Tiered Arithmetic, its Functional Interpretation and Slow Growing Bounds

Naim Çağman

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

The University of Leeds

Department of Pure Mathematics

January 2000

The candidate confirms that the work submitted is his own and that appropriate credit has been given where reference has been made to the work of others.

Acknowledgement

I have enjoyed writing this thesis with the following many people who have helped and it is a pleasure to thank them now.

Firstly and most importantly I would like to thank Professor S.S. Wainer, my supervisor, teacher and friend, for his cheerful, expert, and firm guidance. He skillfully introduced me to the beauty of mathematical logic. He always answered all my question, even stupid ones, and was always willing to explain mysteries of proof theory. He had great influence on the direction of this thesis.

I would like to extend my thanks to every single person who took part directly or indirectly in this work at Leeds University. The very considerable extent to Professor S.B. Cooper, Dr H.D Macpherson, Dr M. Rathjen, Dr H. Simmons and Professor J.K. Truss for their postgraduate Mathematical Logic courses. I learned much from them. They were always kind enough to give the benefit of their advice, asked and unasked. In particular to all of the postgraduate students of Mathematical Logic Group for providing the intellectual environment which made my period of study at Leeds an interesting and rewarding one. A special thanks to my academic brother, G.E. Ostrin, for friendly talks and useful discussions about our work.

I am very grateful to The Higher Educational Council of Turkey and The Gaziosmanpaşa University (in Tokat, Turkey) for their financial support during my research.

Last but not least, I would like to express my sincere thanks to my family, especially my wife Kudret, grandfather, father, mother, brothers and all other relatives, for their love, unending patience, moral support and tolerance throughout the years.

Abstract

In 1992, Bellantoni and Cook developed a new recursion theoretic characterisation of the polytime functions, BC, in which it is shown that a natural two-sorted re-interpretation of the usual primitive recursion schemes characterises polynomially bounded computation over the binary representation of numbers.

This thesis is based on Bellantoni-Cook's variable separation over unary notation. We first define a two-sorted Peano Arithmetic, $PA(;)$, formulated with this variable separation, with quantification allowed only over one sort, output (safe) variables, and induction allowed only over the other sort, input (normal) variables.

From $PA(;)$ we obtained its two-sorted fragment Σ_1-Induction, $\Sigma_1(;) - IND$, by restricting the induction rule to $\Sigma_1(;)$-formulas. We then get the result that the provably terminating functions of the $\Sigma_1(;) - IND$ turn out to be exactly the Grzegorczyk $\mathcal{E}^2$ functions, such functions being computable on a Turing Machine in Linear Space (bounded by a linear function of the length of its binary input).

We develop the two-sorted higher type Gödel primitive recursive functionals $T(;)$ by applying this variable separation. We then get the result that every provably recursive function of two-sorted full $PA(;)$ is characterised by the two-sorted Gödel's Dialectica interpretation of arithmetic, and we show that every elementary function, in Grzegorczyk's class $\mathcal{E}^3$, is definable in $T(;)$. We then prove that the converse of this result, that every definable function in $T(;)$ is elementary. This is proved by using normalisation and transfinite counting by means of infinite terms.

This work is related to other results of Buss, Bellantoni, Beckmann-Weiermann, Leivant and Ostrin but our context and methods are quite different. The bounding functions for the two-sorted theory are now the Slow Growing ones rather than the Fast Growing functions which bound computations in the usual single-sorted versions of Peano Arithmetic.

Contents

Chapter 1

Introduction

Our work may be roughly divided into two parts: *functional classes* and *arithmetical theory*. The aim is to use tiered arithmetic, its functional interpretation and slow growing bounds to explore feasible functional classes and their proof theory. We will introduce and discuss some fundamental connections between *proof-theoretic complexity* and *computational complexity* – the structural links between formal termination proofs and theoretical complexity bounds.

The difficulty is knowing where to start since there are so many preliminary issues that need to be addressed. Our basic functional class is the *primitive recursive functions*, PRIM, introduced explicitly by Skolem [25] in 1923. Although $PRIM$ is a natural and very extensive class and plays an important role in the abstract part of computing science, it does not include all computable functions. $PRIM$ can be analysed by breaking it down into Grzegorczyk's hierarchy $\mathcal{E}^n$, $n > 0$, developed by Grzegorczyk [10] in 1953. In this thesis we are interested particularly in the classes $\mathcal{E}^2$ and $\mathcal{E}^3$, because they differentiate between polynomial and super-exponential complexity. They are very important in computer science since they isolate the feasible computable functions from infeasible. The $PRIM$ and $\mathcal{E}^n$ classes will be introduced respectively in the sections 1.1 and 1.2 of this chapter.

It is well known that $PRIM$ contains many *fast-growing non-feasible* functions. But the recent works of Buss [5], Bellantoni [3], Leivant [15] and others obtained feasibly computable functions by using a natural two-sorted restriction of primitive

recursion. In other words, they show how a natural *two-sorted* restriction of primitive recursion serves to characterise complexity classes such as *polynomial-time* and *linear space*. The main contribution here is to show how classical ideas and methods can be applied quite naturally in more *computationally feasible* contexts to provide proof-theoretic characterisations of well-known complexity classes.

Because of the above results, we shall motivate here especially the elegant *normal/safe* recursion schemes of Bellantoni and Cook. Around 1992, in their original paper [2], they introduced a new class of functions, BC, to develop a more foundational characterisation of polynomial-time. They have done this by taking the *primitive recursive* function schemes and separating the variables into two sorts: *Input (normal)* and *Output (safe)* variables depending on their computational roles. The BC scheme was applied over the binary representation of numbers in order to characterise the class PTIME of functions computable in time bounded by a polynomial in the binary lengths of the inputs. We, however, will work exclusively with unary representations, giving the related class BCu.

It is worth pointing out that a recursion-theoretic characterisation of polynomial time was first given by Cobbam [6] using recursion on notations. He used explicit polynomial bounds to control the size of the output.

The variable separation was also independently introduced by Simmons [24] and Leivant [14] and in many other papers. Simmons used tiering to control the power of recursion in a paper delineating the primitive recursive functions. To achieve this he formulated more restrictive closure schemes, *allowable composition* and *allowable recursion*. Leivant introduce the notion of *tiered/ramified primitive recursion* and defined a sub-recursive hierarchy which manages to isolate polynomial from exponential numeric functions, whereas the natural sub-recursive classification of functions, by counting the recursion-nesting depth fails, since exponentiation can be defined from addition by single recursion.

Around 1995, Handley, in his unpublished note, formulated a new class of functions, BCu, by using unary representation instead of binary in the BC. That is, the BCu is a rework of the Bellantoni and Cook variable separation result over unary no-

tation. The BCu serves to define the class of functions computable in time bounded by a polynomial of their numerical argument, whereby the length of a number is the actual numerical value. Handley and Wainer [11] details this characterisation of Grzegorczyk's $\mathcal{E}^2$, equivalently the class $LINSPACE$ of functions computable on a Turing Machine in linear space. BCu will be introduced briefly in the section 1.3 of this chapter.

In Chapter 2, we shall pass from functional class to a corresponding arithmetical theory in two-sorted form. We will present there a simple two-sorted version of first order Peano Arithmetic, $PA(;)$, in which *induction* is allowed only over *input* variables and *quantification* is allowed only over *output* variables. This system provides a natural setting where classical proof-theoretic methods can be used to characterise the hierarchy of complexity classes between polynomial time and super-exponential time. The theory $PA(;)$ was first developed and studied by Ostrin in his PhD thesis [18]. We then obtain the theory of two-sorted Σ_1-Induction, $\Sigma_1(;) - IND$, from $PA(;)$ by restricting the induction rule to apply only to $\Sigma_1(;)$-formulas, and get the result that the class of provably terminating functions of $\Sigma_1(;) - IND$ is exactly the class BCu. This means that $\Sigma_1(;) - IND$ provides a proof theoretic characterisation of Grzegorczyk's $\mathcal{E}^2$.

In Chapter 3, and in the rest of the thesis, we shall apply the variable separation to Gödel's functional interpretation, yielding a two-sorted version of Gödel's primitive recursive functionals, denoted $T(;)$. (Another different approach is worked by Ostrin [18] where this variable separation is applied to ordinal analysis in a low subrecursive context, showing that $PA(;)$ proves the totality of at most the elementary functions, analogous to classical PA). Our main contribution here is to develop the Gödel-Shoenfield [23] dialectica style interpretion of $PA(;)$, using a BCu-version of higher type primitive recursion. It follows that every provably recursive function of $PA(;)$ is characterised by a two-sorted Gödel-Dialectica functional in $T(;)$. In the final section of this chapter we will show explicitly how every elementary function can be defined in $T(;)$.

In Chapter 4, we prove the converse to the result of the previous chapter: every

definable function in $T(;)$ is elementary. This is proved by using normalisation of infinitary λ-terms with tree-ordinal bounds, and is the main technical work of this thesis. The infinite terms were first introduced by Tait [26] and Schwichtenberg [22]. In order to fulfil our aim we shall first formulate an infinitary version of Gödel's $T(;)$ by replacing the substitution, primitive recursion and finite list by λ-abstraction, infinite sequences and cartesian products respectively. Every functional of $T(;)$ is represented by a term of this new infinitary system $T^{\infty}(;)$ and the tree-ordinal bounds allow us to measure complexity in terms of the Slow Growing Hierarchy below ϵ_0. We then get the result that every number-theoretic function definable in $T(;)$ is elementary, since every function computable within an elementary-bounded number of steps is itself elementary. The result is related to the work of Beckmann and Weiermann [1], but the context and the proof are quite different.

