the compositional term defining f. The result clearly holds if f is any one of the base functions:

$$f(\vec{n}) = 0 \text{ or } 1 \text{ or } n_i \text{ or } n_i + n_j \text{ or } n_i \dot{-} n_j.$$

If f is defined from g by application of bounded sum or product:

$$f(\vec{n}, m) = \sum_{i<m} g(\vec{n}, i) \text{ or } \prod_{i<m} g(\vec{n}, i)$$

where $g(\vec{n}, i) < 2_k(\max(\vec{n}, i))$ then we have

$$f(\vec{n}, m) \leq 2_k(\max(\vec{n}, m))^m < 2_{k+2}(\max(\vec{n}, m))$$

(using $m^m < 2^{2^m}$). If f is defined from $g_0, g_1, \ldots, g_l$ by composition:

$$f(\vec{n}) = g_0(g_1(\vec{n}), \ldots, g_l(\vec{n}))$$

where for each $j \leq l$ we have $g_j(-) < 2_{k_j}(\max(-))$, then with $k = \max_j k_j$,

$$f(\vec{n}) < 2_k(2_k(\max(\vec{n}))) = 2_{2k}(\max(\vec{n}))$$

and this completes the first part.

(b). If $2_n(1)$ were an elementary function of n then by (a) there would be a positive k such that for all n,

$$2_n(1) < 2_k(n)$$

but then putting $n = 2_k(1)$ yields $2_{2_k(1)}(1) < 2_{2k}(1)$, a contradiction. □

1.2.2. Elementary relations. A relation R on $\mathbb{N}^k$ is said to be *elementary* if its characteristic function

$$c_R(\vec{n}) = \begin{cases} 1 & \text{if } R(\vec{n}) \\ 0 & \text{otherwise} \end{cases}$$

is elementary. In particular, the "equality" and "less than" relations are elementary since their characteristic functions can be defined as follows:

$$c_<(m,n) = 1 \dot{-} (1 \dot{-} (n \dot{-} m)) \ ; \ c_=(m,n) = 1 \dot{-} (c_<(m,n) + c_<(n,m))).$$

Furthermore if R is elementary then so is the function

$$f(\vec{n}, m) = \mu_{k<m} R(\vec{n}, k)$$

since $R(\vec{n}, k)$ is equivalent to $1 \dot{-} c_R(\vec{n}, k) = 0$.

LEMMA. *The elementary relations are closed under applications of propositional connectives and bounded quantifiers.*

PROOF. For example, the characteristic function of $\neg R$ is

$$1 \dot{-} c_R(\vec{n}) \ .$$

The characteristic function of $R_0 \wedge R_1$ is

$$c_{R_0}(\vec{n}) \cdot c_{R_1}(\vec{n}).$$

The characteristic function of $\forall_{i<m} R(\vec{n}, i)$ is

$$c_=(m, \mu_{i<m} c_R(\vec{n}, i) = 0). \qquad \square$$

EXAMPLES. The above closure properties enable us to show that many "natural" functions and relations of number theory are elementary; thus

$$\begin{aligned} \lfloor \frac{m}{n} \rfloor \qquad & = \mu_{k<m}(m < (k+1)n) \\ m \bmod n \ & = m \ \dot{-} \ \lfloor \frac{m}{n} \rfloor n \\ \mathrm{Prime}(m) & \leftrightarrow 1 < m \ \wedge \ \neg \exists_{n<m} (1 < n \wedge m \bmod n = 0) \\ p_n \qquad & = \mu_{m<2^{2^n}} \big(\mathrm{Prime}(m) \ \wedge \ n = \sum_{i<m} c_{\mathrm{Prime}}(i) \big) \end{aligned}$$

so $p_0, p_1, p_2, \ldots$ gives the enumeration of primes in increasing order. The estimate $p_n \leq 2^{2^n}$ for the nth prime p_n can be proved by induction on n: For $n = 0$ this is clear, and for $n \geq 1$ we obtain

$$p_n \leq p_0 p_1 \cdots p_{n-1} + 1 \leq 2^{2^0} 2^{2^1} \cdots 2^{2^{n-1}} + 1 = 2^{2^n - 1} + 1 < 2^{2^n}.$$

1.2.3. The class $\mathcal{E}$.

DEFINITION. 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.

The remarks above show immediately that the characteristic functions of the equality and less than relations lie in $\mathcal{E}$, and that (by the proof of the lemma) the relations in $\mathcal{E}$ are closed under propositional connectives and bounded quantifiers.

Furthermore the above examples show that all the functions in the class $\mathcal{E}$ are elementary. We now prove the converse, which will be useful later.

LEMMA. *There are "pairing functions" π, π_1, π_2 in $\mathcal{E}$ with the following properties:*

(a) π *maps* $\mathbb{N} \times \mathbb{N}$ *bijectively onto* $\mathbb{N}$,
(b) $\pi(a,b) < (a+b+1)^2$,
(c) $\pi_1(c), \pi_2(c) \leq c$,
(d) $\pi(\pi_1(c), \pi_2(c)) = c$,
(e) $\pi_1(\pi(a,b)) = a$,
(f) $\pi_2(\pi(a,b)) = b$.

PROOF. Enumerate the pairs of natural numbers as follows:

$$\begin{array}{cccccc} \vdots & & & & & \\ 10 & & & & & \\ 6 & \ldots & & & & \\ 3 & 7 & \ldots & & & \\ 1 & 4 & 8 & \ldots & & \\ 0 & 2 & 5 & 9 & \ldots & \end{array}$$

At position $(0,b)$ we clearly have the sum of the lengths of the preceeding diagonals, and on the next diagonal $a+b$ remains constant. Let $\pi(a,b)$ be the number written at position (a,b). Then we have

$$\pi(a,b) = \big(\sum_{i \leq a+b} i\big) + a = \frac{1}{2}(a+b)(a+b+1) + a.$$

Clearly $\pi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is bijective. Moreover, $a, b \leq \pi(a,b)$ and in case $\pi(a,b) \neq 0$ also $a < \pi(a,b)$. Let

$$\pi_1(c) := \mu_{x \leq c} \exists_{y \leq c} (\pi(x,y) = c),$$
$$\pi_2(c) := \mu_{y \leq c} \exists_{x \leq c} (\pi(x,y) = c).$$

Then clearly $\pi_i(c) \leq c$ for $i \in \{1,2\}$ and

$$\pi_1(\pi(a,b)) = a,$$
$$\pi_2(\pi(a,b)) = b,$$
$$\pi(\pi_1(c), \pi_2(c)) = c.$$

π, π_1 and π_2 are elementary by definiton. □

LEMMA (Gödel's β-function). *There is in $\mathcal{E}$ a function β with the following property: For every sequence $a_0, \ldots, a_{n-1} < b$ of numbers less than b we can find a number $c \leq 4 \cdot 4^{n(b+n+1)^4}$ such that $\beta(c,i) = a_i$ for all $i < n$.*

PROOF. Let

$$a := \pi(b,n) \quad \text{and} \quad d := \prod_{i<n} \big(1 + \pi(a_i, i)a!\big).$$

From $a!$ and d we can, for each given $i < n$, reconstruct the number a_i as the unique $x < b$ such that

$$1 + \pi(x, i)a! \mid d.$$

For clearly a_i is such an x, and if some $x < b$ were to satisfy the same condition, then because $\pi(x, i) < a$ and the numbers $1 + ka!$ are relatively prime for $k \leq a$, we would have $\pi(x, i) = \pi(a_j, j)$ for some $j < n$. Hence $x = a_j$ and $i = j$, thus $x = a_i$.

We can now define the *Gödel β-function* as

$$\beta(c, i) := \pi_1\big(\mu_{y<c}(1 + \pi(\pi_1(y), i) \cdot \pi_1(c)) \cdot \pi_2(y) = \pi_2(c)\big).$$

Clearly β is in $\mathcal{E}$. Furthermore with $c := \pi(a!, d)$ we see that $\pi(a_i, \lceil d/1 + \pi(a_i, i)a!\rceil)$ is the unique such y, and therefore $\beta(c, i) = a_i$. It is then not difficult to estimate the given bound on c, using $\pi(b, n) < (b + n + 1)^2$. □

REMARK. The above definition of β shows that it is subelementary.

1.2.4. Closure Properties of $\mathcal{E}$.

THEOREM. *The class $\mathcal{E}$ is closed under limited recursion. Thus if g, h, k are given functions in $\mathcal{E}$ and f is defined from them according to the scheme*

$$\begin{aligned} f(\vec{m}, 0) &= g(\vec{m}), \\ f(\vec{m}, n + 1) &= h(n, f(\vec{m}, n), \vec{m}), \\ f(\vec{m}, n) &\leq k(\vec{m}, n), \end{aligned}$$

then f is in $\mathcal{E}$ also.

PROOF. Let f be defined from g, h and k in $\mathcal{E}$, by limited recursion as above. Using Gödel's β-function as in the last lemma we can find for any given $\vec{m}, n$ a number c such that $\beta(c, i) = f(\vec{m}, i)$ for all $i \leq n$. Let $R(\vec{m}, n, c)$ be the relation

$$\beta(c, 0) = g(\vec{m}) \wedge \forall_{i<n}(\beta(c, i + 1) = h(i, \beta(c, i), \vec{m}))$$

and note by the remarks above that its characteristic function is in $\mathcal{E}$. It is clear, by induction, that if $R(\vec{m}, n, c)$ holds then $\beta(c, i) = f(\vec{m}, i)$, for all $i \leq n$. Therefore we can define f explicitly by the equation

$$f(\vec{m}, n) = \beta(\mu_c\, R(\vec{m}, n, c), n).$$

f will lie in $\mathcal{E}$ if μ_c can be bounded by an $\mathcal{E}$ function. However, the lemma on Gödel/sβ-function gives a bound $4 \cdot 4^{(n+1)(b+n+2)^4}$, where in this case b can be taken as the maximum of $k(\vec{m}, i)$ for $i \leq n$. But this can be defined in $\mathcal{E}$ as $k(\vec{m}, i_0)$, where $i_0 = \mu_{i\leq n}\forall_{j\leq n}\, k(\vec{m}, j) \leq k(\vec{m}, i)$. Hence μc can be bounded by an $\mathcal{E}$ function. □

REMARK. Notice that it is in this proof only that the exponential function is required, in providing a bound for μ.

COROLLARY. *$\mathcal{E}$ is the class of all elementary functions.*

PROOF. It is sufficient merely to show that $\mathcal{E}$ is closed under bounded sums and bounded products. Suppose for instance, that f is defined from

g in $\mathcal{E}$ by bounded summation: $f(\vec{m}, n) = \sum_{i<n} g(\vec{m}, i)$. Then f can be defined by limited recursion, as follows

$$\begin{aligned} f(\vec{m}, 0) &= 0 \\ f(\vec{m}, n+1) &= f(\vec{m}, n) + g(\vec{m}, n) \\ f(\vec{m}, n) &\leq n \cdot \max_{i<n} g(\vec{m}, i) \end{aligned}$$

and the functions (including the bound) from which it is defined are in $\mathcal{E}$. Thus f is in $\mathcal{E}$ by the last lemma. If instead, f is defined by bounded product, then proceed similarly. □

1.2.5. Coding finite lists. Computation on lists is a practical necessity, so because we are basing everything here on the single data type $\mathbb{N}$ we must develop some means of "coding" finite lists or sequences of natural numbers into $\mathbb{N}$ itself. There are various ways to do this and we shall adopt one of the most traditional, based on the pairing functions π, π_1, π_2.

The empty sequence is coded by the number 0 and a sequence n_0, n_1, $\ldots$, n_{k-1} is coded by the "sequence number"

$$\langle n_0, n_1, \ldots, n_{k-1} \rangle = \pi'(\ldots \pi'(\pi'(0, n_0), n_1), \ldots, n_{k-1})$$

with $\pi'(a, b) := \pi(a, b) + 1$, thus recursively,

$$\begin{aligned} \langle\rangle &:= 0, \\ \langle n_0, n_1, \ldots, n_k \rangle &:= \pi'(\langle n_0, n_1, \ldots, n_{k-1} \rangle, n_k). \end{aligned}$$

Because of the surjectivity of π, every number a can be decoded uniquely as a sequence number $a = \langle n_0, n_1, \ldots, n_{k-1} \rangle$. If a is greater than zero, $\mathrm{hd}(a) := \pi_2(a \dot{-} 1)$ is the "head" (i.e., rightmost element) and $\mathrm{tl}(a) := \pi_1(a \dot{-} 1)$ is the "tail" of the list. The kth iterate of tl is denoted $\mathrm{tl}^{(k)}$ and since $\mathrm{tl}(a)$ is less than or equal to a, $\mathrm{tl}^{(k)}(a)$ is elementarily definable (by limited recursion). Thus we can define elementarily the "length" and "decoding" functions:

$$\begin{aligned} \mathrm{lh}(a) &:= \mu_{k \leq a} \mathrm{tl}^{(k)}(a) = 0, \\ (a)_i &:= \mathrm{hd}(\mathrm{tl}^{(\mathrm{lh}(a) \dot{-} (i+1))}(a)). \end{aligned}$$

Then if $a = \langle n_0, n_1, \ldots, n_{k-1} \rangle$ it is easy to check that

$$\mathrm{lh}(a) = k \text{ and } (a)_i = n_i \text{ for each } i < k.$$

Furthermore $(a)_i = 0$ when $i \geq \mathrm{lh}(a)$. We shall write $(a)_{i,j}$ for $((a)_i)_j$ and $(a)_{i,j,k}$ for $(((a)_i)_j)_k$. This elementary coding machinery will be used at various crucial points in the following.

Note that our previous remarks show that the functions $\mathrm{lh}(\cdot)$ and $(a)_i$ are subelementary, and so is $\langle n_0, n_1, \ldots, n_{k-1} \rangle$ for each fixed k .

Concatenation of sequence numbers $b * a$ is defined thus:

$$\begin{aligned} b * \langle\rangle &:= b, \\ b * \langle n_0, n_1, \ldots, n_k \rangle &:= \pi(b * \langle n_0, n_1, \ldots, n_{k-1} \rangle, n_k) + 1. \end{aligned}$$

To check that this operation is also elementary, define $h(b, a, i)$ by recursion on i as follows.

$$\begin{aligned} h(b, a, 0) &= b, \\ h(b, a, i+1) &= \pi(h(b, a, i), (a)_i) + 1 \end{aligned}$$

and note that since $\pi(h(b,a,i),(a)_i) < (h(b,a,i)+a)^2$ it follows by induction on i that $h(b,a,i)$ is less than or equal to $(b+a+i)^{2^i}$. Thus h is definable by limited recursion from elementary functions and hence is itself elementary. Finally

$$b * a = h(b, a, \mathrm{lh}(a)).$$

LEMMA. *The class $\mathcal{E}$ is closed under limited course-of-values recursion. Thus if h, k are given functions in $\mathcal{E}$ and f is defined from them according to the scheme*

$$f(\vec{m}, n) = h(n, \langle f(\vec{m}, 0), \ldots . f(\vec{m}, n-1)\rangle, \vec{m})$$
$$f(\vec{m}, n) \leq k(\vec{m}, n)$$

then f is in $\mathcal{E}$ also.

PROOF. $\bar{f}(\vec{m}, n) := \langle f(\vec{m}, 0), \ldots . f(\vec{m}, n-1)\rangle$ is definable by

$$\begin{aligned} \bar{f}(\vec{m}, 0) &= 0, \\ \bar{f}(\vec{m}, n+1) &= \bar{f}(\vec{m}, n) * \langle h(n, \bar{f}(\vec{m}, n), \vec{m})\rangle \\ \bar{f}(\vec{m}, n) &\leq \Big(\sum_{i \leq n} k(\vec{m}, i) + 1\Big)^{2^n}, \end{aligned}$$

using $\langle \underbrace{n, \ldots, n}_{k} \rangle < (n+1)^{2^k}$. But $f(\vec{m}, n) = (\bar{f}(\vec{m}.n))_n$. □

1.3. The Normal Form Theorem

1.3.1. Program numbers. The three types of register machine instructions I can be coded by "instruction numbers" $\sharp I$ thus, where v_0, v_1, v_2, ... is a list of all variables used to denote registers:

If I is "$v_j := 0$" then $\sharp I = \langle 0, j\rangle$.
If I is "$v_j := v_j + 1$" then $\sharp I = \langle 1, j\rangle$.
If I is "**if** $v_j = v_l$ **then** I_m **else** I_n" then $\sharp I = \langle 2, j, l, m, n\rangle$.

Clearly, using the sequence coding and decoding apparatus above, we can check elementarily whether or not a given number is an instruction number.

Any register machine program $P = I_0, I_1, \ldots, I_{k-1}$ can then be coded by a "program number" or "index" $\sharp P$ thus:

$$\sharp P \;=\; \langle\, \sharp I_0, \sharp I_1, \ldots, \sharp I_{k-1} \,\rangle$$

and again (although it is tedious) we can elementarily check whether or not a given number is indeed of the form $\sharp P$ for some program P. Tradition has it that e is normally reserved as a variable over putative program numbers.

Standard program constructs such as those in Sec.1.1 have associated "index-constructors", i.e., functions which, given indices of the subprograms, produce an index for the constructed program. The point is that for standard program constructs the associated index-constructor functions are elementary. For example there is an elementary index-constructor comp such that, given programs P_0, P_1 with indices e_0, e_1, $\mathrm{comp}(e_0, e_1)$ is an index of the program P_0 **;** P_1. A moment's thought should convince the reader that the appropriate definition of comp is as follows:

$$\mathrm{comp}(e_0, e_1) \;=\; e_0 * \langle r(e_0, e_1, 0), r(e_0, e_1, 1), \ldots, r(e_0, e_1, \mathrm{lh}(e_1) \dot{-} 1)\rangle$$

where $r(e_0, e_1, i) =$

$$\begin{cases} \langle 2, (e_1)_{i,1}, (e_1)_{i,2}, (e_1)_{i,3} + \mathrm{lh}(e_0), (e_1)_{i,4} + \mathrm{lh}(e_0)\rangle & \text{if } (e_1)_{i,0} = 2 \\ (e_1)_i & \text{otherwise} \end{cases}$$

re-addresses the jump instructions in P_1. Clearly r and hence comp are elementary functions.

DEFINITION. Henceforth, $\varphi_e^{(r)}$ denotes the partial function computed by the register machine program with program number e, operating on the input registers $v_1, \ldots, v_r$ and with output register v_0. There is no loss of generality here, since the variables in any program can always be renamed so that $v_1, \ldots, v_r$ become the input registers and v_0 the output. If e is not a program number, or it is but does not operate on the right variables, then we adopt the convention that $\varphi_e^{(r)}(n_1, \ldots, n_r)$ is undefined for all inputs $n_1, \ldots, n_r$.

1.3.2. Normal form.

THEOREM (Kleene's Normal Form). *For each arity r there is an elementary function U and an elementary relation T such that, for all e and all inputs $n_1, \ldots, n_r$,*

- $\varphi_e^{(r)}(n_1, \ldots, n_r)$ *is defined if and only if* $\exists_s T(e, n_1, \ldots, n_r, s)$,
- $\varphi_e^{(r)}(n_1, \ldots, n_r) = U(\, e, n_1, \ldots, n_r, \mu_s T(e, n_1, \ldots, n_r, s)\,)$.

PROOF. A computation of a register machine program $P(v_1, \ldots, v_r; v_0)$ on numerical inputs $\vec{n} = n_1, \ldots, n_r$ proceeds deterministically, step by step, each step corresponding to the execution of one instruction. Let e be its program number, and let $v_0, \ldots, v_l$ be all the registers used by P, including the "working registers" so $r \leq l$.

The "state" of the computation at step s is defined to be the sequence number

$$\mathrm{state}(e, \vec{n}, s) = \langle e, i, m_0, m_1, \ldots, m_l\rangle$$

where $m_0, m_1, \ldots, m_l$ are the values stored in the registers $v_0, v_1, \ldots, v_l$ after step s is completed, and the next instruction to be performed is the ith one, thus $(e)_i$ is its instruction number.

The "state transition function" $\mathrm{tr}\colon \mathbb{N} \to \mathbb{N}$ computes the "next state". So suppose that $x = \langle e, i, m_0, m_1, \ldots, m_l\rangle$ is any putative state. Then in what follows, $e = (x)_0$, $i = (x)_1$, and $m_j = (x)_{j+2}$ for each $j \leq l$. The definition of $\mathrm{tr}(x)$ is therefore as follows:

$$\mathrm{tr}(x) = \langle e, i', m_0', m_1', \ldots, m_l'\rangle$$

where

- If $(e)_i = \langle 0, j\rangle$ where $j \leq l$ then $i' = i + 1$, $m_j' = 0$, and all other registers remain unchanged, i.e., $m_k' = m_k$ for $k \neq j$.
- If $(e)_i = \langle 1, j\rangle$ where $j \leq l$ then $i' = i + 1$, $m_j' = m_j + 1$, and all other registers remain unchanged.
- If $(e)_i = \langle 2, j_0, j_1, i_0, i_1\rangle$ where $j_0, j_1 \leq l$ and $i_0, i_1 \leq \mathrm{lh}(e)$ then $i' = i_0$ or $i' = i_1$ according as $m_{j_0} = m_{j_1}$ or not, and all registers remain unchanged, i.e., $m_j' = m_j$ for all $j \leq l$.

- Otherwise, if x is not a sequence number, or if e is not a program number, or if it refers to a register v_k with $l < k$, or if $\mathrm{lh}(e) \leq i$, then $\mathrm{tr}(x)$ simply repeats the same state x so $i' = i$, and $m'_j = m_j$ for every $j \leq l$.

Clearly tr is an *elementary* function, since it is defined by elementarily decidable cases, with (a great deal of) elementary decoding and re-coding involved in each case.

Consequently, the "state function" $\mathrm{state}(e, \vec{n}, s)$ is also *elementary* because it can be defined by iterating the transition function by limited recursion on s as follows:

$$\begin{aligned} \mathrm{state}(e, \vec{n}, 0) &= \langle e, 0, n_1, \ldots, n_r, 0, \ldots, 0 \rangle \\ \mathrm{state}(e, \vec{n}, s+1) &= \mathrm{tr}(\,\mathrm{state}(e, \vec{n}, s)\,) \\ \mathrm{state}(e, \vec{n}, s) &\leq h(e, \vec{n}, s) \end{aligned}$$

where for the bounding function h we can take

$$h(e, \vec{n}, s) = \langle e, e \rangle * \langle \max(\vec{n}) + s, \ldots, \max(\vec{n}) + s \rangle,$$

This is because the maximum value of any register at step s cannot be greater than $\max(\vec{n}) + s$. Now this expression clearly is elementary, since $\langle m, \ldots, m \rangle$ with i occurrences of m is definable by a limited recursion with bound $(m+i)^{2^i}$, as is easily seen by induction on i.

Now recall that if program P has program number e then computation terminates when instruction $I_{\mathrm{lh}(e)}$ is encountered. Thus we can define the "termination relation" $T(e, \vec{n}, s)$ meaning "computation terminates at step s", by

$$T(e, \vec{n}, s) \leftrightarrow (\,\mathrm{state}(e, \vec{n}, s)\,)_1 = \mathrm{lh}(e).$$

Clearly T is elementary and

$$\varphi_e^{(r)}(\vec{n}) \text{ is defined} \leftrightarrow \exists_s T(e, \vec{n}, s).$$

The output on termination is the value of register v_0, so if we define the "output function" $U(e, \vec{n}, s)$ by

$$U(e, \vec{n}, s) \;=\; (\,\mathrm{state}(e, \vec{n}, s)\,)_2$$

then U is also elementary and

$$\varphi_e^{(r)}(\vec{n}) \;=\; U(e, \vec{n}, \mu_s\, T(e, \vec{n}, s)). \qquad \square$$

1.3.3. Σ^0_1-definable relations and μ-recursive functions. A relation R of arity r is said to be Σ^0_1-*definable* if there is an elementary relation E, say of arity $r + l$, such that for all $\vec{n} = n_1, \ldots, n_r$,

$$R(\vec{n}) \leftrightarrow \exists_{k_1} \ldots \exists_{k_l}\, E(\vec{n}, k_1, \ldots, k_l).$$

A partial function φ is said to be Σ^0_1-*definable* if its graph

$$\{\, (\vec{n}, m) \mid \varphi(\vec{n}) \text{ is defined and } = m \,\}$$

is Σ^0_1-definable.

To say that a non-empty relation R is Σ^0_1-definable is equivalent to saying that the set of all sequences $\langle \vec{n} \rangle$ satisfying R can be enumerated (possibly with repetitions) by some elementary function $f\colon \mathbb{N} \to \mathbb{N}$. Such relations are

called *elementarily enumerable*. For choose any fixed sequence $\langle a_1, \ldots, a_r \rangle$ satisfying R and define

$$f(m) = \begin{cases} \langle (m)_1, \ldots, (m)_r \rangle & \text{if } E((m)_1, \ldots, (m)_{r+l}) \\ \langle a_1, \ldots, a_r \rangle & \text{otherwise.} \end{cases}$$

Conversely if R is elementarily enumerated by f then

$$R(\vec{n}) \leftrightarrow \exists_m \, (f(m) = \langle \vec{n} \rangle)$$

is a Σ^0_1-definition of R.

The *μ-recursive functions* are those (partial) functions which can be defined from the initial functions: constant 0, successor S, projections (onto the ith coordinate), addition $+$, modified subtraction $\dot{-}$ and multiplication $\cdot$, by applications of composition and unbounded minimization. Note that it is through unbounded minimization that partial functions may arise.

LEMMA. *Every elementary function is μ-recursive.*

PROOF. By simply removing the bounds on μ in the lemmas in 1.2.3 one obtains μ-recursive definitions of the pairing functions π, π_1, π_2 and of Gödel's β-function. Then by removing all mention of bounds from Theorem in 1.2.4 one sees that the μ-recursive functions are closed under (unlimited) primitive recursive definitions: $f(\vec{m}, 0) = g(\vec{m})$, $f(\vec{m}, n+1) = h(n, f(\vec{m}, n))$. Thus one can μ-recursively define bounded sums and bounded products, and hence all elementary functions. □

1.3.4. Computable functions.

DEFINITION. The *while-programs* are those programs which can be built up from assignment statements $x := 0$, $x := y$, $x := y + 1$, $x := y \dot{-} 1$, by Conditionals, Composition, For-Loops and While-Loops as in Sec.1.1 (on program constructs).

THEOREM. *The following are equivalent:*

(a) *φ is register machine computable,*
(b) *φ is Σ^0_1-definable,*
(c) *φ is μ-recursive,*
(d) *φ is computable by a while program.*

PROOF. The Normal Form Theorem shows immediately that every register machine computable function $\varphi_e^{(r)}$ is Σ^0_1-definable since

$$\varphi_e^{(r)}(\vec{n}) = m \leftrightarrow \exists_s \big(T(e, \vec{n}, s) \wedge U(e, \vec{n}, s) = m \big)$$

and the relation $T(e, \vec{n}, s) \wedge U(e, \vec{n}, s) = m$ is clearly elementary. If φ is Σ^0_1-definable, say

$$\varphi(\vec{n}) = m \leftrightarrow \exists_{k_1} \ldots \exists_{k_l} \, E(\vec{n}, m, k_1, \ldots, k_l)$$

then φ can be defined μ-recursively by

$$\varphi(\vec{n}) = (\, \mu_m \, E(\vec{n}, (m)_0, (m)_1, \ldots, (m)_l) \,)_0 \, ,$$

using the fact (above) that elementary functions are μ-recursive. The examples of computable functionals in Sec.1.1 show how the definition of any

μ-recursive function translates automatically into a while program. Finally, Sec.1.1 shows how to implement any while program on a register machine. □

Henceforth *computable* means "register machine computable" or any of its equivalents.

COROLLARY. *The function* $\varphi_e^{(r)}(n_1, \ldots, n_r)$ *is a computable partial function of the* $r+1$ *variables* $e, n_1, \ldots, n_r$.

PROOF. Immediate from the Normal Form. □

LEMMA. *A relation* R *is computable if and only if both* R *and its complement* $\mathbb{N}^n \setminus R$ *are* Σ_1^0*-definable.*

PROOF. We can assume that both R and $\mathbb{N}^n \setminus R$ are not empty, and (for simplicity) also $n = 1$.

$\Rightarrow$. By the theorem above every computable relation is Σ_1^0-definable, and with R clearly its complement is computable.

$\Leftarrow$. Let $f, g \in \mathcal{E}$ enumerate R and $\mathbb{N} \setminus R$, respectively. Then

$$h(n) := \mu_i(f(i) = n \vee g(i) = n)$$

is a total μ-recursive function, and $R(n) \leftrightarrow f(h(n)) = n$. □

1.3.5. Undecidability of the halting problem. The above corollary says that there is a single "universal" program which, given numbers e and $\vec{n}$, computes $\varphi_e^{(r)}(\vec{n})$ if it is defined. However we cannot decide in advance whether or not it will be defined. There is no program which, given e and $\vec{n}$, computes the total function

$$h(e, \vec{n}) \;=\; \begin{cases} 1 & \text{if } \varphi_e^{(r)}(\vec{n}) \text{ is defined,} \\ 0 & \text{if } \varphi_e^{(r)}(\vec{n}) \text{ is undefined.} \end{cases}$$

For suppose there were such a program. Then the function

$$\psi(\vec{n}) \;=\; \mu_m\,(h(n_1, \vec{n}) = 0)$$

would be computable, say with fixed program number e_0, and therefore

$$\varphi_{e_0}^{(r)}(\vec{n}) \;=\; \begin{cases} 0 & \text{if } h(n_1, \vec{n}) = 0 \\ \text{undefined} & \text{if } h(n_1, \vec{n}) = 1. \end{cases}$$

But then fixing $n_1 = e_0$ gives:

$$\varphi_{e_0}^{(r)}(\vec{n}) \text{ defined} \leftrightarrow h(e_0, \vec{n}) = 0 \leftrightarrow \varphi_{e_0}^{(r)}(\vec{n}) \text{ undefined},$$

a contradiction. Hence the relation $R(e, \vec{n})$ which holds if and only if $\varphi_e^{(r)}(\vec{n})$ is defined, is not recursive. It is however Σ_1^0-definable.

There are numerous attempts to classify total computable functions according to the complexity of their termination proofs.

1.4. Recursive Definitions

1.4.1. Least fixed points of recursive definitions. By a *recursive definition* of a partial function φ of arity r from given partial functions $\psi_1, \ldots, \psi_m$ of fixed but unspecified arities, we mean a defining equation of the form

$$\varphi(n_1, \ldots, n_r) \;=\; t(\psi_1, \ldots, \psi_m, \varphi; n_1, \ldots, n_r)$$

where t is any compositional term built up from the numerical variables $\vec{n} = n_1, \ldots, n_r$ and the constant 0 by repeated applications of the successor and predecessor functions, the given functions $\psi_1, \ldots, \psi_m$, the function φ itself, and the "definition by cases" function :

$$\mathrm{dc}(x, y, u, v) \;=\; \begin{cases} u & \text{if } x, y \text{ are both defined and equal} \\ v & \text{if } x, y \text{ are both defined and unequal} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Our notion of recursive definition is essentially a reformulation of the Herbrand-Gödel-Kleene equation calculus; see [Kleene, 1952].

There may be many partial functions φ satisfying such a recursive definition, but the one we wish to single out is the least defined one, i.e., the one whose defined values arise inevitably by *lazy evaluation* of the term t "from the outside in", making only those function calls which are absolutely necessary. This presupposes that each of the functions from which t is constructed already comes equipped with an evaluation strategy. In particular if a subterm $\mathrm{dc}(t_1, t_2, t_3, t_4)$ is called then it is to be evaluated according to the program construct:

$$x := t_1 \;;\; y := t_2 \;;\; [\textbf{if } x := y \textbf{ then } t_3 \textbf{ else } t_4].$$

Some of the function calls demanded by the term t may be for further values of φ itself, and these must be evaluated by repeated unravellings of t (in other words by recursion).

This "least solution" φ will be referred to as *the function defined by that recursive definition* or its *least fixed point*. Its existence and its computability are guaranteed by Kleene's Recursion Theorem below.

1.4.2. The principles of finite support and monotonicity, and the effective index property. Suppose we are given any fixed partial functions $\psi_1, \ldots, \psi_m$ and ψ, of the appropriate arities, and fixed inputs $\vec{n}$. If the term $t = t(\psi_1, \ldots, \psi_m, \psi; \vec{n})$ evaluates to a defined value k then the following principles clearly hold:

Finite Support Principle. Only finitely many values of $\psi_1, \ldots, \psi_m$ and ψ are used in that evaluation of t.

Monotonicity Principle. The same value k will be obtained no matter how the partial functions $\psi_1, \ldots, \psi_m$ and ψ are extended.

Note also that any such term t satisfies the

Effective Index Property. There is an elementary function f such that if $\psi_1, \ldots, \psi_m$ and ψ are computable partial functions with program numbers $e_1, \ldots, e_m$ and e respectively, then according to the lazy evaluation strategy just described,

$$t(\psi_1, \ldots, \psi_m, \psi; \vec{n})$$

defines a computable function of $\vec{n}$ with program number $f(e_1, \ldots, e_m, e)$.

The proof of the Effective Index Property is by induction over the build-up of the term t. The base case is where t is just one of the constants $0, 1$ or a variable n_j, in which case it defines either a constant function $\vec{n} \mapsto 0$ or $\vec{n} \mapsto 1$, or a projection function $\vec{n} \mapsto n_j$. Each of these is trivially computable with a fixed program number, and it is this program number we take as the value of $f(e_1, \ldots, e_m, e)$. Since in this case f is a constant function, it is clearly elementary. The induction step is where t is built up by applying one of the given functions: successor, predecessor, definition by cases or ψ (with or without a subscript) to previously constructed subterms $t_i(\psi_1, \ldots, \psi_m, \psi; \vec{n})$, $i = 1 \ldots l$, thus:

$$t \;=\; \chi(t_1, \ldots, t_l).$$

Inductively we can assume that for each $i = 1 \ldots l$, t_i defines a partial function of $\vec{n} = n_1, \ldots, n_r$ which is register machine computable by some program P_i with program number given by an already-constructed elementary function $f_i = f_i(e_1, \ldots, e_m, e)$. Therefore if χ is computed by a program Q with program number e', we can put $P_1, \ldots, P_l$ and Q together to construct a new program obeying the evaluation strategy for t. Furthermore, by the remark on index-constructions in 1.3.1. we will be able to compute its program number $f(e_1, \ldots, e_m, e)$ from the given numbers $f_1, \ldots, f_l$ and e', by some elementary function.

1.4.3. Recursion Theorem.

THEOREM (Kleene's Recursion Theorem). *For given partial functions $\psi_1, \ldots, \psi_m$, every recursive definition*

$$\varphi(\vec{n}) \;=\; t(\psi_1, \ldots, \psi_m, \varphi; \vec{n})$$

has a least fixed point, i.e., a least defined solution, φ. Moreover if ψ_1, $\ldots$,ψ_m are computable, so is the least fixed point φ.

PROOF. Let $\psi_1, \ldots, \psi_m$ be fixed partial functions of the appropriate arities. Let Φ be the functional from partial functions of arity r to partial functions of arity r defined by lazy evaluation of the term t as described above:

$$\Phi(\psi)(\vec{n}) \;=\; t(\psi_1, \ldots, \psi_m, \psi; \vec{n}).$$

Let $\varphi_0, \varphi_1, \varphi_2, \ldots$ be the sequence of partial functions of arity r generated by Φ thus: φ_0 is the completely undefined function, and $\varphi_{i+1} = \Phi(\varphi_i)$ for each i. Then by induction on i, using the Monotonicity Principle above, we see that each φ_i is a subfunction of φ_{i+1}. That is, whenever $\varphi_i(\vec{n})$ is defined with a value k then $\varphi_{i+1}(\vec{n})$ is defined with that same value. Since their defined values are consistent with one another we can therefore construct the "union" φ of the φ_i's as follows:

$$\varphi(\vec{n}) = k \leftrightarrow \exists_i \, (\varphi_i(\vec{n}) = k).$$

(i) This φ is then the required least fixed point of the recursive definition.

To see that it is a fixed point, i.e., $\varphi = \Phi(\varphi)$, first suppose $\varphi(\vec{n})$ is defined with value k. Then by the definition of φ just given, there is an $i > 0$ such that $\varphi_i(\vec{n})$ is defined with value k. But $\varphi_i = \Phi(\varphi_{i-1})$ so $\Phi(\varphi_{i-1})(\vec{n})$ is defined with value k. Therefore by the Monotonicity Principle for Φ, since

φ_{i-1} is a subfunction of φ, $\Phi(\varphi)(\vec{n})$ is defined with value k. Hence φ is a subfunction of $\Phi(\varphi)$.

It remains to show the converse, that $\Phi(\varphi)$ is a subfunction of φ. So suppose $\Phi(\varphi)(\vec{n})$ is defined with value k. Then by the Finite Support Principle, only finitely many defined values of φ are called for in this evaluation. By the definition of φ there must be some i such that φ_i already supplies all of these required values, and so already at stage i we have $\Phi(\varphi_i)(\vec{n}) = \varphi_{i+1}(\vec{n})$ defined with value k. Since φ_{i+1} is a subfunction of φ it follows that $\varphi(\vec{n})$ is defined with value k. Hence $\Phi(\varphi)$ is a subfunction of φ.