1.1 Primitive Recursive Functions

PRIM is the smallest class of functions that contains the *initial functions*, and is closed under the operations of *substitution* and *primitive recursion.*

Definition 1.1.1 (Initial Functions). There are three initial function which are

(a) The *zero function*, $Z(x) = 0$ for every $x \in \mathbb{N}$

(b) The *successor function*, $S(x) = x + 1$ for every $x \in \mathbb{N}$

(c) The *projection functions*, $U_i^n(\vec{x}) = x_i$ for every $x_i \in \mathbb{N}$ where $\vec{x} = x_1, ..., x_n$

Definition 1.1.2 (Substitution). A common way of manufacturing new functions from old is to substitute into other functions. To be precise, suppose that $g(x_1, ..., x_k)$ and $h_1(\vec{x})$, .., $h_k(\vec{x})$ are functions which are already defined, then *composition* yields the new function $f(\vec{x})$ by

$$f(\vec{x}) = g(h_1(\vec{x}), .., h_k(\vec{x}))$$

Definition 1.1.3 (Primitive Recursion). Recursion is a method of defining a function by specifying each of its values in term of previously defined values, and

possibly using other already defined functions. To be precise, suppose functions g and h are already defined, then f is given by primitive recursion on h with basis g by

$$\begin{cases} f(0,\vec{x}) &= g(\vec{x}) \\ f(z+1,\vec{x}) &= h(z, f(z,\vec{x}), \vec{x}) \end{cases}$$

We can refer to Péter [20], Mendelson [16] and Rose [21] for background details concerning primitive recursion functions.

1.2 Grzegorczyk's Hierarchy

Grzegorczyk's class $\mathcal{E}^n$ is the smallest class containing the *zero*, *maximum* and *projection functions* as well as E_n (defined below) which is closed under *substitution* and *limited recursion*.

Definition 1.2.1 (E_n). E_n arises by primitive recursion from E_{n-1} as follows:

$$\begin{aligned} E_0(x) &= x+1 \\ E_1(x) &= 2x \\ E_2(x) &= x^2 \\ E_3(x) &= 2^x \end{aligned}$$

and for $n > 3$

$$E_n(x) = E_{n-1}^x(2)$$

where, for any function f of one variable f^x is the x-times iterate of f given by

$$\begin{cases} f^0(y) &= y \\ f^{x+1}(y) &= f(f^x(y)) \end{cases}$$

Definition 1.2.2 (Limited Primitive Recursion). If g, h and p are functions which have been introduced previously in the class, then we define a new function f from g, h, p by *limited primitive recursion* as follows

$$\begin{cases} f(0,\vec{x}) &= g(\vec{x}) \\ f(y+1,\vec{x}) &= h(y, f(y,\vec{x}), \vec{x}) \\ f(y,\vec{x}) &\leq p(y,\vec{x}) \end{cases}$$

Here we are interested particularly in the classes $\mathcal{E}^2$ and $\mathcal{E}^3$, because they differentiate between polynomial and super-exponential complexity.

The following are well-known facts about $\mathcal{E}^2$ and $\mathcal{E}^3$. See Rose [21], Chapter 2.

Theorem 1.2.3. *The following are equivalent:*

(i) $f \in \mathcal{E}^2$

(ii) *f is computable by a register-machine within a number of steps bounded by a polynomial of its arguments.*

(iii) *f is computable by a Turing machine within space bounded by a linear function of the binary lengths of its arguments.*

Theorem 1.2.4. *The following are equivalent:*

(i) $f \in \mathcal{E}^3$

(ii) *f is computable within time or space bounded by an iterated exponential function of its arguments.*

(iii) *f is elementary functions.*

Note that the *class of elementary functions* was introduced by Kalmar [13] in 1943 and by Csillag [7] in 1947 intependently. It is definable from the zero, successor, projection, addition and modified subtraction functions, and it is closed under the operations of substitution, bounded summation and bounded products.

Definition 1.2.5 (Super-Exponentiation). The *super-exponentiation* function,

$$exp_m^n(x) = \underbrace{m^{m^{\cdot^{\cdot^{\cdot^{m^x}}}}}}_{n \text{ many } m}$$

is in $\mathcal{E}^3$ for fixed $n \in \mathbb{N}$. It is defined from *exponentiation* function, $E(m,n) = m^n$, by the primitive recursion

$$\begin{cases} exp_m^0(x) & = x \\ exp_m^{n+1}(x) & = E(m, exp_m^n(x)) \end{cases}$$

which we will use throughout the thesis.

1.3 Bellantoni-Cook System

The Bellantoni and Cook *normal/safe* recursion schemes, BC, are formulated by taking the *primitive recursive* function schemes and separating the variables into two sorts, depending on their computational roles:

- *Normal (input)* variables are represented by $x, y, z, ...$ with or without subscripts.
- *Safe (output)* variables are represented by $a, b, c, ...$ with or without subscripts.

To formalise this distinction, in any formula the input variables will always be listed first and followed immediately by a semi-colon to distinguish them from the remaining output variables (if any). We think of $f(x; a)$ as denoting a function of type $I \times O \to O$, taking an input argument x and an output argument a to an output value.

The class BC was formulated with numbers represented in a binary notation. Handley and Wainer [11] present a BCu scheme, which is a unary version of BC, by showing directly that BCu schemes characterise Grzegorczyk's $\mathcal{E}^2$.

Here and in the rest of this thesis, we consider BCu since our work will be based on unary representation of numbers.

Notations 1.3.1. We shall adopt the following notations:

1. $\vec{x}$ stands for a tuple $x_1, x_2, ..., x_n$ of input variables, and henceforth we use $\vec{a}$ to stand for a tuple $a_1, a_2, ..., a_m$ of output variables.
2. To specify that a function f has a n input and m output variables, we write $f \in BCu^{n,m}$.
3. We will use BCu_{normal} for $\bigcup_{n\in\mathbb{N}} BCu^{n,0}$, that is, the subscript "normal" refers to the restriction to function of input arguments only.

Definition 1.3.2 (BCu). BCu is the smallest class which contains the following initial functions:

(a) *Zero*, $Z \in BCu^{0,0}$, where $Z(;) = 0$

(b) *Successor*, $S \in BCu^{0,1}$, where $S(;a) = a + 1$.

(c) *Predecessor*, $P \in BCu^{0,1}$, where $P(;a) = max(a - 1, 0)$.

(d) *Projections*, $U_i^m \in BCu^{0,m}$, where $U_i^m(;\vec{a}) = a_i, \quad 1 \leq i \leq m$

(e) *Choice*, $C \in BCu^{0,3}$, where $C(;a,b,c) = \begin{cases} b & if \quad a = 0 \\ c & if \quad a \neq 0. \end{cases}$

and is closed under *BCu-composition* over output variables and *BCu-recursion* over input variables, as below

Definition 1.3.3 (BCu-**Composition**). In order to preserve the idea of I and O-typing as described above, *BCu-composition* is only allowed over output variables. In this two-sorted formalism, *BCu-compositions* have to be of the form

$$f(\vec{x};\vec{a}) = g(x_{i_1}, ..., x_{i_k}; h_1(\vec{x};\vec{a}), ..., h_l(\vec{x};\vec{a}))$$

since we can only substitute computed values $\vec{h}(\vec{x};\vec{a})$ inside output positions. Thus if $h_1, ..., h_l \in BCu^{n,m}$, $g \in BCu^{k,l}$, $\{i_1, ..., i_k\} \subseteq \{1, 2, ..., n\}$, then $f \in BCu^{n,m}$.

Definition 1.3.4 (BCu-**Recursion**). Similarly, the *BCu-recursion* is only allowed over input variables but the composition has to be at output places, so it takes the form

$$\begin{cases} f(0,\vec{x};\vec{a}) & = g(\vec{x};\vec{a}), \\ f(z+1,\vec{x};\vec{a}) & = h(z,\vec{x};\vec{a}, f(z,\vec{x};\vec{a})), \end{cases}$$

where $g \in BCu^{n,m}$, $h \in BCu^{n+1,m+1}$, and then $f \in BCu^{n+1,m}$.

We furthermore allow "*simultaneous*" BCu-recursion.

Remark 1.3.5. The BCu-composition and BCu-recursion are essentially the so-called *safe composition* and *safe recursion* schemes of Bellantoni and Cook [2]. There is a small difference between them.

- In the *safe composition*, Bellantoni and Cook also allow substitution of functions with input variables only for the input arguments inside the function g in the BCu-composition, thus

$$f(x;a) = g(r(x);h(x;a))$$

It turns out however, that even with these more general schemes the same class of functions would be obtained (see Handley and Wainer [11]). We restrict our attention here to the restrictive form of BCu-composition given above.

Examples 1.3.6.

1. The constant function $C_k(x;a) = k$ is in BCu. By induction on k;

$$\begin{cases} C_0(x;a) &= 0 \\ C_{k+1}(x;a) &= S(;C_k(x;a)) \end{cases}$$

2. $A(x;a) = x + a$ is in BCu.

$$\begin{cases} A(0;a) &= a \\ A(x+1;a) &= S(;A(x;a)) \end{cases}$$

3. The usual primitive recursion definition of $M(x;a) = x \cdot a$

$$\begin{cases} M(0;a) &= 0 \\ M(x+1;a) &= A(a;M(x;a)) \end{cases}$$

is not in BCu since it fails to type correctly, $a \notin I$. Instead, we can define $M'(x,y;a) = x \cdot y + a$ which is in BCu.

$$\begin{cases} M'(0,y;a) &= a \\ M'(x+1,y;a) &= A(y;M'(x,y;a)) \end{cases}$$

Thus we can multiply two input variables.

4. However we cannot multiply an output variable by an input variable, since one more BCu-recursion would then give exponentiation $E(x) = 2^x$ as defined recursively by

$$\begin{cases} E(0) &= 1 \\ E(x+1) &= M(2;E(x)) \end{cases}$$

which is not in BCu because it contradicts the Lemma 1.3.7 below.

Lemma 1.3.7. *For every BCu-function $f(\vec{x};\vec{a})$ there exist a polynomial $p_f(\vec{x})$ such that for all $\vec{x}$ and $\vec{a}$*

$$f(\vec{x};\vec{a}) \leq max(\vec{a}) + p_f(\vec{x})$$

Proof. The proof is by induction on the definition of BCu function. Our polynomials will always have positive coefficients and will therefore be monotone in all their arguments. The claim clearly holds for all of the initial functions. As for the closure properties:

Case (BCu-composition). Suppose that

$$f(\vec{x};\vec{a}) = g(\vec{x}; h_1(\vec{x};\vec{a}), ..., h_m(\vec{x};\vec{a}))$$

Then by induction hypothesis on g and $\vec{h}$ we get

$$\begin{aligned} f(\vec{x};\vec{a}) &\leq max(h_1(\vec{x};\vec{a}), ..., h_m(\vec{x};\vec{a})) + p_g(\vec{x}) \\ &\leq max(max(\vec{a}) + p_{h_1}(\vec{x}), ..., max(\vec{a}) + p_{h_m}(\vec{x})) + p_g(\vec{x}) \\ &\leq max(\vec{a}) + max(p_{h_1}(\vec{x}), ..., +p_{h_m}(\vec{x})) + p_g(\vec{x}) \\ &\leq max(\vec{a}) + \textstyle\sum_{i=1}^{m} p_{h_i}(\vec{x}) + p_g(\vec{x}) \\ &\leq max(\vec{a}) + p_f(\vec{x}) \end{aligned}$$

Case (BCu-recursion). Suppose that

$$\begin{cases} f(0,\vec{x};\vec{a}) &= g(\vec{x};\vec{a}) \\ f(z+1,\vec{x};\vec{a}) &= h(z,\vec{x};\vec{a}, f(z,\vec{x};\vec{a})) \end{cases}$$

Then by induction hypothesis on g and h and by induction on z:

– *Base case*, $z = 0$, we get

$$f(0,\vec{x};\vec{a}) \leq max(\vec{a}) + p_g(\vec{x})$$

– *Induction case*, assume that for $z = n$ we already have

$$f(n,\vec{x};\vec{a}) \leq max(\vec{a}) + p_f(n,\vec{x})$$

Hence for $z = n + 1$ we get

$$\begin{aligned}
f(n+1,\vec{x};\vec{a}) &= h(z,\vec{x};\vec{a},f(z,\vec{x};\vec{a})) \\
&\leq max(\vec{a},f(n,\vec{x};\vec{a}))+p_h(n,\vec{x}) \\
&\leq max(\vec{a},max(\vec{(a)}+p_f(n,\vec{x})+p_h(n,\vec{x}) \quad \text{by ind. hyp.} \\
&\leq max(\vec{a})+p_f(n,\vec{x})+p_h(\vec{x}) \\
&\leq max(\vec{a})+(p_g(\vec{x})+p_h(1,\vec{x})+...+p_h(n,\vec{x}))+p_h(n,\vec{x}) \\
&\leq max(\vec{a})+p_g(\vec{x})+n\cdot p_h(n,\vec{x})+p_h(n,\vec{x}) \\
&\leq max(\vec{a})+p_g(\vec{x})+(n+1)\cdot p_h(n,\vec{x}) \\
&\leq max(\vec{a})+p_f(n+1,\vec{x})
\end{aligned}$$

Theorem 1.3.8 (Handley-Wainer).

$$BCu_{normal} = \mathcal{E}^2$$

Proof. See Handley and Wainer [11].

Chapter 2

$PA(;)$, its Fragment $\Sigma_1(;) - IND$, and $\mathcal{E}^2$

In this chapter we shall obtain a new logical theory, two-sorted Peano Arithmetic $PA(;)$, from the formal proof system of single-sorted Peano Arithmetic, PA, by using the Bellantoni-Cook variable separation. In $PA(;)$, quantification is exclusively applied over output variables whereas induction is formulated only over input variables, which remain free throughout. We then obtain the theory of $\Sigma_1(;) - IND$ from $PA(;)$ by restricting the induction rule to apply only to $\Sigma_1(;)$-formulas, and get the result that the provably terminating functions of $\Sigma_1(;) - IND$ are exactly the functions of BCu.

2.1 $PA(;)$

In this section the notion of the Bellantoni-Cook variable separation is applied to the standard formulation of the Peano Arithmetic to obtain a new logical theory of two-sorted Peano Arithmetic, $PA(;)$. Here the semi-colon is not part of the official syntax, we still use it to clarify the separation of the input variables from the output variables.

Definition 2.1.1 (Variables). The variables in the language of $PA(;)$ are distinguished into two-sorts as in BCu, both intended to range over to natural numbers.

- The *input* variables which are called *I-variables* denoted by $x, y, z, ...$ with subscript.
- The *output* variables which are called *O-variables* denoted by $a, b, c, ...$ with subscript.

Definition 2.1.2 (Terms). The *basic terms* are defined by

(a) 0 is a basic term.

(b) Either sort of variable is a basic term.

(c) If t is a basic term, then St and Pt are basic terms.

Where S and P are the *successor* and *predecessor* function-symbols. Note that we will use Sx and $x + 1$ interchangeable.

More general *terms* denoted by $s, t, r, ...$ are constructed from basic terms with use of function symbols, $f, g, h, ...$, standing for (primitive) recursive functions, formulated over either sort of variables. These are given by their primitive recursive definitions which will be assumed as axioms.

Definition 2.1.3 (Formulas). $PA(;)$ is formulated here classically in the style of Tait [27] with finite sets of formulas for sequences.

Atomic formulas are expressions between terms with respect to the basic relations and exist in complementary pair, $R(s,t)$ and $\bar{R}(s,t)$ for any terms s and t. For example, if u and v are either sort, $u = v$ and $u < v$ are the complement of $u \neq v$ and $u \not< v$ respectively.

More complex *formulas* $A, B, C, ...$ are built out of atomic formulas R and their complements $\bar{R}$ by applying the logical connectives; $\vee$ (or), $\wedge$ (and), $\exists$ (some) and $\forall$ (all). It is sometimes convenient to display the free variables occurring in formula A by writing $A(x_1, ..., x_n; a_1, ..., a_m)$. The Γ denotes finite sets of formulas, $\{A_1, ..., A_n\}$,

the intending meaning being A_1 or A_2 or ... or A_n. Furthermore Γ, A denotes $\Gamma \cup \{A\}$. Hence the *judgements* will be of the form

$$PA(;) \vdash \Gamma, A(x_1, ..., x_n; a_1, ..., a_m)$$

Note that *negation* $\neg$ and *implication* $\rightarrow$ are not included as basic logical symbols. The negation is defined by De Morgan's Laws:

$$\begin{aligned} \neg R &\equiv \bar{R} \\ \neg \bar{R} &\equiv R \\ \neg(A \vee B) &\equiv \neg A \wedge \neg B \\ \neg(A \wedge B) &\equiv \neg A \vee \neg B \\ \neg(\exists a A(;a)) &\equiv \forall a \neg A(;a) \\ \neg(\forall a A(;a)) &\equiv \exists a \neg A(;a) \end{aligned}$$

The implication is defined by

$$A \rightarrow B \equiv \neg A \vee B$$

The reason for presenting logic in this way is merely one of choice, but it makes it possible to exploit the duality between $\vee$ and $\wedge$, and between $\exists$ and $\forall$ for the purposes of simple cut-reduction.

The "$\equiv$" is the *syntactic identity* of terms.

Definition 2.1.4 (Axioms and Rules). For any Γ the arithmetical axioms and logical rules are highlighted in the following list.

Arithmetical Axioms. The substitution-instances (all terms for variables) of

(a) $\vdash \Gamma,\ u = v,\ u \neq v$

(b) $\vdash \Gamma,\ u < v,\ u \not< v$

(c) $\vdash \Gamma,\ Su \neq 0$

(d) $\vdash \Gamma,\ Su \neq Sv,\ u = v$

(e) $\vdash \Gamma,\ P0 = 0$

(f) $\vdash \Gamma,\ u = 0,\ S(Pu) = u$

(g) $\vdash \Gamma,\ P(Su) = u$

(h) $\vdash \Gamma,\ u = u$

(i) $\vdash \Gamma,\ u \neq v,\ v = u$

(j) $\vdash \Gamma,\ u \neq v,\ v \neq w,\ u = w$

(k) $\vdash \Gamma,\ u \not< 0$

(l) $\vdash \Gamma,\ u < Su$

(m) $\vdash \Gamma,\ u \not< Sv,\ u < v,\ u = v$

(n) $\vdash \Gamma,\ u \not< u$

(o) $\vdash \Gamma,\ u \not< v,\ v \not< w,\ u < w$

(p) $\vdash \Gamma,\ u < v,\ u = v,\ v < u$

(q) $\vdash \Gamma,\ u \neq v,\ \neg A(u),\ A(v)$ with A atomic.

Where to simplify the list, both variables safe and normal are represented by u, v, w, and $A(u)$ means appropriate $A(u;)$ or $A(;u)$ in arithmetical axioms. From (q) one can easily derive $\Gamma,\ u \neq v,\ \neg A(u),\ A(v)$ for all formulas A.

Recursion Axioms. For each primitive recursive function f, defined over input or output variables, we add its defining equations as new axioms. These new primitive recursive function-symbols can be used to generate arbitrary terms in the normal way. But these are not basic terms. For each defining equation $f(\vec{x};\vec{a}) = t$, the corresponding axiom is,

$$\vdash \Gamma, f(\vec{x};\vec{a}) = t$$

Further Arithmetical Axioms are added, expressing the associativity, commutativity and distributivity of addition, multiplication, and the usual index laws of exponentiation.

Logical Rules.

$(\vee)$ $\dfrac{\vdash \Gamma, A_i}{\vdash \Gamma, (A_0 \vee A_1)}$ where $i = 0$ or 1.

$(\wedge)$ $\dfrac{\vdash \Gamma, A_0 \qquad \vdash \Gamma, A_1}{\Gamma, (A_0 \wedge A_1)}$

$(\exists)$ $\dfrac{\vdash \Gamma, A(t)}{\vdash \Gamma, \exists a A(;a)}$ where t is a basic term.

$(\forall)$ $\dfrac{\vdash \Gamma, A(;b)}{\vdash \Gamma, \forall a A(;a)}$ where b is not free in Γ.

(Cut) $\dfrac{\vdash \Gamma, A \qquad \vdash \Gamma, \neg A}{\vdash \Gamma}$ where A is the "cut formula".