To see that φ is the least such fixed point, suppose φ' is any fixed point of Φ. Then $\Phi(\varphi') = \varphi'$ so by the Monotonicity Principle, since φ_0 is a subfunction of φ' it follows that $\Phi(\varphi_0) = \varphi_1$ is a subfunction of $\Phi(\varphi') = \varphi'$. Then again by Monotonicity, $\Phi(\varphi_1) = \varphi_2$ is a subfunction of $\Phi(\varphi') = \varphi'$ etcetera so that for each i, φ_i is a subfunction of φ'. Since φ is the union of the φ_i's it follows that φ itself is a subfunction of φ'. Hence φ is the least fixed point of Φ.

(ii) Finally we have to show that φ is computable if the given functions $\psi_1, \ldots, \psi_m$ are. For this we need the Effective Index Property of the term t, which supplies an elementary function f such that if ψ is computable with program number e then $\Phi(\psi)$ is computable with program number $f(e) = f(e_1, \ldots, e_m, e)$. Thus if u is any fixed program number for the completely undefined function of arity r, $f(u)$ is a program number for $\varphi_1 = \Phi(\varphi_0)$, $f^2(u) = f(f(u))$ is a program number for $\varphi_2 = \Phi(\varphi_1)$, and in general $f^i(u)$ is a program number for φ_i. Therefore in the notation of the Normal Form Theorem,

$$\varphi_i(\vec{n}) \;=\; \varphi^{(r)}_{f^i(u)}(\vec{n})$$

and by the corollary (in 1.3.4) to the Normal Form Theorem, this is a computable function of i and $\vec{n}$, since $f^i(u)$ is a computable function of i definable (informally) say by a for-loop of the form "**for** $j = 1 \ldots i$ **do** f **od**". Therefore by the earlier equivalences, $\varphi_i(\vec{n})$ is a Σ^0_1-definable function of i and $\vec{n}$, and hence so is φ itself because

$$\varphi(\vec{n}) = m \leftrightarrow \exists_i \, (\, \varphi_i(\vec{n}) = m \,) \; .$$

So φ is computable and this completes the proof. □

NOTE. The above proof works equally well if φ is a vector-valued function. In other words if, instead of defining a single partial function φ, the recursive definition in fact defines a finite list $\vec{\varphi}$ of such functions *simultaneously*. For example, the individual components of the machine state of any register machine at step s are clearly defined by a simultaneous recursive definition, from zero and successor.

1.4.4. Recursive programs and partial recursive functions. A *recursive program* is a finite sequence of possibly simultaneous recursive definitions:

$$\begin{aligned}
\vec{\varphi}_0(n_1, \ldots, n_{r_0}) &= t_0(\vec{\varphi}_0; n_1, \ldots, n_{r_0}) \\
\vec{\varphi}_1(n_1, \ldots, n_{r_1}) &= t_1(\vec{\varphi}_0, \vec{\varphi}_1; n_1, \ldots, n_{r_1}) \\
\vec{\varphi}_2(n_1, \ldots, n_{r_2}) &= t_2(\vec{\varphi}_0, \vec{\varphi}_1, \vec{\varphi}_2; n_1, \ldots, n_{r_2})
\end{aligned}$$

$$\vdots$$
$$\vec{\varphi}_k(n_1,\ldots,n_{r_k}) = t_k(\vec{\varphi}_0,\ldots,\vec{\varphi}_{k-1},\vec{\varphi}_k;n_1,\ldots,n_{r_k}).$$

A partial function is said to be *partial recursive* if it is one of the functions defined by some recursive program as above. A partial recursive function which happens to be totally defined is called simply a *recursive function*.

THEOREM. *A function is partial recursive if and only if it is computable.*

PROOF. The Recursion Theorem tells us immediately that every partial recursive function is computable. For the converse we use the equivalence of computability with μ-recursiveness already established in 1.3.4. Thus we need only show how to translate any μ-recursive definition into a recursive program:

The constant 0 function is defined by the recursive program

$$\varphi(\vec{n}) \;=\; 0$$

and similarly for the constant 1 function.

The addition function $\varphi(m,n) = m+n$ is defined by the recursive program

$$\varphi(m,n) \;=\; \mathrm{dc}(n,0,m,\varphi(m,n \dot{-} 1)+1)$$

and the subtraction function $\varphi(m,n) = m \dot{-} n$ is defined similarly but with the successor function $+1$ replaced by the predecessor $\dot{-}1$. Multiplication is defined recursively from addition in much the same way. Note that in each case the right hand side of the recursive definition is an allowed term.

The composition scheme is a recursive definition as it stands.

Finally, given a recursive program defining ψ, if we add to it the recursive definition:

$$\varphi(\vec{n},m) \;=\; \mathrm{dc}(\,\psi(\vec{n},m),0,m,\varphi(\vec{n},m+1)\,)$$

followed by

$$\varphi'(\vec{n}) \;=\; \varphi(\vec{n},0)$$

then the computation of $\varphi'(\vec{n})$ proceeds as follows:

$$\begin{aligned}
\varphi'(\vec{n}) &= \varphi(\vec{n},0) \\
&= \varphi(\vec{n},1) && \text{if } \psi(\vec{n},0)\neq 0 \\
&= \varphi(\vec{n},2) && \text{if } \psi(\vec{n},1)\neq 0 \\
&\;\;\vdots \\
&= \varphi(\vec{n},m) && \text{if } \psi(\vec{n},m-1)\neq 0 \\
&= m && \text{if } \psi(\vec{n},m)=0.
\end{aligned}$$

Thus the recursive program for φ' defines unbounded minimization:

$$\varphi'(\vec{n}) \;=\; \mu_m\,(\psi(\vec{n},m)=0). \qquad \square$$

1.5. Primitive Recursion and For-Loops

1.5.1. Primitive recursive functions. A *primitive recursive program* over $\mathbb{N}$ is a recursive program in which each recursive definition is of one of the following five special kinds:

$$(Z) \qquad f_i(n) = 0,$$
$$(S) \qquad f_i(n) = n + 1,$$
$$(U_j^k) \qquad f_i(n_1, \ldots, n_k) = n_j,$$
$$(C_r^k) \qquad f_i(n_1, \ldots, n_k) = f_{i_0}(\, f_{i_1}(n_1, \ldots, n_k), \ldots, f_{i_r}(n_1, \ldots, n_k)\,),$$
$$(PR) \qquad f_i(n_1, \ldots, n_k, 0) = f_{i_0}(n_1, \ldots, n_k),$$
$$f_i(n_1, \ldots, n_k, m+1) = f_{i_1}(n_1, \ldots, n_k, m, f_i(n_1, \ldots, n_k, m)),$$

where, in (C) and (PR), $i_0, i_1, \ldots, i_r < i$. Recall that functions are allowed to be 0-ary, so k may be 0. Note that the two equations in the (PR) scheme can easily be combined into one recursive definition using the dc and $\dot{-}$ function. The reason for using f rather than φ to denote the functions in such a program is that they are obviously totally defined (we try to maintain the convention that $f, g, h, \ldots$ denote total functions).

Definition. The *primitive recursive functions* are those which are definable by primitive recursive programs. The class of all primitive recursive functions is denoted "Prim"

Lemma (Explicit Definitions). *If t is a term built up from numerical constants, variables $n_1, \ldots, n_k$ and function symbols $f_1, \ldots, f_m$ denoting previously defined primitive recursive functions, then the function f defined from them by*

$$f(n_1, \ldots, n_k) = t(f_1, \ldots, f_m; n_1, \ldots, n_k)$$

is also primitive recursive.

Proof. By induction over the generation of term t.
If t is a constant l then using the (Z), (S) and (U) schemes :

$$f(n_1, \ldots, n_k) = (S \circ S \ldots S \circ Z \circ U_1^k)\,(n_1, \ldots, n_k).$$

If t is one of the variables n_j then using the (U_j^k) scheme:

$$f(n_1, \ldots, n_k) = n_j.$$

If t is an applicative term $f_i(t_1, \ldots, t_r)$ then by the (C_r^k) scheme:

$$f(n_1, \ldots, n_k) = f_i(t_1(n_1, \ldots, n_k), \ldots, t_r(n_1, \ldots, n_k)). \qquad \square$$

Lemma. *Every elementary function is primitive recursive, but not conversely.*

Proof. Addition $f(n, m) = n + m$ is defined from successor by the primitive recursion:

$$f(n, 0) \;=\; n, \quad f(n, m+1) \;=\; f(n, m) + 1$$

and modified subtraction $f(n, m) = n \dot{-} m$ is defined similarly, replacing $+1$ by $\dot{-}1$. Note that predecessor $\dot{-}1$ is definable by a trivial primitive recursion:

$$f(0) \;=\; 0, \quad f(m+1) \;=\; m.$$

Bounded sum $f(\vec{n}, m) = \sum_{i<m} g(\vec{n}, i)$ is definable from $+$ by another primitive recursion:

$$f(\vec{n}, 0) \;=\; 0, \quad f(\vec{n}, m+1) \;=\; f(\vec{n}, m) + g(\vec{n}, m).$$

Multiplication is then defined explicitly by a bounded sum, and bounded product by a further primitive recursion. The above lemma then gives closure under all explicit definitions using these principles. Hence every elementary function is primitive recursive.

We have already seen that the function $n \mapsto 2_n(1)$ is not elementary. However it can be defined primitive recursively from the (elementary) exponential function thus:

$$2_0(1) \;=\; 1, \quad 2_{n+1}(1) \;=\; 2^{2_n(1)}. \qquad \square$$

1.5.2. Loop-Programs. The *loop-programs* over $\mathbb{N}$ are built up from

- *assignments* $x := 0$, $x := x + 1$, $x := y$, $x := y \dot{-} 1$ using
- *compositions* $\ldots$ **;** $\ldots$,
- *conditionals* **if** $x = y$ **then** $\ldots$ **else** $\ldots$ **fi**, and
- *for-loops* **for** $i = 1 \ldots y$ **do** $\ldots$ **od**,
  where i is not reset between **do** and **od**.

LEMMA. *Every primitive recursive function is computable by a loop-program.*

PROOF. Composition corresponds to "**;**" and primitive recursion

$$f(\vec{n}, 0) \;=\; g(\vec{n}), \quad f(\vec{n}, m+1) \;=\; h(\vec{n}, m, f(\vec{n}, m))$$

can be recast as a for-loop (with input variables $\vec{x}, y$ and output variable z) thus:

$$z := g(\vec{x});\ \textbf{for}\ i = 1 \ldots y\ \ \textbf{do}\ z := h(\vec{x}, i-1, z)\ \ \textbf{od}. \qquad \square$$

We now describe the *operational semantics of loop programs.* Each loop-program P on "free variables" $\vec{x} = x_1, \ldots, x_k$ (i.e., those not "bound" by for-loops), can be considered as a "state-transformer" function from $\mathbb{N}^k$ to $\mathbb{N}^k$ and we write $P(\vec{n})$ to denote the output state $(n'_1, \ldots, n'_k)$ which results after applying program P to input $(n_1, \ldots, n_k)$. Note that loop-programs always terminate! The definition of $P(\vec{n})$ runs as follows, according to the form of program P:

Assignments. For example if P is "$x_i := x_j \dot{-} 1$" then

$$P(n_1, \ldots, n_i, \ldots, n_k) = (n_1, \ldots, n_j \dot{-} 1, \ldots, n_k).$$

Composition. If P is "Q **;** R" then

$$P(\vec{n}) = (R \circ Q)(\vec{n}).$$

Conditionals. If P is "**if** $x_i = x_j$ **then** Q **else** R **fi**" then

$$P(\vec{n}) = \begin{cases} Q(\vec{n}) & \text{if } n_i = n_j \\ R(\vec{x}) & \text{if } n_i \neq n_j. \end{cases}$$

For-loops. If P is "**for** $i = 1 \dots x_j$ **do** $Q(i, \vec{x})$ **od**" then P is defined by $P(n_1, \dots, n_j, \dots, n_k) = Q^*(n_j, n_1, \dots, n_j, \dots, n_k)$ with Q^* defined by primitive recursion on i thus

$$\begin{cases} Q^*(0, n_1, \dots, n_j, \dots, n_k) & = (n_1, \dots, n_j, \dots, n_k) \\ Q^*(i+1, n_1, \dots, n_j, \dots, n_k) & = Q(i+1, Q^*(i, n_1, \dots, n_j, \dots, n_k)). \end{cases}$$

Notice that the above description actually gives P as a primitive recursive function from $\mathbb{N}^k$ to $\mathbb{N}^k$ and not from $\mathbb{N}^k$ to $\mathbb{N}$ as the formal definition of primitive recursion requires. However this is immaterial when working over $\mathbb{N}$ because we can work with "coded" sequences $\langle \vec{n} \rangle \in \mathbb{N}$ instead of vectors $(\vec{n}) \in \mathbb{N}^k$ so as to define

$$P(n_1, \dots, n_k) = \langle n_1', \dots, n_k' \rangle.$$

The coding and decoding can all be done elementarily, so for any loop-program P the output function $P(\vec{n})$ will always be primitive recursive. We therefore have:

THEOREM. *The primitive recursive functions are exactly those computed by loop-programs.*

1.5.3. Reduction to primitive recursion. Various somewhat more general kinds of recursion can be transformed into ordinary primitive recursion. Two important examples are:

Course of values recursion. A trivial example is the Fibonacci function

$$\begin{cases} f(0) & = 1, \\ f(1) & = 2, \\ f(n+2) & = f(n) + f(n+1), \end{cases}$$

which calls for several "previous" values (in this case two) in order to compute the "next" value. This is not formally a primitive recursion, but it could be transformed into one because it can be computed by the for-loop (with x, y as input and output variables):

$y := 1 \ ; \ z := 1 \ ; \ \textbf{for} \ i = 1 \dots x \ \ \textbf{do} \ u := y \ ; \ y := y + z \ ; \ z := u \ \ \textbf{od}.$

Recursion with parameter substitution. This has the form:

$$\begin{cases} f(n, 0) & = g(n), \\ f(n, m+1) & = h(n, m, f(p(n, m), m)). \end{cases}$$

Again this is not formally a primitive recursion as it stands, but it can be transformed to the following primitive recursive program:

$$\begin{aligned} (PR) \quad & \begin{cases} q(n, m, 0) & = n, \\ q(n, m, i+1) & = p(q(n, m, i), m \dot{-} (i+1)), \end{cases} \\ (C) \quad & g'(n, m) \ = \ g(q(n, m, m)), \\ (C) \quad & h'(n, m, i, j) \ = \ h(q(n, m, m \dot{-} (i+1)), i, j), \\ (PR) \quad & \begin{cases} f'(n, m, 0) & = g'(n, m), \\ f'(n, m, i+1) & = h'(n, m, i, f'(n, m, i)), \end{cases} \\ (C) \quad & f(n, m) \ = \ f'(n, m, m). \end{aligned}$$

We leave it as an exercise to check that this program defines the correct function f.

1.5.4. A complexity hierarchy for Prim**.** Given a register machine program $I_0, I_1, \ldots, I_m \ldots, I_{k-1}$ where, for example, I_m is a jump instruction "**if** $x_p = x_q$ **then** I_r **else** I_s **fi**" and given numerical inputs in the registers $\vec{x}$, the ensuing computation *as far as step* y can be performed by a single for-loop as follows, where j counts the "next instruction" to be obeyed:

$j := 0$ **;**
for $i = 1 \ldots y$ **do**
  if $j = 0$ **then** I_0 **;** $j := 1$ **else**
  if $j = 1$ **then** I_1 **;** $j := 2$ **else**
  ...
  if $j = m$ **then if** $x_p = x_q$ **then** $j := r$ **else** $j := s$ **fi else**
  ...
  ... **fi** ... **fi fi**
od.

Definition. L_k consists of all loop-programs which contain nested for-loops with maximum depth of nesting k. Thus L_0-programs are loop-free and L_{k+1}-programs only contain for-loops of the form **for** $i = 1 \ldots y$ **do** P **od** where P is a L_j-program for some $j \leq k$.

Definition. A bounding function for a loop-program P is an increasing function $B_P \colon \mathbb{N} \to \mathbb{N}$ (that is, $B_P(n) \geq n$) such that for all $n \in \mathbb{N}$ we have

$$B_P(n) \geq n + \max_{\vec{i} \leq n} \#_P(\vec{i})$$

where $\#_P(\vec{i})$ denotes the number of steps executed by P when called with input $\vec{i}$. Note that $B_P(n)$ will also bound the size of the output for any input $\vec{i} \leq n$, since at most 1 can be added to any register at any step. x

With each loop-program there is a naturally associated bounding function as follows :

$P =$ assignment	$B_P(n) = n + 1,$
$P =$ **if** $x_i = x_j$ **then** Q **else** R **fi**	$B_P(n) = \max(B_Q(n), B_R(n)) + 1,$
$P = Q$ **;** R	$B_P(n) = B_R(B_Q(n)),$
$P =$ **for** $i = 1 \ldots x_k$ **do** Q **od**	$B_P(n) = B_Q^n(n),$

where B_Q^n denotes the n-times iterate of B_Q.

It is obvious that the defined B_P is a bounding function when P is an assignment or a conditional. When P is a composed program $P = Q$ **;** R then, given any input $\vec{i} \leq n$ let $s := \#Q(\vec{i})$. Then $n + s \leq B_Q(n)$ and so the output $\vec{j}$ of the computation of Q on $\vec{i}$ is also $\leq B_Q(n)$. Now let $s' := \#_R(\vec{j})$. Then $B_R(B_Q(n)) \geq B_Q(n) + s' \geq n + s + s'$. Hence $B_R(B_Q(n)) \geq n + \max_{\vec{i} \leq n} \#_P(\vec{i})$ and therefore $B_R \circ B_Q$ is an appropriate bounding function for P. Finally if P is a for-loop as indicated, then for any input $\vec{i} \leq n$ the computation simply composes Q a certain number of times, say k, where $k \leq n$. Therefore, by what we just have done for composition,

$B^n_Q(n) \geq B^k_Q(n) \geq n + \#_P(\vec{i})$. Again this justifies our choice of bounding functions for for-loops.

DEFINITION. The sequence $F_0, F_1, \ldots F_k, \ldots$ of Prim functions is given by

$$F_0(n) \;=\; n+1, \quad F_{k+1}(n) \;=\; F^n_k(n).$$

DEFINITION. For each increasing function $g\colon \mathbb{N} \to \mathbb{N}$ let $\mathrm{Comp}(g)$ denote the class of all total functions $f\colon \mathbb{N}^r \to \mathbb{N}$ which can be computed by register machines in such a way that on (all but finitely many) inputs $\vec{n}$, the number of steps required to compute $f(\vec{n})$ is bounded by $g(\max(\vec{n}))$.

THEOREM. *For each $k \geq 1$ we have*

$$L_k\text{-computable} = \bigcup_i \mathrm{Comp}(F^i_k)$$

and hence

$$\mathrm{Prim} = \bigcup_k \mathrm{Comp}(F_k).$$

PROOF. The second part follows immediately from the first since for all $n \geq i$, $F^i_k(n) \leq F^n_k(n) = F_{k+1}(n)$.

To prove the left-to-right containment of the first part, proceed by induction on $k \geq 0$ to show that for every L_k-program P there is a fixed i such that $B_P \leq F^i_k$ where B_P is the bounding function associated with P as above. It then follows that the function computed by P lies in $\mathrm{Comp}(B_P)$ which is contained in $\mathrm{Comp}(F^i_k)$. The basis of the induction is trivial since L_0-programs terminate in a constant number of steps i so that $B_P(n) = n + i = F^i_0(n)$. For the induction step the crucial case is where P is a L_{k+1}-program of the form **for** $j = 1 \ldots x_m$ **do** Q **od** with $Q \in L_k$. By the IH there is a i such that $B_Q \leq F^i_k$ and hence, using $F_1(n) = 2n \leq F_{k+1}(n)$, we have

$$B_P(n) = B^n_Q(n) \leq F^{in}_k(n) \leq F_{k+1}(in) \leq F_{k+1}(2^{i-1}n) \leq F^i_{k+1}(n)$$

as required.

For the right-to-left containment, suppose $f \in \mathrm{Comp}(F^i_k)$ for some fixed i and k. Then there is a register machine which computes $f(\vec{n})$ within $F^i_k(\max(\vec{n}))$ steps. Now F_k is defined by k successive iterations (nested for-loops) starting with $F_0 = \mathrm{succ}$. So F_k is L_k-computable and (by composing i times) so is F^i_k. Therefore if $k \geq 1$ we can compute $f(\vec{n})$ by a L_k-program:

$$x := \max(\vec{n})\ \mathbf{;}\ y := F^i_k(x)\ \mathbf{;}\ \text{compute } y \text{ steps in the computation of f}$$

since, as we have already noted, an L_1 program suffices to perform any predetermined number of steps of a register machine program. This completes the proof. □

COROLLARY. *The "Ackermann-Péter Function" $F\colon \mathbb{N}^2 \to \mathbb{N}$ defined as*

$$F(k,n) = F_k(n)$$

is not primitive recursive.

PROOF. Since every loop-program has one of the F_k^i as a bounding function, it follows that every Prim function f is dominated by some F_k^i and therefore for all $n \geq \max(k+1, i)$ we have

$$f(n) < F_k^i(n) \leq F_k^n(n) = F_{k+1}(n) = F(k+1, n) \leq F(n, n).$$

Thus the binary function F cannot be primitive recursive, for otherwise we could take $f(n) = F(n, n)$ and obtain a contradiction. □

COROLLARY. *The elementary functions are just those definable by L_2-programs, since*

$$\text{Elem} = \bigcup_i \text{Comp}(F_2^i)$$

where $F_2(n) = n \cdot 2^n$.

PROOF. It is very easy to see that the elementary functions (like the primitive recursive ones) form an "honest" class in the sense that every elementary function is computable within a number of steps bounded by some (other) elementary function, and hence by some iterated exponential, and hence by F_2^i for some i. Conversely if $f \in \text{Comp}(F_2^i)$ then by the Normal Form Theorem there is a program number e such that for all $\vec{n}$,

$$f(\vec{n}) \;=\; U(\, e, \vec{n}, \mu_s T(e, \vec{n}, s)\,)$$

and furthermore the number of computation steps $\mu_s T(e, \vec{n}, s)$ is bounded elementarily by $F_2^i(\max(\vec{n}))$. Thus the unbounded minimization is in this case replaced by an elementarily bounded minimization, and since U and T are both elementary, so therefore is f. □

Grzegorczyk was the first to classify the primitive recursive functions by means of a hierarchy $\mathcal{E}^n$, which coincides with levels of L_k-computability for $n = k + 1 \geq 3$. In addition, $\mathcal{E}^2$ is the class of subelementary functions.

1.6. The Arithmetical Hierarchy

The goal of this section is to give a classification of the relations definable by arithmetical formulas. We have already made a step in this direction when we discussed the Σ_1^0-definable relations.

As a preparatory step we prove the Substitution Lemma and as its corollary the Fixed Point Lemma, also known as Kleene's Second Recursion Theorem.

1.6.1. Kleene's Second Recursion Theorem.

LEMMA (Substitution Lemma). *There is a binary elementary function S such that*

$$\varphi_e^{(q+1)}(m, \vec{n}) = \varphi_{S(e,m)}^{(q)}(\vec{n}).$$

PROOF. The details are left as an exercise; we only describe the basic idea here. To construct $S(e, m)$ we view e as code of a register machine program computing an $q+1$-ary function φ. Then $S(e, m)$ is to be a code of a register machine program computing the q-ary function obtained from φ by fixing its first argument to be m. So the program coded by $S(e, m)$ should work as follows. Shift all inputs one register to the right, and write m in the first register. Then compute as prescribed by e. □

THEOREM (Fixed Point Lemma or Kleene's Second Recursion Theorem). *Fix an arity q. Then for every e we can find an e_0 such that for all $\vec{n} = n_1, \ldots, n_r$*

$$\varphi_{e_0}^{(q)}(\vec{n}) = \varphi_e^{(q+1)}(e_0, \vec{n}).$$

PROOF. Let $\varphi_h(m, \vec{n}) = \varphi_e(S(m,m), \vec{n})$ and $e_0 := S(h,h)$. Then by the Substitution Lemma

$$\varphi_{e_0}(\vec{n}) = \varphi_{S(h,h)}(\vec{n}) = \varphi_h(h, \vec{n}) = \varphi_e(S(h,h), \vec{n}) = \varphi_e(e_0, \vec{n}). \qquad \square$$

1.6.2. Characterization of Σ_1^0-definable and recursive relations. We now give a useful characterization of the Σ_1^0-definable relations, which will lead us to the arithmetical hierarchy. Let

$$W_e^{(q)} := \{\, \vec{n} \mid \exists_s T(e, \vec{n}, s) \,\}.$$

The Σ_1^0-definable relations are also called *recursively enumerable* (r.e.) relations.

LEMMA. (a) *The $W_e^{(q)}$ enumerate for $e = 0, 1, 2, \ldots$ the q-ary Σ_1^0-definable relations.*

(b) *For fixed arity q, $W_e^{(q)}(\vec{n})$ as a relation of $e, \vec{n}$ is Σ_1^0-definable, but not recursive.*

PROOF. (a). If $R = W_e^{(q)}$, then R is Σ_0^1-definable by definition. For the converse assume that R is Σ_1^0-definable, i.e., that there is an elementary relation E, say of arity $q + r$, such that for all $\vec{n} = n_1, \ldots, n_q$,

$$R(\vec{n}) \;\leftrightarrow\; \exists_{k_1} \ldots \exists_{k_r} E(\vec{n}, k_1, \ldots, k_r).$$

Then clearly R is the domain of the partial recursion function φ given the following μ-recursive definition:

$$\varphi(\vec{n}) \;=\; \mu_m \,[\mathrm{lh}(m) = r \wedge E(\vec{n}, (m)_0, (m)_1, \ldots, (m)_{r-1})].$$

For $\varphi = \varphi_e$ we have by the Normal Form Theorem $R(\vec{n}) \leftrightarrow \exists_s T(e, \vec{n}, s)$.

(b) It suffices to show that $W_e(\vec{n})$ is not recursive. So assume that it would be. Then we could pick e_0 such that

$$W_{e_0}(e, \vec{n}) \;\leftrightarrow\; \neg W_e(e, \vec{n});$$

for $e = e_0$ we obtain a contradiction. $\square$

From the Substitution Lemma above we can immediately infer

$$W_e^{(q+1)}(m, \vec{n}) \;\leftrightarrow\; W_{S(e,m)}^{(q)}(\vec{n});$$

this fact is sometimes called *Substitution Lemma for Σ_1^0-definable relations*.

NOTE. We have already seen in 1.3.4 that a relation R is recursive if and only if both R and its complement $\neg R$ are Σ_1^0-definable.

1.6.3. Arithmetical relations. A relation R of arity q is said to be *arithmetical* if there is an elementary relation E, say of arity $q+r$, such that for all $\vec{n} = n_1, \ldots, n_q$,

$$R(\vec{n}) \;\leftrightarrow\; (Q_1)_{k_1} \ldots (Q_r)_{k_r}\, E(\vec{n}, k_1, \ldots, k_r) \quad \text{with } Q_i \in \{\forall, \exists\}.$$

Note that we may assume that the quantifiers Q_i are alternating, since e.g.

$$\forall_n \forall_m R(n,m) \;\leftrightarrow\; \forall_k R((k)_0, (k)_1).$$

A relation R of arity q is said to be Σ^0_r-*definable* if there is an elementary relation E such that for all $\vec{n}$,

$$R(\vec{n}) \;\leftrightarrow\; \exists_{k_1} \forall_{k_2} \ldots Q_{k_r}\, E(\vec{n}, k_1, \ldots, k_r)$$

with $Q = \forall$ if r is even and $Q = \exists$ if r is odd. Similary, a relation R of arity q is said to be Π^0_r-*definable* if there is an elementary relation E such that for all $\vec{n}$,

$$R(\vec{n}) \;\leftrightarrow\; \forall_{k_1} \exists_{k_2} \ldots Q_{k_r}\, E(\vec{n}, k_1, \ldots, k_r)$$

with $Q = \exists$ if r is even and $Q = \forall$ if r is odd. A relation R is said to be Δ^0_r-*definable* if it is Σ^0_r-definable as well as Π^0_r-definable.

A partial function φ is said to be *arithmetical* (Σ^0_r-*definable*, Π^0_r-*definable*, Δ^0_r-*definable*) if its graph $\{\, (\vec{n}, m) \mid \varphi(\vec{n}) \text{ is defined and } = m \,\}$ is.

By the note above a relation R is Δ^0_1-definable if and only if it is recursive.

Example. Let $\mathrm{Tot} := \{\, e \mid \varphi_e^{(1)} \text{ is total} \,\}$. Then we have

$$\begin{aligned} e \in \mathrm{Tot} \;&\leftrightarrow\; \varphi_e^{(1)} \text{ is total} \\ &\leftrightarrow\; \forall_n \exists_m (\varphi_e(n) = m) \\ &\leftrightarrow\; \forall_n \exists_m \exists_s (T(e,n,s) \wedge U(e,n,s) = m). \end{aligned}$$

Therefore Tot is Π^0_2-definable. We will show below that Tot is *not* Σ^0_2-definable.

1.6.4. Closure properties.

Lemma. *Σ^0_r, Π^0_r and Δ^0_r-definable relations are closed under conjunction, disjunction and bounded quantifiers $\exists_{m<n}$, $\forall_{m<n}$. The Δ^0_r-definable relations are closed against negation. Moreover, for $r > 0$ the Σ^0_r-definable relations are closed against the existential quantifier $\exists$ and the Π^0_r-definable relations are closed against the universal quantifier $\forall$.*

Proof. This can be seen easily. For instance, closure under the bounded universal quantifier $\forall_{m<n}$ follows from

$$\forall_{m<n} \exists_k R(\vec{n}, n, m, k) \;\leftrightarrow\; \exists_l \forall_{m<n} R(\vec{n}, n, m, (l)_m). \qquad \square$$

The relative positions of the Σ^0_r, Π^0_r and Δ^0_r-definable relations are shown in Fig. 1 on page 27.

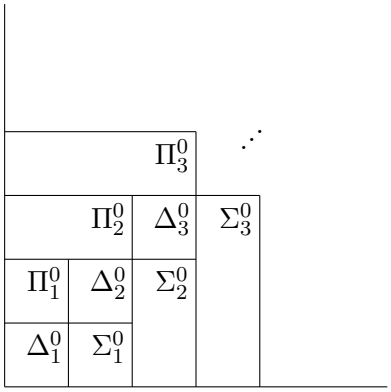

FIGURE 1. The arithmetical hierarchy

1.6.5. Universal Σ^0_{r+1}-definable relations. We now generalize the enumeration $W^{(q)}_e$ of the unary Σ^0_1-definable relations and construct bunary *universal* Σ^0_{r+1}-definable relations U^0_{r+1}:

$$U^0_1(e,n) :\leftrightarrow \exists_s T(e,n,s) \quad (\leftrightarrow\ n \in W^{(1)}_e),$$
$$U^0_{r+1}(e,n) :\leftrightarrow \exists_m \neg U^0_r(e, n * \langle m\rangle).$$

For example,

$$U^0_3(e,n) :\leftrightarrow \exists_{m_1}\forall_{m_2}\exists_s T(e, n * \langle m_1, m_2\rangle, s),$$
$$U^0_2(e,n) :\leftrightarrow \exists_m \forall_s \neg T(e, n * \langle m\rangle, s).$$

Clearly the relations $U^0_{r+1}(e, \langle \vec{n}\rangle)$ enumerate for $e = 0, 1, 2, \ldots$ the q-ary Σ^0_{r+1}-definable relations, and their complements the q-ary Π^0_{r+1}-definable relations,

Now it easily follows that all inclusions in Fig. 1 are proper. To see this, assume for example that $\exists_m\forall_s\neg T(e, \langle n, m\rangle, s)$ would be Π^0_2. Pick e_0 such that

$$\forall_m\exists_s T(e_0, \langle n, m\rangle, s) \ \leftrightarrow\ \exists_m\forall_s\neg T(n, \langle n, m\rangle, s);$$

for $n := e_0$ we obtain a contradiction. As another example, assume

$$A := \{\, 2\langle e, n\rangle \mid \exists_m\forall_s\neg T(e, \langle n, m\rangle, s)\,\} \cup \{\, 2\langle e, n\rangle + 1 \mid \forall_m\exists_s T(e, \langle n, m\rangle, s)\,\},$$

which is a Δ^0_3-set, would be Σ^0_2. Then we would have a contradiction

$$\forall_m\exists_s T(e, \langle n, m\rangle, s) \ \leftrightarrow\ 2\langle e, n\rangle + 1 \in A,$$

and hence $\{\, (e, n) \mid \forall_m\exists_s T(e, \langle n, m\rangle, s)\,\}$ would be a Σ^0_2-definable relation, a contradiction.

1.6.6. Σ^0_r-complete relations. We now develop an easy method to obtain precise classifications in the arithmetical hierarchy. Since by sequence-coding we can pass in an elementary way between relations R of arity q and relations $R'(n) \leftrightarrow R((n)_1, \ldots, (n)_q)$ of arity 1, it is no real loss of generality if we henceforth restrict to $q = 1$ and only deal with sets $A, B \subseteq \mathbb{N}$ (i.e., unary relations). First we introduce the notion of (many-one) reducibility.

Let $A, B \subseteq \mathbb{N}$. B is said to be *reducible* to A if there is a total recursive function f such that for all n

$$n \in B \;\leftrightarrow\; f(n) \in A.$$

A set A is said to be Σ^0_r-*complete* if

(1) A is Σ^0_r-definable, and
(2) every Σ^0_r-definable set B is reducible to A.

LEMMA. *If A is Σ^0_r-complete, then A is Σ^0_r-definable but not Π^0_r-definable.*

PROOF. Let A be Σ^0_r-complete and assume that A is Π^0_r-definable. Pick a set B which is Σ^0_r-definable but not Π^0_r-definable. By Σ^0_r-completeness of A the set B is reducible to A via a recursive function f:

$$n \in B \;\leftrightarrow\; f(n) \in A.$$

But then B would be Π^0_r-definable too, contradicting the choice of B. □

REMARK. In the definition and the lemma above we can replace Σ^0_r by Π^0_r. This gives the notion of Π^0_1-completeness, and the proposition that every Π^0_r-complete set A is Π^0_r-definable but not Σ^0_r-definable.

EXAMPLE. We have seen above that the set $\mathrm{Tot} := \{\, e \mid \varphi^{(1)}_e \text{ is total} \,\}$ is Π^0_2-definable. We now can show that Tot is *not* Σ^0_2-definable. By the lemma it suffices to prove that Tot is Π^0_2-complete. So let B be an arbitrary Π^0_2-definable set. Then, for some $e \in \mathbb{N}$,

$$n \in B \;\leftrightarrow\; \forall_m \exists_s T(e, n, m, s).$$

Consider the partial recursive function

$$\varphi_e(n, m) := U(e, n, m, \mu_s T(e, n, m, s)).$$

By the Substitution Lemma we have

$$\begin{aligned} n \in B &\leftrightarrow \forall_m(\varphi_e(n, m) \text{ is defined}) \\ &\leftrightarrow \forall_m(\varphi_{S(e,n)}(m) \text{ is defined}) \\ &\leftrightarrow \varphi_{S(e,n)} \text{ is total} \\ &\leftrightarrow S(e, n) \in \mathrm{Tot}. \end{aligned}$$

Therefore B is reducible to Tot.

CHAPTER 2

Constructive Ordinals

The complexity hierarchy for the primitive recursive functions was indexed by the natural numbers. Later we will consider other hierarchies of functions, which cannot be indexed by $\mathbb{N}$ any more, but need the ordinals instead, or more precisely the constructive ordinals. An ordinal is said to be constructive simply if it can be represented by a recursive well-ordering of $\mathbb{N}$. However, for a closer study of the recursive ordinals some preparations are necessary. First of all, we need to extend our treatment of computable functions to *functionals*, i.e., we need to also consider function arguments. This will be done in the first section below. Then we generalize the arithmetical hierarchy to the *analytical* hierarchy, by also allowing function quantifiers. After the definition and initial discussion of the constructive ordinals we define a first hierarchy – due to Kleene – of recursive functionals indexed by the constructive ordinals. Finally we determine the exact location in the analytical hierarchy of various predicates concerning the constructive ordinals.

2.1. Relative Recursiveness

2.1.1. Oracles. We now extend our treatment of computable functions to functionals. Again we start with register machines, since this provides the most intuitive approach. However, this time we suppose that an additional "given function" g is available, whose values are to be supplied immediately (in one step, by a magic "oracle") whenever they are called for by a register machine program with a new kind of instruction:

$$\textit{Oracle.}\ \ x := g(x).$$

Oracle register machine programs can be numbered in just the same way as before (taking $\langle 3, i\rangle$ as the number of the oracle instruction where x is the i-th register variable).

We let $\Phi_e^{(r)}(g)$ denote the partial function of arity r computed relative to g by the oracle program with index e. Since by sequence-coding we can pass in an elementary way between functions ψ of arity r and functions $\psi'(n) = \psi((n)_1, \ldots, (n)_r)$ of arity 1, it is no real loss of generality if we restrict to $r = 1$ and write simply $\Phi_e(g)$ instead of $\Phi_e^{(1)}(g)$.