(Ind) $\dfrac{\vdash \Gamma, A(0;) \qquad \vdash \Gamma, \neg A(x;), A(x+1;)}{\vdash \Gamma, A(t;)}$ where x is not free in Γ.

Where in all the rules the active variables are the output variables, except in the induction rule. In the induction rule, A is said to be the induction formula and t *is any basic term over an input variable.* Thus if $t \equiv x$ the logic prevent us from subsequently quantifying over x, because it is not output, but we may quentify existentially. We have to point out that

- *Quantification* is exclusively over output variables.
- *Induction* is formulated over input variables.

We are allowed to introduce any function or term, but writing the defining schemes down is not enough. To use the function, we must first prove that it terminates.

Definition 2.1.5 (Defined Term). Apart from the variable separation another major difference between this arithmetic and the more usual formulations of *single sorted* Peano Arithmetic is the non-standard existential rule. Whereas in PA we allow any term to act as a witness for an existential rule, (similar for the conclusion of an induction), here we restrict our witnesses to be only *basic terms*. On the other

hand, we can obtain such a rule for more general witnessing terms, the only proviso being that we must first prove

$$\vdash \exists b(t = b)$$

In such circumstances we call such a term *defined* and often we will use the shorthand

$$t\downarrow$$

Remark 2.1.6. If t is a basic term, then we can prove $t\downarrow$ immediately by applying the $\exists$-rule to the axiom $t = t$. Furthermore, from the equality axiom, with b a new variable not in t,

$$t \neq b, \neg A(t), A(b)$$

we easily obtain (since the term b is basic)

$$\neg(t\downarrow), \neg A(t), \exists b A(b)$$

In other words we can only use provably-defined terms to witness existential formulas, that is

$$\frac{\Gamma, t\downarrow \qquad \Gamma, A(t)}{\Gamma, \exists b A(b)}$$

Lemma 2.1.7 ($\forall$-inversion). *Assume that* $\vdash \Gamma, \forall a A(x;a)$ *where* $\forall a A(x;a)$ *is nowhere used as an induction formula. Then* $\vdash \Gamma, A(x;b)$, *and the height of the new proof is not increased.*

Proof. By induction on the height of the given $PA(;)$-derivation of $\Gamma, \forall a A(;a)$, according to the last rule which is applied:

Case 1. If $\Gamma, \forall a A(;a)$ is an axiom, so is $\Gamma, A(;b)$.

Case 2. If last rule applied $(\vee)$, $(\wedge)$, $(\exists)$ and (Cut), then $\forall a A(;a)$ is side formula in premises. For instance

$$\frac{\Gamma, \forall a A(;a), B, C}{\Gamma, \forall a A(;a), (B \vee C)}$$

We can use the induction hypothesis to replace $\Gamma, \forall a A(;a)$ by $\Gamma, A(;b)$ in the premises. Then re-apply that rule

$$\frac{\Gamma, A(;b), B, C}{\Gamma, A(;b), (B \vee C)}$$

Case 3. If last rule applied ($\forall$), and $\forall a A(;a)$ is a side formula in premises:

$$\frac{\forall a A(;a), \Gamma, B(;c)}{\forall a A(;a), \Gamma, \forall a B(;a)}$$

We can use the induction hypothesis to replace $\Gamma, \forall a A(;a)$ by $\Gamma, A(;b)$ in the premises ($c \neq b$, if $c = b$ we can change it). Then re-apply that rule

$$\frac{A(;b), \Gamma, B(;c)}{A(;b), \Gamma, \forall a B(;a)}$$

Case 4. If last rule applied ($\forall$), $\forall a A(;a)$ is the main formula in premise.

$$\frac{\Gamma, A(;c)}{\Gamma, \forall a A(;a)}$$

we can change variable c to b, then $\Gamma, A(;b)$.

Case 5. If last rule applied is (Ind), then $\forall a A(x;a)$ will only occur as a side fromula, so we can preceed as in *Case* 2.

2.2 $\Sigma_1(;) - IND$

In this section, we obtain the two-sorted fragment $\Sigma_1(;) - IND$ from $PA(;)$ by restricting induction formulas to being $\Sigma_1(;)$-formulas.

Definition 2.2.1 ($\Sigma_1(;)$-formula). Two-sorted $\Sigma_1(;)$-formulas $A(\vec{x};\vec{b})$ will be of the form

$$\exists \vec{a} A'(\vec{x};\vec{a},\vec{b})$$

where A' is a quantifier free formula.

Definition 2.2.2 (Truth). To say that a $\Sigma_1(;)$-formula $\exists \vec{a} A'(\vec{x};\vec{a},\vec{b})$ is *true*, means that there are selection functions $\vec{f}(\vec{x};\vec{b}) = f_1(\vec{x};\vec{b}), ..., f_m(\vec{x};\vec{b})$ which satisfy the formula in the sense that for all $\vec{x}$ and $\vec{b}$ the formula $A'(\vec{x};\vec{f}(\vec{x};\vec{b}),\vec{b})$ is true in the standard model. We often denote this as

$$\vec{f}(\vec{x};\vec{b}) \vDash \exists \vec{a} A'(\vec{x};\vec{a},\vec{b}) \qquad \text{or} \qquad \vDash A'(\vec{x};\vec{f}(\vec{x};\vec{b}),\vec{b})$$

If $\Gamma(\vec{x};\vec{b}) = \{A_1(\vec{x},\vec{b}), ..., A_n(\vec{x},\vec{b})\}$ and $\vec{g}(\vec{x};\vec{b})$ is any sequence of functions (all with arguments $\vec{x}$, $\vec{b}$) then we write

$$\vec{g} \vDash \Gamma$$

to mean that for arbitrary, fixed value of $\vec{x}$ and $\vec{b}$, there is an i such that

$$\vec{f}(\vec{x};\vec{b}) \vDash A_i(\vec{x};\vec{b})$$

for some subsequence $\vec{f}$ of functions from the sequence $\vec{g}$.

Definition 2.2.3 (Provably Recursive). Let a function f be defined by a primitive recursive program. Then we say that it is *provably recursive* (*provably terminating*) in a two-sorted theory, if for each function g introduced in the program with variables of the appropriate sorts we have

$$\vdash \exists c.g(\vec{x};\vec{a}) = c$$

In particular, if f is the final function defined by the program, with input variables only, we have also

$$\vdash \exists c.f(\vec{x}) = c$$

We will actually be interested in extracting those functions f which have input variables only.

We ask how is a proof of $\exists \vec{a} A'(\vec{x};\vec{a},\vec{b})$ related to the complexity of the selection function $\vec{f}(\vec{x},\vec{b})$? It is an old and classical result of logic which was given by Parsons [19], Mints [17] and others that the $PRIM$ functions are exactly those which are provably recursive in the fragment $\Sigma_1 - IND$ of PA. Below we investigate the two-sorted analogue of this result.

2.3 Provably Recursive Functions of $\Sigma_1(;) - IND$ same as BCu

In this section, we want to prove that if a two-sorted function $f(x;a)$ is provably terminating in $\Sigma_1(;) - IND$, then f is definable in BCu, and vice versa. It gives us

that the provably recursive functions of $\Sigma_1(;) - IND$ are the same as the function of $\mathcal{E}^2$ since $BCu_{normal} = \mathcal{E}^2$.

Theorem 2.3.1. *Assume that* $\Sigma_1(;) - IND \vdash \Gamma$ *where* Γ *is a set of* $\Sigma_1(;)$*-formulas. Then there are BCu functions* $\vec{f}(\vec{x};\vec{b})$ *which satisfies* Γ*, i.e.,*

$$\vec{f} \vDash \Gamma$$

Proof. The lemma says that for every $\Sigma_1(;)$-formula provable in the theory of $\Sigma_1(;) - IND$, there are BCu functions $\vec{f}(\vec{x};\vec{b})$ such that $A'(\vec{x};\vec{f}(\vec{x};\vec{b}),\vec{b})$ is true for every $\vec{x}$ and $\vec{b}$.

Let $\Gamma = \{\ \exists \vec{a_i} A_i'(\vec{x};\vec{a_i},\vec{b})\ \}_{i=1,\ldots,k}$ be the set of $\Sigma_1(;)$-formulas. Then we proceed by induction on the height of the derivation in $\Sigma_1(;) - IND$.

Note that a standard cut elimination argument shows that any $\Sigma_1(;) - IND$ derivation can be transformed into one with only $\Sigma_1(;)$ cut formulas. The idea is to prove that given derivations of $\Gamma, \neg C$ and Γ, C where C is either $D \vee E$ or $\exists a D(a)$ and Γ, C does not arise by an induction on formula C, we can obtain a derivation of Γ wherein cut formulas are less complex than C. The proof goes by induction on the height of derivation $\vdash \Gamma, C$. For example, suppose $C \equiv \exists a D(a)$ and by the $\exists$-rule

$$\frac{\Gamma, D(t)}{\Gamma, D}$$

where t is a basic term. By the $\forall$-inversion we obtain $\Gamma, \neg D(t)$ from $\Gamma, \forall a \neg D(a)$. Then by the cut-rule

$$\frac{\Gamma, \neg D(t) \qquad \Gamma, D(t)}{\Gamma}$$

with cut-formula D less complex than C. By repeating this process we can transform the original proof into one in which cut formulas are at worst Σ_1 and Π_1. We now return to the proof.

Case (Axs). The axioms all contain true quantifier-free formulas, so any BCu function f will do for them.

Case $(\vee)$*,*$(\wedge)$*.* The $(\vee)$ and $(\wedge)$-rules respectively

$$\frac{\Gamma, A_0}{\Gamma, (A_0 \vee A_1)} \qquad \frac{\Gamma, A_0 \qquad \Gamma, A_1}{\Gamma, (A_0 \wedge A_1)}$$

They are trivial since Γ contains only $\Sigma_1(;)$-formulas and A_0 and A_1 must be quantifier free otherwise $A_0 \vee A_1$ and $A_0 \wedge A_1$ can not be $\Sigma_1(;)$-formula. We are therefore concerned merely that their truth is preserved.

Case ($\exists$). The ($\exists$)-rule

$$\frac{\Gamma, A(\vec{x}; t, \vec{b})}{\Gamma, \exists d A(\vec{x}; d, \vec{b})} \qquad t \text{ is a basic term}$$

where $A(\vec{x}; t, \vec{b}) \equiv \exists \vec{a} A'(\vec{x}; \vec{a}, t, \vec{b})$ and then $\exists d A(\vec{x}; d, \vec{b}) \equiv \exists d \exists \vec{a} A'(\vec{x}; \vec{a}, d, \vec{b})$. Then the ($\exists$)-rule will be

$$\frac{\Gamma, \exists \vec{a} A'(\vec{x}; \vec{a}, t, \vec{b})}{\Gamma, \exists d \exists \vec{a} A'(\vec{x}; \vec{a}, d, \vec{b})}$$

Applying the induction hypothesis we have BCu functions $\vec{f}(\vec{x}; \vec{b})$ such that for all $\vec{x}$ and $\vec{b}$

$$\vDash \Gamma, A'(\vec{x}; \vec{f}(\vec{x}; \vec{b}), t, \vec{b})$$

where t is a basic term, $\exists d(t = d)$ (see Remark 2.1.6), so we have a BCu function $g(\vec{x}; \vec{b})$ such that for all $\vec{x}, \vec{b}$

$$g(\vec{x}; \vec{b}) = t$$

Now we define BCu functions $\vec{h}(\vec{x}; \vec{b})$ such that for all $\vec{x}, \vec{b}$

$$\vDash \Gamma, A'(\vec{x}; \vec{h}(\vec{x}; \vec{b}), h_{k+1}(\vec{x}; \vec{b}), \vec{b})$$

where $h_i(\vec{x}; \vec{b}) = f_i(\vec{x}; \vec{b})$, i=1,...,k, and $h_{k+1}(\vec{x}; \vec{b}) = g(\vec{x}; \vec{b})$. This completes the proof of this case.

Case ($\forall$). The ($\forall$)-rule

$$\frac{\Gamma, A(\vec{x}; b)}{\Gamma, \forall d A(\vec{x}; d)} \qquad b \text{ is not free in } \Gamma$$

which is not applicable since in conclusion $\forall d A(\vec{x}; d)$ can not be $\Sigma_1(;)$-formula because it contains universal quantifier $\forall d$.

Case (Cut). The (Cut)-rule

$$\frac{\Gamma, A \qquad \Gamma, \neg A}{\Gamma}$$

where we can crucially assume that the (Cut)-formula A is in $\Sigma_1(;)$-form, say $A \equiv \exists \vec{a} A'(\vec{x}; \vec{a}, \vec{b})$, then $\neg A \equiv \forall \vec{a} \neg A'(\vec{x}; \vec{a}, \vec{b})$. But the right premise now contains a $\forall \vec{a}$ which must be removed in order to continue the proof. $\forall$-inversion allows the proof of $\Gamma, \forall \vec{a} \neg A'(\vec{x}; \vec{a}, \vec{b})$ to be replaced by a proof of $\Gamma, \neg A'(\vec{x}; \vec{c}, \vec{b})$ but containing new variable $\vec{c}$. So the application of (Cut) will be

$$\frac{\Gamma, \exists \vec{a} A'(\vec{x}; \vec{a}, \vec{b}) \qquad \Gamma, \neg A'(\vec{x}; \vec{c}, \vec{b})}{\Gamma}$$

Applying the induction hypothesis to left premise, we have BCu functions $\vec{f}_1(\vec{x}; \vec{b})$, ..., $\vec{f}_k(\vec{x}; \vec{b})$, $\vec{f}_{k+1}(\vec{x}; \vec{b})$ such that

$$\vDash \{ A'_i(\vec{x}; \vec{f}_i(\vec{x}; \vec{b}), \vec{b}) \}_{i=1,...,k}, A'(\vec{x}; \vec{f}_{k+1}(\vec{x}; \vec{b}), \vec{b}) \tag{2.1}$$

and from right premise, we have BCu functions $\vec{g}_1(\vec{x}; \vec{c}, \vec{b}), ..., \vec{g}_l(\vec{x}; \vec{c}, \vec{b})$ such that

$$\vDash \{ A'_i(\vec{x}; \vec{g}_i(\vec{x}; \vec{c}, \vec{b}), \vec{b}) \}_{i=1,...,l}, \neg A'(\vec{x}; \vec{c}, \vec{b}) \tag{2.2}$$

Now we add to the sequence $\vec{f}_1, ..., \vec{f}_k$ new BCu functions $\vec{h}_1(\vec{x}; \vec{b}), ..., \vec{h}_l(\vec{x}; \vec{b})$ defined by

$$\vec{h}_i(\vec{x}; \vec{b}) := \vec{g}_i(\vec{x}; \vec{f}_{k+1}(\vec{x}; \vec{b}), \vec{b})$$

such that if $\vec{f}_1, ..., \vec{f}_k$ fail to satisfy Γ then $\vec{h}_1, ..., \vec{h}_l$ succeed in satisfying Γ, thus

$$\vDash \{ A'_i(\vec{x}; \vec{h}_i(\vec{x}; \vec{b}), \vec{b}) \}_{i=1,...,l}$$

Case (Ind). The (*Ind*)-rule

$$\frac{\Gamma, A(0, \vec{x}; \vec{b}) \qquad \Gamma, \neg A(z, \vec{x}; \vec{b}), A(z+1, \vec{x}; \vec{b})}{\Gamma, A(y, \vec{x}; \vec{b})}$$

where z not free in Γ. The A is in $\Sigma_1(;)$-form, $A(z, \vec{x}; \vec{b}) \equiv \exists \vec{a} A'(z, \vec{x}; \vec{a}, \vec{b})$ and $\neg A(z, \vec{x}; \vec{b}) \equiv \forall \vec{a} \neg A'(z, \vec{x}; \vec{a}, \vec{b})$. In order to continue the proof $\forall \vec{a}$ must be

removed. $\forall$-inversion allows the formula to be replaced by $\neg A'(z, \vec{x}; \vec{c}, \vec{b})$ but containing new variables $\vec{c}$. The (Ind)-rule becomes

$$\frac{\exists \vec{a} A'(0, \vec{x}; \vec{a}, \vec{b}) \qquad \neg A'(z, \vec{x}; \vec{c}, \vec{b}), \exists \vec{a} A'(z+1, \vec{x}; \vec{a}, \vec{b})}{\exists \vec{a} A'(y, \vec{x}; \vec{a}, \vec{b})}$$

where we ignored the Γ for the sake of brevity since it does not change the proof. Applying the induction hypothesis to left premise, we have BCu functions $\vec{f}(\vec{x}; \vec{b})$, such that for all $\vec{x}, \vec{b}, \vec{c}$

$$\vDash A'(0, \vec{x}; \vec{f}(\vec{x}; \vec{b}), \vec{b}) \tag{2.3}$$

and applying the induction hypothesis to right premise, we have BCu functions $\vec{g}(\vec{x}; \vec{c}, \vec{b})$, such that for $\vec{x}, \vec{b}$

$$\vDash \neg A'(z, \vec{x}; \vec{c}, \vec{b}), A'(z+1, \vec{x}; \vec{g}(\vec{x}; \vec{c}, \vec{b}), \vec{b}) \tag{2.4}$$

Now we define BCu functions $\vec{h}(z, \vec{x}; \vec{b})$ by the primitive recursion from $\vec{f}$ and $\vec{g}$ by

$$\begin{cases} \vec{h}(0, \vec{x}; \vec{b}) &= \vec{f}(\vec{x}; \vec{b}) \\ \vec{h}(z+1, \vec{x}; \vec{b}) &= \vec{g}(\vec{x}; \vec{h}(z, \vec{x}; \vec{b}), \vec{b}) \end{cases}$$

such that $\vDash A'(y, \vec{x}; \vec{h}(y, \vec{x}; \vec{b}), \vec{b})$ for every y. We can prove this by induction on y:

– *Base case*, $y = 0$ is (2.3).

– *Induction case*, for y we suppose that

$$\vDash A'(y, \vec{x}; \vec{h}(y, \vec{x}; \vec{b}), \vec{b})$$

Then by butting $\vec{c} = \vec{h}(y, \vec{x}; \vec{b})$ in (2.4) we obtain

$$\vDash \neg A'(y, \vec{x}; \vec{h}(y, \vec{x}; \vec{b}), \vec{b}), A'(y+1, \vec{x}; \vec{h}(y+1, \vec{x}; \vec{b}), \vec{b})$$

because $\vec{h}(y+1, \vec{x}; \vec{b}) = \vec{g}(\vec{x}; \vec{h}(y, \vec{x}; \vec{b}), \vec{b})$ by definition. Thus we have

$$\vDash A'(y+1, \vec{x}; \vec{h}(y+1, \vec{x}; \vec{b}), \vec{b})$$

as required.

Corollary 2.3.2. *All provably recursive functions of* $\Sigma_1(;) - IND$, *with input arguments only, are BCu functions.*

Proof. We mentioned in the Remark 1.3.5 that in the *more general (safe) composition* of Bellantoni and Cook, substitution of functions with input variables, at input places is allowed.

Suppose $\Sigma_1(;) - IND \vdash \exists b(f(\vec{x}) = b)$ and similarly for all other functions used in the given primitive recursive program defining f.

By the Theorem 2.3.1 we have BCu-definable functions $h_1(\vec{x}), ..., h_n(\vec{x})$ such that for all $\vec{x}$,

$$h_1(\vec{x}) \models \exists b(f(\vec{x}) = b) \quad \text{or} \quad ... \quad \text{or} \quad h_n(\vec{x}) \models \exists b(f(\vec{x}) = b)$$

that is,

$$h_1(\vec{x}) = f(\vec{x}) \quad \text{or} \quad ... \quad \text{or} \quad h_n(\vec{x}) = f(\vec{x})$$

Now assume, inductively, that we already have established a BCu definition of each of the functions $\vec{g}$ used in the given program before the final line defining f itself.