2.1.2. Relativized normal form.

THEOREM (Relativized Normal Form). *There is an elementary relation T_1 and an elementary function U_1 such that for all oracle program numbers e, inputs n, and oracle functions g,*

- $\Phi_e(g)(n)$ *is defined* $\leftrightarrow\ \exists_s T_1(e, n, \overline{g}(s))$, *and*

- $\Phi_e(g)(n) \;=\; U_1(e, n, \overline{g}(s_0))$ *where* $s_0 = \mu_s\, T_1(e, n, \overline{g}(s))$.

In the above we take $\overline{g}(s) := \langle g(0), g(1), \ldots, g(s-1)\rangle$.

PROOF. If for arbitrary but fixed e, g and n, $\Phi_e(g)(n)$ is defined, then the computation will only make finitely many oracle calls, all of which will be contained in the sequence number $m = \overline{g}(s)$ provided s is large enough. In that case every oracle call $x := g(x)$ could be replaced by a non-oracle subroutine:

$$\textbf{if } x < \mathrm{lh}(m) \textbf{ then } x := (m)_x \textbf{ else } \text{undefined } \textbf{fi}$$

and the resulting non-oracle program, say with index e_1, would compute the same value. However it now has two numerical arguments n and m instead of one numerical argument n and one oracle g. Thus if the sequence number $m = \overline{g}(s)$ is long enough then

$$\Phi_e(g)(n) \text{ defined} \Rightarrow\; \Phi_e(g)(n) = \varphi^{(2)}_{e_1}(n, m).$$

The above transformation from oracle program to non-oracle program simply inserts the fixed subroutine above, in place of each oracle instruction in the original program, so the corresponding index constructor $e, m \mapsto e_1$ is an elementary function. Now from the T predicate of the original Normal Form Theorem, define

$$T_1(e, n, m) \;:\leftrightarrow\; T(e_1, n, m, \mathrm{lh}(m)).$$

Clearly T_1 is elementary since T is. Furthermore $T_1(e, n, \overline{g}(s))$ holds if and only if the computation of $\varphi_{e_1}(n, \overline{g}(s))$ terminates by step s. But this computation simulates the oracle computation of $\Phi_e(g)(n)$. Therefore $\Phi_e(g)(n)$ is defined if and only if $T_1(e, n, \overline{g}(s))$ holds for some (large enough) s, and in that case its value is $U(e_1, n, \overline{g}(s), s)$ for any such s. So if we define

$$U_1(e, n, m) \;:=\; U(e_1, n, m, \mathrm{lh}(m))$$

then U_1 is elementary too, and it clearly has the required property. □

Clearly we can generalize this to the case where arbitrary many function and number arguments are admitted. We write

$$\Phi_e(\vec{g}, \vec{n}) = \Phi^{(p,q)}_e(g_1, \ldots, g_p, n_1, \ldots, n_q)$$

for the functional computed by an oracle register machine program with program number e, where the function variables are $g_1, \ldots, g_p$ and the number (input) variables are $n_1, \ldots, n_q$.

We now give a second form of the relativized normal form theorem, where we allow function arguments to appear explicitly. Observe first that it is obvious how to extend the notion of elementary functions and relations to also allow function arguments.

COROLLARY (Relativized Normal Form, Second Form). *There is an elementary relation* T_2 *(possibly with function arguments) and an elementary functional* U_2 *such that for all oracle program numbers* e*, inputs* n*, and oracle functions* g*,*

- $\Phi_e(\vec{g}, \vec{n})$ *is defined* $\leftrightarrow\; \exists_s\, T_2(e, \vec{g}, \vec{n}, s)$*, and*
- $\Phi_e(\vec{g}, \vec{n}) \;=\; U_2(e, \vec{g}, \vec{n}, \mu_s\, T_2(e, \vec{g}, \vec{n}, s))$.

PROOF. For simplicity we assume $\vec{g} = g$ and $\vec{n} = n$. Define

$$T_2(e, g, n, s) :\leftrightarrow T_1(e, n, \overline{g}(s)) \quad (\leftrightarrow T(e_1, n, \overline{g}(s), s)),$$
$$U_2(e, g, n, s) :\leftrightarrow U_1(e, n, \overline{g}(s)) \quad (\leftrightarrow U(e_1, n, \overline{g}(s), s)).$$

Then claim then follows from the Relativized Normal Form Theorem. □

COROLLARY. *The functional $\Phi_e^{(p,q)}(g_1, \ldots, g_p, n_1, \ldots, n_q)$ is a computable partial functional of the p function variables $g_1, \ldots, g_p$ and the $q + 1$ number variables $e, n_1, \ldots, n_q$.*

PROOF. Immediate from the normal form. □

2.1.3. Relativization of Σ_1^0-definable relations. The definition of Σ_1^0-definable relations can now be extended in the obvious way to also include function arguments. The same is true for μ-recursive functions (to be called μ-recursive functionals) and while-programs. With the same proof as before we then have as a corollary to the Relativized Normal Form Theorem:

COROLLARY. *The following properties of partial functionals Φ are equivalent:*

- *Φ is register machine computable,*
- *Φ is Σ_1^0-definable,*
- *Φ is μ-recursive,*
- *Φ is computable by a while program.*

Recall that recursive programs define the computable functions by successive recursive definitions in which the three basic functions: successor, predecessor and definition-by-cases, are assumed given. Again we can introduce an additional given function g, whose values are to be supplied immediately (by our magic "oracle") whenever they are called for by any program. The functions defined by recursive programs with this new oracle are said to be *partial recursive in (or relative to) g*. By the previously established equivalences, which again relativize to g without much change, this is the same as saying that the function is computable by a register machine with the new kind of oracle instruction $x := g(x)$. Clearly this notion of a partial recursive functional extends to the case where we have p function arguments $\vec{g} = g_1, \ldots, g_p$ and q number arguments $\vec{n} = n_1, \ldots, n_q$.

It will be useful below to add some more characterizations of the Σ_1^0-definable relations:

LEMMA. *For a total relation R the following are equivalent:*

(a) *R is the domain of a partial recursive functional.*
(b) *R can be written in the form $R(\vec{g}, \vec{n}) \leftrightarrow \exists_m \Gamma(\vec{g}, \vec{n}, m)$ with a partial recursive relation Γ.*
(c) *R can be written in the form $R(\vec{g}, \vec{n}) \leftrightarrow \exists_m Q(\vec{g}, \vec{n}, m)$ with a (total) recursive relation Q.*
(d) *R can be written in the form $R(\vec{g}, \vec{n}) \leftrightarrow \exists_m E(\vec{g}, \vec{n}, m)$ with an elementary relation E.*
(e) *R can be written in the form $R(\vec{g}, \vec{n}) \leftrightarrow \exists_s T_2(e, \vec{g}, \vec{n}, s)$.*

PROOF. From (a) we get (e), since

$$R(\vec{g}, \vec{n}) \leftrightarrow \exists_s T_2(e, \vec{g}, \vec{n}, s).$$

From (e) we obviously obtain (d), (c) and then (b). To get from (b) to (a) note that

$$\begin{aligned} R(\vec{g},\vec{n}) &\leftrightarrow \exists_m\,(\Phi_e(\vec{g},\vec{n},m)=1)\\ &\leftrightarrow \exists_m\exists_s[T_2(e,\vec{g},\vec{n},m,s)\wedge U_2(e,\vec{g},\vec{n},m,s)=1]\\ &\leftrightarrow \exists_m\exists_s\,E(e,\vec{g},\vec{n},m,s)\quad \text{for an elementary relation } E\\ &\leftrightarrow \mu_k\,E(e,\vec{g},\vec{n},(k)_0,(k)_1)\quad \text{is defined.} \end{aligned}$$ □

REMARK. Note that a relation R is Σ^0_1-definable if and only if it can be written in the form

$$R(g,n)\ \leftrightarrow\ \exists_s\,T_1(e,n,\overline{g}(s));$$

this follows immediately from the Relativized Normal Form Theorem. Hence one can develop from the theory of recursive functions first the theory of Σ^0_1-definable relations with function arguments and then the theory of partial recursive functionals. This is the route followed by Shoenfield [1967]. Here we have preferred a more intuitive approach, using oracle register machines.

2.1.4. Substitution in function arguments.

LEMMA. *For all partial recursive functionals Φ, Ψ_1, ..., Ψ_p we can find a partial recursive functional Φ' such that for all $\vec{g},\vec{n}$ with $\lambda x\Psi_i(\vec{g},\vec{n},x)$ total we have*

$$\Phi'(\vec{g},\vec{n})\ =\ \Phi(\lambda x\Psi_1(\vec{g},\vec{n},x),\dots,\lambda x\Psi_p(\vec{g},\vec{n},x),\vec{n}).$$

PROOF. Induction on the definition of Φ as a μ-recursive functional. □

REMARK. The lemma is false without the restriction on the arguments $\vec{g},\vec{n}$ For otherwise the partial function φ defined by

$$\begin{aligned} \varphi(e) &:= 0^2(\lambda x\varphi_e^{(1)}(x))\quad \text{with } 0^2(g):=0\\ &= \begin{cases} 0 & \text{if } e\in \mathrm{Tot}\\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned}$$

would be partial recursive, and cleary Tot is the domain of φ. However, this contradicts the fact observed above that Tot is Π^0_2-complete, hence not Σ^0_1-definable.

2.1.5. Post's theorem. We now prove a well-known theorem of Post, which gives a characterization of the arithmetical hierarchy for sets in terms of the notions "Σ^0_1-definable in" and "recursive in".

THEOREM (Post). *For every set $A\subseteq\mathbb{N}$ the following are equivalent.*

(a) *A is Σ^0_{r+1}-definable.*
(b) *A is Σ^0_1-definable in B for some Σ^0_r-definable set $B\subseteq\mathbb{N}$.*
(c) *A is Σ^0_1-definable in B for some Π^0_r-definable set $B\subseteq\mathbb{N}$.*
(d) *A is Σ^0_1-definable in B for some Δ^0_{r+1}-definable set $B\subseteq\mathbb{N}$.*

PROOF. The equivalence of (b) and (c) is clear. (a) implies (b), since by (a)

$$n\in A\ \leftrightarrow\ \exists_m\neg R(n,m)$$

for some Σ^0_r-definable relation R, hence A is Σ^0_1-definable in B with $n \in B :\leftrightarrow R((n)_0, (n)_1)$. Clearly (b) implies (d). To see that (d) implies (a) observe that

$$n \in A \leftrightarrow \exists_m R(\overline{c_B}(m), n, m) \quad \text{with } B\ \Delta^0_{r+1}\text{-definable and } R \text{ recursive}$$
$$\leftrightarrow \exists_m \exists_k \big(k = \overline{c_B}(m) \wedge R(k, n, m)\big).$$

Now $k = \overline{c_B}(m)$ can be written in the form

$$k = \overline{c_B}(m) \ \leftrightarrow\ \mathrm{lh}(k) = m \wedge \forall_{i<m}[(B(i) \to (k)_i = 1) \wedge (\neg B(i) \to (k)_i = 0)].$$

Then claim then follows since by assumption both B and $\neg B$ are Σ^0_{r+1}-definable. □

Corollary (Post). *For sets $A, B \subseteq \mathbb{N}$ the following propositions are equivalent.*

(a) *A is Δ^0_{r+1}-definable.*
(b) *A is recursive in B for some Σ^0_r-definable B.*
(c) *A is recursive in B for some Π^0_r-definable B.*

Proof. The equivalence of (b) and (c) is clear. To see the equivalence of (a) and (b), observe that the following are equivalent.

A is Δ^0_{r+1}-definable

$A, \neg A$ are Σ^0_{r+1}-definable

$A, \neg A$ are Σ^0_1-definable in (we may assume) the same Σ^0_r-definable set B

A is recursive in B for some Σ^0_r-definable B.

The last equivalence follows from the relativised form of a note in 1.6.2. □

2.1.6. Reducibility. The notion of f being recursive in g gives rise to a quite natural equivalence relation on unary functions: call f and g equivalent (written $f \sim g$) if f is recursive in g and g is recursive in f. The equivalence classes with respect to this relation are called *recursive degrees*. There is a rich literature on such degrees (for instance [Soare, 1987]); however, we will not embark into this subject here.

We now want to relate the notion of reducibility introduced above to the notion of relative recursiveness. Recall that a set B is said to be reducible to a set A if there is a total recursive function f such that for all n

$$n \in B \ \leftrightarrow\ f(n) \in A. \tag{2.1}$$

Therefore if B is reducible to A, then B is recursive in A. However, the converse is false To see this, pick a Σ^0_1-definable set A that is not recursive, and let $B := \neg A$. Then clearly B is recursive in A. Now if B would be reducible to A, then by (2.1) B would be Σ^0_1-definable, contradicting the assumption on A.

For a fixed set A let us now consider all sets B which are Σ^0_1-definable in A. We will show that amongst those there is a maximal one with respect to reducibility. Let A', the *recursive successor* or *jump* of A, be defined by

$$n \in A' \ :\leftrightarrow\ \exists_s T_2((n)_0, c_A, (n)_1, s).$$

Theorem. *A set B is Σ^0_1-definable in a set A if and only if B is reducible to A'.*

PROOF. If B is Σ^0_1-definable in a set A, then

$$n \in B \;\leftrightarrow\; \exists_s T_2(e, c_A, n, s) \;\leftrightarrow\; \langle e, n\rangle \in A'.$$

Conversely, assume that B is reducible to A'. Then for some recursive function f we have

$$n \in B \;\leftrightarrow\; f(n) \in A' \;\leftrightarrow\; \exists_s T_2((f(n))_0, c_A, (f(n))_1, s)$$

and hence B is Σ^0_1-definable in A. □

Note that as a consequence A' cannot be recursive in A. For if it were, then by the theorem every set B Σ^0_1-definable in A would be recursive in A. This contradicts the relativization of the theorem asserting the existence of Σ^0_1-definable, but not recursive sets.

2.1.7. Jumps. We give a characterization of the arithmetical hierarchy for sets in terms of the iterated successors or jumps $\emptyset^{(r)}$ of the empty set, defined by

$$\emptyset^{(0)} := \emptyset, \quad \emptyset^{(r+1)} := \big(\emptyset^{(r)}\big)'.$$

LEMMA. *$\emptyset^{(r)}$ is Σ^0_r-definable.*

PROOF. We use induction on r. By the definition of A' we have

$$n \in \emptyset^{(r+1)} \;\leftrightarrow\; \exists_s T_2((n)_0, c_{\emptyset^{(r)}}, (n)_1, s).$$

Hence $\emptyset^{(r+1)}$ is Σ^0_1-definable in $\emptyset^{(r)}$, which by IH is Σ^0_r-definable (or recursive if $r = 0$). Therefore by Post's Theorem $\emptyset^{(r+1)}$ is Σ^0_{r+1}-definable. □

THEOREM. *For every set $A \subseteq \mathbb{N}$ the following are equivalent:*
(a) *A is Σ^0_{r+1}-definable.*
(b) *A is Σ^0_1-definable in $\emptyset^{(r)}$.*
(c) *A is reducible to $\emptyset^{(r+1)}$.*

PROOF. The equivalence of (b) and (c) is a consequence of the theorem above (on A'). The implication from (b) to (a) follows from Post's Theorem with the lemma above. For the implication from (a) to (b) we use induction on r. For $r = 0$ this is obvious, and in case $r + 1$ we have

$$n \in A \;\leftrightarrow\; \exists_m \neg R(n, m)$$

with a Σ^0_r-definable relation R. Now by II $R(n, m)$ can be written in the form

$$R(n, m) \;\leftrightarrow\; f(n, m) \in \emptyset^{(r)}$$

with a recursive function f. Therefore A is Σ^0_1-definable in $\emptyset^{(r)}$. □

COROLLARY. *For every set $A \subseteq \mathbb{N}$ the following are equivalent:*
(a) *A is Δ^0_{r+1}-definable.*
(b) *A is recursive in $\emptyset^{(r)}$.*

2.2. The Analytical Hierarchy

We now generalize the arithmetical hierarchy and give a classification of the relations definable by analytical formulas, i.e., formulas involving number as well as function quantifiers.

2.2.1. Analytical relations. First note that the Substitution Lemma as well as the Fixed Point Lemma in 1.6.1 continue to hold if function arguments are present, with the same function S in the Substitution Lemma. We also extend the enumeration $W_e^{(q)}$ of the Σ_1^0-definable relations: By part (e) of the lemma in 2.1.3 the sets

$$W_e^{(p,q)} := \{\, (\vec{g}, \vec{n}) \mid \exists_s T_2(e, \vec{g}, \vec{n}, s) \,\}$$

enumerate for $e = 0, 1, 2, \ldots$ the (p, q)-ary Σ_1^0-definable relations. With the same argument as in Sec.1.6 we see that for fixed arity (p, q), $W_e^{(p,q)}(\vec{g}, \vec{n})$ as a relation of $\vec{g}, e, \vec{n}$ is Σ_1^0-definable, but not recursive. The treatment of the arithmetical hierarchy can now be extended without difficulties to (p, q)-ary relations.

EXAMPLES. (a) The set $\mathcal{R}$ of all recursive functions is Σ_3^0-definable, since

$$\mathcal{R}(f) \;\leftrightarrow\; \exists_e \forall_n \exists_s [T(e, n, s) \wedge U(e, n, s) = f(n)].$$

(b) Let LinOrd denote the set of all functions f such that

$$\leq_f \;:=\; \{\, (n, m) \mid f\langle n, m\rangle = 1 \,\}$$

is a linear ordering of its *field* $M_f := \{\, n \mid \exists_m (f\langle n, m\rangle = 1 \vee f\langle m, n\rangle = 1) \,\}$. LinOrd is Π_1^0-definable, since

$$\begin{aligned} \mathrm{LinOrd}(f) \leftrightarrow\; & \forall_n (n \in M_f \to f\langle n, n\rangle = 1) \wedge \\ & \forall_{n,m} (f\langle n, m\rangle = 1 \wedge f\langle m, n\rangle = 1 \to n = m)) \wedge \\ & \forall_{n,m,k} (f\langle n, m\rangle = 1 \wedge f\langle m, k\rangle = 1 \to f\langle n, k\rangle = 1) \wedge \\ & \forall_{n,m} (n, m \in M_f \to f\langle n, m\rangle = 1 \vee f\langle m, n\rangle = 1). \end{aligned}$$

Here we have written $n \in M_f$ for $\exists_m (f\langle n, m\rangle = 1 \vee f\langle m, n\rangle = 1)$.

A relation R of arity (p, q) is said to be *analytical* if there is an arithmetical relation P, say of arity $(r + p, q)$, such that for all $\vec{g} = g_1, \ldots, g_p$ and $\vec{n} = n_1, \ldots, n_q$,

$$R(\vec{g}, \vec{n}) \;\leftrightarrow\; (Q_1)_{f_1} \ldots (Q_r)_{f_r}\, P(f_1, \ldots, f_r, \vec{g}, \vec{n}) \quad \text{with } Q_i \in \{\forall, \exists\}.$$

Note that we may assume that the quantifiers Q_i are alternating, since for instance

$$\forall_f \forall_g R(f, g) \;\leftrightarrow\; \forall_h R((h)_0, (h)_1),$$

where $(h)_i(n) := (h(n))_i$. A relation R of arity (p, q) is said to be Σ_r^1-*definable* if there is an $(r + p, q)$-ary arithmetical relation P such that for all $\vec{g}, \vec{n}$,

$$R(\vec{g}, \vec{n}) \;\leftrightarrow\; \exists_{f_1} \forall_{f_2} \ldots Q_{f_r}\, P(f_1, \ldots, f_r, \vec{g}, \vec{n})$$

with $Q = \forall$ if r is even and $Q = \exists$ if r is odd. Similarly, a relation R of arity (p, q) is said to be Π_r^1-*definable* if there is an arithmetical relation P such that for all $\vec{g}, \vec{n}$,

$$R(\vec{g}, \vec{n}) \;\leftrightarrow\; \forall_{f_1} \exists_{f_2} \ldots Q_{f_r}\, P(f_1, \ldots, f_r, \vec{g}, \vec{n})$$

with $Q = \exists$ if r is even and $Q = \forall$ if r is odd. A relation R is said to be Δ_r^1-*definable* if it is Σ_r^1-definable as well as Π_r^1-definable.

A partial functional Φ is said to be *analytical* (Σ_r^1-*definable*, Π_r^1-*definable*, Δ_r^1-*definable*) if its graph $\{\, (\vec{g}, \vec{n}, m) \mid \Phi(\vec{g}, \vec{n})$ is defined and $= m \,\}$ is.

LEMMA. *A relation R is Σ^1_r-definable if and only if it can be written in the form*

$$R(\vec{g},\vec{n}) \leftrightarrow \exists_{f_1}\forall_{f_2}\dots Q_{f_r}\overline{Q}_m\, P(f_1,\dots,f_r,\vec{g},\vec{n},m)$$
$$\text{with } Q\in\{\forall,\exists\} \text{ and } \overline{Q} := \begin{cases}\exists & \text{if } Q=\forall\\ \forall & \text{if } Q=\exists\end{cases}$$

with an elementary relation P. Similarly, a relation R is Π^1_r-definable if and only if it can be written in the form

$$R(\vec{g},\vec{n}) \leftrightarrow \forall_{f_1}\exists_{f_2}\dots Q_{f_r}\overline{Q}_m\, P(f_1,\dots,f_r,\vec{g},\vec{n},m)$$

with $Q,\overline{Q}$ as above and an elementary relation P.

PROOF. Use

$$\forall_n\exists_f R(f,n)\leftrightarrow\exists_g\forall_n R((g)_n,n) \quad \text{with } (g)_n(m):=f\langle n,m\rangle,$$
$$\forall_n R(n)\leftrightarrow\forall_f R(f(0)).$$

E.g., the prefix $\forall_f\exists_n\forall_m$ is transformed first into $\forall_f\exists_n\forall_g$, then into $\forall_f\forall_h\exists_n$, and finally into $\forall_g\exists_n$. □

EXAMPLE. Define

$$\mathrm{WOrd}(f) :\leftrightarrow (\leq_f \text{ is a well-ordering of its field } M_f).$$

Then WOrd satisfies

$$\mathrm{WOrd}(f)\leftrightarrow \mathrm{LinOrd}(f)\wedge\forall_g[\forall_n f\langle g(n{+}1),g(n)\rangle)=1\rightarrow\exists_m g(m{+}1)=g(m)].$$

Hence WOrd is Π^1_1-definable.

2.2.2. Closure Properties.

LEMMA (Closure Properties). *The Σ^1_r, Π^1_r and Δ^1_r-definable relations are closed against conjunction, disjunction and numerical quantifiers $\exists_n$, $\forall_n$. The Δ^1_r-definable relations are closed against negation. Moreover, for $r>0$ the Σ^1_r-definable relations are closed against the existential function quantifier $\exists_f$ and the Π^1_r-definable relations are closed against the universal function quantifier $\forall_f$.*

PROOF. This can be seen easily. For instance, closure of the Σ^1_1-definable relations against universal numerical quantifiers follows from the transformation of $\forall_n\exists_f\forall_m$ first into $\exists_g\forall_n\forall_m$ and then into $\exists_g\forall_k$. □

The relative positions of the Σ^1_r, Π^1_r and Δ^1_r-definable relations are shown in Fig. 1 on page 37. Here

$$\Delta^0_\infty := \bigcup_{r\geq 1}\Sigma^0_r \quad \left(=\bigcup_{r\geq 1}\Pi^0_r\right)$$

is the set of all arithmetical relations, and

$$\Delta^1_\infty := \bigcup_{r\geq 1}\Sigma^1_r \quad \left(=\bigcup_{r\geq 1}\Pi^1_r\right)$$

is the set of all analytical relations.

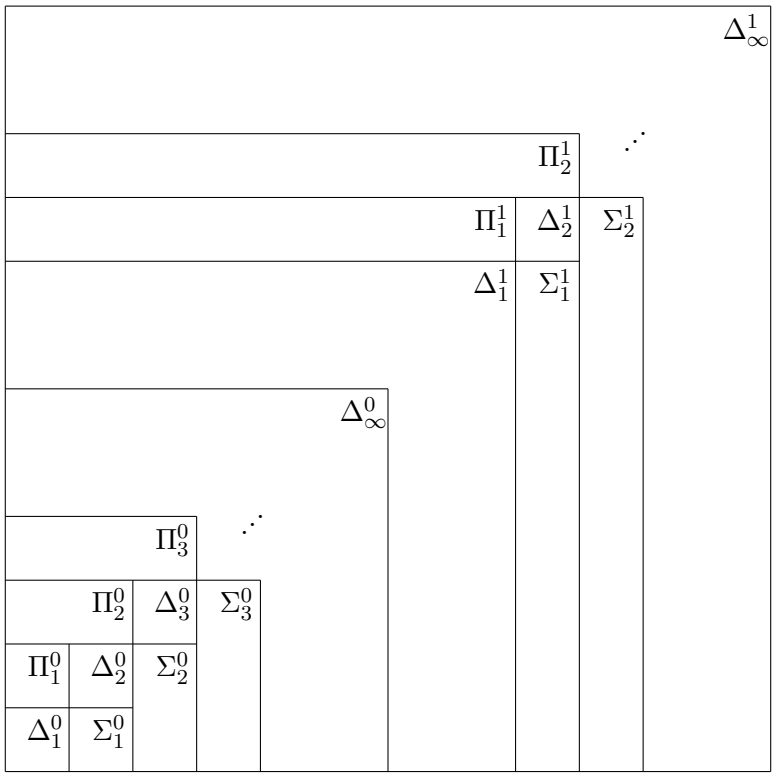

FIGURE 1. The analytical hierarchy

2.2.3. Universal Σ^1_{r+1}-definable relations.

LEMMA (Universal relations). *Among the Σ^1_{r+1} (Π^1_{r+1})-definable relations there is a $(p, q+1)$-ary relation enumerating all (p, q)-ary Σ^1_{r+1} (Π^1_{r+1}) -definable relations.*

PROOF. As an example, we prove the lemma for Σ^1_2 and Σ^1_1. All Σ^1_2-definable relations are enumerated by

$$\exists_g \forall_h \exists_s T_2(e, \vec{f}, \vec{n}, g, h, s),$$

and all Σ^1_1-definable relations are enumerated by

$$\exists_g \forall_s \neg T_2(e, \vec{f}, \vec{n}, g, s). \qquad \square$$

LEMMA. *All inclusions in Fig.1 on page 37 are proper.*

PROOF. We postpone (to 2.4.8) the proof of $\Delta^0_\infty \subsetneq \Delta^1_1$. The rest of the proof is obvious from the following examples. Assume $\exists_g \forall_h \exists_s T_2(e, n, g, h, s)$ would be Π^1_2. Pick e_0 such that

$$\forall_g \exists_h \forall_s \neg T_2(e_0, n, g, h, s) \;\leftrightarrow\; \exists_g \forall_h \exists_s T_2(n, n, g, h, s);$$

for $n := e_0$ we obtain a contradiction. As another example, assume

$$\begin{aligned} A := &\{\, 2\langle e, n\rangle \mid \exists_g \forall_h \exists_s T_2(e, n, g, h, s) \,\} \cup \\ &\{\, 2\langle e, n\rangle + 1 \mid \forall_g \exists_h \forall_s \neg T_2(e, n, g, h, s) \,\}, \end{aligned}$$

which is a Δ^1_3-set, would be Σ^1_2. Then from

$$\forall_g \exists_h \forall_s \neg T_2(e, n, g, h, s) \;\leftrightarrow\; 2\langle e, n\rangle + 1 \in A,$$

it would follow that $\{\, (e, n) \mid \forall_g \exists_h \forall_s \neg T_2(e, n, g, h, s) \,\}$ is a Σ^1_2-definable relation, a contradiction. □

2.2.4. Σ^1_r-complete relations. A set $A \subseteq \mathbb{N}$ is said to be Σ^1_r-*complete* if

(1) A is Σ^1_r-definable, and
(2) every Σ^1_r-definable set $B \subseteq \mathbb{N}$ is reducible to A.

LEMMA. *If $A \subseteq \mathbb{N}$ is Σ^1_r-complete, then A is Σ^1_r-definable but not Π^1_r-definable.*

PROOF. Let A be Σ^1_r-complete and assume that A is Π^1_r-definable. Pick a set $B \subseteq \mathbb{N}$ which is Σ^1_r-definable but not Π^1_r-definable. By Σ^1_r-completeness of A the set B is reducible to A via a recursive function f:

$$n \in B \;\leftrightarrow\; f(n) \in A.$$

But then B would be Π^1_r-definable too, contradicting the choice of B. □

REMARK. In the definition and the lemma above we can replace Σ^1_r by Π^1_r. This gives the notion of Π^1_r-completeness, and the proposition that every Π^1_r-complete set A is Π^1_r-definable but not Σ^1_r-definable.

2.3. Recursive Type-2 Functionals and Wellfoundedness

2.3.1. Computation trees. To each oracle program with index e, associate its "tree of non-past-secured sequence numbers":

$$\mathrm{Tree}(e) \;:=\; \{\, \langle n_0, \ldots, n_{l-1}\rangle \mid \forall_{k<l}\, \neg T_1(e, n_0, \langle n_1, \ldots, n_{k-1}\rangle) \,\}$$

called the *computation tree* of the given program.

We imagine the computation tree as growing downwards by extension, that is if σ and τ are any two sequence numbers (or nodes) in the tree then σ comes below τ if and only if σ is a proper extension of τ, that is $\mathrm{lh}(\tau) < \mathrm{lh}(\sigma)$ and $\forall_{i<\mathrm{lh}(\tau)}((\sigma)_i = (\tau)_i)$. We write $\sigma \supset \tau$ to denote this. Note that if σ is in the tree and $\sigma \supset \tau$ then τ is automatically in the tree, by definition. An infinite branch of the tree is thus determined by a number n and a function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\forall_s \neg T_1(e, n, \overline{g}(s))$. Therefore by the Relativized Normal Form in 2.1.2, an infinite branch is a witness to the fact that for some n and some g, $\Phi_e(g)(n)$ is not defined. To say that the tree is "well founded" is to say that there are no infinite branches, and hence:

THEOREM. *Φ_e is total if and only if* $\mathrm{Tree}(e)$ *is well-founded.*

2.3.2. Ordinal assignments; recursive ordinals. This equivalence is the basis for a natural theory of ordinal assignments, measuring (in some sense) the "complexity" of those oracle programs which terminate "everywhere" (on all oracles and all numerical inputs). We shall later investigate in some detail these ordinal assignments and the ways in which they measure complexity, but to begin with we shall merely describe the hierarchy which immediately arises. It is due to Kleene (1958), but appears there only as a brief footnote to the first page.

DEFINITION. If Tree(e) is well-founded we can assign to each of its nodes τ an ordinal $\|\tau\|$ by recursion "up the tree" as follows: if τ is a terminal node (no extension of it belongs to the tree) then $\|\tau\| = 0$; otherwise $\|\tau\| = \sup\{\, \|\sigma\| + 1 \mid \sigma \supset \tau \wedge \sigma \in \mathrm{Tree}(e) \,\}$.

Then we can assign an ordinal to the whole tree by defining $\|e\| := \|\langle\rangle\|$.

EXAMPLE. The for-loop (with input variable x and output variable y):

$$y := 0 \ \mathbf{;}\ \mathbf{for}\ i = 1 \ldots x\ \mathbf{do}\ y := g(y)\ \mathbf{od}$$

computes the iteration functional $It(g)(n) = g^n(0)$. For fixed g and n the branch through its computation tree will terminate in a node

$$\langle n, g(0), \ldots, g^2(0), \ldots, g^{n-1}(0), \ldots, g^n(0), \ldots, g(s-1)\rangle$$

where s is the least number such that (i) $\overline{g}(s)$ contains all the necessary oracle information concerning g, so $s > g^{n-1}(0)$, and (ii) computation of the program terminates by step s.

Working down this g-branch (and remembering that g is any function at all) we see that for $i < n$, once the value of $g^i(0)$ is chosen, it determines the length of the ensuing segment as far as $g^{i+1}(0)$. The greater the value of $g^i(0)$, the greater is the length of this segment. Therefore as we take the supremum over all branches issuing from a node

$$\langle n, g(0), \ldots, g^2(0), \ldots, g(g^{i-1}(0) - 1)\rangle$$

the successive segments $g^i(0), \ldots, g^{i+1}(0)$ have unbounded length, depending on the value of $g^i(0)$. So each such segment adds one more ω to the ordinal height of the tree. Since there are $n-1$ such segments, the height of the subtree below node $\langle n\rangle$ will be $\omega \cdot (n-1)$. Therefore the height of the computation tree for this loop-program is $\sup_n \omega \cdot (n-1) = \omega^2$.

DEFINITION. An ordinal is *recursive* if it is the order-type of some recursive well-ordering relation $\subseteq \mathbb{N} \times \mathbb{N}$. Any predecessor of a recursive ordinal is recursive and so is its successor, so the recursive ordinals form an initial segment of the countable ordinals. The least non-recursive ordinal is a limit, denoted ω_1^{CK}, the "CK" standing for Church-Kleene.

Note that if Φ_e is total recursive then Tree(e) can be well ordered by the so-called *Kleene-Brouwer ordering*: $\sigma <_{\mathrm{KB}} \tau$ if and only if either $\sigma \supset \tau$ or else there is an $i < \min(\mathrm{lh}(\sigma), \mathrm{lh}(\tau))$ such that $\forall_{j<i}((\sigma)_j = (\tau)_j)$ and $(\sigma)_i < (\tau)_i$. This is a recursive (in fact elementary) well ordering with order-type $\geq \|e\|$. Hence $\|e\|$ is a recursive ordinal.

2.3.3. A hierarchy of total recursive functionals. Kleene's hierarchy of total recursive functionals consists of the classes

$$\mathcal{R}_2(\alpha) := \{\, \Phi_e \mid \Phi_e \text{ total } \wedge\ \|e\| < \alpha \,\}$$

where α ranges over all recursive ordinals. Thus $\mathcal{R}_2(\alpha) \subseteq \mathcal{R}_2(\beta)$ if $\alpha < \beta$.

THEOREM (Hierarchy Theorem). *Every total recursive functional belongs to $\mathcal{R}_2(\alpha)$ for some recursive ordinal α. Furthermore the hierarchy continues to expand as α increases through ω_1^{CK}, that is, for every recursive ordinal α there is a total recursive functional F such that $F \notin \mathcal{R}_2(\alpha)$.*

PROOF. The first part is immediate since if Φ_e is total it belongs to $\mathcal{R}_2(\alpha+1)$ where α is the order-type of the Kleene-Brouwer ordering on Tree(e).

For the second part suppose α is any fixed recursive ordinal, and let $\prec_\alpha$ be a fixed recursive well-ordering with that order type. We define a total recursive functional $V_\alpha(f,g,e,\sigma)$ with two unary function arguments f and g, where e ranges over indices for oracle programs and σ ranges over sequence numbers. Note first that if $\sigma = \langle n_0, n_1, \ldots, n_{k-1}\rangle$ is a non-terminal node in Tree(e) then for any function $g\colon \mathbb{N}\to\mathbb{N}$ the sequence number

$$\sigma * g(\mathrm{lh}(\sigma)-1) := \langle n_0, n_1, \ldots, n_{k-1}, g(k-1)\rangle$$

is also a node in Tree(e), below σ. The definition of V_α is as follows, by recursion down the g-branch of Tree(e) starting with node σ, but controlled by the well-ordering $\prec_\alpha$ via the other function argument f:

$$V_\alpha(f,g,e,\sigma) =$$
$$\begin{cases} V_\alpha(f,g,e,\sigma * g(\mathrm{lh}(\sigma)-1)) & \text{if } \sigma\in \mathrm{Tree}(e) \text{ and} \\ & f(\sigma * g(\mathrm{lh}(\sigma)-1)) \prec_\alpha f(\sigma) \\ U_1(e,(\sigma)_0,\langle(\sigma)_1,\ldots,(\sigma)_{k-1}\rangle) & \text{otherwise.}\end{cases}$$

This is a recursive definition and furthermore it always is defined since repeated application of the first clause leads to a descending sequence

$$\cdots \prec_\alpha f(\sigma'') \prec_\alpha f(\sigma') \prec_\alpha f(\sigma)$$

which must terminate after finitely many steps because $\prec_\alpha$ is a well-ordering. Hence the second clause must eventually apply and the computation terminates. Therefore V_α is total recursive.