There are two cases to consider (the initial cases being easy):

Case (Composition). Suppose $f(\vec{x})$ is defined explicitly from g_0 and g_1 by

$$f(\vec{x}) = g_0(\vec{x}, g_1(\vec{x}))$$

By the induction hypothesis, both g_0 and g_1 are BCu-definable. Therefore, immediately, so is f, provided we use the more general (safe) composition scheme of Bellantoni-Cook (in the case where g_0 might be previously defined by recursion on its last argument).

Case (Recursion). Suppose $f(z, \vec{x})$ is defined explicitly from g_0 and g_1 by

$$\begin{cases} f(0, \vec{x}) &= g_0(\vec{x}) \\ f(z+1, \vec{x}) &= g_1(z, \vec{x}, f(z, \vec{x})) \end{cases}$$

Again by the induction hypothesis, g_0 and g_1 are BCu-definable, but g_1 is previously defined by recursion on its last argument which must be "*input*".

In this case we again use the more general composition scheme to define in BCu,

$$g_{1i}(z,\vec{x}) = g_1(z,\vec{x},h_i(z,\vec{x})) \qquad \text{for each} \quad i = 1,...,n$$

Then we can give the following BCu definition for f:

$$\begin{cases} f(0,\vec{x}) &= g_0(\vec{x}) \\ f(z+1,\vec{x}) &= F(z,\vec{x},f(z,\vec{x})) \end{cases}$$

where

$$F(z,\vec{x},a) = \begin{cases} g_{11}(z,\vec{x}) & \text{if} \quad a = h_1(z,\vec{x}) \\ g_{12}(z,\vec{x}) & \text{if} \quad a = h_2(z,\vec{x}) \\ \vdots & \\ g_{1n}(z,\vec{x}) & \text{if} \quad a = h_n(z,\vec{x}) \end{cases}$$

Hence F is definable in BCu by using definition by cases, where the cases are determined by the characteristic function of equality $a = y$ where y is an input variable (later substituted by $h_i(z,\vec{x})$). But the characteristic function of $a = y$ is just $C(a \dot{-} y, y \dot{-} a, 1)$ and this is BCu-definable because both of the functions $a \dot{-} y$ and $y \dot{-} a$ are easily defined by BCu-recursion over y,

$$\begin{cases} a \dot{-} 0 &= a \\ a \dot{-} (y+1) &= P(a \dot{-} y) \end{cases}$$

and

$$\begin{cases} o \dot{-} a &= 0 \\ (y+1) \dot{-} a &= C(a \dot{-} y, S(y \dot{-} a), 0) \end{cases}$$

Thus the above safe recursion now gives a BCu definition of f.

Theorem 2.3.3. *If $f(\vec{x};\vec{b})$ is a BCu-definable function, then it is provably recursive in $\Sigma_1(;) - IND$, i.e.,*

$$\Sigma_1(;) - IND \vdash \exists a(f(\vec{x};\vec{b}) = a)$$

Proof. The lemma says that for every BCu function $f(\vec{x};\vec{b})$, there is a $\Sigma_1(;)$-formula $\exists \vec{a} A'(\vec{x};\vec{a},\vec{b})$ in theory of $\Sigma_1(;) - IND$ such that $A'(\vec{x};f(\vec{x};\vec{b}),\vec{b})$ is true for every x and b.

It can be proved by induction on the length of the BCu definition:

Case (Initial Functions). Suppose f is one of the initial functions respectively,

(a) $\Sigma_1(;) - IND \vdash \exists a(Z(\vec{x};\vec{b}) = a)$ with basic witness $a = 0$,

(b) $\Sigma_1(;) - IND \vdash \exists a(S(\vec{x};b) = a)$ with basic witness $a = S(;b)$,

(c) $\Sigma_1(;) - IND \vdash \exists a(P(\vec{x};b) = a)$ with basic witness $a = P(;b)$,

(d) $\Sigma_1(;) - IND \vdash \exists a(U_i(\vec{x};\vec{b}) = a)$ with basic witness $a = b_i$,

(e) $\Sigma_1(;) - IND \vdash \exists a(C(\vec{x};b,c,d) = a)$ with witness

$$a = \begin{cases} c & if \quad b = 0 \\ d & if \quad b \neq 0 \end{cases}$$

Since $b = 0 \vee b = S(P(b))$ is an axiom, and $C(\vec{x};0,c,d) = c$ and $C(\vec{x};S(P(b)),c,d) = d$, so by logic

$$\vdash \exists a(C(\vec{x};b,c,d) = a)$$

Case (BCu-Composition). Suppose $f, g, h \in BCu$ and f is defined explicitly from g and h by

$$f(\vec{x};b) = g(\vec{x};h(\vec{x};b)) \tag{2.5}$$

where g and h are already assumed that

$$\Sigma_1(;) - IND \vdash \exists c(h(\vec{x};b) = c)$$
$$\Sigma_1(;) - IND \vdash \exists a(g(\vec{x};c) = a)$$

We have to show that $\Sigma_1(;) - IND \vdash \exists a(f(\vec{x};\vec{b}) = a)$. We start with the arithmatical axiom (q) which is $\vdash \Gamma, u \neq v, \neg A(u), A(v)$. If we choose c and $h(\vec{x};b)$ instead of u, v, then for g we get $\neg\exists a(g(\vec{x};c) = a)$ and $\exists a(g(\vec{x};h(\vec{x};b)) = a)$ instead of $\neg A(u)$ and $A(v)$. So the axiom (q) will be

$$\vdash h(\vec{x};b) \neq c, \neg\exists a(g(\vec{x};c) = a), \exists a(g(\vec{x};h(\vec{x};b)) = a)$$

by using (2.5) the axiom

$$\vdash h(\vec{x};b) \neq c, \neg\exists a(g(\vec{x};c) = a), \exists a(f(\vec{x};b) = a)$$

hence

$$\dfrac{\dfrac{\exists a(g(\vec{x};c)=a) \qquad h(\vec{x};b)\neq c, \neg\exists a(g(\vec{x};c)=a), \exists a(f(\vec{x};b)=a)}{h(\vec{x};b)\neq c, \exists a(f(\vec{x};b)=a)}\ (Cut)}{\dfrac{\forall c\neg(h(\vec{x};b)=c), \exists a(f(\vec{x};b)=a)}{\exists a(f(\vec{x};b)=a)}\ (\forall)}\ (Cut)$$

where we used $(\forall)$ since c is not free in the side formula.

Case (BCu-Recursion). Suppose $f, g, h \in BCu$ and f is defined explicitly from g and h by

$$f(0,\vec{x};\vec{b}) = g(\vec{x};\vec{b}) \tag{2.6}$$

$$f(z+1,\vec{x};\vec{b}) = h(z,\vec{x};f(z,\vec{x};\vec{b}),\vec{b}) \tag{2.7}$$

where g and h are already assumed that

$$\Sigma_1(;) - IND \vdash \exists a(g(\vec{x};\vec{b})=a)$$
$$\Sigma_1(;) - IND \vdash \exists a(h(\vec{x};c,\vec{b})=a)$$

First using (2.6) and first assumption we get

$$\vdash \exists a(f(0,\vec{x};\vec{b})=a)$$

Now we have to show that $\Sigma_1(;)-IND \vdash \exists a(f(z,\vec{x};\vec{b})=a)$. We start with the arithmetical axiom (q) as in previous case. If we choose c and $f(z,\vec{x};\vec{b})$ instead of u, v, then for h we get $\neg\exists a(h(\vec{x};c,\vec{b})=a)$ and $\exists a(h(\vec{x};f(z,\vec{x};\vec{b}),\vec{b})=a)$ instead of $\neg A(u)$ and $A(v)$. So the axiom (q) will be

$$\vdash f(z,\vec{x};\vec{b})\neq c, \neg\exists a(h(\vec{x};c,\vec{b})=a), \exists a(h(\vec{x};f(z,\vec{x};\vec{b}),\vec{b})=a)$$

by using (2.7) the axiom

$$\vdash f(z,\vec{x};\vec{b})\neq c, \neg\exists a(h(\vec{x};c,\vec{b})=a), \exists a(f(z+1,\vec{x};\vec{b})=a)$$

hence

$$\dfrac{\dfrac{\exists a(h(\vec{x};c,\vec{b})=a) \quad f(z,\vec{x};\vec{b})\neq c, \neg\exists a(h(\vec{x};c,\vec{b})=a), \exists a(f(z+1,\vec{x};\vec{b})=a)}{f(z,\vec{x};\vec{b})\neq c, \exists a(f(z+1,\vec{x};\vec{b})=a)}\ (\forall)}{\dfrac{\forall a\neg(f(z,\vec{x};\vec{b})=a), \exists a(f(z+1,\vec{x};\vec{b})=a)}{\exists a(f(0,\vec{x};\vec{b})=a) \quad \neg\exists a(f(z,\vec{x};\vec{b})=a), \exists a(f(z+1,\vec{x};\vec{b})=a)}\ (Ind)}\ (Cut)$$

$$\exists a(f(z,\vec{x};\vec{b})=a)$$

where we used $(\forall)$ since c is not free in the side formulas.

Theorem 2.3.4. *The functions of input arguments only which are provably recursive in $\Sigma_1(;) - IND$ are exactly the $\mathcal{E}^2$ functions.*

Proof. The one way the Corollary 2.3.2 shows that all provable terminating recursive functions of $\Sigma_1(;) - IND$ are BCu function. The other way the Theorem 2.3.3 shows that all BCu functions are provable terminating recursive functions of $\Sigma_1(;) - IND$. By Theorem 1.3.8 we have that $BCu_{normal} = \mathcal{E}^2$.