Now if Φ_e is any total recursive functional such that $\|e\| < \alpha$ then there will be an order preserving map from Tree(e) into α, and hence a function $f_e\colon \mathbb{N}\to\mathbb{N}$ such that whenever $\tau \supset \sigma$ in Tree(e) then $f_e(\tau) \prec_\alpha f_e(\sigma)$. For this particular e and f_e it is easy to see by induction up the computation tree, and using the Relativized Normal Form Theorem, that for all g and n,

$$\Phi_e(g)(n) \;=\; V_\alpha(f_e,g,e,\langle n\rangle).$$

Consequently the total recursive functional F defined from V_α by

$$F(g)(n) \;=\; V_\alpha(\lambda x.g(x+1),g,g(0),\langle n\rangle)+1$$

cannot lie in $\mathcal{R}_2(\alpha)$. For if it did there would be an e and f_e as above such that $F=\Phi_e$ and hence for all g and all n,

$$V_\alpha(\lambda x.g(x+1),g,g(0),\langle n\rangle)+1 \;=\; V_\alpha(f_e,g,e,\langle n\rangle).$$

A contradiction follows immediately by choosing g so that $g(0)=e$ and $g(x+1)=f_e(x)$. This completes the proof. □

REMARK. For relatively simple but fundamental reasons based in effective descriptive set theory, no such "nice" hierarchy exists for the recursive *functions*. For whereas the class of all indices e of total recursive functionals is definable by the Π^1_1 condition

$$\forall_g \forall_n \exists_s T_1(e,n,\overline{g}(s))$$

the set of all indices of total recursive functions is given merely by an arithmetical Π^0_2 condition:

$$\forall_n \exists_s T(e, n, s).$$

So by the so-called "boundedness property" of hyperarithmetic theory, any inductive hierarchy classification of all the recursive functions is sure to "collapse" before ω_1^{CK}. In practice this usually occurs at the very first limit stage ω and the hierarchy gives no interesting information.

Nevertheless if we adopt a more constructive view and take into account also the *ways* in which a countable ordinal may be presented as a well-ordering, rather than just accepting its set-theoretic existence, then interesting hierarchies of proof theoretically important *sub-classes* of recursive functions begin to emerge.

2.4. Inductive Definitions

We have already used an inductive definition in our proof of Kleene's Recursion Theorem in 1.4.3. Now we treat inductive definitions quite generally, and discuss how far we can get in the analytical hierarchy using inductive definitions.

2.4.1. Monotone operators. Let U be a fixed non-empty set. A map $\Gamma\colon \mathcal{P}(U) \to \mathcal{P}(U)$ is called an *operator* on U. Γ is called *monotone* if for all $X, Y \subseteq U$ from $X \subseteq Y$ we can conclude $\Gamma(X) \subseteq \Gamma(Y)$.

$$\mathcal{I}_\Gamma := \bigcap \{\, X \subseteq U \mid \Gamma(X) \subseteq X \,\}$$

is the set defined inductively by the monotone operator Γ; so $\mathcal{I}_\Gamma$ is the intersection of all Γ-closed subsets of U. Definitions of this kind are called (generalized) *monotone inductive definitions*.

Theorem (Knaster-Tarski). *Let Γ be a monotone operator. Then*

(a) *If $\Gamma(X) \subseteq X$, then $\mathcal{I}_\Gamma \subseteq X$.*
(b) $\Gamma(\mathcal{I}_\Gamma) = \mathcal{I}_\Gamma$.

In particular $\mathcal{I}_\Gamma$ is the least fixed point *of Γ.*

Proof. (a) follows immediately from the definition of $\mathcal{I}_\Gamma$. (b). From $\Gamma(X) \subseteq X$ we can conclude $\mathcal{I}_\Gamma \subseteq X$ by (a), hence $\Gamma(\mathcal{I}_\Gamma) \subseteq \Gamma(X) \subseteq X$ by the monotonicity of Γ. By definition of $\mathcal{I}_\Gamma$ we obtain $\Gamma(\mathcal{I}_\Gamma) \subseteq \mathcal{I}_\Gamma$. Using monotonicity of Γ we can infer $\Gamma(\Gamma(\mathcal{I}_\Gamma)) \subseteq \Gamma(\mathcal{I}_\Gamma)$, hence $\mathcal{I}_\Gamma \subseteq \Gamma(\mathcal{I}_\Gamma)$ again by definition of $\mathcal{I}_\Gamma$. □

Example. Let $0 \in U$ and consider an arbitrary function $S\colon U \to U$. For every set $X \subseteq U$ we define

$$\Gamma(X) := \{0\} \cup \{\, S(v) \mid v \in X \,\}.$$

Clearly Γ is monotone, and $\mathcal{I}_\Gamma$ consists of the (not necessarily distinct) elements $0, S(0), S(S(0)), \ldots$.

2.4.2. An induction principle for monotone inductive definitions. The premise $\Gamma(X) \subseteq X$ in part (a) of the Knaster-Tarski Theorem is in the special case of the example above equivalent to

$$\forall_u\,[u = 0 \vee \exists_{v\in X}(u = S(v)) \to u \in X],$$

i.e., to

$$0 \in X \wedge \forall_{v\in X}(S(v) \in X),$$

and the conclusion is $\forall_{u\in\mathcal{I}_\Gamma}(u \in X)$. Hence part (a) of the Knaster-Tarski Theorem expresses some kind of a general induction principle. However, in the "induction step" we do not quite have the desired form: instead of $\forall_{v\in X}(S(v) \in X)$ we would like to have $\forall_{v\in\mathcal{I}_\Gamma}(v \in X \to S(v) \in X)$. But this can be achieved easily. The theorem below formulates this in the general case.

THEOREM (Induction principle for monotone inductive definitions). *Let* Γ *be a monotone operator. If* $\Gamma(X \cap \mathcal{I}_\Gamma) \subseteq X$*, then* $\mathcal{I}_\Gamma \subseteq X$*.*

PROOF. Because of $\Gamma(X\cap\mathcal{I}_\Gamma) \subseteq \Gamma(\mathcal{I}_\Gamma) = \mathcal{I}_\Gamma$ we obtain from the premise $\Gamma(X \cap \mathcal{I}_\Gamma) \subseteq X \cap \mathcal{I}_\Gamma$. Therefore we have $\mathcal{I}_\Gamma \subseteq X \cap \mathcal{I}_\Gamma$ by definition of $\mathcal{I}_\Gamma$, hence $\mathcal{I}_\Gamma \subseteq X$. □

Note that because of $\Gamma(X\cap\mathcal{I}_\Gamma) \subseteq \Gamma(\mathcal{I}_\Gamma) = \mathcal{I}_\Gamma$ the premise of the theorem could further be weakened to $\mathcal{I}_\Gamma \cap \Gamma(X \cap \mathcal{I}_\Gamma) \subseteq X$.

2.4.3. Approximation of the least fixed point. The least fixed point $\mathcal{I}_\Gamma$ of the monotone operator Γ was defined "from above", as intersection of all sets X such that $\Gamma(X) \subseteq X$. We now show that it can also be obtained by stepwise approximation "from below". In the general situation considered here we need a transfinite iteration of the approximation steps along the ordinals. For an arbitrary operator $\Gamma\colon \mathcal{P}(U) \to \mathcal{P}(U)$ we define $\Gamma{\uparrow}\alpha$ by transfinite recursion on ordinals α:

$$\begin{aligned}
\Gamma{\uparrow}0 &:= \emptyset,\\
\Gamma{\uparrow}(\alpha+1) &:= \Gamma(\Gamma{\uparrow}\alpha),\\
\Gamma{\uparrow}\lambda &:= \bigcup_{\xi<\lambda} \Gamma{\uparrow}\xi, \quad \text{where } \lambda \text{ denotes a limit ordinal.}
\end{aligned}$$

It turns out that not only monotone, but also certain other not necessarily monotone operators Γ have fixed points that can be approximated by these $\Gamma{\uparrow}\alpha$. Call an operator Γ *inclusive* if $X \subseteq \Gamma(X)$ for all $X \subseteq U$.

LEMMA. *Let* Γ *be a monotone or inclusive operator.*

(a) $\Gamma{\uparrow}\alpha \subseteq \Gamma{\uparrow}(\alpha+1)$ *for all ordinals* α*.*
(b) *If* $\Gamma{\uparrow}\alpha = \Gamma{\uparrow}(\alpha+1)$*, then* $\Gamma{\uparrow}(\alpha+\beta) = \Gamma{\uparrow}\alpha$ *for all ordinals* β*.*
(c) $\Gamma{\uparrow}\alpha = \Gamma{\uparrow}(\alpha+1)$ *for some* α *such that* $\mathrm{Card}(\alpha) \leq \mathrm{Card}(U)$*.*

So $\overline{\Gamma} := \bigcup_{\beta\in\mathrm{On}} \Gamma{\uparrow}\beta = \Gamma{\uparrow}\alpha$*, where* α *is the least ordinal such that* $\Gamma{\uparrow}\alpha = \Gamma{\uparrow}(\alpha+1)$*, and* On *denotes the class of all ordinals. This* α *is called the* closure ordinal *of* Γ *and is denoted by* $|\Gamma|$*. The set* $\overline{\Gamma}$ *is called the* closure *of the operator* Γ*. Clearly* $\overline{\Gamma}$ *is a fixed point of* Γ*.*

PROOF. (a). For monotone Γ we use transfinite induction on α. The case $\alpha = 0$ is trivial. In the successor case we have

$$\Gamma{\uparrow}\alpha = \Gamma(\Gamma{\uparrow}(\alpha - 1)) \subseteq \Gamma(\Gamma{\uparrow}\alpha) = \Gamma{\uparrow}(\alpha + 1).$$

Here we have used the IH and the monotonicity of Γ. In the limit case we obtain

$$\Gamma{\uparrow}\lambda = \bigcup_{\xi<\lambda} \Gamma{\uparrow}\xi \subseteq \bigcup_{\xi<\lambda} \Gamma{\uparrow}(\xi + 1) = \bigcup_{\xi<\lambda} \Gamma(\Gamma{\uparrow}\xi) \subseteq \Gamma\big(\bigcup_{\xi<\lambda} \Gamma{\uparrow}\xi\big) = \Gamma{\uparrow}(\lambda + 1).$$

Again we have used the IH and the monotonicity of Γ. – In case Γ is inclusive we simply have

$$\Gamma{\uparrow}\alpha \subseteq \Gamma(\Gamma{\uparrow}\alpha) = \Gamma{\uparrow}(\alpha + 1).$$

(b). By transfinite induction on β. The case $\beta = 0$ is trivial. In the successor case we have by IH

$$\Gamma{\uparrow}(\alpha + \beta + 1) = \Gamma(\Gamma{\uparrow}(\alpha + \beta)) = \Gamma(\Gamma{\uparrow}\alpha) = \Gamma{\uparrow}(\alpha + 1) = \Gamma{\uparrow}\alpha,$$

and in the limit case again by IH

$$\Gamma{\uparrow}(\alpha + \beta) = \bigcup_{\gamma<\beta} \Gamma{\uparrow}(\alpha + \gamma) = \Gamma{\uparrow}\alpha.$$

(c). Assume that for all α such that $\mathrm{Card}(\alpha) \leq \mathrm{Card}(U)$ we have $\Gamma{\uparrow}\alpha \subsetneq \Gamma{\uparrow}(\alpha+1)$, and let $u_\alpha \in \Gamma{\uparrow}(\alpha+1) \setminus \Gamma{\uparrow}\alpha$. This defines an injective map from $\{\, \alpha \mid \mathrm{Card}(\alpha) \leq \mathrm{Card}(U) \,\}$ into U. But this set $\{\, \alpha \mid \mathrm{Card}(\alpha) \leq \mathrm{Card}(U) \,\}$ is exactly the least cardinal *larger* than $\mathrm{Card}(U)$, so this is impossible. □

We now show that for a monotone operator Γ its closure $\overline{\Gamma}$ is in fact its least fixed point $\mathcal{I}_\Gamma$.

LEMMA. *Let Γ be a monotone operator. Then for all ordinals α we have*
(a) $\Gamma{\uparrow}\alpha \subseteq \mathcal{I}_\Gamma$.
(b) *If $\Gamma{\uparrow}\alpha = \Gamma{\uparrow}(\alpha + 1)$, then $\Gamma{\uparrow}\alpha = \mathcal{I}_\Gamma$.*

PROOF. (a). By transfinite induction on α. The case $\alpha = 0$ is trivial. In the successor case we have by IH $\Gamma{\uparrow}(\alpha - 1) \subseteq \mathcal{I}_\Gamma$. Since Γ is monotone this implies

$$\Gamma{\uparrow}\alpha = \Gamma(\Gamma{\uparrow}(\alpha - 1)) \subseteq \Gamma(\mathcal{I}_\Gamma) = \mathcal{I}_\Gamma.$$

In the limit case we obtain from the IH $\Gamma{\uparrow}\xi \subseteq \mathcal{I}_\Gamma$ for all $\xi < \lambda$. This implies

$$\Gamma{\uparrow}\lambda = \bigcup_{\xi<\lambda} \Gamma{\uparrow}\xi \subseteq \mathcal{I}_\Gamma.$$

(b). Let $\Gamma{\uparrow}\alpha = \Gamma{\uparrow}(\alpha + 1)$, hence $\Gamma{\uparrow}\alpha = \Gamma(\Gamma{\uparrow}\alpha)$. Then $\Gamma{\uparrow}\alpha$ is a fixed point of Γ, hence $\mathcal{I}_\Gamma \subseteq \Gamma{\uparrow}\alpha$. The reverse inclusion follows from (a). □

2.4.4. Continuous operators. We now consider the important special case of *continuous* operators. A subset $\mathcal{Z} \subseteq \mathcal{P}(U)$ is called *directed* if for every finite $\mathcal{Z}_0 \subseteq \mathcal{Z}$ there is an $X \in \mathcal{Z}$ such that $Y \subseteq X$ for all $Y \in \mathcal{Z}_0$. An operator $\Gamma\colon \mathcal{P}(U) \to \mathcal{P}(U)$ is called *continuous* if

$$\Gamma\big(\bigcup \mathcal{Z}\big) = \bigcup\{\, \Gamma(X) \mid X \in \mathcal{Z} \,\}$$

for every directed subset $\mathcal{Z} \subseteq \mathcal{P}(U)$.

LEMMA. *Every continuous operator Γ is monotone.*

PROOF. For $X, Y \subseteq U$ such that $X \subseteq Y$ we obtain $\Gamma(Y) = \Gamma(X \cup Y) = \Gamma(X) \cup \Gamma(Y)$ from the continuity of Γ, and hence $\Gamma(X) \subseteq \Gamma(Y)$. □

For a continuous operator the transfinite approximation of its least fixed point stops after ω steps. Hence in this case we have an easy characterization of the least fixed point "from below".

LEMMA. *Let Γ be a continuous operator. Then $\mathcal{I}_\Gamma = \Gamma{\uparrow}\omega$.*

PROOF. It suffices to show $\Gamma{\uparrow}(\omega + 1) = \Gamma{\uparrow}\omega$.

$$\Gamma{\uparrow}(\omega+1) = \Gamma(\Gamma{\uparrow}\omega) = \Gamma(\bigcup_{n<\omega} \Gamma{\uparrow}n) = \bigcup_{n<\omega} \Gamma(\Gamma{\uparrow}n) = \bigcup_{n<\omega} \Gamma{\uparrow}(n+1) = \Gamma{\uparrow}\omega,$$

where in the third to last equation we have used the continuity of Γ. □

2.4.5. The accessible part of a relation. An important example of a monotone inductive definition is the following construction of the accessible part of a binary relation $\prec$ on U. Note that $\prec$ is *not* required to be transitive, so $(U, \succ)$ may be viewed as a *reduction system*. For $X \subseteq U$ let $\Gamma_\prec(X)$ be the set of all $\prec$-predecessors of u:

$$\Gamma_\prec(X) := \{\, u \mid \forall_{v \prec u}(v \in X) \,\}.$$

Clearly $\Gamma_\prec$ is monotone; its least fixed point $\mathcal{I}_{\Gamma_\prec}$ is called the *accessible part* of $(U, \prec)$ and denoted by $\mathrm{acc}(\prec)$ or $\mathrm{acc}_\prec$. If $\mathcal{I}_{\Gamma_\prec} = U$, then the relation $\prec$ is called *well-founded*; the inverse relation $\succ$ is called *noetherian* or *terminating*. In this special case the Knaster-Tarski Theorem and Induction Principle in Sec.2.4.2 can be combined as follows.

$$\forall_u[\forall_{v \prec u}(v \in X \cap \mathrm{acc}_\prec) \to u \in X] \to \forall_{u \in \mathrm{acc}_\prec}(u \in X). \tag{2.2}$$

(2.3) $\mathrm{acc}_\prec$ is $\Gamma_\prec$-closed, i.e., $\forall_{v \prec u}(v \in \mathrm{acc}_\prec)$ implies $u \in \mathrm{acc}_\prec$.

(2.4) Every $u \in \mathrm{acc}_\prec$ is from $\Gamma_\prec(\mathrm{acc}_\prec)$, i.e., $\forall_{u \in \mathrm{acc}_\prec} \forall_{v \prec u}(v \in \mathrm{acc}_\prec)$.

Notice that (2.2) expresses an induction principle: to show that all elements in $u \in \mathrm{acc}_\prec$ are in a set X it suffices to prove the "induction step": we can infer $u \in X$ from the assumption that all smaller $v \prec u$ are accessible and in X.

By a *reduction sequence* we mean a finite or infinite sequence $u_1, u_2, \ldots$ such that $u_i \succ u_{i+1}$. As an easy application one can show that $u \in \mathrm{acc}_\prec$ if and only if every reduction sequence starting with u terminates after finitely many steps. For the direction from left to right we use induction on $u \in \mathrm{acc}_\prec$. So let $u \in \mathrm{acc}_\prec$ and assume that for every u' such that $u \succ u'$ every reduction sequence starting starting with u' terminates after finitely many steps. Then clearly also every reduction sequence starting with u must terminate, since its second member is such a u'. Conversely, suppose we would have a $u \notin \mathrm{acc}_\prec$. We construct an infinite reduction sequence $u = u_1, u_2, \ldots, u_n, \ldots$ such that $u_n \notin \mathrm{acc}_\prec$; this yields the desired contradiction. So let $u_n \notin \mathrm{acc}_\prec$. By (2.3) we then have a $v \notin \mathrm{acc}_\prec$ such that $u_n \succ v$; pick u_{n+1} as such a v.

2.4.6. Inductive definitions over $\mathbb{N}$. We now turn to inductive definitions over the set $\mathbb{N}$ and their relation to the arithmetical and analytical hierarchies. An operator $\Gamma\colon \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ is called Σ^0_r*-definable* if there is a Σ^0_r-definable relation Q_Γ such that for all $A \subseteq \mathbb{N}$ and all $n \in \mathbb{N}$

$$n \in \Gamma(A) \;\leftrightarrow\; Q_\Gamma(c_A, n).$$

Π^0_r, Δ^0_r, Σ^1_r, Π^1_r and Δ^1_r-definable operators are defined similarly.

It is easy to show that every Σ^0_1-definable monotone operator Γ is continuous, and hence by a lemma in 2.4.4 has closure ordinal $|\Gamma| \leq \omega$. We now show that this consequence still holds for inclusive operators.

Lemma. *Let Γ be a monotone or inclusive Σ^0_1-definable operator. Then $|\Gamma| \leq \omega$.*

Proof. By assumption

$$n \in \Gamma(A) \;\leftrightarrow\; \exists_s\, T_1(e, n, \overline{c_A}(s))$$

for some $e \in \mathbb{N}$. It suffices to show that $\Gamma(\Gamma{\uparrow}\omega) \subseteq \Gamma{\uparrow}\omega$. Suppose $n \in \Gamma(\Gamma{\uparrow}\omega)$, so $T_1(e, n, \overline{c_{\Gamma\uparrow\omega}}(s))$ for some s. Since $\Gamma{\uparrow}\omega$ is the union of the increasing chain $\Gamma{\uparrow}0 \subseteq \Gamma{\uparrow}1 \subseteq \Gamma{\uparrow}2 \subseteq \ldots$, for some r we must have $\overline{c_{\Gamma\uparrow\omega}}(s) = \overline{c_{\Gamma\uparrow r}}(s)$. Therefore $n \in \Gamma(\Gamma{\uparrow}r) = \Gamma{\uparrow}(r+1) \subseteq \Gamma{\uparrow}\omega$. □

2.4.7. Definability of least fixed points for monotone operators. Next we prove that the closure of a *monotone* Σ^0_1-definable operator is Σ^0_1-definable as well (this will be seen to be false for inclusive operators). As a tool in the proof we need König's Lemma. Here and later we use starred function variables $f^*, g^*, h^*, \ldots$ to range over 0-1-valued functions.

Lemma (König). *Let T be a binary tree, i.e., T consists of (codes for) sequences of 0 and 1 only and is closed against the formation of initial segments. Then*

$$\forall_n \exists_x [\mathrm{lh}(x) = n \wedge \forall_{i<n} (x)_i \leq 1 \wedge x \in T] \;\leftrightarrow\; \exists_{f^*} \forall_s (\overline{f^*}(s) \in T).$$

Proof. The direction from right to left is obvious. For the converse assume the left hand side and let

$$M := \{\, y \mid \forall_{i<\mathrm{lh}(y)} (y)_i \leq 1 \wedge \forall_m \exists_z [\mathrm{lh}(z) = m \wedge \forall_{i<m} (z)_i \leq 1 \wedge \forall_{j \leq \mathrm{lh}(y)+m} \mathrm{Init}(y * z, j) \in T] \,\}.$$

M can be seen as the set of all "fertile" nodes, possessing arbitrary long extensions within T. To construct the required infinite path f^* we use the *axiom of dependent choice*:

$$\exists_y A(0, y) \to \forall_{n,y}(A(n, y) \to \exists_z A(n+1, z)) \to \exists_f \forall_n A(n, f(n)),$$

with $A(n, y)$ expressing that y is a fertile node of length n:

$$A(n, y) :\leftrightarrow y \in M \wedge \mathrm{lh}(y) = n.$$

Now $\exists_y A(0, y)$ is obvious (take $y := \langle\rangle$). For the step case assume that y is a fertile node of length n. Then at least one of the two possible extensions $y * \langle 0\rangle$ and $y * \langle 1\rangle$ must be fertile, i.e., in M; pick z accordingly. □

Corollary. *If R is Π^0_1-definable, then so is*

$$Q(\vec{g}, \vec{n}) \;:\leftrightarrow\; \exists_{f^*} \forall_s\, R(\overline{f^*}(s), \vec{g}, \vec{n}).$$

PROOF. By König's Lemma we have

$$Q(\vec{g},\vec{n}) \;\leftrightarrow\; \forall_n \exists_{x\le\langle 1,\dots,1\rangle}[\mathrm{lh}(x)=n \wedge \forall_{i<n}\,(x)_i \le 1 \wedge R(x,\vec{g},\vec{n})]. \qquad \square$$

We now show that the Π^1_1 and Σ^0_1-definable relations are closed against monotone inductive definitions.

THEOREM. *Let $\Gamma\colon \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ be a monotone operator.*
(a) *If Γ is Π^1_1-definable, then so is its least fixed point $\mathcal{I}_\Gamma$.*
(b) *If Γ is Σ^0_1-definable, then so is its least fixed point $\mathcal{I}_\Gamma$.*

PROOF. Let $\Gamma\colon \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ be a monotone operator and $n \in \Gamma(A) \;\leftrightarrow\; Q_\Gamma(c_A, n)$.

(a). Assume Q_Γ is Π^1_1-definable. Then $\mathcal{I}_\Gamma$ is the intersection of all Γ-closed sets, so

$$n \in \mathcal{I}_\Gamma \;\leftrightarrow\; \forall_f[\forall_m(Q_\Gamma(f,m) \to f(m)=1) \to f(n)=1].$$

This shows that $\mathcal{I}_\Gamma$ is Π^1_1-definable.

(b). Assume Q_Γ is Σ^0_1-definable. Then $\mathcal{I}_\Gamma$ can be represented in the form

$$\begin{aligned} n \in \mathcal{I}_\Gamma &\leftrightarrow \forall_{f^*}[\forall_m(Q_\Gamma(f^*,m) \to f^*(m)=1) \to f^*(n)=1] \\ &\leftrightarrow \forall_{f^*}\exists_m\, R(f^*,m,n) \quad \text{with } R \text{ recursive} \\ &\leftrightarrow \forall_{f^*}\exists_s\, T_1(e,n,\overline{f^*}(s)) \quad \text{for some } e. \end{aligned}$$

By the corollary to König's Lemma $\mathcal{I}_\Gamma$ is Σ^0_1-definable. $\square$

2.4.8. Some counterexamples. If Γ is a non monotone but only inclusive Σ^0_1-definable operator, then its closure $\overline{\Gamma}$ need not even be arithmetical. Recall from 1.6.5 the definition of the universal Σ^0_{r+1}-definable relations $U^0_{r+1}(e,n)$:

$$\begin{aligned} U^0_1(e,n) &:\leftrightarrow \exists_s\, T(e,n,s) \quad (\leftrightarrow\; n \in W^{(1)}_e), \\ U^0_{r+1}(e,n) &:\leftrightarrow \exists_m \neg U^0_r(e, n*\langle m\rangle). \end{aligned}$$

Let

$$U^0_\omega \;:=\; \{\, \langle r,e,\vec{n}\rangle \mid U^0_{r+1}(e,\langle\vec{n}\rangle)\,\}.$$

Clearly for every arithmetical relation R there are r,e such that $R(\vec{n}) \leftrightarrow \langle r,e,\vec{n}\rangle \in U^0_\omega$. Hence U^0_ω can not be arithmetical, for if it would be say Σ^0_{r+1}-definable, then every arithmetical relation R would be Σ^0_{r+1}-definable, contradicting the fact that the arithmetical hierarchy is properly expanding. On the other hand we have

LEMMA. *There is an inclusive Σ^0_1-definable operator Γ such that $\overline{\Gamma} = U^0_\omega$, hence $\overline{\Gamma}$ is not arithmetical.*

PROOF. We define Γ such that

$$\Gamma{\uparrow}r \;=\; \{\, \langle t,e,\vec{n}\rangle \mid 0<t\le r \wedge U^0_t(e,\langle\vec{n}\rangle)\,\}. \tag{2.5}$$

Let

$$\begin{aligned} &s \in \Gamma(A) :\leftrightarrow \\ &\quad s \in A \vee \\ &\quad \exists_{e,\vec{n}}\,[s=\langle 1,e,\vec{n}\rangle \wedge U^0_1(e,\langle\vec{n}\rangle)] \vee \end{aligned}$$

$$\exists_{t,e,\vec{n}}\,[s = \langle t+1, e, \vec{n}\rangle \wedge \exists_{e_1,\vec{m}}\,\langle t, e_1, \vec{m}\rangle \in A \wedge \exists_m \langle t, e, \vec{n}, m\rangle \notin A].$$

We now prove (2.5) by induction on r. The base case $r = 0$ is obvious. In the step case we have

$$\begin{aligned}
s \in \Gamma(\Gamma{\uparrow}r) \leftrightarrow\ & \\
& s \in \Gamma{\uparrow}r \vee \\
& \exists_{e,\vec{n}}\,[s = \langle 1, e, \vec{n}\rangle \wedge U_1^0(e, \langle\vec{n}\rangle)] \vee \\
& \exists_{t,e,\vec{n}}[s = \langle t+1, e, \vec{n}\rangle \wedge \exists_{e_1,\vec{m}}\,\langle t, e_1, \vec{m}\rangle \in \Gamma{\uparrow}r \wedge \exists_m \langle t, e, \vec{n}, m\rangle \notin \Gamma{\uparrow}r] \\
\leftrightarrow\ & s \in \Gamma{\uparrow}r \vee \\
& \exists_{e,\vec{n}}\,[s = \langle 1, e, \vec{n}\rangle \wedge U_1^0(e, \langle\vec{n}\rangle)] \vee \\
& \exists_{t,e,\vec{n}}\,[s = \langle t+1, e, \vec{n}\rangle \wedge 0 < t \leq r \wedge \exists_m \neg U_t^0(e, \langle\vec{n}, m\rangle)] \\
\leftrightarrow\ & \exists_{t,e,\vec{n}}\,[s = \langle t, e, \vec{n}\rangle \wedge 0 < t \leq r \wedge U_t^0(e, \langle\vec{n}\rangle)] \vee \\
& \exists_{e,\vec{n}}\,[s = \langle 1, e, \vec{n}\rangle \wedge U_1^0(e, \langle\vec{n}\rangle)] \vee \\
& \exists_{t,e,\vec{n}}\,[s = \langle t+1, e, \vec{n}\rangle \wedge 0 < t \leq r \wedge U_{t+1}^0(e, \langle\vec{n}\rangle)] \\
\leftrightarrow\ & s \in \{\, \langle t, e, \vec{n}\rangle \mid 0 < t \leq r+1 \wedge U_t^0(e, \langle\vec{n}\rangle)]\,\}.
\end{aligned}$$

Clearly Γ is a Σ_1^0-definable inclusive operator. By 2.4.6 its closure ordinal $|\Gamma|$ is $\leq \omega$, so $\Gamma{\uparrow}\omega = \overline{\Gamma}$. But clearly $\Gamma{\uparrow}\omega = \bigcup_r \Gamma{\uparrow}r = U_\omega^0$. □

On the positive side we have

LEMMA. *For every inclusive Δ_1^1-definable operator Γ its closure $\overline{\Gamma}$ is Δ_1^1-definable.*

PROOF. Let Γ be an inclusive operator such that $n \in \Gamma(A) \leftrightarrow Q_\Gamma(c_A, n)$ for some Δ_1^1-definable relation Q_Γ. Let

$$f^*(p) := \begin{cases} 1 & \text{if } p = \langle r+1, n\rangle \text{ and } n \in \Gamma{\uparrow}r \\ 0 & \text{otherwise} \end{cases}$$

and consider the following Δ_1^1-definable relation R.

$$\begin{aligned}
R(g) :\leftrightarrow\ & \forall_p\,[g(p) \leq 1] \wedge \\
& \forall_p\,[\mathrm{lh}(p) \neq 2 \vee (p)_0 = 0 \to g(p) = 0] \\
& \forall_r \forall_n\,[g\langle r+1, n\rangle = 1 \leftrightarrow Q_\Gamma(\lambda x g\langle r, x\rangle, n)].
\end{aligned}$$

Clearly $R(f^*)$. Moreover, for every g such that $R(g)$ we have $g(p) = 0$ for all p not of the form $\langle r+1, n\rangle$, and it is easy to prove that

$$g\langle r+1, n\rangle = 1 \leftrightarrow n \in \Gamma{\uparrow}r.$$

Therefore f^* is the unique member of R, and we have

$$\begin{aligned}
n \in \overline{\Gamma} \leftrightarrow \exists_r n \in \Gamma{\uparrow}r &\leftrightarrow \exists_r g\langle r+1, n\rangle = 1 \\
&\leftrightarrow \exists_g\,[R(g) \wedge \exists_r g\langle r+1, n\rangle = 1] \\
&\leftrightarrow \forall_g\,[R(g) \to \exists_r g\langle r+1, n\rangle = 1].
\end{aligned}$$

Hence $\overline{\Gamma}$ is Δ_1^1-definable. □

COROLLARY. $U_\omega^0 \in \Delta_1^1 \setminus \Delta_\infty^0$.

2.5. Notations for Constructive Ordinals

In this section we prove the equivalence of two systems of notations for ordinals: Kleene's system $\mathcal{O}$ and the set $\mathcal{W}$ of indices of recursive well-orderings. In both cases we assign natural numbers to ordinals from a certain section of the countable ordinals.

2.5.1. Kleene's notation system $\mathcal{O}$. The definition of Kleene's notation system $\mathcal{O}$ distinguishes cases according to whether an ordinal is zero, a successor or a limit. A (countable) limit ordinal α can be given by a (countable) increasing sequence of ordinals converging to α. If the notations of such a sequence can be enumerated by a recursive function φ_e, then e determines a notation for α in $\mathcal{O}$.

Definition. By transfinite recursion on the countable ordinals α we simultaneously define

(a) a set $\mathcal{O}_\alpha \subseteq \mathbb{N}$ of *ordinal notations* for α, and
(b) a relation $<_{\mathcal{O}_\alpha} \subseteq \mathbb{N} \times \mathbb{N}$

such that for $\alpha < \beta$ the sets $\mathcal{O}_\alpha$ and $\mathcal{O}_\beta$ are disjoint, and $b <_{\mathcal{O}_\alpha} a$ implies $a \in \mathcal{O}_\alpha$ and $b \in \mathcal{O}_\beta$ for some $\beta < \alpha$. From the the disjointness of the $\mathcal{O}_\alpha$'s it follows that any $a \in \mathcal{O}_\alpha$ uniquely determines the denoted ordinal α; we write $\alpha = |a|_{\mathcal{O}}$ in this case. If $a \in \mathcal{O}_\alpha$, then we also write $b <_{\mathcal{O}} a$ for $b <_{\mathcal{O}_\alpha} a$.

$$\mathcal{O}_0 := \{1\}, \qquad \mathcal{O}_{\alpha+1} := \{\, 2^a \mid a \in \mathcal{O}_\alpha \,\},$$
$$<_{\mathcal{O}_0} := \emptyset, \qquad <_{\mathcal{O}_{\alpha+1}} := \{\, (b, 2^a) \mid a \in \mathcal{O}_\alpha \wedge (b <_{\mathcal{O}} a \vee b = a) \,\}$$

and in case α is a limit ordinal

$$\begin{aligned} \mathcal{O}_\alpha := \{\, 3 \cdot 5^e \mid\ & \varphi_e \text{ is a total unary function} \wedge \\ & \forall_n \exists_{\gamma<\alpha}\, \varphi_e(n) \in \mathcal{O}_\gamma \wedge \\ & \forall_n\, \varphi_e(n) <_{\mathcal{O}} \varphi_e(n+1) \wedge \\ & \forall_{\beta<\alpha} \exists_{n,\gamma}\, (\beta \le \gamma < \alpha \wedge \varphi_e(n) \in \mathcal{O}_\gamma) \,\}, \end{aligned}$$
$$<_{\mathcal{O}_\alpha} := \{\, (b, 3 \cdot 5^e) \mid 3 \cdot 5^e \in \mathcal{O}_\alpha \wedge \exists_n\, b <_{\mathcal{O}} \varphi_e(n) \,\}.$$

Let

$$\mathcal{O} := \bigcup_{\alpha \in \mathrm{On}} \mathcal{O}_\alpha, \quad <_{\mathcal{O}} := \bigcup_{\alpha \in \mathrm{On}} <_{\mathcal{O}_\alpha}.$$

This is compatible with the notation $b <_{\mathcal{O}} a$ introduced above. We write $a \le_{\mathcal{O}} b$ for $a <_{\mathcal{O}} b \vee a = b$.

Note. From $3 \cdot 5^e \in \mathcal{O}$ we can conclude that $\varphi_e(n) <_{\mathcal{O}} 3 \cdot 5^e$, since by definition we then have $\varphi_e(n) <_{\mathcal{O}} \varphi_e(n+1)$.

Intuitively $\mathcal{O}$ can be viewed as follows. All finite ordinals get a unique notation in $\mathcal{O}$. But already the first limit ordinal ω has infinitely many notations, for two reasons. First, there are infinitely many recursive sequences converging to ω, and second, each of these sequences has infinitely many indices.

The choice of numbers of the form $3 \cdot 5^e$ as notation for limit ordinals has historical reasons; of course one could take for instance $\langle 3, e \rangle$ or 3^e as well.

2.5.2. Properties of $<_{\mathcal{O}}$. We first show that $<_{\mathcal{O}}$ is a transitive and irreflexive relation on $\mathcal{O}$ (i.e., a strict partial ordering); clearly $<_{\mathcal{O}}$ is not a linear ordering.

LEMMA. (a) *If $a <_{\mathcal{O}} b$ and $b <_{\mathcal{O}} c$, then $a <_{\mathcal{O}} c$.*
(b) *$a \not<_{\mathcal{O}} a$.*

PROOF. (a). We use induction on $|c|_{\mathcal{O}}$. The case $|c|_{\mathcal{O}} = 0$ (i.e., $c = 1$) is obvious. Now let $c = 2^d$. We distinguish cases according to $b <_{\mathcal{O}} 2^d$: in case $b <_{\mathcal{O}} d$ the claim follows from the IH and in case $b = d$ from the first assumption. Finally let $c = 3 \cdot 5^e$. Then $b <_{\mathcal{O}} \varphi_e(n) \in \mathcal{O}_\gamma$ for some $\gamma < |c|_{\mathcal{O}}$, hence $a <_{\mathcal{O}} \varphi_e(n)$ by IH. Now $a <_{\mathcal{O}} 3 \cdot 5^e$ by definition.

(b). By induction on $|a|_{\mathcal{O}}$, using (a). The case $|a|_{\mathcal{O}} = 0$ is obvious. *Case* 2^a. Assume $2^a <_{\mathcal{O}} 2^a$. Then by definition we have $2^a <_{\mathcal{O}} a \vee 2^a = a$. In the first case from $a <_{\mathcal{O}} 2^a$ and (a) we obtain $a <_{\mathcal{O}} a$ and hence a contradiction by IH; the second case clearly is impossible. *Case* $3{\cdot}5^e$. Assume $3{\cdot}5^e <_{\mathcal{O}} 3{\cdot}5^e$. Then by definition $3 \cdot 5^e <_{\mathcal{O}} \varphi_e(n)$ for some n. From $\varphi_e(n) <_{\mathcal{O}} 3 \cdot 5^e$ (cf. the remark above) and (a) we obtain $\varphi_e(n) <_{\mathcal{O}} \varphi_e(n)$ and hence a contradiction by IH. □

Clearly 1 is the least element of the partial ordering $<_{\mathcal{O}}$: by induction on $|a|_{\mathcal{O}}$ one easily shows $1 \leq_{\mathcal{O}} a$.

2.5.3. Addition of ordinal notations. Next we introduce a function $+_{\mathcal{O}}$, which corresponds on $\mathcal{O}$ to the addition of ordinals.