Chapter 3

The Functional Interpretation of $PA(;)$

The aim of this chapter is to develop a Gödel dialectica style interpretion of the full theory of $PA(;)$, using a BCu-version of higher type primitive recursive functionals. Our system of two-sorted higher-type Gödel primitive recursive functionals, is denoted $T(;)$, and the result is that every provably recursive function of $PA(;)$ is definable in $T(;)$. In the final section of this chapter we show explicitly that every elementary function is definable in $T(;)$. The converse, that every function definable in $T(;)$ is elementary, will be dealt with in the next chapter.

3.1 Gödel's System $T(;)$

The Gödel primitive recursive functionals were first given by Gödel in [9]. We now define two-sorted higher-type Gödel primitive recursive functionals by using the variable separation and in fact we follow closely the development by Shoenfield [23]. This two-sorted Gödel system is denoted by $T(;)$ and has two crucial distinctions

- The *composition* is exclusively over output variables.
- The *recursion* is formulated over input variables.

as in $PA(;)$.

Definition 3.1.1 (Types). The types fall into two classes; *input* and *output*. They are generated from two ground-types ι (input) and o (output) using "$\rightarrow$", according to the following inductive definition:

(a) o is an output type, and ι is the only input type.

(b) If τ and σ are output types, then $\sigma \rightarrow \tau$ and $\iota \rightarrow \tau$ are output types.

Definition 3.1.2 (Functionals). A functional of type $\sigma \rightarrow \tau$ is any mapping from the set of functionals of type σ into the set of functionals of type τ. The objects of type ι and o are natural numbers. These functionals constitute the *maximal* type-structure over $\mathbb{N}$, which means: each type σ will have a stock of variables $v_1 : \sigma, v_2 : \sigma, \ldots$ which are to be thought of as ranging over the set N_σ where N_0 or N_ι are sets of natural numbers, and $N_{\sigma\rightarrow\tau}$ is set of functions defined from N_σ to N_τ.

A *functional* will mean function of some type. We shall not always give the type of a functional explicitly, but it is to be understood that whatever *functionals* we write are properly typed, and context should make the typing clear.

We have three function constants:

1. *Successor* S of type $o \rightarrow o$

2. *Predecessor* P of type $o \rightarrow o$

3. *Choice function* C of type $o \rightarrow o \rightarrow o \rightarrow o$

which are defined by their intending meaning as in the definition of BCu in Chapter 1, Section 1.3.

Definition 3.1.3 (Terms). The *terms* of $T(;)$ fall into two classes:

• *I-terms.* The only terms of type ι are those built up from I-variables and the constant 0 by (repeated) application of $+1$ or $\dot{-}1$. For notational convenience we shall often denote an I-term simply x (although strictly-speaking it is an I-variable).

• *O-terms.* The output variables of type σ are called *O-variables*, denoted by u^σ, v^σ, ... with or without subscripts. The *O-terms*, denoted by r, s, t with or without subscripts, are defined by the generalised inductive definition:

(a) Every *O-variable* is an O-term.

(b) The *constants* $0,\ S,\ P,\ C$ are O-terms of type $o,\ o \to o,\ o \to o,\ o \to o \to o \to o$ respectively.

(c) If t_1 and t_2 are O-terms of type $\sigma \to \tau$ and τ respectively, then the *application* $t_1 t_2$ is an O-term of type τ.

(d) If t is an O-term of type τ all of whose variables occur among I-variables $x_1, ..., x_n$ and O-variables $v_1, ..., v_m$ of type $\sigma_1, ..., \sigma_m$, then we introduce a new constant O-term f of type $\iota \to ... \to \iota \to \sigma_1 \to ... \to \sigma_m \to \tau$ by composition

$$f x_1 ... x_n v_1 ... v_m = t$$

(e) If g and h are constant O-terms of type τ and $\iota \to \tau \to \tau$ respectively, then we introduce a new constant O-term f of type $\iota \to \tau$ by the *primitive recursion*

$$\begin{cases} f(0;\vec{v}) &= g(x;\vec{v}) \\ f(x+1;\vec{v}) &= h(x; f(x;\vec{v})) \end{cases}$$

Notations 3.1.4. Here and in the rest of the thesis, we shall adopt the following notations whenever the context makes the arguments clear.

1. *$T(;)$-terms* will be used for either *I-terms* or *O-terms* whenever there is no confusion.

2. Denote the type of a term t by writing $t : \sigma$ or sometimes t^σ.

3. Parentheses will be omitted from types and terms according to the usual conventions, by association to the right for types and association to the left for terms. For examples,

$$\begin{aligned} \rho \to \sigma \to \tau &= (\rho \to (\sigma \to \tau)) \\ tsr &= ((ts)r) \end{aligned}$$

4. We shall write $\sigma_1, ..., \sigma_n \to \tau$ for $\sigma_1 \to ... \to \sigma_n \to \tau$ which will also be abbreviated by $\vec{\sigma} \to \tau$. Hence $\vec{t} : \vec{\sigma}$ will mean $\vec{t} = t_1, ..., t_n$ are of types $\sigma_1, ..., \sigma_n$ respectively.

5. We shall omit to write the types with terms whenever the context makes the arguments clear. In such a case every term, of course, has an appropriate type. We shall do this just for the sake of brevity.

3.2 Two-sorted Generalised Formulas

The *two-sorted generalised formulas* are to be viewed as higher-type extensions of $\Sigma_1(;)$-formulas in $PA(;)$.

Definition 3.2.1 (Generalised Formulas). To each $PA(;)$-formula $A(\vec{x};\vec{a})$ is associated a *generalised formula*

$$A^D = \forall\vec{u}\exists\vec{v}A'(\vec{x};\vec{a},\vec{u},\vec{v})$$

defined below, where A' is quantifier free, and where $\vec{u},\vec{v}$ are now lists of (possibly) higher-type variables. That is, the variables range not just over the ground type of natural numbers, but over higher function types, so that an $\forall\exists$ quantifier prefix can be transformed into $\exists\forall$ by *skolemization* thus

$$\forall\vec{u}\exists\vec{v}A'(\vec{x};\vec{a},\vec{u},\vec{v}) \equiv \exists\vec{f}\forall\vec{u}A'(\vec{x};\vec{a},\vec{u},\vec{f})$$

Systematic application of this idea will then transform any aritmetical formula into a corresponding generalized formula.

Let $B(\vec{x};\vec{a})$, $C(\vec{x};\vec{a_1})$ be any arithmetical formulas with generalised formulas $B^D = \forall\vec{u}\exists\vec{v}B'(\vec{x};\vec{a},\vec{u},\vec{v})$, $C^D = \forall\vec{u_1}\exists\vec{v_1}C'(\vec{x};\vec{a_1},\vec{u_1},\vec{v_1})$ respectively. Then the definition of A^D is by induction over the logical structure of the $PA(;)$-formula A, as follows:

(a) A is atomic $\Rightarrow$ $A^D = A$

(b) $A = \neg B$ $\Rightarrow$ $A^D = \forall \vec{f} \exists \vec{u} \neg B'(\vec{x}; \vec{a}, \vec{u}, \vec{f}\vec{u})$

(c) $A = B \vee C$ $\Rightarrow$ $A^D = \forall \vec{u}\vec{u_1} \exists \vec{v}\vec{v_1}(B'(\vec{x}; \vec{a}, \vec{u}, \vec{v}) \vee C'(\vec{x}; \vec{a_1}, \vec{u_1}, \vec{v_1}))$

(d) $A = \forall b B$ $\Rightarrow$ $A^D = \forall b \vec{u} \exists \vec{v} B'(\vec{x}; \vec{a}, b, \vec{u}, \vec{v})$

Where in clause (b), $(\neg B)^D$ can be shown by using skolemization

$$\begin{aligned}(\neg B)^D &\equiv \neg B^D \\ &\equiv \neg(\forall \vec{u} \exists \vec{v} B'(\vec{x}; \vec{a}, \vec{u}, \vec{v})) \\ &\equiv \neg(\exists \vec{f} \forall \vec{u} B'(\vec{x}; \vec{a}, \vec{u}, \vec{f}\vec{u})) \\ &\equiv \forall \vec{f} \exists \vec{u} \neg B'(\vec{x}; \vec{a}, \vec{u}, \vec{f}\vec{u})\end{aligned}$$

where $\vec{f}$ is a sequence of new variables of the appropriate types and $\vec{f}\vec{u}$ denotes the finite lists $f_1\vec{u}, f_2\vec{u}, ..., f_k\vec{u}$ where k is the length of the list $\vec{v}$.

We therefore also can obtain $(B \to C)^D$ classically as

$$\begin{aligned}(B \to C)^D &\equiv (\neg B \vee C)^D \\ &\equiv (\neg B)^D \vee C^D \\ &\equiv \forall \vec{f} \exists \vec{u} \neg B'(\vec{x}; \vec{a}, \vec{u}, \vec{f}\vec{u}) \vee \forall \vec{u_1} \exists \vec{v_1} C'(\vec{x}; \vec{a_1}, \vec{u_1}, \vec{v_1}) \\ &\equiv \forall \vec{f} \forall \vec{u_1} \exists \vec{u}\vec{v_1}(\neg B'(\vec{x}; \vec{a}, \vec{u}, \vec{f}\vec{u}) \vee C'(\vec{x}; \vec{a_1}, \vec{u_1}, \vec{v_1})) \\ &\equiv \forall \vec{f} \forall \vec{u_1} \exists \vec{u}\vec{v_1}(B'(\vec{x}; \vec{a}, \vec{u}, \vec{f}\vec{u}) \to C'(\vec{x}; \vec{a_1}, \vec{u_1}, \vec{v_1}))\end{aligned}$$

Note that $\forall \vec{a}\vec{b}\vec{c}$ ... means $\forall \vec{a} \forall \vec{b} \forall \vec{c}$ ..., similarly $\exists \vec{a}\vec{b}\vec{c}$ ... means $\exists \vec{a} \exists \vec{b} \exists \vec{c}$ ... .