DEFINITION.

$$a +_{\mathcal{O}} b := \begin{cases} a & \text{if } b = 1 \text{ and } a \neq 0, \\ 2^{a +_{\mathcal{O}} c} & \text{if } b = 2^c \text{ and } c \neq 0, \\ 3 \cdot 5^{S(\underline{+_{\mathcal{O}}}, a) \circ e} & \text{if } b = 3 \cdot 5^e, \\ 7 & \text{otherwise.} \end{cases}$$

Here $\underline{+_{\mathcal{O}}}$ is an index of $+_{\mathcal{O}}$, S is the function from the Substitution Lemma and $\circ$ is the elementary function comp from 1.3.1; hence

$$\varphi_{e_1 \circ e_2}(\vec{n}) = \varphi_{e_1}(\varphi_{e_2}(\vec{n})).$$

By Kleene's Second Recursion Theorem for recursive functions there is a recursive solution $+_{\mathcal{O}}$ of the equation above.

REMARK. In case $b = 1$ we need the assumption $a \neq 0$, since we later want to infer $a, b \in \mathcal{O}$ from $a +_{\mathcal{O}} b \in \mathcal{O}$; without the assumption we would have

$$0 +_{\mathcal{O}} 2^1 = 2^{0 +_{\mathcal{O}} 1} = 2^0 = 1 \in \mathcal{O}, \quad \text{but} \quad 0 \notin \mathcal{O}.$$

LEMMA. *For $a, b \in \mathcal{O}$ we have*
(a) *$a +_{\mathcal{O}} b \in \mathcal{O}$.*
(b) *If $c <_{\mathcal{O}} b$, then $a +_{\mathcal{O}} c <_{\mathcal{O}} a +_{\mathcal{O}} b$.*
(c) *$|a +_{\mathcal{O}} b|_{\mathcal{O}} = |a|_{\mathcal{O}} + |b|_{\mathcal{O}}$.*

PROOF. All three parts are proved simultaneously by transfinite induction on $|b|_{\mathcal{O}}$. The case $|b|_{\mathcal{O}} = 0$ (i.e., $b = 1$) is obvious.

Case $b = 2^d$ with $d \in \mathcal{O}$. (a). By IH we have $a +_\mathcal{O} d \in \mathcal{O}$. hence $2^{a+_\mathcal{O} d} = a +_\mathcal{O} 2^d \in \mathcal{O}$. (b). Assume $c <_\mathcal{O} b$. Then either $c <_\mathcal{O} d$ or $c = d$. In the first case we have, using the IH

$$a +_\mathcal{O} c <_\mathcal{O} a +_\mathcal{O} d <_\mathcal{O} 2^{a+_\mathcal{O} d} = a +_\mathcal{O} b$$

The second case is even simpler. (c).

$$|a +_\mathcal{O} b|_\mathcal{O} = |a +_\mathcal{O} d|_\mathcal{O} + 1 = |a|_\mathcal{O} + |d|_\mathcal{O} + 1 = |a|_\mathcal{O} + |b|_\mathcal{O}.$$

Case $b = 3 \cdot 5^e$. With φ_e also $\varphi_{S(\underline{+_\mathcal{O}},a)\circ e}$ is total. By IH(b) we have

$$\varphi_{S(\underline{+_\mathcal{O}},a)\circ e}(n) = a +_\mathcal{O} \varphi_e(n) <_\mathcal{O} a +_\mathcal{O} \varphi_e(n+1) = \varphi_{S(\underline{+_\mathcal{O}},a)\circ e}(n+1)$$

Moreover,

$$\begin{aligned} |a +_\mathcal{O} b|_\mathcal{O} &= |3 \cdot 5^{S(\underline{+_\mathcal{O}},a)\circ e}|_\mathcal{O} \\ &= \sup_{n<\omega} |a +_\mathcal{O} \varphi_e(n)|_\mathcal{O} \\ &= \sup_{n<\omega} (|a|_\mathcal{O} + |\varphi_e(n)|_\mathcal{O}) \quad \text{by IH(c)} \\ &= |a|_\mathcal{O} + \sup_{n<\omega} |\varphi_e(n)|_\mathcal{O} \\ &= |a|_\mathcal{O} + |b|_\mathcal{O}. \end{aligned}$$

Now also (a) and (b) can be seen easily. □

2.5.4. Equivalence of the two notation systems. We now show that the ordinals denoted by notations $a \in \mathcal{O}$ are precisely the recursive ordinals. The essential tool in the proof is the Recursion Lemma, an easy but important consequence of Kleene's Second Recursion Theorem.

Lemma (Recursion Lemma). *Let $<_M$ be a well-founded partial ordering of a set $M \subseteq \mathbb{N}$ and $R \subseteq M \times \mathbb{N}$ some binary relation. Assume that there is a partial recursive function χ such that for every $n \in M$ and t*

$$\forall_{m<_M n} R(m, \varphi_t(m)) \to R(n, \chi(t, n)).$$

Then we can find a partial recursive function φ such that $\forall_{n\in M} R(n, \varphi(n))$. Moreover, if χ is total, then so is φ.

Proof. By Kleene's Second Recursion Theorem we can find $e_0 \in \mathbb{N}$ such that for all n we have $\varphi_{e_0}(n) = \chi(e_0, n)$. For $\varphi := \varphi_{e_0}$ we can easily prove the claim by induction on $<_M$. □

Let

$$\mathcal{W} := \{ e \mid \varphi_e \in \mathrm{WOrd} \}.$$

Clearly $\mathcal{W}$ is Π^1_1-definable (cf. the example of $\mathrm{WOrd}(f)$ in 2.2.1). For $e \in \mathcal{W}$ we write $|e|_\mathcal{W}$ for the order type of the well-ordering $<_e$ denoted by e, so $n <_e m \leftrightarrow \varphi_e\langle n, m\rangle = 1$.

Theorem. *An ordinal α is recursive if and only if $\alpha = |a|_\mathcal{O}$ for some $a \in \mathcal{O}$.*

Proof. First assume $\alpha = |a|_\mathcal{O}$ for some $a \in \mathcal{O}$. We apply the Recursion Lemma to $\mathcal{O}$ and $<_\mathcal{O}$ with

$$R(a, e) :\leftrightarrow a \in \mathcal{O} \wedge e \in \mathcal{W} \wedge |a|_\mathcal{O} < |e|_\mathcal{W}.$$

Since the recursive ordinals form an initial segment of the (countable) ordinals it suffices to find a partial recursive function φ such that $\forall_{a\in\mathcal{O}} R(a,\varphi(a))$. By the Recursion Lemma this will follow from the existence of a partial recursion function χ such that for all $a \in \mathcal{O}$ and t

$$\forall_{b<_{\mathcal{O}} a}\, R(b,\varphi_t(b)) \to R(a,\chi(t,a)). \tag{2.6}$$

We prove (2.6) by cases on a, simultaneously with the construction of χ.

For $a = 1$ let $\chi(t,a)$ be an index of a one-element well-ordering. *Case* 2^a. By assumption $\varphi_t(a)$ is an index of a well-ordering of order type $> |a|_{\mathcal{O}}$. Let $\chi(t,2^a)$ be an index of a new well-ordering obtained from this one by adding one element at the end. *Case* $3\cdot 5^e$. For every n by assumption $\varphi_t(\varphi_e(n))$ is an index of a well-ordering of order type $> |\varphi_e(n)|_{\mathcal{O}}$. Let $\chi(t,3\cdot 5^e)$ be an index of the new recursive well-ordering obtained by appending all these (with n increasing).

For the converse we again apply the Recursion Lemma. First we have to construct from the given well-ordering $\prec$ a new one $\prec'$, in order to be able to recursively distinguish the successor and the limit case.

$$\langle n_1,m_1\rangle \prec' \langle n_2,m_2\rangle \;:\leftrightarrow\; n_1 \prec n_2 \vee (n_1 = n_2 \wedge m_1 < m_2).$$

We then clearly have $|\prec'| = \omega\cdot|\prec|$. Now let

$$R(p,a) \;:\leftrightarrow\; p \in \mathrm{Field}(\prec') \wedge a \in \mathcal{O} \wedge |p|_{\prec'} \le |a|_{\mathcal{O}}.$$

Here $|p|_{\prec'}$ is the order type of the well-ordering obtained from $\prec'$ by "cutting off" at p, i.e., of $\prec'$ restricted to $\{\, q \mid q \prec' p\,\}$. We may assume that for some p we have $|\prec| = |p|_{\prec'}$; hence it suffices to construct a partial recursive φ such that $\forall_{p\in\mathrm{Field}(\prec')} R(p,\varphi(p))$. Again by the Recursion Lemma this will follow from the existence of a partial recursion function χ such that for all $p \in \mathrm{Field}(\prec')$ and t

$$\forall_{q\prec' p}\, R(q,\varphi_t(q)) \to R(p,\chi(t,p)). \tag{2.7}$$

We prove (2.7) by cases on p, simultaneously with the construction of χ. So assume $\forall_{q\prec' p} R(q,\varphi_t(q))$.

Case $p = \langle n_0,0\rangle$ with n_0 minimal element in the well-ordering $\prec$. Let $\chi(t,p) := 1$.

Case $p = \langle n, m+1\rangle$. Let $\chi(t,p) := 2^{\varphi_t\langle n,m\rangle}$.

Case $p = \langle n_1,0\rangle$ with $n_1 \neq n_0$. Then $|p|_{\prec'}$ is a limit ordinal. We define a fundamental sequence f for p in $\prec'$ by

$$f(0) \;:=\; \langle n_0,0\rangle$$

and for $f(k) = \langle n,m\rangle$ we let

$$f(k+1) \;:=\; \begin{cases} \langle k,0\rangle & \text{if } n \prec k \prec n_1 \\ \langle n,m+1\rangle & \text{otherwise.}\end{cases}$$

f is a total recursive function, and an e such that $f = \varphi_e$ can be computed elementarily from p. Moreover we have

$$\forall_{\langle n,m\rangle\prec' p}\exists_k\, \langle n,m\rangle \prec' f(k).$$

Let

$$g(0) := 1,$$

$$g(k+1) := g(k) +_{\mathcal{O}} \varphi_{toe}(k) +_{\mathcal{O}} 2^1.$$

Again an index e_1 of g can be computed elementarily from p, t. Now with $\chi(t,p) := 3 \cdot 5^{e_1}$ we have $R(p, \chi(t,p))$. □

2.6. Complexity of the Two Notation Systems

The two systems $\mathcal{O}$ and $\mathcal{W}$ are equivalent not only with respect to the ordinals they denote, but also with respect to their complexity: both are complete Π^1_1-definable sets. This can be proven easily for $\mathcal{W}$, essentially because $\mathcal{W}$ is defined explicitely. For $\mathcal{O}$ we need some knowledge about inductive definitions.

2.6.1. Π^1_1-completeness of $\mathcal{W}$.

LEMMA. *$\mathcal{W}$ is Π^1_1-complete.*

PROOF. We already noted that $\mathcal{W}$ is Π^1_1-definable. Now let $B \subseteq \mathbb{N}$ be an arbitrary Π^1_1-definable set. Then for some $e \in \mathbb{N}$

$$n \in B \;\leftrightarrow\; \forall_g \exists_s T_1(e, n, \overline{g}(s)).$$

Let

$$\mathrm{Tree}(e,n) \;:=\; \{\, \langle n_0, \ldots, n_{l-1}\rangle \mid \forall_{k<l}\, \neg T_1(e, n, \langle n_0, \ldots, n_{k-1}\rangle) \,\}.$$

So $\mathrm{Tree}(e,n)$ is the tree of non-past-secured sequence numbers for the oracle programs with index e and input n. (Its relation to the similar construction in Sec.2.3 is that $\mathrm{Tree}(e) = \{\, \langle n\rangle * \sigma \mid \sigma \in \mathrm{Tree}(e,n) \,\}$.) Let $<_{\mathrm{KB}}$ be the Kleene-Brouwer ordering of $\mathrm{Tree}(e,n)$; clearly $<_{\mathrm{KB}}$ is a well-ordering if and only if $\mathrm{Tree}(e,n)$ is well-founded. Therefore

$$n \in B \;\leftrightarrow\; \mathrm{Tree}(e,n) \text{ is well-founded } \;\leftrightarrow\; <_{\mathrm{KB}} \text{ is a well-ordering.}$$

Let f be a recursive function associating with any n an index of the Kleene-Brouwer ordering $<_{\mathrm{KB}}$ of $\mathrm{Tree}(e,n)$. Then

$$n \in B \;\leftrightarrow\; f(n) \in \mathcal{W}.$$

This shows that $\mathcal{W}$ is Π^1_1-complete. □

2.6.2. A Σ^0_1-definable extension $<'_{\mathcal{O}}$ of Kleene's $<_{\mathcal{O}}$. In order to obtain estimates of the complexity of $\mathcal{O}$ and $<_{\mathcal{O}}$ in the analytical hierarchy we inductively define an extension $<'_{\mathcal{O}}$ of Kleene's $<_{\mathcal{O}}$ by the clauses

(a) If $a \neq 0$, then $a <'_{\mathcal{O}} 2^a$.
(b) If $T(e,n,s)$ and $U(e,n,s) = a$, then $a <'_{\mathcal{O}} 3 \cdot 5^e$.
(c) $<'_{\mathcal{O}}$ is transitive.

LEMMA. *$<'_{\mathcal{O}}$ is Σ^0_1-definable.*

PROOF. We apply the theorem in 2.4.7 to the relation Q defined as follows.

$$\begin{aligned} Q(f,x) :\leftrightarrow\; & \exists_a(a \neq 0 \wedge x = \langle a, 2^a\rangle) \vee \\ & \exists_{e,n,s,a}\,[T(e,n,s) \wedge U(e,n,s) = a \wedge x = \langle a, 3\cdot 5^e\rangle] \vee \\ & \exists_{a,b,c}\,[f\langle a,b\rangle = 1 \wedge f\langle b,c\rangle = 1 \wedge x = \langle a,c\rangle]. \end{aligned}$$

Clearly Q is Σ^0_1-definable and monotone. Hence by the theorem the binary relation $<'_{\mathcal{O}}$ defined inductively by Q is Σ^0_1-definable as well. □

LEMMA. *If $b \in \mathcal{O}$, then $a <_{\mathcal{O}} b \leftrightarrow a <'_{\mathcal{O}} b$.*

PROOF. Let $b \in \mathcal{O}$. For the direction from left to right we use transfinite induction on $\beta := |b|_{\mathcal{O}}$. In the successor case $\beta = \alpha + 1$ we habe $b = 2^c$ with $c \in \mathcal{O}_\alpha$ and $a <_{\mathcal{O}} c \vee a = c$. In the first case we have by IH $a <'_{\mathcal{O}} c$, so $a <'_{\mathcal{O}} 2^c$ by $c <'_{\mathcal{O}} 2^c$ and transitivity of $<'_{\mathcal{O}}$. In the second case we have $a = c$, so $a <'_{\mathcal{O}} 2^c$ follows from $c <'_{\mathcal{O}} 2^c$. Now assume that β is a limit, so $b = 3 \cdot 5^e \in \mathcal{O}_\beta$ and $a <_{\mathcal{O}} \varphi_e(n)$ for some n. Then we have by IH $a <'_{\mathcal{O}} \varphi_e(n)$ and $\varphi_e(n) <'_{\mathcal{O}} 3 \cdot 5^e$ be definition of $<'_{\mathcal{O}}$, hence $a <'_{\mathcal{O}} 3 \cdot 5^e$ by transitivity.

For the converse we use the induction principle associated with the monotone inductive definition of $<'_{\mathcal{O}}$. If $a <'_{\mathcal{O}} 2^a$, then from $2^a \in \mathcal{O}$ we can conclude $a \in \mathcal{O}$ and hence $a <_{\mathcal{O}} 2^a$. If $a <'_{\mathcal{O}} 3 \cdot 5^e$ and $a = \varphi_e(n)$, then from $a = \varphi_e(n) <_{\mathcal{O}} \varphi_e(n+1)$ we can infer $a <_{\mathcal{O}} 3 \cdot 5^e$ by the definition of $<_{\mathcal{O}}$. Finally assume that $a <'_{\mathcal{O}} b$ is obtained by the transitivity rule, so $a <'_{\mathcal{O}} c$ and $c <'_{\mathcal{O}} b$ for some c. By IH we have $c <_{\mathcal{O}} b$, hence $c \in \mathcal{O}$ and again by IH $a <_{\mathcal{O}} c$. The transitivity of $<_{\mathcal{O}}$ implies $a <_{\mathcal{O}} b$. □

2.6.3. Π^1_1-definability of $\mathcal{O}$ and $<_{\mathcal{O}}$. Using the relation $<'_{\mathcal{O}}$ we are now able to define the set $\mathcal{O}$ in such a way that the theorem in 2.4.7 can be applied to obtain an estimate on its complexity in the analytical hierarchy. To this end we first introduce a set $\mathcal{O}'$ which can be seen easily to be Π^1_1-definable, and then prove $\mathcal{O} = \mathcal{O}'$. We define $\mathcal{O}'$ inductively by the clauses

(a) $1 \in \mathcal{O}'$.
(b) If $a \in \mathcal{O}'$, then also $2^a \in \mathcal{O}'$.
(c) If φ_e is total and for all n we have $\varphi_e(n) \in \mathcal{O}'$ and $\varphi_e(n) <'_{\mathcal{O}} \varphi_e(n+1)$, then $3 \cdot 5^e \in \mathcal{O}'$.

It is now easy to see that $\mathcal{O}'$ is Π^1_1-definable. Let

$$\begin{aligned} Q(f,a) \;:\leftrightarrow\; & a = 1 \vee \\ & \exists_b\,[f(b) = 1 \wedge a = 2^b] \vee \\ & \exists_e\,[\varphi_e \text{ total} \wedge \forall_n\, f(\varphi_e(n)) = 1 \wedge \forall_n\, \varphi_e(n) <'_{\mathcal{O}} \varphi_e(n+1) \wedge a = 3 \cdot 5^e]. \end{aligned}$$

Q is arithmetical, hence Π^1_1-definable and clearly $\mathcal{O}'$ is the set defined inductively by Q. Therefore $\mathcal{O}'$ is Π^1_1-definable.

THEOREM. *$\mathcal{O}$ and $<_{\mathcal{O}}$ are Π^1_1-definable.*

PROOF. From the lemma above (on the equivalence of $<_{\mathcal{O}}$ and $<'_{\mathcal{O}}$ below $b \in \mathcal{O}$) one can easily show $a \in \mathcal{O} \leftrightarrow a \in \mathcal{O}'$, using transfinite induction in one direction and induction on the definition of $\mathcal{O}'$ in the other direction. Hence $\mathcal{O}$ is Π^1_1-definable. Because of $a <_{\mathcal{O}} b \leftrightarrow b \in \mathcal{O} \wedge a <'_{\mathcal{O}} b$ also $<_{\mathcal{O}}$ is Π^1_1-definable. □

2.6.4. Π^1_1-completeness of $\mathcal{O}$ and $<_{\mathcal{O}}$. The last goal of this section is a proof of the Π^1_1-completeness of $\mathcal{O}$ and $<_{\mathcal{O}}$. This requires some additional observations concerning $+_{\mathcal{O}}$.

LEMMA. (a) *If $a +_{\mathcal{O}} b \in \mathcal{O}$, then also $a \in \mathcal{O}$ and $b \in \mathcal{O}$.*
(b) *If $a +_{\mathcal{O}} b = a +_{\mathcal{O}} c \in \mathcal{O}$, then $b = c$.*
(c) *For every d such that $a \leq_{\mathcal{O}} d <_{\mathcal{O}} a +_{\mathcal{O}} b$ we can find $c \in \mathcal{O}$ such that $c <_{\mathcal{O}} b$ and $d = a +_{\mathcal{O}} c$.*

PROOF. All three parts are proved simultaneously by transfinite induction on $|a +_{\mathcal{O}} b|_{\mathcal{O}}$.

(a). Assume $a +_{\mathcal{O}} b \in \mathcal{O}$. We distinguish cases according to the definition of $a +_{\mathcal{O}} b$. *Case* $a +_{\mathcal{O}} b = a$ with $b = 1$ and $a \neq 0$. Then clearly $a, b \in \mathcal{O}$. *Case* $a +_{\mathcal{O}} b = 2^{a+_{\mathcal{O}} c}$ with $b = 2^c$ and $c \neq 0$. Then $a +_{\mathcal{O}} c \in \mathcal{O}$ and $|a +_{\mathcal{O}} c|_{\mathcal{O}} < |a +_{\mathcal{O}} b|_{\mathcal{O}}$, hence by IH $a, c \in \mathcal{O}$, so also $b = 2^c \in \mathcal{O}$. *Case* $a +_{\mathcal{O}} b = 3 \cdot 5^{S(\underline{+}_{\mathcal{O}}, a) \circ e} \in \mathcal{O}$ with $b = 3 \cdot 5^e$. Then for every n we have $\varphi_{S(\underline{+}_{\mathcal{O}}, a) \circ e}(n) = a +_{\mathcal{O}} \varphi_e(n) \in \mathcal{O}$ and $|a +_{\mathcal{O}} \varphi_e(n)|_{\mathcal{O}} < |a +_{\mathcal{O}} b|_{\mathcal{O}}$. Hence by IH $a \in \mathcal{O}$ and $\varphi_e(n) \in \mathcal{O}$. It remains to show that $3 \cdot 5^e \in \mathcal{O}$. We first prove that $\varphi_e(n) <_{\mathcal{O}} \varphi_e(n+1)$. Because of

$$a \leq_{\mathcal{O}} a +_{\mathcal{O}} \varphi_e(n) <_{\mathcal{O}} a +_{\mathcal{O}} \varphi_e(n+1)$$

by IH(c) there exists $c \in \mathcal{O}$ such that $c <_{\mathcal{O}} \varphi_e(n+1)$ and $a +_{\mathcal{O}} \varphi_e(n) = a +_{\mathcal{O}} c$. Now IH(b) yields $\varphi_e(n) = c <_{\mathcal{O}} \varphi_e(n+1)$. Hence with $\beta := \sup_n |\varphi_e(n)|_{\mathcal{O}}$ we have $3 \cdot 5^e \in \mathcal{O}_\beta$. *Case* $a +_{\mathcal{O}} b = 7$. This contradicts our assumption $a +_{\mathcal{O}} b \in \mathcal{O}$.

(b). Assume $a +_{\mathcal{O}} b = a +_{\mathcal{O}} c \in \mathcal{O}$. Then by (a) $a, b, c \in \mathcal{O}$. *Case* $b = 1$. Then $a = a +_{\mathcal{O}} c$, hence $c = 1$ (we need $1 \leq_{\mathcal{O}} c$ and part (b) of the lemma in 2.5.3 here). *Case* $c = 1$. Similarly we obtain $b = 1$. *Case* $b = 2^{b_1}$ and $c = 2^{c_1}$. Then $a +_{\mathcal{O}} b = 2^{a+_{\mathcal{O}} b_1} = 2^{a+_{\mathcal{O}} c_1} = a +_{\mathcal{O}} c$, hence $a +_{\mathcal{O}} b_1 = a +_{\mathcal{O}} c_1 \in \mathcal{O}$ and therefore $b_1 = c_1$ by IH. *Case* $b = 3 \cdot 5^{e_1}$ and $c = 3 \cdot 5^{e_2}$. Then $S(\underline{+}_{\mathcal{O}}, a) \circ e_1 = S(\underline{+}_{\mathcal{O}}, a) \circ e_2$, hence $e_1 = e_2$ by the definition of $\circ$ (which can be assumed to be injective).

(c). Assume $a \leq_{\mathcal{O}} d <_{\mathcal{O}} a +_{\mathcal{O}} b$. By (a) we have $a \in \mathcal{O}$ and $b \in \mathcal{O}$. *Case* $a +_{\mathcal{O}} b = 2^{a+_{\mathcal{O}} c}$ with $b = 2^c$ and $c \neq 0$. Then $d <_{\mathcal{O}} a +_{\mathcal{O}} c$ or $d = a +_{\mathcal{O}} c$. In the first case by IH we can find $c_1 \in \mathcal{O}$ such that $c_1 <_{\mathcal{O}} c <_{\mathcal{O}} b$ and $d = a +_{\mathcal{O}} c_1$. In the second case we immediately have $c <_{\mathcal{O}} b$ and $d = a +_{\mathcal{O}} c$. *Case* $a +_{\mathcal{O}} b = 3 \cdot 5^{S(\underline{+}_{\mathcal{O}}, a) \circ e}$ with $b = 3 \cdot 5^e$. Then $a \leq_{\mathcal{O}} d <_{\mathcal{O}} a +_{\mathcal{O}} \varphi_e(n) \in \mathcal{O}$ for some n, hence by IH we can find $c \in \mathcal{O}$ such that $c <_{\mathcal{O}} \varphi_e(n)$ and $d = a +_{\mathcal{O}} c$. But clearly $c <_{\mathcal{O}} 3 \cdot 5^e = b$. □

Now we can show

THEOREM. *$\mathcal{O}$ and $<_{\mathcal{O}}$ are Π^1_1-complete.*

PROOF. By 2.6.3 we know that $\mathcal{O}$ is Π^1_1-definable. To prove the Π^1_1-completeness of $\mathcal{O}$ we begin as in the proof of the Π^1_1-completeness of $\mathcal{W}$ in 2.6.1. So let $B \subseteq \mathbb{N}$ be an arbitrary Π^1_1-definable set. Then for some $e \in \mathbb{N}$

$$n \in B \leftrightarrow \forall_g \exists_s T_1(e, n, \overline{g}(s)).$$

Let

$$\mathrm{Tree}(e, n) := \{ \langle n_0, \ldots, n_{l-1} \rangle \mid \forall_{k<l} \neg T_1(e, n, \langle n_0, \ldots, n_{k-1} \rangle) \}.$$

We are done if we can define a recursive function g such that

$$\mathrm{Tree}(e, n) \text{ is a well-founded tree } \leftrightarrow g(n) \in \mathcal{O}.$$

We prove more generally that there exists a recursive function h such that (2.8)

$$\mathrm{Tree}_y(e, n) := \{ z \mid y * z \in \mathrm{Tree}(e, n) \} \text{ is well-founded } \leftrightarrow h(y, n) \in \mathcal{O}.$$

With $g(n) := h(\langle\rangle, n)$ we then obtain our desired function g.

We define h via Kleene's Second Recursion Theorem as follows.

$$h(y,n) = \begin{cases} 1 & \text{if } y \notin \mathrm{Tree}(e,n) \\ 3 \cdot 5^{e_1} & \text{if } y \in \mathrm{Tree}(e,n) \end{cases}$$

with e_1 an index of the sequence f defined by

$$f(0) := 1,$$
$$f(m+1) := f(m) +_{\mathcal{O}} h(y * \langle m \rangle, n) +_{\mathcal{O}} 2^1.$$

Let us now prove (2.8). For the direction from left to right we use induction on $\mathrm{Tree}_y(e,n)$. So let $z \in \mathrm{Tree}_y(e,n)$, i.e., $y * z \in \mathrm{Tree}(e,n)$. Then for all m by IH or the definition of h we have $h(y * z * \langle m \rangle, n) \in \mathcal{O}$, hence $h(y * z, n) \in \mathcal{O}$. For the converse we use induction on $|h(y,n)|_{\mathcal{O}}$. So assume $h(y,n) \in \mathcal{O}$. If $y \notin \mathrm{Tree}(e,n)$, then $\mathrm{Tree}_y(e,n) = \emptyset$ and we are done. If however $y \in \mathrm{Tree}(e,n)$, then with $h(y,n) = 3 \cdot 5^e$ all $\varphi_e(m) = f(m)$ are in $\mathcal{O}$, hence by part (a) of the lemma in 2.6.4 all $h(y * \langle m \rangle, n)$ are in $\mathcal{O}$. Because of $|h(y * \langle m \rangle, n)|_{\mathcal{O}} < |h(y,n)|_{\mathcal{O}}$, by IH $\mathrm{Tree}_{y * \langle m \rangle}(e,n)$ is well-founded for all m, hence also $\mathrm{Tree}_y(e,n)$.

Now the Π^1_1-completeness of $<_{\mathcal{O}}$ follows from $a \in \mathcal{O} \leftrightarrow a <_{\mathcal{O}} 2^a$. $\square$

2.7. Notes

The example in 2.4.8 of a Δ^1_1-definable non-arithmetical set is from Hinman [1978].

Constructive ordinals have been introduced by Kleene; in our exposition in Sec.2.5 we have made use of Rogers [1967], Spector [1955] and Enderton and Luckham [1964]. The theory of inductive definitions has first been made explicit in work of Spector [1961]; however, many of the relevant notions and arguments were already present in previous papers of Kleene. In Sec.2.6 we followed [Kleene, 1944, 1955].

CHAPTER 3

Hyperarithmetical Sets and Functions

We have seen that the arithmetical relations can be exhausted by iterating the recursive successor A'. So if we define

$$A_0 := \emptyset,$$
$$A_{n+1} := A'_n,$$

then every arithmetical set is recursive in some A_n and conversely. Now if we define

$$A_\omega := \{\, n \mid (n)_0 \in A_{(n)_1} \,\},$$

then we obtain a non-arithmetical set. If we again apply the successor operation, we obtain

$$A_{\omega+1} := A'_\omega,$$
$$A_{\omega+2} := A''_\omega \text{ etc.}$$

Clearly this process can be extended into the transfinite. In the present chapter we will study the system of sets obtained in this way, where as index set we use the *constructive* ordinals. We obtain the hyperarithmetical hierarchy of Kleene, which – as we will prove – exhausts the Δ^1_1-definable sets. A further characterization has been given by Kreisel: the hyperarithmetical functions form the least class of functions closed against "recursive in" and the axiom of choice in the form (with R arithmetical)

$$\forall_n \exists_f R(f,n) \to \exists_g \forall_n R((g)_n, n).$$

Moreover we will prove in this chapter the Hyperarithmetical Quantifier Theorem of Kleene and as applications some results on paths in $\mathcal{O}$.

3.1. The Hyperarithmetical Hierarchy

The hyperarithmetical hierarchy is formed by recursive closure of its "backbone" $\{\, H_a \mid a \in \mathcal{O} \,\}$. The sets H_a themselves are built by iterating the recursive successor (or jump) along the constructive ordinals, with diagonalization in the limit case. It turns out – and this is the most important result in this section – that modulo recursive equivalence H_a does not depend on the special choice of a but only on the ordinal denoted by a. More precisely, if $|a|_\mathcal{O} = |b|_\mathcal{O}$, then H_a is recursive in H_b and conversely.

3.1.1. Lifting relative recursiveness. We shall write $A \leq [e]\, B$ for $\forall_n(c_A(n) = \Phi_e(c_B, n))$, read as "$A$ is recursive in B with index e".

LEMMA. *There is an elementary function $\circ$ such that from $A \leq [e_1]B$ and $B \leq [e_2]C$ we can conclude $A \leq [e_1 \circ e_2]C$.*

PROOF.

$$\begin{aligned} c_A(n) &= \Phi_{e_1}(c_B, n) \\ &= \Phi_{e_1}(\lambda x c_B(x), n) \\ &= \Phi_{e_0}(e_1, e_2, c_C, n) \quad \text{for a fixed } e_0 \\ &= \Phi_{S(e_0,e_1,e_2)}(c_C, n), \end{aligned}$$

where $S(e_0, e_1, e_2)$ of course stands for $S(S(e_0, e_1), e_2)$. So let $e_1 \circ e_2 := S(e_0, e_1, e_2)$. □

Note that the function $\circ$ introduced here is is *not* the same as the function also denoted by $\circ$ introduced in Sec.2.5 (with the property $\varphi_{e_1 \circ e_2}(\vec{n}) = \varphi_{e_1}(\varphi_{e_2}(\vec{n}))$); it will always be clear from the context which one is meant. We write $e^{(i)}$ for $e \circ e \cdots \circ e$ with i occurrences of e (associated to the left).

PROPOSITION. *If $A \leq [e]B$, then $A' \leq [N(e)]B'$ with an elementary function N.*

PROOF. We have

$$\begin{aligned} n \in A' &\leftrightarrow \exists_s T_2((n)_0, c_A, (n)_1, s) \\ &\leftrightarrow \exists_s T_2((n)_0, \lambda x \Phi_e(c_B, x), (n)_1, s) \\ &\leftrightarrow \exists_s R(c_B, e, n, s) \quad \text{for a recursive } R, \text{ since } \lambda x \Phi_e(c_B, x) \text{ is total} \\ &\leftrightarrow W_{e_0}(c_B, e, n) \quad \text{for a fixed } e_0 \\ &\leftrightarrow W_{S(e_0,e)}(c_B, n) \\ &\leftrightarrow \exists_s T_2(S(e_0, e), c_B, n, s) \\ &\leftrightarrow \langle S(e_0, e), n \rangle \in B', \end{aligned}$$

hence

$$c_{A'}(n) = c_{B'}(\langle S(e_0, e), n \rangle) = \Phi_{e_1}(c_{B'}, e, n) = \Phi_{S(e_1,e)}(c_{B'}, n)$$

for a fixed e_1. Let $N(e) := S(e_1, e)$. □

3.1.2. Definition and simple properties of the sets H_a. We now define the *hyperarithmetical hierarchy*, consisting of sets $H_a \subseteq \mathbb{N}$ for every $a \in \mathcal{O}$. They are defined by

$$\begin{aligned} H_1 &:= \emptyset, \\ H_{2^a} &:= \big(H_a\big)', \\ H_{3 \cdot 5^e} &:= \{\, n \mid (n)_0 <_{\mathcal{O}} 3 \cdot 5^e \wedge (n)_1 \in H_{(n)_0} \,\}. \end{aligned}$$

A set A is called *hyperarithmetical* if A is recursive in some H_a with $a \in \mathcal{O}$.

It will be useful to introduce the following notations. We shall say that *a denotes a successor number* if $a = 2^{(a)_0}$, and that *a denotes a limit* if $a = 3 \cdot 5^{(a)_2}$. Let $a^* := 2^a$, called *ordinal successor* of a. Finally let

$$a^- := \begin{cases} (a)_0 & \text{if } a \text{ denotes a successor number} \\ a & \text{otherwise.} \end{cases}$$

a^- is called *ordinal predecessor* of a.

Our goal in the present section is to show that H_a is recursive in H_b and conversely, provided $|a|_{\mathcal{O}} = |b|_{\mathcal{O}}$; this will be achieved in 3.1.3. Let us begin with the special case where a and b are comparable with respect to $\leq_{\mathcal{O}}$.

LEMMA. *If $a \leq_{\mathcal{O}} b$, then H_a is recursive in H_b, effectively in a and b. In fact, there is an elementary function f such that $H_a \leq [f(a,b)]\, H_b$.*

PROOF. *Case* 1: $a = b$. Then clearly $H_a \leq [e_0]\, H_b$ with a fixed e_0.

Case 2: b can be reached from a in finitely many successor steps, i.e., $b = a^{*\cdots *}$ with i stars. Since for a certain e_1 we have $A \leq [e_1]A'$, we can conclude from the proposition in 3.1.1 that $H_a \leq [e_1^{(i)}]\, H_b$.

Case 3: Otherwise. Then for some c denoting a limit we have $a \leq_{\mathcal{O}} c \leq_{\mathcal{O}} b$ and $b = c^{*\cdots *}$ with $i \geq 0$ stars. So again we have $H_c \leq [e_1^{(i)}]\, H_b$. Moreover

$$\begin{aligned} n \in H_a &\leftrightarrow \langle a, n\rangle \in H_c \\ &\leftrightarrow \Phi_{e_2}(c_{H_c}, a, n) = 1 \quad \text{for some fixed } e_2 \\ &\leftrightarrow \Phi_{S(e_2,a)}(c_{H_c}, n) = 1. \end{aligned}$$

Therefore $H_a \leq [S(e_2, a)]\, H_c$ and hence $H_a \leq [e_1^{(i)} \circ S(e_2, a)]\, H_b$. □

The following auxiliary proposition will be needed in 3.1.3.

PROPOSITION. *For every $b \in \mathcal{O}$ the set*

$$\mathcal{O}_{|b|} := \{\, a \in \mathcal{O} \mid |a|_{\mathcal{O}} < |b|_{\mathcal{O}} \,\}$$

is recursive in H_{b^*}, effectively in b.

PROOF. We apply the Recursion Lemma to $\mathcal{O}$ and $<_{\mathcal{O}}$, with respect to the relation

$$R(b, e) :\leftrightarrow\ b \in \mathcal{O} \wedge \mathcal{O}_{|b|} \leq [e]\, H_{b^*}.$$

So we need to construct a recursive function χ such that for arbitrary t and $b \in \mathcal{O}$ from the assumption

$$\forall_{c <_{\mathcal{O}} b}\, R(c, \varphi_t(c)) \tag{3.1}$$

we can infer $R(b, \chi(t, b))$.

Case 1: $b = 1$ or $b = 1^*$. Then we have $\mathcal{O}_{|1|} = \emptyset \leq [e_0]\, H_{1^*}$ and $\mathcal{O}_{|1^*|} = \{1\} \leq [e_1]\, H_{1^{**}}$ for certain e_0, e_1, since $\emptyset$ and $\{1\}$ are recursive.

Case 2: b and b^- both denote successor numbers. Then we have

$$\begin{aligned} a \in \mathcal{O}_{|b|} &\leftrightarrow a \in \mathcal{O} \wedge |a|_{\mathcal{O}} < |b|_{\mathcal{O}} \\ &\leftrightarrow (a \in \mathcal{O} \wedge |a|_{\mathcal{O}} < |b^-|_{\mathcal{O}}) \vee (a \in \mathcal{O} \wedge a = 2^{(a)_0} \wedge |a^-|_{\mathcal{O}} < |b^-|_{\mathcal{O}}) \\ &\leftrightarrow a \in \mathcal{O}_{|b^-|} \vee (a = 2^{(a)_0} \wedge a^- \in \mathcal{O}_{|b^-|}). \end{aligned}$$

Hence $\mathcal{O}_{|b|} \leq [e_2]\mathcal{O}_{|b^-|}$ for some e_2.. Now from our assumption (3.1) we know that $\mathcal{O}_{|b^-|} \leq [\varphi_t(b^-)]\, H_b$, and finally $H_b \leq [e_3]\, H_{b^*}$ for some e_3. Putting these together we obtain

$$\mathcal{O}_{|b|} \leq [e_2 \circ \varphi_t(b^-) \circ e_3]\, H_{b^*}.$$

So in this case we define $\chi(t, b) := e_2 \circ \varphi_t(b^-) \circ e_3$.

Case 3: b denotes a successor number and b^- denotes a limit. In this case

$$\begin{aligned}
a \in \mathcal{O}_{|b|} &\leftrightarrow a \in \mathcal{O} \wedge |a|_{\mathcal{O}} < |b|_{\mathcal{O}} \\
&\leftrightarrow a \in \mathcal{O}_{|b^-|} \vee \big(a \in \mathcal{O} \wedge a = 3 \cdot 5^{(a)_2} \wedge |a|_{\mathcal{O}} \leq |b^-|_{\mathcal{O}}\big) \\
&\leftrightarrow a \in \mathcal{O}_{|b^-|} \vee \exists_{e \leq a} \big[a = 3 \cdot 5^e \wedge \varphi_e \text{ is total} \wedge \\
&\qquad\qquad \forall_{n,u,v} \left(\varphi_e(n) = u \wedge \varphi_e(n+1) = v \to u <'_{\mathcal{O}} v\right) \wedge \\
&\qquad\qquad \forall_{n,v} \left(\varphi_e(n) = v \to v \in \mathcal{O}_{|b^-|}\right)\big] \\
&\leftrightarrow a \in \mathcal{O}_{|b^-|} \vee \exists_{e \leq a} \left[\langle e, a\rangle \in A \wedge e \in B\right]
\end{aligned}$$

where A is Π^0_2-definable and B is Π^0_1-definable in $\mathcal{O}_{|b^-|}$. Because of (3.1)

$$\mathcal{O}_{|b^-|} \leq [\varphi_t(b^-)]\, H_b \leq [e_3]\, H_{b^*},$$
$$A \leq [e_4]\, \emptyset'' = H_{1^{**}} \leq [f(1^{**}, b^*)]\, H_{b^*} \quad \text{with } f \text{ from the lemma above,}$$
$$B \leq [e_5]\big(\mathcal{O}_{|b^-|}\big)' \leq [N(\varphi_t(b^-))]\, H_{b^*} \quad \text{with } N \text{ from the proposition in 3.1.1.}$$

From all this we obtain

$$\begin{aligned}
a \in \mathcal{O}_{|b|} &\leftrightarrow \Phi_{\varphi_t(b^-) \circ e_3}(c_{H_{b^*}}, a) = 1 \vee \\
&\qquad \exists_{e \leq a} \left[\Phi_{e_4 \circ f(1^{**}, b^*)}(c_{H_{b^*}}, a) = 1 \wedge \Phi_{e_5 \circ N(\varphi_t(b^-))}(c_{H_{b^*}}, a) = 1\right] \\
&\leftrightarrow \Phi_{e_6}(c_{H_{b^*}}, t, b, a) = 1 \quad \text{for some } e_6 \\
&\leftrightarrow \Phi_{S(e_6,t,b)}(c_{H_{b^*}}, a) = 1.
\end{aligned}$$

So we can define $\chi(t, b) := S(e_6, t, b)$.

Case 4: b denotes a limit. First note that

$$a \in \mathcal{O}_{|b|} \;\leftrightarrow\; a \in \mathcal{O} \wedge |a|_{\mathcal{O}} < |b|_{\mathcal{O}} \;\leftrightarrow\; \exists_n \left(a \in \mathcal{O}_{|\varphi_{(b)_2}(n)|}\right).$$

From our assumption (3.1) and the lemma above we obtain for every n

$$\mathcal{O}_{|\varphi_{(b)_2}(n)|} \leq [\varphi_t(\varphi_{(b)_2}(n))]\, H_{(\varphi_{(b)_2}(n))^*} \leq [f(\varphi_{(b)_2}(n)^*, b)]\, H_b.$$

Hence we have

$$\begin{aligned}
a \in \mathcal{O}_{|b|} &\leftrightarrow \exists_n \left(a \in \mathcal{O}_{|\varphi_{(b)_2}(n)|}\right) \\
&\leftrightarrow \exists_n\, \Phi_{\varphi_t(\varphi_{(b)_2}(n)) \circ f((\varphi_{(b)_2}(n))^*, b)}(c_{H_b}, a) = 1 \\
&\leftrightarrow \exists_s\, T_2(c_{H_b}, e_7, \langle t, b, a\rangle, s) \quad \text{for some } e_7 \\
&\leftrightarrow \langle e_7, \langle t, b, a\rangle\rangle \in H_{b^*} \\
&\leftrightarrow \Phi_{e_8}(c_{H_{b^*}}, t, b, a) = 1 \quad \text{for some } e_8 \\
&\leftrightarrow \Phi_{S(e_8,t,b)}(c_{H_{b^*}}, a) = 1.
\end{aligned}$$

So let $\chi(t, b) := S(e_8, t, b)$. This proves the proposition. □

3.1.3. Spector's theorem.

THEOREM (Spector). *If $a, b \in \mathcal{O}$ and $|a|_{\mathcal{O}} \leq |b|_{\mathcal{O}}$, then H_a is recursive in H_b, effectively in a and b.*

PROOF. Again we use the Recursion Lemma. Let

$$M := \{\, \langle a, b\rangle \mid a \in \mathcal{O} \wedge b \in \mathcal{O} \wedge |a|_{\mathcal{O}} \leq |b|_{\mathcal{O}} \,\}.$$

For $\langle a,b\rangle, \langle c,d\rangle \in M$ let

$$\langle a,b\rangle \prec \langle c,d\rangle \;:\leftrightarrow\; (a <_{\mathcal{O}} c \wedge b \leq_{\mathcal{O}} d) \vee (a \leq_{\mathcal{O}} c \wedge b <_{\mathcal{O}} d).$$

Clearly $\prec$ is a well-founded ordering on M. Let

$$R(p,e) \;:\leftrightarrow\; p \in M \wedge H_{(p)_0} \leq [e]\, H_{(p)_1}.$$

Now fix $a, b \in \mathcal{O}$ such that $|a|_{\mathcal{O}} \leq |b|_{\mathcal{O}}$ and assume

$$\forall_{c,d}\,(\langle c,d\rangle \prec \langle a,b\rangle \to H_c \leq [\varphi_t\langle c,d\rangle]\, H_d). \tag{3.2}$$

We want to define a recursive function χ such that

$$H_a \leq [\chi(t,a,b)]\, H_b.$$

Case 1: $a = 1$. Then $H_1 = \emptyset \leq [e_0]\, H_b$ for some e_0.

Case 2: a, b both denote a successor numbers. Then $\langle a^-, b^-\rangle \prec \langle a,b\rangle$, hence because of our assumption (3.2) $H_{a^-} \leq [\varphi_t\langle a^-, b^-\rangle] H_{b^-}$. We obtain $H_a \leq [N(\varphi_t\langle a^-, b^-\rangle)]\, H_b$. So let $\chi(t,a,b) := N(\varphi_t\langle a^-, b^-\rangle)$.

Case 3: a denotes a successor number and b denotes a limit. Then we have $|a|_{\mathcal{O}} < |b|_{\mathcal{O}}$. Using 2.6.2 we obtain

$$<'_{\mathcal{O}} \leq [e_1]\, \emptyset' = H_{1^*} \leq [f(1^*, b)]\, H_b$$

and for $c <_{\mathcal{O}} b$ with $|a|_{\mathcal{O}} < |c|_{\mathcal{O}}$ using 3.1.2

$$\mathcal{O}_{|c|} \leq [g(c)] H_{c^*} \leq [f(c^*, b)]\, H_b.$$

Hence for all c we have (recall that a, b are fixed)

$$\begin{aligned} c <_{\mathcal{O}} b \wedge |a|_{\mathcal{O}} < |c|_{\mathcal{O}} &\leftrightarrow c <'_{\mathcal{O}} b \wedge a \in \mathcal{O}_{|c|} \\ &\leftrightarrow \Phi_{e_1 \circ f(1^*,b)}(c_{H_b}, c, b) = 1 \wedge \Phi_{g(c)\circ f(c^*,b)}(c_{H_b}, a) = 1 \\ &\leftrightarrow \Phi_{e_2}(c_{H_b}, a, b, c) = 1 \quad \text{for some } e_2. \end{aligned}$$

Hence there is a number e_3 such that

$$c_{a,b} := \Phi_{e_3}(c_{H_b}, a, b) = \mu_c\,(c <_{\mathcal{O}} b \wedge |a|_{\mathcal{O}} < |c|_{\mathcal{O}}).$$

From our assumption (3.2), $\langle a, c_{a,b}\rangle \prec \langle a,b\rangle$ and 3.1.2 we know that

$$H_a \leq [\varphi_t\langle a, c_{a,b}\rangle]\, H_{c_{a,b}} \leq [f(c_{a,b}, b)]\, H_b.$$

Hence we obtain

$$\begin{aligned} n \in H_a &\leftrightarrow \Phi_{\varphi_t\langle a, c_{a,b}\rangle \circ f(c_{a,b},b)}(c_{H_b}, n) = 1 \\ &\leftrightarrow \Phi_{e_4}(c_{H_b}, t, a, b, n) = 1 \quad \text{for some } e_4 \\ &\leftrightarrow \Phi_{S(e_4,t,a,b)}(c_{H_b}, n) = 1. \end{aligned}$$

So let $\chi(t,a,b) := S(e_4, t, a, b)$.

Case 4: a denotes a limit:

$$\begin{aligned} n \in H_a &\leftrightarrow (n)_0 <_{\mathcal{O}} a \wedge (n)_1 \in H_{(n)_0} \\ &\leftrightarrow (n)_0 <'_{\mathcal{O}} a \wedge \Phi_{\varphi_t\langle (n)_0, b\rangle}(c_{H_b}, (n)_1) = 1 \quad \text{because of (3.2)} \\ &\leftrightarrow \Phi_{e_1 \circ f(1^*,b)}(c_{H_b}, (n)_0, a) = 1 \wedge \Phi_{\varphi_t\langle (n)_0, b\rangle}(c_{H_b}, (n)_1) = 1 \\ &\leftrightarrow \Phi_{e_5}(c_{H_b}, t, a, b, n) = 1 \quad \text{for some } e_5 \\ &\leftrightarrow \Phi_{S(e_5,t,a,b)}(c_{H_b}, n) = 1. \end{aligned}$$

So let $\chi(t,a,b) := S(e_5, t, a, b)$. □

3.2. The Characterization Theorem of Souslin/Kleene

We now show that the class of Δ^1_1-definable sets is exactly the class of all hyperarithmetical sets. For one direction we shall make use of the facts that the class of all Δ^1_1-definable sets is recursively closed (in fact Δ^1_1-closed) and that H_a is Δ^1_1-definable for every $a \in \mathcal{O}$. In the other direction we essentially use the Boundedness Lemma, which says that every Δ^1_1-definable subset of $\mathcal{O}$ is bounded (in $\mathcal{O}$).

3.2.1. Δ^1_1-definability of hyperarithmetical sets.

LEMMA. *If a set A is Δ^1_1-definable in a Δ^1_n-definable set B, then A is Δ^1_n-definable.*

PROOF. Assume that A is Δ^1_1-definable in a Δ^1_n-definable set B. Then there is a recursive relation R such that

$$\begin{aligned} n \in A &\leftrightarrow \forall_f \exists_s R(\overline{c_B}(s), \overline{f}(s), n) \\ &\leftrightarrow \forall_f \exists_s \exists_x [R(x, \overline{f}(s), n) \wedge \mathrm{lh}(x) = s \wedge \forall_{i<s} ((x)_i \leq 1) \wedge \\ &\qquad \forall_{i<s} ((x)_i = 1 \leftrightarrow i \in B)]. \end{aligned}$$

This shows that A is Π^1_n-definable. Now since there is a similar representation of the complement $\mathbb{N} \setminus A$ we obtain a Δ^1_n-definition of A. □

Call a number e a Π^1_1-*index* of a set A if

$$n \in A \;\leftrightarrow\; \forall_f \exists_s T_2(f, e, n, s).$$

PROPOSITION. *For every $a \in \mathcal{O}$ the set H_a is Δ^1_1-definable effectively in a, i.e., there is a recursive function f such that for every $a \in \mathcal{O}$ the number $(f(a))_0$ is a Π^1_1-index of H_a and $(f(a))_1$ is a Π^1_1-index of $\mathbb{N} \setminus H_a$.*

PROOF. We apply the Recursion Lemma to $\mathcal{O}$ and $<_{\mathcal{O}}$ with respect to the relation

$$R(a, e) \;:\leftrightarrow\; a \in \mathcal{O} \wedge (e)_0 \text{ is } \Pi^1_1\text{-index of } H_a \wedge (e)_1 \text{ is } \Pi^1_1\text{-index of } \mathbb{N} \setminus H_a.$$

So fix $a \in \mathcal{O}$ and assume

$$\forall_{b <_{\mathcal{O}} a} R(b, \varphi_t(b)). \tag{3.3}$$

We want to define a recursive function χ such that $R(a, \chi(t, a))$.

Case 1: $a = 1$. Let $\chi(t, a) := \langle e_0, e_1 \rangle$ with e_0 Π^1_1-index of $H_1 = \emptyset$ and e_1 Π^1_1-index of $\mathbb{N} \setminus H_1 = \mathbb{N}$.

Case 2: a denotes a successor number. From (3.3) we obtain that

$$\begin{aligned} e_0^- &:= (\varphi_t(a^-))_0 \quad \text{is } \Pi^1_1\text{-index of } H_{a^-}, \text{ and} \\ e_1^- &:= (\varphi_t(a^-))_1 \quad \text{is } \Pi^1_1\text{-index of } \mathbb{N} \setminus H_{a^-}. \end{aligned}$$

Since H_a is Σ^0_1-definable in H_{a^-} we obtain for a recursive relation P:

$$\begin{aligned} n \in H_a &\leftrightarrow \exists_s P(\overline{c_{H_{a^-}}}(s), n) \\ &\leftrightarrow \exists_s \exists_x (P(x, n) \wedge \wedge \mathrm{lh}(x) = s \wedge \\ &\qquad \forall_{i<s} [((x)_i = 1 \wedge i \in H_{a^-}) \vee ((x)_i = 0 \wedge i \notin H_{a^-})]) \\ &\leftrightarrow \exists_s \exists_x (P(x, n) \wedge \wedge \mathrm{lh}(x) = s \wedge \\ &\qquad \forall_{i<s} [((x)_i = 1 \wedge \forall_f \exists_z T_2(f, e_0^-, i, z)) \vee \end{aligned}$$

$$((x)_i = 0 \wedge \forall_f \exists_z T_2(f, e_1^-, i, z))]$$
$$\leftrightarrow \forall_f \exists_m T_2(f, e_2, e_0^-, e_1^-, n, m) \quad \text{for some } e_2$$
$$\leftrightarrow \forall_f \exists_m T_2(f, S(e_2, e_0^-, e_1^-), n, m).$$

Let $e_0 := S(e_2, e_0^-, e_1^-)$. Similarly we can construct a Π^1_1-index e_1 of $\mathbb{N} \setminus H_a$. Let $\chi(t, a) := \langle e_0, e_1 \rangle$.

Case 3: a denotes a limit. Then

$$\begin{aligned} n \in H_a &\leftrightarrow (n)_0 <_{\mathcal{O}} a \wedge (n)_1 \in H_{(n)_0} \\ &\leftrightarrow (n)_0 <_{\mathcal{O}} a \wedge \forall_f \exists_s T_2(f, (\varphi_t((n)_0))_0, (n)_1, s) \quad \text{because of (3.3)} \\ &\leftrightarrow \forall_f \exists_m T_2(f, e_3, t, a, n, m) \quad \text{for some } e_3 \\ &\leftrightarrow \forall_f \exists_m T_2(f, S(e_3, t, a), n, m). \end{aligned}$$

Let $e_0 := S(e_3, t, a)$. Similarly we can construct a Π^1_1-index e_1 of $\mathbb{N} \setminus H_a$. Let $\chi(t, a) := \langle e_0, e_1 \rangle$. □

COROLLARY. *Every hyperarithmetical set is Δ^1_1-definable.*

PROOF. Let A be hyperarithmetical, i.e., recursive in H_a for some $a \in \mathcal{O}$. Then H_a is Δ^1_1-definable by the proposition above, hence also A is Δ^1_1-definable by the lemma. □

3.2.2. Boundedness and separation.

LEMMA (Boundedness). *Every Σ^1_1-definable subset of $\mathcal{O}$ is bounded in $\mathcal{O}$.*

PROOF. Recall that for $e \in \mathcal{W}$ we write $|e|_{\mathcal{W}}$ for the order type of the well-ordering $<_{\mathcal{W}}$ denoted by e, so $n <_e m \leftrightarrow \varphi_e\langle n, m \rangle = 1$. Assume that $A \subseteq \mathcal{O}$ is Σ^1_1-definable and unbounded, i.e., $\forall_{b \in \mathcal{O}} \exists_{a \in A} (|b|_{\mathcal{O}} \leq |a|_{\mathcal{O}})$. Then we could write

$$\begin{aligned} e \in \mathcal{W} \leftrightarrow \varphi_e \in \mathrm{LinOrd} \wedge \exists_f \exists_{a \in A} (&f \text{ injective order isomorphism between} \\ &<_e \text{ and } <'_{\mathcal{O}} \restriction \{\, b \mid b <'_{\mathcal{O}} a \,\}). \end{aligned}$$

This would be a Σ^1_1-definition of $\mathcal{W}$, contradicting 2.6.1. □

LEMMA (Separation). *Any two disjoint Σ^1_1-definable sets A and B can be separated by a hyperarithmetical set C.*

PROOF. Because of the Π^1_1-completeness of $\mathcal{O}$ (cf. 2.6.4) there is a recursive function f such that $n \notin B \leftrightarrow f(n) \in \mathcal{O}$. Let $D := \{\, f(n) \mid n \in A \,\}$. Then D is a Σ^1_1-definable subset of $\mathcal{O}$, since

$$a \in D \leftrightarrow \exists_n (n \in A \wedge f(n) = a).$$

Now the Boundedness Lemma yields $b \in \mathcal{O}$ with $D \subseteq \mathcal{O}_{|b|}$. Let $C := \{\, n \mid f(n) \in \mathcal{O}_{|b|} \,\}$. C is hyperarithmetical by 3.1.2, and clearly we have $A \subseteq C$ and $C \subseteq \mathbb{N} \setminus B$. □

3.2.3. The theorem of Souslin and Kleene.

THEOREM (Souslin/Kleene). *The hyperarithmetical sets are exactly the Δ^1_1-definable sets.*

PROOF. By the Corollary in 3.2.1 every hyperarithmetical set is Δ^1_1-definable. Now assume that A is Δ^1_1-definable. Then A and $\mathbb{N} \setminus A$ are Σ^1_1-definable and hence by the Separation Lemma can be separated by a hyperarithmetical set C. Therefore A is hyperarithmetical. □

3.3. Hyperarithmetical Functions and the Axiom of Choice

The goal of this section is to prove that the class $\mathcal{H}$ of all hyperarithmetical functions is the smallest recursively closed class $\mathcal{C}$ of functions satisfying the following version of the axiom of choice

$$\forall_n \exists_{f \in \mathcal{C}} P(f, n) \to \exists_{g \in \mathcal{C}} \forall_n P((g)_n, n)$$

with an arithmetical relation P.

3.3.1. Π^1_1-Uniformization. As a tool we use the fact that every binary Π^1_1-definable relation contains a Π^1_1-definable subrelation which is the graph of a partial function.

LEMMA (Π^1_1-Uniformization). *For every binary Π^1_1-definable relation P we can find effectively in P a binary Π^1_1-definable relation Q such that*

(a) $Q(n, m) \to P(n, m)$,
(b) $Q(n, m_1) \wedge Q(n, m_2) \to m_1 = m_2$,
(c) $\exists_m P(n, m) \to \exists_m Q(n, m)$.

PROOF. From the Π^1_1-completeness of $\mathcal{W}$ we can conclude that there is a recursive function h such that

$$P(n, m) \;\leftrightarrow\; h(n, m) \in \mathcal{W}.$$

Let

$$\begin{aligned} Q(n, m) :\leftrightarrow\; & P(n, m) \wedge \\ & \forall_k \left(P(n, k) \to |h(n, m)|_{\mathcal{W}} \leq |h(n, k)|_{\mathcal{W}}\right) \wedge \\ & \forall_{k<m} \left(P(n, k) \to |h(n, m)|_{\mathcal{W}} < |h(n, k)|_{\mathcal{W}}\right). \end{aligned}$$

If $Q(n, m)$, then m is the numerically least l such that the ordinal $|h(n, l)|_{\mathcal{W}}$ is minimal in the set $\{\, |h(n, k)|_{\mathcal{W}} \mid P(n, k) \,\}$. Clearly Q has the properties (a)–(c) stated in the Π^1_1-Uniformization Lemma.

We claim that there exist Σ^1_1-definable relations $\mathcal{W}_{\leq}$ and $\mathcal{W}_{<}$ such that for all $e_1 \in \mathbb{N}$ and $e_2 \in \mathcal{W}$

$$\begin{aligned} \mathcal{W}_{\leq}(e_1, e_2) &\leftrightarrow e_1 \in \mathcal{W} \wedge |e_1|_{\mathcal{W}} \leq |e_2|_{\mathcal{W}}, \\ \mathcal{W}_{<}(e_1, e_2) &\leftrightarrow e_1 \in \mathcal{W} \wedge |e_1|_{\mathcal{W}} < |e_2|_{\mathcal{W}}. \end{aligned}$$

To see this let

$$\begin{aligned} \mathcal{W}_{\leq}(e_1, e_2) :\leftrightarrow\; & \varphi_{e_1} \in \mathrm{LinOrd} \wedge \exists_f (f \text{ is an injective order isomorphism} \\ & \qquad \text{from } <_{e_1} \text{ into } <_{e_2}), \\ \mathcal{W}_{<}(e_1, e_2) :\leftrightarrow\; & \varphi_{e_1} \in \mathrm{LinOrd} \wedge \exists_f \exists_k (f \text{ is an order isomorphism} \\ & \qquad \text{between } <_{e_1} \text{ and } <_{e_2} \restriction \{\, n \mid n <_{e_2} k \,\}). \end{aligned}$$

Now for Q we can give the following Π^1_1-definition.

$$\begin{aligned} Q(n,m) \leftrightarrow P(n,m) \wedge \\ \neg\exists_k \mathcal{W}_<(h(n,k),h(n,m)) \wedge \\ \neg\exists_{k<m} \mathcal{W}_\leq(h(n,k),h(n,m)). \end{aligned}$$ □

From the definition of Q we can see immediately that there is a recursive function k assigning to every Π^1_1-index e of P a Π^1_1-index $k(e)$ of Q. Let $\mathcal{W}^1_e$ denote the relation with Π^1_1-index e. So if $P = \mathcal{W}^1_e$, then $Q = \mathcal{W}^1_{k(e)}$. Let φ^1_e denote the partial function whose graph is $\mathcal{W}^1_{k(e)}$.

3.3.2. Hyperarithmetical functions. A unary total function f is said to be *hyperarithmetical* if its graph

$$G_f \;:=\; \{\,(n,m) \mid f(n) = m\,\}$$

is hyperarithmetical. Let $\mathcal{H}$ be the class of all hyperarithmetical functions.

Notice that the following are equivalent.

(a) $f \in \mathcal{H}$.
(b) G_f is Σ^1_1-definable.
(c) G_f is Π^1_1-definable.

This follows from the Theorem of Souslin/Kleene and the representation

$$\neg G_f(n,m) \;\leftrightarrow\; \exists_k\,(k \neq m \wedge G_f(n,k)).$$

Corollary. *If P is a Π^1_1-definable relation, then*

$$\forall_n\exists_m P(n,m) \to \exists_{f\in\mathcal{H}}\forall_n P(n,f(n)).$$

Proof. This is an easy consequence of the fact above together with Π^1_1-Uniformization Lemma. □

The following proposition gives a stronger version of this fact; it already is one half of Kreisel's Theorem.

Proposition. *If P is a Π^1_1-definable relation, then*

$$\forall_n\exists_{f\in\mathcal{H}}\, P(f,n) \to \exists_{g\in\mathcal{H}}\forall_n\, P((g)_n,n).$$

Here $(g)_n(m) := g(\langle n,m\rangle)$.

Proof. Assume $\forall_n\exists_{f\in\mathcal{H}}\, P(f,n)$. Then we have

$$\forall_n\exists_e\,[\varphi^1_e \text{ is total} \wedge \forall_f\,(\forall_{m,k}\,(\varphi^1_e(m) = k \to f(m) = k) \to P(f,n))].$$

Since "φ^1_e is total" has the form $\forall_u\exists_v\forall_f\exists_s\, T_2(f,k(e),u,v,s)$, the above formula can be written in the form $\forall_n\exists_e S(n,e)$ with a Π^1_1-definable relation S. Now the Corollary above (to Π^1_1-Uniformization) yields $h \in \mathcal{H}$ such that $\forall_n S(n,h(n))$, hence $\forall_n P(\varphi^1_{h(n)},n)$. We must find $g \in \mathcal{H}$ such that $(g)_n = \varphi^1_{h(n)}$. Define $g(p) := \varphi^1_{h((p)_0)}((p)_1)$. Because of

$$\begin{aligned} g(n) = m &\leftrightarrow \varphi^1_{h((n)_0)}((n)_1) = m \\ &\leftrightarrow \forall_f\exists_s\, T(f,k(h((n)_0)),(n)_1,m,s) \end{aligned}$$

we have $g \in \mathcal{H}$, and clearly $\forall_n P((g)_n,n)$. □

3.3.3. Implicit definitions. Let us say that a $(1,0)$-ary relation Q *implicitely defines* a function f_0 if

$$\forall_f(Q(f) \leftrightarrow f = f_0).$$

We define a relation J_a for every $a \in \mathcal{O}$ by

$$J_a(n,b) :\leftrightarrow b \leq_{\mathcal{O}} a \wedge n \in H_b.$$

LEMMA. *There is a (1,1)-ary arithmetical relation Q such that for every $a \in \mathcal{O}$ the $(1,0)$-ary relation $Q_{(a)}$ implicitely defines $\langle c_{J_a}\rangle$; here $Q_{(a)}(f) :\leftrightarrow Q(f,a)$.*

PROOF. Fix $a \in \mathcal{O}$.

$$\begin{aligned}
&c_{J_a}(n,b) = 1\\
&\leftrightarrow J_a(n,b)\\
&\leftrightarrow b \leq_{\mathcal{O}} a \wedge n \in H_b\\
&\leftrightarrow b \leq'_{\mathcal{O}} a \wedge\\
&\quad [(b = 1 \wedge n \neq n) \vee\\
&\quad (b = 2^{(b)_0} \wedge n \in (H_{b^-})') \vee\\
&\quad (b = 3\cdot 5^{(b)_2} \wedge (n)_0 <'_{\mathcal{O}} b \wedge (n)_1 \in H_{(n)_0})]\\
&\leftrightarrow b \leq'_{\mathcal{O}} a \wedge\\
&\quad [(b = 2^{(b)_0} \wedge \exists_s R(\overline{c_{H_b^-}}(s), n)) \vee\\
&\quad (b = 3\cdot 5^{(b)_2} \wedge (n)_0 <'_{\mathcal{O}} b \wedge c_{J_a}((n)_1,(n)_0) = 1)] \text{ for some recursive } R\\
&\leftrightarrow b \leq'_{\mathcal{O}} a \wedge\\
&\quad [(b = 2^{(b)_0} \wedge \exists_s\exists_x[R(x,n) \wedge \wedge \mathrm{lh}(x) = s \wedge \forall_{i<s}((x)_i = c_{J_a}(i,b^-))]) \vee\\
&\quad (b = 3\cdot 5^{(b)_2} \wedge (n)_0 <'_{\mathcal{O}} b \wedge c_{J_a}((n)_1,(n)_0) = 1)].
\end{aligned}$$

Therefore we define

$$\begin{aligned}
Q(f,a) :\leftrightarrow\ &\forall_n(f(n) = f\langle(n)_0,(n)_1\rangle \leq 1) \wedge\\
&\forall_n\forall_b\,(f\langle n,b\rangle = 1 \leftrightarrow\\
&\quad b \leq'_{\mathcal{O}} a \wedge\\
&\quad [(b = 2^{(b)_0} \wedge \exists_s\exists_x[R(x,n) \wedge \mathrm{lh}(x) = s \wedge \forall_{i<s}((x)_i = f\langle i,b^-\rangle)]) \vee\\
&\quad (b = 3\cdot 5^{(b)_2} \wedge (n)_0 <'_{\mathcal{O}} b \wedge f\langle(n)_1,(n)_0\rangle = 1)]).
\end{aligned}$$

Q is arithmetical and for $a \in \mathcal{O}$ we have $Q(\langle c_{J_a}\rangle, a)$. Conversely, fix $a \in \mathcal{O}$ and assume $Q(f,a)$. We must show that $f = \langle c_{J_a}\rangle$. Because of $f(n) = f\langle(n)_0,(n)_1\rangle \leq 1$ it suffices to show that

$$f\langle n,b\rangle = 1 \leftrightarrow J_a(n,b).$$

This is done by $<_{\mathcal{O}}$-induction on $b \leq_{\mathcal{O}} a$, since for $b \not\leq_{\mathcal{O}} a$ both sides of the equivalence are false. □

3.3.4. Kreisel's theorem.

THEOREM (Kreisel). *$\mathcal{H}$ is the least non-empty set $\mathcal{C}$ of unary total functions such that*

(a) *If $f \in \mathcal{C}$ and g is recursive in f, then $g \in \mathcal{C}$.*
(b) *If P is a Π^1_1-definable relation, then*

$$\forall_n \exists_{f\in\mathcal{C}} P(f,n) \to \exists_{g\in\mathcal{C}} \forall_n P((g)_n, n).$$

PROOF. (a) is true for $\mathcal{H}$ by the lemma in 3.2.1, and (b) holds for $\mathcal{H}$ by 3.3.2. For the converse assume that $\mathcal{C}$ satisfies (a) and (b). We must show $\mathcal{H} \subseteq \mathcal{C}$. Since because of $n \in H_a \leftrightarrow J_a(n,a)$ the set H_a is recursive in J_a. Hence by (a) it suffices to show that $\langle c_{J_a}\rangle \in \mathcal{C}$ for every $a \in \mathcal{O}$. We prove this by $<_{\mathcal{O}}$-induction on a.

Case 1: $a = 1$. This follows immediately from (a), since $\mathcal{C} \neq \emptyset$ and $\langle c_{J_1}\rangle$ is recursive.

Case 2: a denotes a successor number. Because of

$$\begin{aligned} J_a(n,b) &\leftrightarrow b \leq_{\mathcal{O}} a \wedge n \in H_b \\ &\leftrightarrow (b = a \wedge n \in (H_{a^-})') \vee (b \leq'_{\mathcal{O}} a^- \wedge n \in H_b) \end{aligned}$$

the relation J_a is Σ^0_1-definable in J_{a^-}, i.e., for some recursive relation R we have

$$\langle c_{J_a}\rangle(n) = 1 \leftrightarrow \exists_z R(\langle c_{J_{a^-}}\rangle, n, z).$$

Therefore with the arithmetical Q from 3.3.3 we have by IH:

$$\forall_n \exists_m \exists_{f\in\mathcal{C}} [Q_{(a^-)}(f) \wedge ([m = 1 \wedge \exists_z R(f,n,z)] \vee [m = 0 \wedge \neg\exists_z R(f,n,z)])].$$

The relation P described in the kernel $[\ldots]$ is arithmetical and we have $P(f,n,m) \to \langle c_{J_a}\rangle(n) = m$. Because of (a) we further obtain

$$\forall_n \exists_{g\in\mathcal{C}} P(\lambda x.g(x+1), n, g(0))$$

and hence using (b)

$$\exists_{h\in\mathcal{C}} \forall_n P(\lambda x.(h)_n(x+1), n, (h)_n(0)).$$

Hence for all n we have $\langle c_{J_a}\rangle(n) = (h)_n(0) =: f(n)$ with $f \in \mathcal{C}$.

Case 3: a denotes a limit.

$$\begin{aligned} J_a(n,b) &\leftrightarrow b \leq_{\mathcal{O}} a \wedge n \in H_b \\ &\leftrightarrow (b = a \wedge (n)_0 <'_{\mathcal{O}} a \wedge (n)_1 \in H_{(n)_0}) \vee (b <'_{\mathcal{O}} a \wedge n \in H_b) \\ &\leftrightarrow \exists_{d<'_{\mathcal{O}}a}[(b = a \wedge d = (n)_0 <'_{\mathcal{O}} a \wedge (n)_1 \in H_d) \vee \\ &\qquad (d = b <'_{\mathcal{O}} a \wedge n \in H_d)]. \end{aligned}$$

Therefore for some Σ^0_1-definable relation R we have

$$\langle c_{J_a}\rangle(n) = 1 \leftrightarrow \exists_{d<'_{\mathcal{O}}a} R(\langle c_{J_d}\rangle, n, d).$$

By IH $\forall_{d<'_{\mathcal{O}}a}\exists_{f\in\mathcal{C}}(f = \langle c_{J_d}\rangle)$, hence $\forall_d \exists_{f\in\mathcal{C}}(d <'_{\mathcal{O}} a \to Q(f,d))$. This implies by (b)

$$\exists_{g\in\mathcal{C}} \forall_{d<'_{\mathcal{O}}a} Q((g)_d, d).$$

Let

$$\begin{aligned} P(g,n,m) :\leftrightarrow\ & \forall_{d<'_{\mathcal{O}}a} Q((g)_d, d) \wedge \\ & [(m = 1 \wedge \exists_{d<'_{\mathcal{O}}a} R((g)_d, n, d)) \vee \end{aligned}$$

$$(m = 0 \wedge \neg\exists_{d<'_{\mathcal{O}}a}\, R((g)_d, n, d))].$$

P is arithmetical and we have $\forall_n\exists_y\exists_{g\in\mathcal{C}}P(g,n,y)$ and also $P(g,n,m) \to \langle c_{J_a}\rangle(n) = m$. This implies the claim just as in the second case. □

From the proof it is clear that in (b) we only need to require the validity of the axiom of choice for arithmetical relations.

3.4. The Hyperarithmetical Quantifier Theorem

3.4.1. $\Sigma^1_{1,\mathcal{H}}$- and $\Delta^1_{1,\mathcal{H}}$-definable sets. A set A is called $\Sigma^1_{1,\mathcal{H}}$-*definable* if there is a recursive relation R such that

$$n \in A \;\leftrightarrow\; \exists_{f\in\mathcal{H}}\forall_s\, R(f,n,s).$$

$\Pi^1_{1,\mathcal{H}}$- and $\Delta^1_{1,\mathcal{H}}$-definable sets are defined similarly.

Our goal is to show that a set is Π^1_1-definable if and only if it is $\Sigma^1_{1,\mathcal{H}}$-definable. To do this we need an auxiliary lemma.

LEMMA. (a) *If $a \in \mathcal{O}$, then $\mathcal{O}_{|a|}$ is Δ^1_1-definable*
(b) *If $a \in \mathcal{O}$, then $\mathcal{O}_{|a|}$ is $\Delta^1_{1,\mathcal{H}}$-definable.*
(c) *If $a \in \mathcal{O}^A$, then $\mathcal{O}^B_{|a|^A}$ is $\Delta^{1,A,B}_1$-definable.*
(d) *If $a \in \mathcal{O}^A$, then $\mathcal{O}^B_{|a|^A}$ is $\Delta^{1,A,B}_{1,\mathcal{H}^{A,B}}$-definable.*

PROOF. (a). This immediately follows from the proposition in 3.1.2 together with the theorem of Souslin/Kleene.

(b). We apply the Recursion Lemma to $\mathcal{O}$ and $<_{\mathcal{O}}$, with respect to the relation

$$\begin{aligned} R(b,e) \;:\leftrightarrow\; & b \in \mathcal{O} \text{ and } (e)_0 \text{ is } \Pi^1_{1,\mathcal{H}}\text{-index of } \mathcal{O}_{|a|} \\ & \text{and } (e)_1 \text{ is } \Pi^1_{1,\mathcal{H}}\text{-index of } \neg\mathcal{O}_{|a|}. \end{aligned}$$

So we need to construct a recursive function χ such that for arbitrary t and $b \in \mathcal{O}$ from the assumption

$$\forall_{c<_{\mathcal{O}}b}\, R(c, \varphi_t(c)) \tag{3.4}$$

we can infer $R(b, \chi(t,b))$.

Case 1: $b = 1$ or $b = 1^*$. Then we have $\mathcal{O}_{|1|} = \emptyset$ and $\mathcal{O}_{|1^*|} = \{1\}$, and hence the claim is trivial.

Case 2: b and b^- both denote successor numbers. As in the proof of the proposition in 3.1.2 we have

$$\begin{aligned} a \in \mathcal{O}_{|b|} &\leftrightarrow a \in \mathcal{O}_{|b^-|} \vee (a = 2^{(a)_0} \wedge a^- \in \mathcal{O}_{|b^-|}) \\ &\leftrightarrow \forall_{f\in\mathcal{H}}\exists_s\, T_2(f, (\varphi_t(b^-))_0, a, s) \vee \\ &\quad (a = 2^{(a)_0} \wedge \forall_{f\in\mathcal{H}}\exists_s\, T_2(f, (\varphi_t(b^-))_0, a^-, s) \\ &\leftrightarrow \forall_{f\in\mathcal{H}}\exists_s\, T_2(f, e_0, t, b, a, s) \quad \text{for some } e_0 \\ &\leftrightarrow \forall_{f\in\mathcal{H}}\exists_s\, T_2(f, S(e_0,t,b), a, s). \end{aligned}$$

Let $e'_0 := S(e_0,t,b)$. Because of $a \notin \mathcal{O}_{|b|} \;\leftrightarrow\; a \notin \mathcal{O}_{|b^-|} \wedge (a \neq 2^{(a)_0} \vee a^- \notin \mathcal{O}_{|b^-|})$ for the complement $\mathbb{N} \setminus \mathcal{O}_{|b|}$ of $\mathcal{O}_{|b|}$ we obtain a similar representation as for $\mathcal{O}_{|b|}$ above, and hence a Π^1_1-index e'_1. So in this case we define $\chi(t,a) := \langle e'_0, e'_1\rangle$.

Case 3: b denotes a successor number and b^- denotes a limit. Here we obtain as in the proposition in 3.1.2

$$a \in \mathcal{O}_{|b|} \;\leftrightarrow\; a \in \mathcal{O}_{|b^-|} \vee \exists_{e \leq a}\,[\langle e, a\rangle \in A \wedge e \in B],$$

where A is Π^0_2-definable and B is Π^0_1-definable in $\mathcal{O}_{|b^-|}$. Using Kreisel's Theorem and the IH we obtain that A, B are $\Pi^1_{1,\mathcal{H}}$-definable, hence also $\mathcal{O}_{|b|}$ (effectively in b). The representation of the complement of $\mathcal{O}_{|b|}$ can again be obtained by negating the right hand side of the above equivalence.

Case 4: b denotes a limit. Then we have

$$\begin{aligned}
a \in \mathcal{O}_{|b|} &\leftrightarrow \exists_n\,(a \in \mathcal{O}_{|\varphi_{(b)_2}(n)|}) \\
&\leftrightarrow \exists_n \forall_{f\in\mathcal{H}} \exists_s\, T_2(f, (\varphi_t(\varphi_{(b)_2}(n)))_0, a, n, s) \\
&\leftrightarrow \forall_{f\in\mathcal{H}} \exists_n \exists_s\, T_2((f)_n, (\varphi_t(\varphi_{(b)_2}(n)))_0, a, n, s) \text{ by Kreisel's Theorem} \\
&\leftrightarrow \forall_{f\in\mathcal{H}} \exists_s\, T_2(f, e_1, t, b, a, s) \quad \text{for some } e_1 \\
&\leftrightarrow \forall_{f\in\mathcal{H}} \exists_s\, T_2(f, S(e_1, t, b), a, s).
\end{aligned}$$

For the complement of $\mathcal{O}_{|b|}$ we proceed dually.

(c). Read the proof of (b) without restricting the function quantifiers, and relativize.

(d). Again by reading of the proof of (b), this time with the function quantifiers relativized to $\mathcal{H}^{A,B}$. □

PROPOSITION. *The set $\mathcal{H}$ of all hyperarithmetical functions is Π^1_1-, but not Σ^1_1-definable.*

PROOF. We first show that $\mathcal{H}$ is Π^1_1-definable. This follows from

$$\begin{aligned}
f \in \mathcal{H} &\leftrightarrow \exists_e \forall_n (f(n) = \varphi^1_e(n)) \\
&\leftrightarrow \exists_e \forall_n \forall_f \exists_s\, T_2(g, k(e), n, f(n), s),
\end{aligned}$$

using Π^1_1-Uniformization. Now assume that $\mathcal{H}$ is Σ^1_1-definable. Let

$$P(f, a) \;:\leftrightarrow\; (f \in \mathcal{H} \wedge a \in \mathcal{O} \wedge f \text{ recursive in } \mathcal{O}_{|a|}) \vee (f \notin \mathcal{H} \wedge a = 1).$$

Since P is Π^1_1-definable, Π^1_1-Uniformization (which easily generalizes to relations with function arguments) yields a Π^1_1-definable relation $Q \subseteq P$ such that Q is the graph of a total function and its image $\Im(Q)$ is an unbounded subset of $\mathcal{O}$. Moreover $\Im(Q)$ is Σ^1_1-definable, since

$$\begin{aligned}
m \in \Im(Q) &\leftrightarrow \exists_f Q(f, m) \\
&\leftrightarrow \exists_f \forall_n (n \neq m \to \neg Q(f, n)).
\end{aligned}$$

But this contradicts the Boundedness Lemma; hence $\mathcal{H}$ is not Σ^1_1-definable. □

3.4.2. The hyperarithmetical quantifier theorem.

THEOREM (Hyperarithmetical Quantifier Theorem). *A set A is Π^1_1-definable if and only if it is $\Sigma^1_{1,\mathcal{H}}$-definable.*

PROOF. Assume first that the set A is $\Sigma^1_{1,\mathcal{H}}$-definable. Then

$$\begin{aligned}
&n \in A \\
&\leftrightarrow \exists_{f\in\mathcal{H}} \forall_s R(\overline{f}(s), n) \quad \text{for some recursive } R
\end{aligned}$$

$$\leftrightarrow \exists_e\,[\varphi^1_e \text{ is total} \wedge \forall_g\,(\forall_{m,k}\,(\varphi^1_e(m) = k \to g(m) = k) \to \forall_s R(\overline{g}(s), s))].$$

Hence because of $\varphi^1_e(m) = k \;\leftrightarrow\; \forall_f \exists_s T_2(f, k(e), u, v, s)$ the set A is Π^1_1-definable (cf. the first few lines of the proof of the proposition in 3.3.2).

For the converse it suffices to show that $\mathcal{O}$ is $\Sigma^1_{1,\mathcal{H}}$-definable, since $\mathcal{O}$ is Π^1_1-complete. Since $\mathcal{H}$ is Π^1_1-definable, for some e we have

$$c_A \in \mathcal{H} \;\leftrightarrow\; \forall_f \exists_s T_2(c_A, e, \overline{f}(s)).$$

Now by relativizing the proof of the Π^1_1-completeness of $\mathcal{O}$ in 2.6.4 we have a function h recursive in A such that

$$\{\, z \mid y * z \in \mathrm{Tree}(e, c_A) \,\} \text{ is well-founded } \leftrightarrow\; h(y) \in \mathcal{O}^A.$$

So with $e_0 := h(\langle\rangle)$ we have

$$c_A \in \mathcal{H} \;\leftrightarrow\; e_0 \in \mathcal{O}^A.$$

But there is no $a \in \mathcal{O}$ such that

$$c_A \in \mathcal{H} \;\leftrightarrow\; e_0 \in \mathcal{O}^A_{|a|}.$$

To see this assume that there would be such an $a \in \mathcal{O}$. Then $\mathcal{O}^A_{|a|}$ would be $\Delta^{1,A}_1$-definable by part (c) of the lemma in 3.4.1, hence

$$n \in \mathcal{O}^A_{|a|} \;\leftrightarrow\; \forall_f \exists_s T_2(f, c_A, e_1, n, s) \;\leftrightarrow\; \exists_f \forall_s \neg T_2(f, c_A, e_2, n.s).$$

Therefore

$$c_A \in \mathcal{H} \;\leftrightarrow\; \forall_f \exists_s T_2(f, c_A, e_1, e_0, s) \;\leftrightarrow\; \exists_f \forall_s \neg T_2(f, c_A, e_2, e_0, s).$$

Hence $\mathcal{H}$ would be Δ^1_1-definable, which contradicts the proposition in 3.4.1.

Hence $\{\, |e_0|^A \mid c_A \in \mathcal{H} \,\}$ is unbounded in $\mathcal{O}$. We now obtain

$n \in \mathcal{O}$

$\leftrightarrow \exists_{c_A \in \mathcal{H}}\; n \in \mathcal{O}_{|e_0|^A}$

$\leftrightarrow \exists_{c_A \in \mathcal{H}} \exists_{f \in \mathcal{H}^A} \forall_s\, R(c_A, \overline{f}(s), n)$ $\quad R$ recursive by (d) of the lemma in 3.4.1

$\leftrightarrow \exists_{g \in \mathcal{H}} \forall_s\, R'(\overline{g}(s), n)$ $\quad R'$ recursive, by closure of $\mathcal{H}$ under Δ^1_1-definability.

This yields the required representation of $\mathcal{O}$. □

3.5. Paths in Kleene's $\mathcal{O}$

A subset $P \subseteq \mathcal{O}$ is called *path in* $\mathcal{O}$ if P is linearly ordered by $<_{\mathcal{O}}$ and with y also all $x <_{\mathcal{O}} y$ are in P. P is called *path through* $\mathcal{O}$ if P is path in $\mathcal{O}$ and every constructive ordinal is denoted by an element of P.

From the Boundedness Lemma we immediately obtain

LEMMA. *There is no Σ^1_1-path through $\mathcal{O}$.* □

THEOREM (Feferman, Spector). *There is a Π^1_1-path through $\mathcal{O}$.*

PROOF. Let $C(a) := \{\, x \mid x <'_{\mathcal{O}} a \,\}$. We claim that $\mathcal{O}$ is the set of all $a \in \mathbb{N}$ such that

(a) $C(a)$ is well-ordered by $<'_{\mathcal{O}}$, and
(b) every $b \in C(a)$ either has the form $b = 1$, or $b = 2^y$ for some $y \neq 0$, or $b = 3 \cdot 5^e$ with φ_e total and $\forall_n(\varphi_e(n) <'_{\mathcal{O}} \varphi_e(n+1))$.

The inclusion from left to right follows from 2.6.2. For the converse inclusion we distinguish cases according to the form of $b \in C(a)$, and use transfinite induction on the well-ordering $<'_{\mathcal{O}}$ on $C(a)$.

Now define $\mathcal{O}^*$ to be the set of all $a \in \mathbb{N}$ such that

(a) $C(a)$ is well-ordered by $<'_{\mathcal{O}}$ w.r.t. descending sequences from $\mathcal{H}$, and
(b) as above.

We claim that $\mathcal{O}^*$ is Σ^1_1-definable. Condition (b) clearly is arithmetical, and (a) says that $<'_{\mathcal{O}}$ is a linear ordering on $C(a)$, and $\forall_{\alpha\in\mathcal{H}}\exists_s P(\alpha, a, s)$ with P recursive, hence by the hyperarithmetic quantifier theorem also in the form $\exists_f \forall_s Q(f, a, s)$ with Q recursive.

Since clearly $\mathcal{O} \subseteq \mathcal{O}^*$, we must have $\mathcal{O} \subsetneq \mathcal{O}^*$. Pick $a^* \in \mathcal{O}^* \setminus \mathcal{O}$, and let

$$Z := C(a^*) \cap \mathcal{O}.$$

Clearly Z is a Π^1_1-definable path in $\mathcal{O}$. We claim that Z is a path *through* $\mathcal{O}$.

So assume $Z \subseteq \mathcal{O}_{|b|}$ for some $b \in \mathcal{O}$; we need to derive a contradiction. Then $Z = C(a^*) \cap \mathcal{O}_{|b|}$, hence $Z \in \Delta^1_1$ by the proposition in 3.1.2. Therefore also $M := C(a^*) \setminus \mathcal{O} = C(a^*) \setminus Z \in \Delta^1_1$. Because of $M \neq \emptyset$, $M \subseteq C(a^*)$ and $M \in \Delta^1_1$ there is a $<'_{\mathcal{O}}$-minimal element $a_0 \in M$.

Case $a_0 = 1$. Then $a_0 \in \mathcal{O}$, contradicting the definition of M.

Case $a_0 = 2^b$ for some $b \neq 0$. Then $b <'_{\mathcal{O}} a_0$, so $b \in \mathcal{O}$ and therefore $a_0 \in \mathcal{O}$, a contradiction.

Case $a_0 = 3 \cdot 5^e$ with φ_e total and $\forall_n(\varphi_e(n) <'_{\mathcal{O}} \varphi_e(n+1))$. Then we would have $\varphi_e(n) <'_{\mathcal{O}} a_0$, hence $\varphi_e(n) \in \mathcal{O}$ for every n and therefore $a_0 \in \mathcal{O}$, which again is a contradiction. □

Note that the proof gives us an interesting corollary.

Corollary. *There is a recursively enumerable linear ordering which is a well-ordering w.r.t. hyperarithmetic decreasing sequences but not w.r.t. arbitrary decreasing sequences.*

Proof. Take $<'_{\mathcal{O}}$ on $C(a^*) = \{\, x \mid x <'_{\mathcal{O}} a^* \,\}$, with $a^* \in \mathcal{O}^* \setminus \mathcal{O}$ as constructed in the proof above. □

CHAPTER 4

Computation with Partial Continuous Functionals

The logic considered up to now is very general, and for instance does not allow to speak of natural numbers. We therefore introduce inductive types (or free algebras) as base domains; they are given by constructors. We also allow function spaces, with the inductive types as base types. We specialize our minimal logic to a simply typed one, where the variables are typed, and the formation of terms is adapted. We add induction axioms, to express the minimality of the inductive types. Every inductive type comes with a recursion operator, which has certain conversion or definitional equality rules associated with it. We prove that every term (possibly with free variables) can be converted into normal (or canonical) form.

We describe a constructive theory of computable functionals, based on the partial continuous functionals as their intendend domain. Such a task had been started by Scott [1969] and continued by Scott and Strachey [1971] and continued (and implemented) by Milner [1972, 1973], who also coined the name LCF. However, the prime example of such a theory, the type theory of Martin-Löf [1984], in its present form deals with total (structural recursive) functionals only. An early attempt of Martin-Löf [1983] to give a domain theoretic interpretation of his type theory has not even been published, probably because it was felt that a more general approach – such as formal topology, see Coquand et al. [2003] – would be more appropriate.

Here we try to make a fresh start, and do full justice to the fundamental notion of computability in finite types, with the partial continuous functionals as underlying domains. The total ones then appear as a dense subset [Kreisel, 1959, Ershov, 1972, Berger, 1993, Stoltenberg-Hansen et al., 1994, Schwichtenberg, 1996, Kristiansen and Normann, 1997], and seem to be best treated in this way.

Computable functionals and logic. Types are built from base types by the formation of function types, $\rho \Rightarrow \sigma$. As domains for the base types we choose non-flat (cf. Fig. 2 on page 81) and possibly infinitary free algebras, given by their constructors. The main reason for taking non-flat base domains is that we want the constructors to be injective and with disjoint ranges.

The naive model of such a finitely typed theory is the full set theoretic hierarchy of functionals of finite types. However, this immediately leads to higher cardinalities, and does not lend itself well for a theory of computability. A more appropriate semantics for typed languages has its roots in [Kreisel, 1959] (which used formal neighborhoods) and [Kleene, 1959]. This line of research was taken up and developed in a mathematically more satisfactory way by Scott [1970] and Ershov [1974]. Today this theory is

usually presented in the context of abstract domain theory (see Stoltenberg-Hansen et al. [1994], Abramsky and Jung [1994]); it is based on classical logic.

The present work can be seen as an attempt to develop a constructive theory of formal neighborhoods for continuous functionals, in a direct and intuitive style. The task is to replace abstract domain theory by a more concrete and (in case of finitary free algebras) finitary theory of representations. As a framework we use Scott's information systems (see Scott [1982], Larsen and Winskel [1991], Stoltenberg-Hansen et al. [1994]). It turns out that we only need to deal with "atomic" and "coherent" information systems (abbreviated acis), which simplifies matters considerably. In this setup the basic notion is that of a "token", or unit of information. The elements of the domain appear as abstract or "ideal" entitites: possibly infinite sets of tokens, which are "consistent" and "deductively closed".

Total functionals. One reason to be interested in total functionals is that for base types, that is free algebras, we can prove properties of total objects by structural induction. This is also true for the more general class of *structure-total* objects, where the arguments at parameter positions in constructor terms need not be total. An example is a list whose length is determined, but whose elements need not be total.

We show that the standard way to single out the total functionals from the partial ones works with non-flat base domains as well, and that Berger's proof [1993] of Kreisel's [1959] density theorem can be adapted.

Terms and their denotational and operational semantics. Since we have introduced domains via concrete representations, it is easy to define the computable functionals, simply as recursively enumerable ideals (= sets of tokens). However, this way to deal with computability is too general for concrete applications. In practice, one wants to define computable functionals by recursion equations. We show that and how computation rules (see Berger et al. [2003], Berger [2005]) can be used to achieve this task. The meaning $[\![\lambda\vec{x}\,M]\!]$ of a term M (with free variables in $\vec{x}$) involving constants D defined by computation rules will be an inductively defined set of tokens $(\vec{U}, b)$, of the type of $\lambda\vec{x}\,M$.

So we extend the term language of Plotkin's PCF [1977], by constants defined via "computation rules". One instance of such rules is the definition of the fixed point operators $\mathcal{Y}_\rho$ of type $(\rho \Rightarrow \rho) \Rightarrow \rho$, by $\mathcal{Y}_\rho f = f(\mathcal{Y}_\rho f)$. Another instance is the structural recursion operator $\mathcal{R}^\tau_{\mathrm{nat}}$, defined by

$$\mathcal{R}^\tau_{\mathrm{nat}}(f, g, 0) = f, \qquad \mathcal{R}^\tau_{\mathrm{nat}}(f, g, Sn) = g(n, \mathcal{R}^\tau_{\mathrm{nat}}(f, g, n)).$$

Operationally, the term language provides some natural conversion rules to "simplify" terms: β, η, and – for every defined constant D – the defining equations $D\vec{P} \mapsto M$ with non-overlapping constructor patterns $\vec{P}$; the equivalence generated by these conversions is called operational semantics. We show that the (denotational) values are preserved under conversions, including computation rules.

Computational adequacy. Clearly we want to know that the conversions mentioned above give rise to a "computationally adequate" operational semantics: If $[\![M]\!] = k$, then the conversion rules suffice to actually reduce M

to the numeral k. We show that this holds true in our somewhat extended setting as well, with computation rules and non-flat base domains.

Structural recursion. An important example of computation rules are those of the (Gödel) structural recursion operators. We prove their totality, by showing that the rules are strongly normalizing. A predicative proof of this fact has been given by Abel and Altenkirch [2000], based on Aczel's notion of a set-based relation. Our proof is predicative as well, but – being based on an extension of Tait's method of strong computability predicates – more along the standard line of such proofs. Moreover, it extends the result to the present setting.

Related work. The development of constructive theories of computable functionals of finite type began with Gödel's [1958]. There the emphasis was on particular computable functionals, the structural (or primitive) recursive ones. In contrast to what was done later by Kreisel, Kleene, Scott and Ershov, the domains for these functionals were not constructed explicitly, but rather considered as described axiomatically by the theory.

Denotational semantics for PCF-like languages is well-developed, and usually (as in Plotkin's [1977]) done in a domain-theoretic setting. The study of the semantics of non-overlapping higher type recursion equations - called here computation rules - has been initiated in Berger et al. [2003], again in a domain-theoretic setting. Recently Berger [2005] he has introduced a "strict" variant of this domain-theoretic semantics, and used it to prove strong normalization of extensions of Gödel's T by different versions of bar recursion. Information systems have been conceived by Scott [1982], as an intuitive approach to domains for denotational semantics. The idea to consider atomic information systems is due to Ulrich Berger (unpublished work); coherent information systems have been introduced by Plotkin [1978, p.210]. Taking up Kreisel's [1959] idea of neighborhood systems, Martin-Löf developed in unpublished (but somewhat distributed) notes [1983] a domain theoretic interpretation of his type theory. The intersection type discipline of Barendregt, Coppo, and Dezani-Ciancaglini [1983] can be seen as a different style of presenting the idea of a neighborhood system. The desire to have a more general framework for these ideas has lead Martin-Löf, Sambin and others to develop a formal topology; cf. Coquand, Sambin, Smith, and Valentini [2003].

It seems likely that the method in [Kristiansen and Normann, 1997, Section 3.5] (which is based on an idea of Ulrich Berger) can be used to prove density in the present case, but this would require some substantial rewriting.

The first proof of an adequacy theorem (not under this name) is due to Plotkin [1977, Theorem 3.1]; Plotkin's proof is by induction on the types, and uses a computability predicate. A similar result in a type-theoretic setting is in Martin-Löf's notes [1983, Second Theorem]. Adequacy theorems have been proved in many contexts, by Abramsky [1991], Amadio and Curien [1998], Barendregt et al. [1983], Martin-Löf [1983]. Coquand and Spiwack [2005] – building on the work of Martin-Löf [1983] and Berger [2005] – observed that the adequacy result even holds for untyped languages, hence also for dependently typed ones.

The problem of proving strong normalization for extensions of typed λ-calculi by higher order rewrite rules has been studied extensively in the literature: Tait [1971], Girard [1971], Troelstra [1973], Blanqui et al. [1999], Abel and Altenkirch [2000], Berger [2005]. Most of these proofs use impredicative methods (e.g., by reducing the problem to strong normalization of second order propositional logic, called system F by Girard [1971]). Our definition of the strong computability predicates and also the proof are related to Zucker's [1973] proof of strong normalization of his term system for recursion on the first three number or tree classes. However, Zucker uses a combinatory term system and defines strong computability for closed terms only. Following some ideas in an unpublished note of Berger, Benl (in his diploma thesis [1998]) transferred this proof to terms in simply typed λ-calculus, possibly involving free variables. Here it is adapted to the present context.

Organization of the chapter. In Sec.4.1 atomic coherent information systems are defined, and used as a concrete representation of the relevant domains, based on non-flat and possibly infinitary free algebras. The structural recursion operators are defined in Sec.4.2. Sec.4.3 deals with total and structure-total ideals; it is shown that the density theorem holds.

4.1. Partial Continuous Functionals

Information systems have been introduced by Scott [1982], as an intuitive approach to deal constructively with ideal, infinite objects in function spaces, by means of their finite approximations. One works with atomic units of information, called *tokens*, and a notion of *consistency* for finite sets of tokens. Finally there is an *entailment* relation, between consistent finite sets of tokens and single tokens. The *ideals* (or *objects*) of an information system are defined to be the consistent and deductively closed sets of tokens; we write $|\boldsymbol{A}|$ for the set of ideals of $\boldsymbol{A}$. One shows easily that $|\boldsymbol{A}|$ is a domain w.r.t. the inclusion relation. Conversely, every domain with countable basis can be represented as the set of all ideals of an appropriate information system [Larsen and Winskel, 1991].

Here we take Scott's notion of an information system as a basis to introduce the partial continuous functionals. Call an information system *atomic* if the entailment relation $U \vdash b$ is given by $\exists_{a\in U}\{a\} \vdash b$ and hence determined by a transitive relation on A (namely $\{a\} \vdash b$, written $a \geq b$). Call it *coherent* [Plotkin, 1978, p.210] when a finite set U of tokens is consistent if and only if every two-element subset of it is. We will show below that if $\boldsymbol{B}$ is atomic (coherent), then so is the "function space" $\boldsymbol{A} \to \boldsymbol{B}$. Since our algebras will be given by atomic coherent information systems, this is the only kind of information systems we will have to deal with.

4.1.1. 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 $\rho \Rightarrow \sigma$ and the formation of inductively generated types $\mu\vec{\alpha}\,\vec{\kappa}$, where $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ is a list of distinct "type variables", and $\vec{\kappa} = (\kappa_i)_{i=1,\dots,k}$ is a list of "constructor types", whose argument types contain $\alpha_1, \dots, \alpha_N$ in strictly positive positions only.

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

DEFINITION. Let $\vec{\alpha} = (\alpha_j)_{j=1,\dots,N}$ 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 by

$$\frac{\vec{\rho}, \vec{\sigma}_1, \dots, \vec{\sigma}_n \in \mathrm{Ty}}{\vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \dots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \in \mathrm{KT}(\vec{\alpha})} \quad (n \geq 0)$$

$$\frac{\kappa_1, \dots, \kappa_n \in \mathrm{KT}(\vec{\alpha})}{(\mu\vec{\alpha}\,(\kappa_1, \dots, \kappa_n))_j \in \mathrm{Ty}} \quad (n \geq 1) \qquad \frac{\rho, \sigma \in \mathrm{Ty}}{\rho \Rightarrow \sigma \in \mathrm{Ty}}$$

Here $\vec{\rho} \Rightarrow \sigma$ means $\rho_1 \Rightarrow \dots \Rightarrow \rho_m \Rightarrow \sigma$, associated to the right. We reserve μ for types of the form $(\mu\vec{\alpha}\,(\kappa_1, \dots, \kappa_k))_j$. The *parameter types* of μ are the members of all $\vec{\rho}$ appearing in its constructor types $\kappa_1, \dots, \kappa_k$.

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.

$$\begin{aligned}
&\mathrm{unit} && := \mu\alpha\,\alpha,\\
&\mathrm{boole} && := \mu\alpha\,(\alpha, \alpha),\\
&\mathrm{nat} && := \mu\alpha\,(\alpha, \alpha \Rightarrow \alpha),\\
&\mathrm{list}(\rho) && := \mu\alpha\,(\alpha, \rho \Rightarrow \alpha \Rightarrow \alpha),\\
&\rho \otimes \sigma && := \mu\alpha\,(\rho \Rightarrow \sigma \Rightarrow \alpha),\\
&\rho + \sigma && := \mu\alpha\,(\rho \Rightarrow \alpha, \sigma \Rightarrow \alpha),\\
&(\mathrm{tree}, \mathrm{tlist}) && := \mu(\alpha, \beta)\,(\mathrm{nat} \Rightarrow \alpha, \beta \Rightarrow \alpha, \beta, \alpha \Rightarrow \beta \Rightarrow \beta),\\
&\mathrm{bin} && := \mu\alpha\,(\alpha, \alpha \Rightarrow \alpha \Rightarrow \alpha),\\
&\mathcal{O} && := \mu\alpha\,(\alpha, \alpha \Rightarrow \alpha, (\mathrm{nat} \Rightarrow \alpha) \Rightarrow \alpha),\\
&\mathcal{T}_0 && := \mathrm{nat},\\
&\mathcal{T}_{n+1} && := \mu\alpha\,(\alpha, (\mathcal{T}_n \Rightarrow \alpha) \Rightarrow \alpha).
\end{aligned}$$

Notice that there are many equivalent ways to define these types. For instance, we could take unit + unit to be the type of booleans, and list(unit) to be the type of natural numbers.

A type is called *finitary* if it is a μ-type with all its parameter types $\vec{\rho}$ finitary, and all its constructor types are of the form

$$\vec{\rho} \Rightarrow \alpha_{j_1} \Rightarrow \ldots \Rightarrow \alpha_{j_n} \Rightarrow \alpha_j, \tag{4.1}$$

so the $\vec{\sigma}_1, \ldots, \vec{\sigma}_n$ in the general definition are all empty. In the examples above unit, boole, nat, tree, tlist and bin are all finitary, whereas $\mathcal{O}$ and $\mathcal{T}_{n+1}$ are not. list(ρ), $\rho \otimes \sigma$ 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.

A non-simultaneous type $\mu\alpha(\kappa_1, \ldots, \kappa_n)$ is said to have a *nullary constructor* if one of its constructor types is of the form $\vec{\rho} \Rightarrow \alpha$. More generally, a type $\mu\alpha_1, \ldots, \alpha_N(\kappa_1, \ldots, \kappa_n)$ is said to have *nullary constructors* if for every j $(1 \leq j \leq N)$ there is a constructor type κ_{i_j} of form (4.1) with $j_1, \ldots, j_n < j$. All types in our examples list have nullary constructors.

In what follows we restrict ourselves to finitary μ-types with nullary constructors.

4.1.2. Atomic coherent information systems.

Definition. An *atomic coherent information system* (abbreviated *acis*) is a triple $(A, \smile, \geq)$ with A a countable set (the *tokens*, denoted $a, b, \ldots$), $\smile$ a reflexive and symmetric relation on A (the *consistency relation*), and $\geq$ a reflexive and transitive relation on A (the *entailment relation*) which satisfy

$$a \smile b \to b \geq c \to a \smile c. \tag{4.2}$$

A subset $x \subseteq A$ is called *consistent* if $\forall_{a,b \in x}\, a \smile b$. Let Con be the set of all finite consistent subsets of A. The elements of Con are called *formal neighborhoods* and are denoted $U, V, \ldots$.

We write $U \geq a$ for $\exists_{b \in U}\, b \geq a$, and $U \geq V$ for $\forall_{a \in V}\, U \geq a$. – Every acis is an information system in the sense of Scott [1982]; this follows from

Lemma. *Let $\boldsymbol{A} = (A, \smile, \geq)$ be an acis. $U \geq V_1, V_2$ implies $V_1 \cup V_2 \in$ Con.*

Proof. Let $b_1 \in V_1$, $b_2 \in V_2$. Then we have $a_1, a_2 \in U$ such that $a_i \geq b_i$. From $a_1 \smile a_2$ we obtain $a_1 \smile b_2$ by (c), hence $b_1 \smile b_2$ again by (c). □

Definition. Let $\boldsymbol{A} = (A, \smile_A, \geq_A)$ and $\boldsymbol{B} = (B, \smile_B, \geq_B)$ be acis's. Define $\boldsymbol{A} \to \boldsymbol{B} = (C, \smile, \geq)$ by

$$\begin{aligned} C &:= \mathrm{Con}_A \times B, \\ (U, b) \smile (V, c) &:\leftrightarrow (U \cup V \in \mathrm{Con}_A \to b \smile c), \\ (U, b) \geq (V, c) &:\leftrightarrow V \geq_A U \wedge b \geq_B c. \end{aligned}$$

Lemma. *Let $\boldsymbol{A} = (A, \smile_A, \geq_A)$ and $\boldsymbol{B} = (B, \smile_B, \geq_B)$ be acis's. Then $\boldsymbol{A} \to \boldsymbol{B}$ is an acis again.*

Proof. Clearly $\geq$ is reflexive and transitive, and $\smile$ is reflexive and symmetric; it remains to check (4.2). So let $(U_1, b_1) \smile (U_2, b_2)$ and $(U_2, b_2) \geq (V, c)$, hence $V \geq U_2$ and $b_2 \geq c$. We must show $(U_1, b_1) \smile (V, c)$. So assume $U_1 \cup V \in$ Con; we must show $b_1 \smile c$. Now $U_1 \cup V \in$ Con and $V \geq U_2$ by the previous lemma imply $U_1 \cup U_2 \in$ Con. But then $b_1 \smile b_2$, hence $b_1 \smile c$ by (4.2). □

Scott [1982] introduced the notion of an *approximable map* from $\boldsymbol{A}$ to $\boldsymbol{B}$. Such a map is given by a relation r between Con_A and B, where $r(U,b)$ intuitively means that whenever we are given the information $U \in \mathrm{Con}_A$ on the argument, then we know that at least the token b appears in the value.

Definition (Approximable map). Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be acis's. A relation $r \subseteq \mathrm{Con}_A \times B$ is an *approximable map* from $\boldsymbol{A}$ to $\boldsymbol{B}$ (written $r\colon \boldsymbol{A} \to \boldsymbol{B}$) if and only if

(a) if $r(U,b_1)$ and $r(U,b_2)$, then $b_1 \smile_B b_2$, and
(b) if $r(U,b)$, $V \geq_A U$ and $b \geq_B c$, then $r(V,c)$.

Call a (possibly infinite) set x of tokens *consistent* if $U \in \mathrm{Con}$ for every finite subset $U \subseteq x$, and *deductively closed* if $\forall_{a\in x}\forall_{b\leq a} b \in x$. The *ideals* (or *objects*) of an information system are defined to be the consistent and deductively closed sets of tokens; we write $|\boldsymbol{A}|$ for the set of ideals of $\boldsymbol{A}$.

Theorem. *Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be acis's. The ideals of $\boldsymbol{A} \to \boldsymbol{B}$ are exactly the approximable maps from $\boldsymbol{A}$ to $\boldsymbol{B}$.*

Proof. We show that $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ satisfies the axioms for approximable maps. (a). Let $r(U,b_1)$ and $r(U,b_2)$. Then $b_1 \smile_B b_2$ by the consistency of r. (b). Let $r(U,b)$, $V \geq_A U$ and $b \geq_B c$. Then $(U,b) \geq (V,c)$ by definition, hence $r(V,c)$ by the deductive closure of r.

For the other direction suppose $r\colon \boldsymbol{A} \to \boldsymbol{B}$ is an approximable map. We must show that $r \in |\boldsymbol{A} \to \boldsymbol{B}|$. Consistency of r: Suppose $r(U_1,b_1)$, $r(U_2,b_2)$ and $U = U_1 \cup U_2 \in \mathrm{Con}_A$. We must show that $b_1 \smile_B b_2$. Now by definition of approximable maps, from $r(U_i,b_i)$ and $U \geq_A U_i$ we obtain $r(U,b_i)$, and hence $b_1 \smile_B b_2$. Deductive closure of r: Suppose $r(U,b)$ and $(U,b) \geq (V,c)$, i.e., $V \geq_A U \wedge b \geq_B c$. Then $r(V,c)$ by definition of approximable maps. □

The set $|\boldsymbol{A}|$ of ideals for $\boldsymbol{A}$ carries a natural topology (the Scott topology), which has the cones $\tilde{U} := \{\, z \mid z \supseteq U \,\}$ generated by the formal neighborhoods U as basis. The continuous maps $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ and the ideals $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ are in a bijective correspondence. With any $r \in |\boldsymbol{A} \to \boldsymbol{B}|$ we can associate a continuous $|r|\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$:

$$|r|(z) := \{\, b \in B \mid r(U,b) \text{ for some } U \subseteq z \,\},$$

and with any continuous $f\colon |\boldsymbol{A}| \to |\boldsymbol{B}|$ we can associate $\hat{f} \in |\boldsymbol{A} \to \boldsymbol{B}|$:

$$\hat{f}(U,b) \quad :\Longleftrightarrow \quad b \in f(\overline{U}).$$

These assignments are inverse to each other, i.e., $f = |\hat{f}|$ and $r = \widehat{|r|}$. – We will usually write $r(z)$ for $|r|(z)$, and similarly $f(U,b)$ for $\hat{f}(U,b)$. It will be clear from the context where the mods and hats should be inserted.

4.1.3. Algebras with approximations. Recall that we restrict ourselves to finitary μ-types with nullary constructors.

We now define the acis $\boldsymbol{C}_{\mu_j}$ of an algebra μ_j, given by constructors C_i.

- The *tokens* are all type correct constructor expressions with an outermost C_i, such that at every argument position we have either a token or else special symbol – written $*$ –, which carries no information. – By an *extended token* a^* we mean a token or $*$.

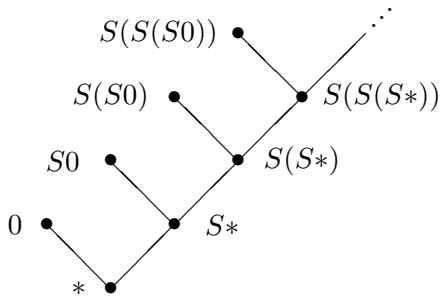

FIGURE 1. Tokens and entailment for nat

- Two tokens are in the *entailment* relation $\geq$ if they start with the same constructor, and for every argument position the arguments located there are extended tokens a^*, b^* such that $a^* \geq b^*$. – $a^* \geq b^*$ means that b^* is $*$, or both are tokens and the entailment relation holds
- Two tokens are *consistent* if both start with the same constructor and have consistent extended tokens at corresponding argument positions.

For example, the (extended) tokens for the algebra nat are as shown in Fig. 1 on page 80. A token a entails another one b if and only if there is a path from a (up) to b (down). In this case (and similarly for every finitary algebra) a finite set U of tokens is consistent if and only if it has an upper bound. Every constructor C generates

$$r_{\mathrm{C}} := \{\, (\vec{U}, C\vec{b^*}) \mid \vec{U} \geq \vec{b^*} \,\}.$$

The continuous map $|r_{\mathrm{C}}|$ is defined by

$$|r_{\mathrm{C}}|(\vec{z}) := \{\, b \mid (\vec{U}, b) \in r_{\mathrm{C}} \text{ for some } \vec{U} \subseteq \vec{z} \,\}.$$

Hence the (continuous maps corresponding to) constructors are injective and their ranges are disjoint, which is what we wanted to achieve.

The ideals x for μ are – as for any information system – the consistent and deductively closed sets of tokens. Clearly all tokens in x begin with the same constructor. For instance, $\{S(S0), S(S*), S*\}$, $\{S(S*), S*\}$, $\{0\}$ are ideals for nat, but also the infinite set $\{\, S^n* \mid n > 0 \,\}$. The ideals for nat and their inclusion relation are pictured in Fig. 2 on page 81. Here we have denoted the ideals $\emptyset$, $\{0\}$, $\{\, S^n* \mid n > 0 \,\}$ by $\bot$, 0, ∞, respectively, and any other ideal by applications of (the continuous map corresponding to) the constructor S to 0 or $\bot$. The ambiguous notation – S denotes a symbol in constructor expressions and also the continuous map $|r_S|$ – should not lead to confusion.

DEFINITION (Partial continuous functionals). Fix a set B of finitary μ-types with nullary constructors. For types ρ over base types from B, define the acis $\boldsymbol{C}_\rho$ inductively, by

$$\boldsymbol{C}_\mu \text{ as above, for } \mu \in B,$$
$$\boldsymbol{C}_{\rho\Rightarrow\sigma} := \boldsymbol{C}_\rho \to \boldsymbol{C}_\sigma.$$

The ideals $x \in |\boldsymbol{C}_\rho|$ are called *partial continuous functionals* of type ρ.

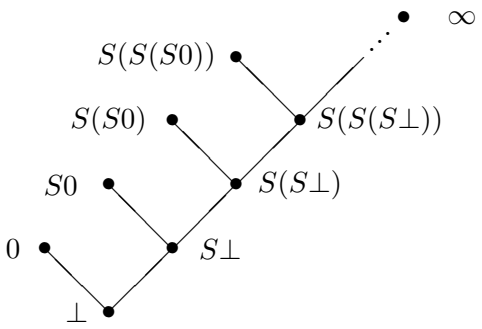

FIGURE 2. Ideals and inclusion for nat, i.e., its domain

REMARK (Coquand). Let C be the binary and 0 the unary constructor of the type bin of binary trees. Then one might think that the two tokens $C{*}0$ and $C0{*}$ should entail $C00$, which in our approach with atomic information system and a binary entailment relation this is cannot be the case. However, we are not interested in a complete representation of all possible formal neighborhoods but in the end only in the limits, that is, the ideals. In this particular case we have to consistently extend the neighborhood to $\{C{*}0, C0{*}, C00\}$, to achive that it entails $C00$.

4.2. Structural Recursion

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 construct elements of a type μ_j, and with the *recursion operators* $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ we can construct mappings from μ_j to τ_j by recursion on the structure of $\vec{\mu}$.

4.2.1. Recursion operators. 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

$$\kappa_i = \vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \alpha_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \alpha_{j_n}) \Rightarrow \alpha_j \in \mathrm{KT}(\vec{\alpha})$$

the *step type*

$$\begin{aligned}\delta_i^{\vec{\mu},\vec{\tau}} := \vec{\rho} \Rightarrow (\vec{\sigma}_1 \Rightarrow \mu_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \mu_{j_n}) \Rightarrow \\ (\vec{\sigma}_1 \Rightarrow \tau_{j_1}) \Rightarrow \ldots \Rightarrow (\vec{\sigma}_n \Rightarrow \tau_{j_n}) \Rightarrow \tau_j.\end{aligned}$$

Here $\vec{\rho}, (\vec{\sigma}_1 \Rightarrow \mu_{j_1}), \ldots, (\vec{\sigma}_n \Rightarrow \mu_{j_n})$ correspond to the *components* of the object of type μ_j under consideration, and $(\vec{\sigma}_1 \Rightarrow \tau_{j_1}), \ldots, (\vec{\sigma}_n \Rightarrow \tau_{j_n})$ to the previously defined values. The recursion operator $\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}}$ has type

$$\mathcal{R}_{\mu_j}^{\vec{\mu},\vec{\tau}} \colon \delta_1^{\vec{\mu},\vec{\tau}} \Rightarrow \ldots \Rightarrow \delta_k^{\vec{\mu},\vec{\tau}} \Rightarrow \mu_j \Rightarrow \tau_j$$

(recall that k is the total number of constructors for all types $\mu_1, \ldots, \mu_N$).

NOTE. The type $\mu_j \Rightarrow \delta_1^{\vec{\mu},\vec{\tau}} \Rightarrow \ldots \Rightarrow \delta_k^{\vec{\mu},\vec{\tau}} \Rightarrow \tau_j$ would probably be better, because it corresponds to the structure necessary for conversion.

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 $\mathcal{R}_{\mu}^{\tau}$.

4.2.2. Examples.

$$\mathtt{tt}^{\mathrm{boole}} := \mathrm{C}_1^{\mathrm{boole}}, \quad \mathtt{ff}^{\mathrm{boole}} := \mathrm{C}_2^{\mathrm{boole}},$$
$$\mathcal{R}^\tau_{\mathrm{boole}} \colon \tau \Rightarrow \tau \Rightarrow \mathrm{boole} \Rightarrow \tau,$$

$$0^{\mathrm{nat}} := \mathrm{C}_1^{\mathrm{nat}}, \quad \mathrm{S}^{\mathrm{nat}\Rightarrow\mathrm{nat}} := \mathrm{C}_2^{\mathrm{nat}},$$
$$\mathcal{R}^\tau_{\mathrm{nat}} \colon \tau \Rightarrow (\mathrm{nat} \Rightarrow \tau \Rightarrow \tau) \Rightarrow \mathrm{nat} \Rightarrow \tau,$$

$$\mathrm{nil}^{\mathrm{list}(\alpha)} := \mathrm{C}_1^{\mathrm{list}(\alpha)}, \quad \mathrm{cons}^{\alpha\Rightarrow\mathrm{list}(\alpha)\Rightarrow\mathrm{list}(\alpha)} := \mathrm{C}_2^{\mathrm{list}(\alpha)},$$
$$\mathcal{R}^\tau_{\mathrm{list}(\alpha)} \colon \tau \Rightarrow (\alpha \Rightarrow \mathrm{list}(\alpha) \Rightarrow \tau \Rightarrow \tau) \Rightarrow \mathrm{list}(\alpha) \Rightarrow \tau,$$

$$\big(\mathrm{inl}_{\rho\sigma}\big)^{\rho\Rightarrow\rho+\sigma} := \mathrm{C}_1^{\rho+\sigma},$$
$$\big(\mathrm{inr}_{\rho\sigma}\big)^{\sigma\Rightarrow\rho+\sigma} := \mathrm{C}_2^{\rho+\sigma},$$
$$\mathcal{R}^\tau_{\rho+\sigma} \colon (\rho \Rightarrow \tau) \Rightarrow (\sigma \Rightarrow \tau) \Rightarrow \rho + \sigma \Rightarrow \tau,$$

$$\big(\otimes^+_{\rho\sigma}\big)^{\rho\Rightarrow\sigma\Rightarrow\rho\otimes\sigma} := \mathrm{C}_1^{\rho\otimes\sigma},$$
$$\mathcal{R}^\tau_{\rho\otimes\sigma} \colon (\rho \Rightarrow \sigma \Rightarrow \tau) \Rightarrow \rho\otimes\sigma \Rightarrow \tau.$$

Terms are inductively defined from typed variables x^ρ and the constants, that is, constructors $\mathrm{C}_i^{\vec{\mu}}$ and recursion operators $\mathcal{R}^{\vec{\mu},\vec{\tau}}_{\mu_j}$, by abstraction $\lambda x^\rho M^\sigma$ and application $M^{\rho\Rightarrow\sigma}N^\rho$. One can see easily that for instance the following functions can be "expressed" by means of terms involving recursion operators: *existence* $E_{\mathrm{nat}} \colon \mathrm{nat} \Rightarrow \mathrm{boole}$ and $E_{\mathrm{list}(\alpha)} \colon \mathrm{nat} \Rightarrow \mathrm{boole}$, and *equality* $= \colon \mathrm{nat} \Rightarrow \mathrm{nat} \Rightarrow \mathrm{boole}$.

$$E_{\mathrm{nat}}(0) := \mathtt{tt}, \qquad (0 = 0) := \mathtt{tt},$$
$$E_{\mathrm{nat}}(\mathrm{S}(n)) := E_{\mathrm{nat}}(n); \qquad (0 = \mathrm{S}(n)) := \mathtt{ff},$$
$$E_{\mathrm{list}(\alpha)}(\mathrm{nil}) := \mathtt{tt}, \qquad (\mathrm{S}(m) = 0) := \mathtt{ff},$$
$$E_{\mathrm{list}(\alpha)}(\mathrm{cons}(x,l)) := E_{\mathrm{list}(\alpha)}(l); \qquad (\mathrm{S}(m) = \mathrm{S}(n)) := \ (n = m).$$

4.3. Total Functionals

We now single out the total continuous functionals from the partial ones. Our main goal will be the density theorem, which says that every finite functional can be extended to a total one. It was first stated by Kreisel [1959]. Proofs of various versions of the density theorem have been given by Ershov [1972], Berger [1993], Stoltenberg-Hansen et al. [1994], Schwichtenberg [1996] and Kristiansen and Normann [1997]. Here we give a proof for the practically important case where the base domains are not just the flat domain of natural numbers, but non-flat and possibly parametrized free algebras.

4.3.1. Total and structure-total ideals. It is well-known how one can single out the total functionals from the partial ones.

Definition. The *total* ideals of type ρ are defined inductively.

- *Case* μ. For an algebra μ, the total ideals x are those of the form $\mathrm{C}\vec{z}$ with C a constructor of μ and $\vec{z}$ total (C denotes the continuous function $|r_{\mathrm{C}}|$).

- *Case* $\rho \Rightarrow \sigma$. An ideal r of type $\rho \Rightarrow \sigma$ is total if and only if for all total z of type ρ, the result $|r|(z)$ of applying r to z is total.

The *structure-total* ideals are defined similarly; the difference is that in case μ the ideals at parameter positions of C need not be total. – We write $x \in G_\rho$ to mean that x is a total ideal of type ρ.

For instance, for nat the ideals 0, $S0$, $S(S0)$ etc. in Fig. 2 on page 81 are total, but $\bot$, $S\bot$, $S(S\bot)$, $\ldots$, ∞ are not. For list(ρ), precisely all ideals of the form $\mathrm{cons}(x_1, \ldots \mathrm{cons}(x_n, \mathrm{nil}) \ldots)$ are structure-total. The total ones are those where in addition all list elements $x_1, \ldots, x_n$ are total.

Remark. Notice that in the arrow case of the definition of totality, we have made use of the universal quantifier "for all total z of type ρ" with an implication is its kernel. So using the concept of a total computable functional to explain the meaning of the logical connectives – as it is done in the well-known Brouwer-Heyting-Kolmogorov interpretation – is in this sense circular.

For non-flat base domains it is easy to see that there are maximal but not total ideals: ∞ is an example for nat. This is less easy for flat base domains; a counterexample has been given by Ershov [1974]; a more perspicious one (at type (nat $\Rightarrow$ nat) $\Rightarrow$ nat) is in [Stoltenberg-Hansen et al., 1994].

Conversely, the total continuous functionals need not be maximal ideals in $\boldsymbol{C}_\rho$: A counterexample is $\{\, (S^n 0, 0) \mid n \in \mathrm{nat} \,\}$, which clearly is a total object of type nat $\Rightarrow$ nat representing the constant function with value 0. However, addition of the pair $(\emptyset, 0)$ yields a different total object of type nat $\Rightarrow$ nat. However, it is easy to show both functionals are "equivalent" in the sense that they have the same behaviour on total arguments.

4.3.2. Equality for total functionals.

Definition. An *equivalence* $\sim_\rho$ between total ideals $x_1, x_2 \in G_\rho$ is defined inductively.

- *Case* μ. For an algebra μ, two total ideals x_1, x_2 are equivalent if both are of the form $\mathrm{C}\vec{z}_i$ with the same constructor C of μ, and we have $z_{1j} \sim_\tau z_{2j}$ for all j.
- *Case* $\rho \Rightarrow \sigma$. Two ideals f, g of type $\rho \Rightarrow \sigma$ are equivalent if and only if $\forall_{x \in G_\rho} f(x) \sim_\sigma g(x)$.

Clearly $\sim_\rho$ is an equivalence relation. Similarly, one can define an equivalence relation $\approx_\rho$ between structure-total ideals x_1, x_2.

We obviously want to know that $\sim_\rho$ (and similarly $\approx_\rho$) is compatible with application; we only treat $\sim_\rho$ here. The nontrivial part of this argument is to show that $x \sim_\rho y$ implies $f(x) \sim_\sigma f(y)$. First we need some lemmata. Recall that our partial continuous functionals are ideals (i.e., certain sets of tokens) in the information systems $\boldsymbol{C}_\rho$.

Lemma. *If* $f \in G_\rho$, $g \in |\boldsymbol{C}_\rho|$ *and* $f \subseteq g$, *then* $g \in G_\rho$.

Proof. By induction on ρ. For base types μ the claim easily follows from the IH. $\rho \Rightarrow \sigma$: Assume $f \in G_{\rho \Rightarrow \sigma}$ and $f \subseteq g$. We must show $g \in G_{\rho \Rightarrow \sigma}$. So let $x \in G_\rho$. We have to show $g(x) \in G_\sigma$. But $g(x) \supseteq f(x) \in G_\sigma$, so the claim follows by IH. $\square$

LEMMA.

(4.3) $(f_1 \cap f_2)(x) = f_1(x) \cap f_2(x)$, *for* $f_1, f_2 \in |\boldsymbol{C}_{\rho\Rightarrow\sigma}|$ *and* $x \in |\boldsymbol{C}_\rho|$.

PROOF. By the definition of $|r|$,

$$\begin{aligned}
&|f_1 \cap f_2|(x)\\
&= \{\, b \in C_\sigma \mid \exists_{U\subseteq x}\,(U,b) \in f_1 \cap f_2 \,\}\\
&= \{\, b \in C_\sigma \mid \exists_{U_1\subseteq x}\,(U_1,b) \in f_1 \,\} \cap \{\, b \in C_\sigma \mid \exists_{U_2\subseteq x}\,(U_2,b) \in f_2 \,\}\\
&= |f_1|(x) \cap |f_2|(x).
\end{aligned}$$

The part $\subseteq$ of the middle equality is obvious. For $\supseteq$, let $U_i \subseteq x$ with $(U_i, b) \in f_i$ be given. Choose $U = U_1 \cup U_2$. Then clearly $(U, b) \in f_i$ (as $(U_i, b) \geq (U, b)$ and f_i is deductively closed). □

LEMMA. $f \sim_\rho g$ *if and only if* $f \cap g \in G_\rho$, *for* $f, g \in G_\rho$.

PROOF. By induction on ρ. For base types μ the claim easily follows from the IH. $\rho \Rightarrow \sigma$:

$$\begin{aligned}
f \sim_{\rho\Rightarrow\sigma} g &\iff \forall_{x\in G_\rho}\, f(x) \sim_\sigma g(x)\\
&\iff \forall_{x\in G_\rho}\, f(x) \cap g(x) \in G_\sigma \quad \text{by IH}\\
&\iff \forall_{x\in G_\rho}\, (f \cap g)x \in G_\sigma \quad \text{by (4.3)}\\
&\iff f \cap g \in G_{\rho\Rightarrow\sigma}.
\end{aligned}$$

This completes the proof. □

THEOREM. $x \sim_\rho y$ *implies* $f(x) \sim_\sigma f(y)$, *for* $x, y \in G_\rho$ *and* $f \in G_{\rho\Rightarrow\sigma}$.

PROOF. Since $x \sim_\rho y$ we have $x \cap y \in G_\rho$ by the last lemma. Now $f(x), f(y) \supseteq f(x \cap y)$ and hence $f(x) \cap f(y) \in G_\sigma$. But this implies $f(x) \sim_\sigma f(y)$ again by the last lemma. □

4.3.3. Dense and separating sets. We now prove the density theorem, which says that every finitely generated functional (i.e., every $\overline{U}$ with $U \in \mathrm{Con}_\rho$) can be extended to a total functional.

However, we need some assumptions on the base types for this theorem to hold. Otherwise, density might fail for the trivial reason that there are no total ideals at all (e.g., in $\mu\alpha\,(\alpha \to \alpha)$). A type $\mu\alpha_1, \ldots, \alpha_N(\kappa_1, \ldots, \kappa_n)$ is said to *have total ideals* if for every j $(1 \leq j \leq N)$ there is a constructor type κ_{i_j} of form (4.1) with $j_1, \ldots, j_n < j$. Then clearly for every j we have a total ideal of type α_j; call it z_j. Moreover, we assume that all base types are finitary. Then their total ideals are finite and maximal, which will be used in the proof.

THEOREM (Density). *Assume that all base types are finitary and have total ideals. Then for every* $U \in \mathrm{Con}_\rho$ *we can find an* $x \in G_\rho$ *such that* $U \subseteq x$.

PROOF. Call a type ρ *dense* if $\forall_{U\in\mathrm{Con}_\rho}\exists_{x\in G_\rho}\, U \subseteq x$, and *separating* if

$$\forall_{U_1,U_2\in\mathrm{Con}_\rho}\big(U_1 \cup U_2 \notin \mathrm{Con}_\rho \Rightarrow \exists_{\vec{z}\in G}\,\mathrm{InCon}(\overline{U_1}(\vec{z}) \cup \overline{U_2}(\vec{z}))\big).$$

Here $\vec{z} \in G$ means that $\vec{z}$ is a sequence of total z_i such that $U_j(\vec{z})$ is of a base type μ. Cleary $\mathrm{InCon}(x)$ means that the (possibly infinite) set x of atoms

contains a finite subset $U \notin \mathrm{Con}$. We prove by simultaneous induction on ρ that every type ρ is dense and separating. This extended claim is needed for the inductive argument.

For base types μ both claims are easy: the fact that μ is separating is obvious, and density for μ can be inferred from the IH, as follows. For simplicity of notation assume that μ is non-simultaneously defined. Let $U \in \mathrm{Con}_\mu$. Then (since μ is finitary) $\exists_b \forall_{a \in U}\, b \geq a$. In the token b, replace every constructor symbol by its corresponding continuous function, every token at a parameter argument position by a total ideal of its type (which exists by IH), and every $*$ at a type-μ-position by the total ideal z of type μ (which exists by assumption). The result is the required total ideal.

$\rho \Rightarrow \sigma$ is separating: This will follow from the inductive hypotheses that ρ is dense and σ is separating. So let $W, W' \in \mathrm{Con}_{\rho\Rightarrow\sigma}$ such that $W \cup W' \notin \mathrm{Con}_{\rho\Rightarrow\sigma}$. Then there are $(U, a) \in W$ and $(U', a') \in W'$ such that $U \cup U' \in \mathrm{Con}_\rho$ but $a \not\succ a'$. Since ρ is dense, we have a $z \in G_\rho$ such that $U \cup U' \subseteq z$. Hence $a \in \overline{W}(z)$ and $a' \in \overline{W'}(z)$. Now since σ is separating there are $\vec{z} \in G$ such that

$$\mathrm{InCon}(\overline{\{a\}}(\vec{z}) \cup \overline{\{a'\}}(\vec{z})),$$

hence also

$$\mathrm{InCon}(\overline{W}(z, \vec{z}) \cup \overline{W'}(z, \vec{z})).$$

This concludes the proof that $\rho \Rightarrow \sigma$ is separating.

$\rho \Rightarrow \sigma$ is dense: This will follow from the inductive hypotheses that ρ is separating and σ is dense. So fix $W = \{\, (U_i, a_i) \mid i \in I \,\} \in \mathrm{Con}_{\rho\Rightarrow\sigma}$. Consider i, j such that $a_i \not\succ a_j$. Then $U_i \cup U_j \notin \mathrm{Con}_\rho$. Since ρ is separating, there are $\vec{z}_{ij} \in G$ and $k_{ij}, l_{ij} \in G_\mu$ such that with $k_{ij} := \overline{U_i}(\vec{z}_{ij})$ and $l_{ij} := \overline{U_j}(\vec{z}_{ij})$

$$\mathrm{InCon}(k_{ij} \cup l_{ij}).$$

We clearly may assume that $\vec{z}_{ij} = \vec{z}_{ji}$ and $(k_{ij}, l_{ij}) = (l_{ji}, k_{ji})$.

Now define for every $U \in \mathrm{Con}_\rho$ a set I_U of indices $i \in I$ such that "U behaves as U_i with respect to the $\vec{z}_{ij}$". More precisely, let

$$I_U := \{\, i \in I \mid \forall_j (a_i \not\succ a_j \to \overline{U}(\vec{z}_{ij}) = k_{ij}) \,\}.$$

We first show that

$$\{\, a_i \mid i \in I_U \,\} \in \mathrm{Con}_\sigma. \tag{4.4}$$

It suffices to show that $a_i \smile a_j$ for all $i, j \in I_U$. So let $i, j \in I_U$ and assume $a_i \not\succ a_j$. Then $\overline{U}(\vec{z}_{ij}) = k_{ij}$ as $i \in I_U$ and $\overline{U}(\vec{z}_{ji}) = k_{ji}$ as $j \in I_U$, and because of $\vec{z}_{ij} = \vec{z}_{ji}$ and $\mathrm{InCon}(k_{ij} \cup k_{ji})$ (recall $l_{ij} = k_{ji}$) we could conclude that $\overline{U}(\vec{z}_{ij})$ would be inconsistent. This contradiction proves $a_i \smile a_j$ and hence (4.4).

Since (4.4) holds and σ is dense by IH, we can find $y_{I_U} \in G_\sigma$ such that $a_i \in y_{I_U}$ for all $i \in I_U$. Define $r \subseteq \mathrm{Con}_\rho \times C_\sigma$ by

$$r(U, a) \iff \begin{cases} a \in y_{I_U}, & \text{if } \overline{U}(\vec{z}_{ij}) \text{ is finite and maximal for all } \vec{z}_{ij}; \\ \exists_{i \in I_U}\, a_i \geq_\sigma a, & \text{otherwise.} \end{cases}$$

We will show that $r \in G_{\rho\Rightarrow\sigma}$ and $W \subseteq r$.

For $W \subseteq r$ we have to show $r(U_i, a_i)$ for all $i \in I$. But this holds, since clearly $i \in I_{U_i}$ and also $a_i \in y_{I_{U_i}}$.

We now show that r is an approximable map, i.e., that $r \in |\boldsymbol{C}_{\rho\Rightarrow\sigma}|$. To prove this we have to verify the defining properties of approximable maps.

(a). $r(U, b_1)$ and $r(U, b_2)$ implies $b_1 \smile b_2$. If $\overline{U}(\vec{z}_{ij})$ is finite and maximal for all $\vec{z}_{ij}$, the claim follows from the consistency of y_{I_U}. If not, the claim follows from the general properties of acis's.

(b). $r(U, b)$, $V \geq_A U$ and $b \geq_B c$ implies $r(V, c)$. First assume that $\overline{U}(\vec{z}_{ij})$ is finite and maximal for all $\vec{z}_{ij}$. Then also $\overline{V}(\vec{z}_{ij})$ is maximal for all $\vec{z}_{ij}$. From $r(U, b)$ we get $b \in y_{I_U}$. We have to show that $c \in y_{I_V}$. But since $\overline{U}(\vec{z}_{ij})$ and $\overline{V}(\vec{z}_{ij})$ are maximal for all $\vec{z}_{ij}$ and $V \geq_\rho U$, they must have the same values on the $\vec{z}_{ij}$, hence $I_U = I_V$, so $y_{I_U} = y_{I_V}$ and therefore $c \in y_{I_V}$ by deductive closure. Now assume the contrary. From $r(U, b)$ we get $a_i \geq_\sigma b$ for some $i \in I_U$. From $V \geq_\rho U$ we can conclude $I_U \subseteq I_V$, by the definition of I_U. Hence $i \in I_V$, and also $b \in y_{I_V}$ (since $a_i \in y_{I_U}$ for all $i \in I_U$, and y_{I_V} is deductively closed). Therefore $r(V, b)$ and hence $r(V, c)$.

This concludes the proof that r is an approximable map. It remains to prove $r \in G_{\rho\Rightarrow\sigma}$. So let $x \in G_\rho$. We must show

$$|r|(x) = \{\, a \in C_\sigma \mid \exists_{U\subseteq x}\, r(U, a) \,\} \in G_\sigma.$$

Now $x(\vec{z}_{ij})$ is total for all i, j, hence by our assumption on base types finite and maximal. So there is some $U_{ij} \subseteq x$ such that $\overline{U_{ij}}(\vec{z}_{ij}) = x(\vec{z}_{ij})$. Let $U \subseteq x$ be the union of all the U_{ij}. Then by definition $r(U, a)$ for all $a \in y_{I_U}$. Therefore $y_{I_U} \subseteq |r|(x)$ and hence $|r|(x) \in G_\sigma$. □

As an application of the density theorem we prove a choice principle for total continuous functionals.

Theorem (Choice principle for total functionals). *There is an ideal* $\Gamma \in |\boldsymbol{C}_{(\rho\Rightarrow\sigma\Rightarrow\mathrm{boole})\Rightarrow\rho\Rightarrow\sigma}|$ *such that for every* $F \in G_{\rho\Rightarrow\sigma\Rightarrow\mathrm{boole}}$ *satisfying*

$$\forall_{x\in G_\rho}\exists_{y\in G_\sigma}\, F(x, y) = \mathtt{tt}$$

we have $\Gamma(F) \in G_{\rho\Rightarrow\sigma}$ *and*

$$\forall_{x\in G_\rho}\, F(x, \Gamma(F, x)) = \mathtt{tt}.$$

Proof. Let $V_0, V_1, V_2, \ldots$ be an enumeration of Con_σ. By the density theorem we can find $y_n \in G_\sigma$ such that $V_n \subseteq y_n$. Define a relation $r \subseteq \mathrm{Con}_{\rho\Rightarrow\sigma\Rightarrow\mathrm{boole}} \times C_{\rho\Rightarrow\sigma}$ by

$$r(W, U, a) \iff \exists_m \forall_{i<m}\, (\overline{W}(\overline{U}, y_i) = \mathtt{ff} \wedge \overline{W}(\overline{U}, y_m) = \mathtt{tt} \wedge a \in y_m).$$

We first show that $\Gamma := r$ is an approximable map. To prove this we have to verify the clauses of the definition of approximable maps.

(a). $r(W, U_1, a_1)$ and $r(W, U_2, a_2)$ imply $U_1, a_1) \smile (U_2, a_2)$. Assume the premise and $U := U_1 \cup U_2 \in \mathrm{Con}_\rho$. We show $a_1 \smile a_2$. The numbers m_i in the definition of $r(W, U_i, a_i)$ are the same, $= m$ say. Hence $a_1, a_2 \in y_m$, and the claim follows from the consistency of y_m.

(b). $r(W', U, a)$, $W \geq W'$ and $(U, a) \geq (V, b)$ implies $r(W, V, b)$. Then $V \geq U$ and $a \geq b$. The claim follows from the definition of r, using the deductive closure of y_m. The m from $r(W', U, a)$ can be used for $r(W, U, a)$.

We finally show that for all $F \in G_{\rho\Rightarrow\sigma\Rightarrow\mathrm{boole}}$ satisfying

$$\forall_{x\in G_\rho}\exists_{y\in G_\sigma} F(x,y) = \mathtt{tt}$$

and all $x \in G_\rho$ we have $\Gamma(F,x) \in G_\sigma$ and $F(x,\Gamma(F,x)) = \mathtt{tt}$. So let F and x with these properties be given. By assumption there is a $y \in G_\sigma$ such that $F(x,y) = \mathtt{tt}$. Hence by the definition of application there is a $V_n \in \mathrm{Con}_\sigma$ such that $F(x,\overline{V_n}) = \mathtt{tt}$. Since $V_n \subseteq y_n$ we also have $F(x,y_n) = \mathtt{tt}$. Clearly we may assume here that n is minimal with this property, i.e., that

$$F(x,y_0) = \mathtt{ff}, \dots, F(x,y_{n-1}) = \mathtt{ff}.$$

We show that $\Gamma(F,x) \supseteq y_n$; this suffices because the extension of a total ideals is total. Recall that

$$\Gamma(F) = \{\, (U,a) \in \mathrm{Con}_\rho \times C_\sigma \mid \exists_{W\subseteq F}\, r(W,U,a) \,\}$$

and

$$\begin{aligned}\Gamma(F,x) &= \{\, a \in C_\sigma \mid \exists_{U\subseteq x}\,(U,a) \in \Gamma(F) \,\}\\ &= \{\, a \in C_\sigma \mid \exists_{U\subseteq x}\exists_{W\subseteq F}\, r(W,U,a) \,\}.\end{aligned}$$

Let $a \in y_n$. By the choice of n we get $U \subseteq x$ and $W \subseteq F$ such that

$$\forall_{i<n}\, \overline{W}(\overline{U},y_i) = \mathtt{ff} \quad \text{and} \quad \overline{W}(\overline{U},y_n) = \mathtt{tt}.$$

Therefore $r(W,U,a)$ and hence $a \in \Gamma(F,x)$. □

4.3.4. Effectiveness. From the proofs of both theorems it can be seen that the functionals constructed are in fact computable. More precisely we have:

THEOREM (Effective density). *For every $U \in \mathrm{Con}_\rho$ we can find a computable $x \in G_\rho$ such that $U \subseteq x$.*

PROOF. By inspection of the proof of the density theorem. To see that r (in the proof that $\rho \Rightarrow \sigma$ is dense) is Σ^0_1-definable observe that $\exists_{i\in I_U} a_i \geq a$ implies $a \in y_{I_U}$ for all U and a, since by definition $a_i \in y_{I_U}$ for all $i \in I_U$. Hence

$r(U,a) \iff$

$\exists_{i\in I_U} a_i \geq a$ or ($a \in y_{I_U}$ and $\overline{U}(\vec{z}_{ij})$ is finite and maximal for all $\vec{z}_{ij}$).

Moreover, if $\overline{U}(\vec{z}_{ij})$ is finite and maximal for all $\vec{z}_{ij}$, one can actually compute I_U (and not only an enumeration procedure for I_U). □

THEOREM (Effective choice principle). *There is a computable Γ of type $(\rho \Rightarrow \sigma \Rightarrow \mathrm{boole}) \Rightarrow \rho \Rightarrow \sigma$ such that for every $F \in G_{\rho\Rightarrow\sigma\Rightarrow\mathrm{boole}}$ satisfying*

$$\forall_{x\in G_\rho}\exists_{y\in G_\sigma} F(x,y) = \mathtt{tt}$$

we have $\Gamma(F) \in G_{\rho\Rightarrow\sigma}$ and

$$\forall_{x\in G_\rho} F(x,\Gamma(F,x)) = \mathtt{tt}.$$

PROOF. Immediate from the proof of the choice principle for total continuous functionals. □

The effective choice principle generalizes the simple fact that whenever we know the truth of $\forall_{x\in\mathbb{N}}\exists_{y\in\mathbb{N}} P(x,y)$ with $P(x,y)$ decidable, then given x we can just search for a y such that $P(x,y)$ holds; the truth of $\forall_{x\in\mathbb{N}}\exists_{y\in\mathbb{N}} P(x,y)$ guarantees termination of the search.

Bibliography

A. Abel and T. Altenkirch. A predicative strong normalization proof for a λ-calculus with interleaving inductive types. In *Types for Proofs and Programs, International Workshop, TYPES '99, Lökeberg, Sweden, June 1999*, volume 1956 of *LNCS*, pages 21–40. Springer Verlag, Berlin, Heidelberg, New York, 2000.

S. Abramsky. Domain theory in logical form. *Annals of Pure and Applied Logic*, 51:1–77, 1991.

S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

R. M. Amadio and P.-L. Curien. *Domains and Lambda-Calculi*. Cambridge University Press, 1998.

H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The Journal of Symbolic Logic*, 48(4):931–940, 1983.

H. Benl. Konstruktive Interpretation induktiver Definitionen. Master's thesis, Mathematisches Institut der Universität München, 1998.

U. Berger. Total sets and objects in domain theory. *Annals of Pure and Applied Logic*, 60:91–117, 1993.

U. Berger. Continuous semantics for strong normalization. In *Proc. CiE 2005*, volume 3526 of *LNCS*, pages 23–34, 2005.

U. Berger, M. Eberl, and H. Schwichtenberg. Term rewriting for normalization by evaluation. *Information and Computation*, 183:19–42, 2003.

F. Blanqui, J.-P. Jouannaud, and M. Okada. The Calculus of Algebraic Constructions. In *RTA'99. LNCS 1631*, 1999.

T. Coquand, G. Sambin, J. Smith, and S. Valentini. Inductively generated formal topologies. *Annals of Pure and Applied Logic*, 124:71–106, 2003.

T. Coquand and A. Spiwack. Proof of normalisation using domain theory. Slides of a talk, October 2005.

H. B. Enderton and D. Luckham. Hierarchies over recursive well-orderings. *The Journal of Symbolic Logic*, 29:183–190, 1964.

Y. L. Ershov. Everywhere defined continuous functionals. *Algebra i Logika*, 11(6):656–665, 1972.

Y. L. Ershov. Maximal and everywhere defined functionals. *Algebra i Logika*, 13(4):374–397, 1974.

J.-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92. North–Holland, Amsterdam, 1971.

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

P. G. Hinman. *Recursion–Theoretic Hierarchies*. Springer Verlag, Berlin, Heidelberg, New York, 1978.

S. C. Kleene. On the forms of the predicates in the theory of constructive ordinals. *Amer. J. Math.*, 66:41–58, 1944.

S. C. Kleene. *Introduction to Metamathematics*. D. van Nostrand Comp., New York, 1952.

S. C. Kleene. On the forms of the predicates in the theory of constructive ordinals (second paper). *Amer. J. Math.*, 77:405–428, 1955.

S. C. Kleene. Countable functionals. In A. Heyting, editor, *Constructivity in Mathematics*, pages 81–100. North–Holland, Amsterdam, 1959.

G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North–Holland, Amsterdam, 1959.

L. Kristiansen and D. Normann. Total objects in inductively defined types. *Archive for Mathematical Logic*, 36(6):405–436, 1997.

K. G. Larsen and G. Winskel. Using information systems to solve recursive domain equations. *Information and Computation*, 91:232–258, 1991.

P. Martin-Löf. The domain interpretation of type theory. Talk at the workshop on semantics of programming languages, Chalmers University, Göteborg, August 1983.

P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

R. Milner. Implementation and applications of Scott's logic for computable functions. In *Proc ACM Conf on Proving Assertions About Programs, Las Cruces, New Mexico*, 1972.

R. Milner. Models of LCF. Technical Report Memo Aim-186, Stanford Artificial Intelligence Laboratory, January 1973.

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

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

H. Rogers. *Theory of recursive functions and effective computability*. Mc Graw Hill, 1967.

H. Schwichtenberg. Density and choice for total continuous functionals. In P. Odifreddi, editor, *Kreiseliana. About and Around Georg Kreisel*, pages 335–362. A.K. Peters, Wellesley, Massachusetts, 1996.

H. Schwichtenberg. Recursion on the partial continuous functionals. In C. Dimitracopoulos, L. Newelski, D. Normann, and J. Steel, editors, *Logic Colloquium '05*, volume 28 of *Lecture Notes in Logic*, pages 173–201. Association for Symbolic Logic, 2006.

D. Scott. A type theoretical alternative to ISWIM, CUCH, OWHY. Published in Theoret. Comput. Sci. 121 (1993), 411–440, 1969.

D. Scott. Outline of a mathematical theory of computation. Technical Monograph PRG–2, Oxford University Computing Laboratory, 1970.

D. Scott. Domains for denotational semantics. In E. Nielsen and E. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *LNCS*, pages 577–613. Springer Verlag, Berlin, Heidelberg, New York, 1982.

A corrected and expanded version of a paper prepared for ICALP'82, Aarhus, Denmark.

D. Scott and C. Strachey. Toward a mathematical semantics for computer languages. Technical Monograph PRG–6, Oxford University Computing Laboratory, 1971.

J. Shepherdson and H. Sturgis. Computability of recursive functions. *J. Ass. Computing Machinery*, 10:217–255, 1963.

J. R. Shoenfield. *Mathematical Logic*. Addison–Wesley Publ. Comp., Reading, Massachusetts, 1967.

R. I. Soare. *Recursively Enumerable Sets and Degrees*. Perspectives in Mathematical Logic. Springer Verlag, Berlin, Heidelberg, New York, 1987.

C. Spector. Recursive well–orderings. *The Journal of Symbolic Logic*, 20: 151–163, 1955.

C. Spector. Inductively defined sets of natural numbers. In *Infinitistic Methods. Proceedings of the Symposium on Foundations of Mathematics*, pages 97–102. Panstwowe Wydawnictwo Naukowe (PWN), Warschau, 1961.

V. Stoltenberg-Hansen, E. Griffor, and I. Lindström. *Mathematical Theory of Domains*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1994.

W. W. Tait. Normal form theorem for bar recursive functions of finite type. In J. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 353–367. North–Holland, Amsterdam, 1971.

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.

J. Zucker. Iterated inductive definitions, trees and ordinals. In A. Troelstra, editor, *Mathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*, pages 392–453. Springer Verlag, Berlin, Heidelberg, New York, 1973.

Index