Definition 3.2.2 (Valid). We say that a generalised formula $\forall \vec{u} \exists \vec{v} A'(\vec{x}; \vec{a}, \vec{u}, \vec{v})$ is *valid*, if there are selection functionals $\vec{F}$ of the appropriate types such that for all $\vec{x}, \vec{a}$ and $\vec{u}$, the formula $A'(\vec{x}; \vec{a}, \vec{u}, \vec{v})$ is true for $\vec{v} = \vec{F}\vec{x}\vec{a}\vec{u}$, when interpreted in the maximal type-structure as explained in Definition 3.1.2.
It is *valid in* $T(;)$ if the selection functionals $\vec{F}$ can be chosen so as to be definable in $T(;)$. In this case we write

$$\vDash A'(\vec{x}; \vec{a}, \vec{u}, \vec{F}\vec{x}\vec{a}\vec{u})$$

3.3 $PA(;)$-functions Definable in $T(;)$

In this section we shall show that every provably recursive function of $PA(;)$ is definable in $T(;)$.

Theorem 3.3.1. *Assume that* $PA(;) \vdash A(\vec{x};\vec{a})$. *Then the generalised formula* $A^D = \forall\vec{u}\exists\vec{v}A'(\vec{x};\vec{a},\vec{u},\vec{v})$ *is weakly valid in* $T(;)$, *in the sense that there are sequences of functionals* $\vec{F}_1,...,\vec{F}_n$ *such that for all* $\vec{x},\vec{a},\vec{u}$ *there is an* i *with*

$$\vDash A'(\vec{x};\vec{a},\vec{u},\vec{F}_i\vec{x}\vec{a}\vec{u})$$

Corollary 3.3.2. *If* $f(\vec{x})$ *is a provably recursive function of* $PA(;)$, *then there is a sequence of* $T(;)$*-definable functions* $g_1,...,g_n$ *such that for every* $\vec{x}$, *there is an* i *with* $g_i(\vec{x})$ *satisfying the generalised formula of* $\exists b(f(\vec{x}) = b)$. *Hence for every* $\vec{x}$, *either*

$$f(\vec{x}) = g_1(\vec{x}), \quad or \quad ..., \quad or \quad f(\vec{x}) = g_n(\vec{x})$$

But then, by methods similar to the proof of Corollary 2.3.2, we obtain a definition of f *in* $T(;)$.

Proof of Theorem 3.3.1. Although for the purposes of cut-elimination it is useful to formalise the logic underlying $PA(;)$ in Tait-style, it is much more convenient here to follow the treatment given in Shoenfield's book [23], using a logic based on the symbols $\neg$, $\vee$, $\forall$, $\rightarrow$. The proof follows Shoenfield closely, but we consider just a few of the main cases in order to show the roles played by the two sorts of variables.

Note that it is because of contraction, $A \vee A \vdash A$, that we only obtain sequences of functionals one of which satisfies the required generalised formula.

Case (Substitution Axiom). Assume that A is a substitution axiom

$$\forall bB(\vec{x};\vec{a},b) \rightarrow B(\vec{x};\vec{a},t)$$

where t is a basic term in accordance with the quantifier rules of $PA(;)$. Let $B^D = \forall\vec{u}\exists\vec{v}B'(\vec{x};\vec{a},b,\vec{u},\vec{v})$. Then by using Definition 3.2.1 we have

$$\begin{aligned}(\neg\forall bB(\vec{x};\vec{a},b))^D &= \forall\vec{f}\exists b\vec{u_1}B'(\vec{x};\vec{a},b,\vec{u_1},\vec{f}b\vec{u_1})\\ (B(\vec{x};\vec{a},t))^D &= \forall\vec{u}\exists\vec{v}B'(\vec{x};\vec{a},t,\vec{u},\vec{v})\end{aligned}$$

Hence $(\forall \vec{b} B(\vec{x};\vec{a},b) \to B(\vec{x};\vec{a},t))^D$ will be

$$\forall \vec{f}\vec{u} \exists b \vec{u_1} \vec{v} (B'(\vec{x};\vec{a},b,\vec{u_1},\vec{f}b\vec{u_1}) \to B'(\vec{x};\vec{a},t,\vec{u},\vec{v})) \tag{3.1}$$

and we must find a sequence of $T(;)$-functionals $F_0, \vec{F_1}, \vec{F_2}$ such that for all $\vec{x},\vec{a},\vec{f},\vec{u}$ the following holds

$$\vDash B'(\vec{x};\vec{a},F_0\vec{x}\vec{a}\vec{f}\vec{u},\vec{F_1}\vec{x}\vec{a}\vec{f}\vec{u},\vec{f}(F_0\vec{x}\vec{a}\vec{f}\vec{u})(\vec{F_1}\vec{x}\vec{a}\vec{f}\vec{u})) \to B'(\vec{x};\vec{a},t,\vec{u},\vec{F_2}\vec{x}\vec{a}\vec{f}\vec{u})$$

We can choose $F_0\vec{x}\vec{a}\vec{f}\vec{u} = t$, $\vec{F_1}\vec{x}\vec{a}\vec{f}\vec{u} = \vec{u}$ and $\vec{F_2}\vec{x}\vec{a}\vec{f}\vec{u} = \vec{f}t\vec{u}$. Then we get

$$\vDash B'(\vec{x};\vec{a},b,\vec{u},\vec{f}t\vec{u}) \to B'(\vec{x};\vec{a},b,\vec{u},\vec{f}t\vec{u})$$

which is a tautology, and so $F_0, \vec{F_1}, \vec{F_2}$ satisfies (3.1).

Case (Cut). Assume that A is inferred by (Cut)-rule. Say that A is $C \vee D$, and is inferred from $B \vee C$ and $\neg B \vee D$ by

$$\frac{B \vee C \qquad \neg B \vee D}{C \vee D}\ (Cut)$$

Suppose that $B^D = \forall \vec{u_1} \exists \vec{v_1} B'(\vec{x};\vec{a},\vec{u_1},\vec{v_1})$, $C^D = \forall \vec{u_2} \exists \vec{v_2} C'(\vec{x};\vec{a},\vec{u_2},\vec{v_2})$ and $D^D = \forall \vec{u_3} \exists \vec{v_3} D'(\vec{x};\vec{a},\vec{u_3},\vec{v_3})$. Then by using Definition 3.2.1 we have

$$\begin{aligned}
(B \vee C)^D &= \forall \vec{u_1}\vec{u_2} \exists \vec{v_1}\vec{v_2} (B'(\vec{x};\vec{a},\vec{u_1},\vec{v_1}) \vee C'(\vec{x};\vec{a},\vec{u_2},\vec{v_2})) \\
(\neg B \vee D)^D &= \forall \vec{f}\vec{u_3} \exists \vec{u_1}\vec{v_3} (\neg B'(\vec{x};\vec{a},\vec{u_1},\vec{f}\vec{u_1}) \vee D'(\vec{x};\vec{a},\vec{u_3},\vec{v_3})) \\
(C \vee D)^D &= \forall \vec{u_2}\vec{u_3} \exists \vec{v_2}\vec{v_3} C'(\vec{x};\vec{a},\vec{u_2},\vec{v_2}) \vee D'(\vec{x};\vec{a},\vec{u_3},\vec{v_3}))
\end{aligned}$$

By induction hypothesis both the left premise $(B \vee C)^D$ and the right premise $(\neg B \vee D)^D$ are valid. Hence for the left premise we have a sequence of $T(;)$-functionals $\vec{G_1}, \vec{G_2}$ such that for all $\vec{x},\vec{a},\vec{u_2},\vec{u_1}$

$$\vDash B'(\vec{x};\vec{a},\vec{u_1},\vec{G_1}\vec{x}\vec{a}\vec{u_2}\vec{u_1}) \vee C'(\vec{x};\vec{a},\vec{u_2},\vec{G_2}\vec{x}\vec{a}\vec{u_2}\vec{u_1}) \tag{3.2}$$

and for the right premise $(\neg B \vee D)^D$ we have a sequence of $T(;)$-functionals $\vec{H_1}, \vec{H_2}$ such that for all $\vec{x},\vec{a},\vec{f},\vec{u_3}$

$$\vDash \neg B'(\vec{x};\vec{a},\vec{H_1}\vec{x}\vec{a}\vec{f}\vec{u_3},\vec{f}(\vec{H_1}\vec{x}\vec{a}\vec{f}\vec{u_3})) \vee D'(\vec{x};\vec{a},\vec{u_3},\vec{H_2}\vec{x}\vec{a}\vec{f}\vec{u_3})$$

where we can put $\vec{G}_1\vec{x}\vec{a}\vec{u}_2$ for $\vec{f}$, then we get

$$\begin{aligned} \vDash \neg B'(\vec{x};\vec{a},\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3,\vec{G}_1\vec{x}\vec{a}\vec{u}_2(\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3)) \\ \vee D'(\vec{x};\vec{a},\vec{u}_3,\vec{H}_2\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3) \end{aligned} \tag{3.3}$$

We can also put $\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3$ for $\vec{u_1}$ in (3.2) , then then get

$$\begin{aligned} \vDash \neg B'(\vec{x};\vec{a},\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3,\vec{G}_1\vec{x}\vec{a}\vec{u}_2(\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3)) \\ \vee C'(\vec{x};\vec{a},\vec{u}_2,\vec{G}_2\vec{x}\vec{a}\vec{u}_2(\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3)) \end{aligned} \tag{3.4}$$

So from (3.3) and (3.4) our conclusion $(C \vee D)^D$ will be

$$\vDash C'(\vec{x};\vec{a},\vec{u}_2,\vec{G}_2\vec{x}\vec{a}\vec{u}_2(\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3)) \vee D'(\vec{x};\vec{a},\vec{u}_3,\vec{H}_2\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3)$$

Thus if we define sequence of functionals $\vec{F}_1,\vec{F}_2$ from already defined $T(;)$-functionals

$$\begin{aligned} \vec{F}_1\vec{x}\vec{a}\vec{u}_2\vec{u}_3 &= \vec{G}_2\vec{x}\vec{a}\vec{u}_2(\vec{H}_1\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3) \\ \vec{F}_2\vec{x}\vec{a}\vec{u}_2\vec{u}_3 &= \vec{H}_2\vec{x}\vec{a}(\vec{G}_1\vec{x}\vec{a}\vec{u}_2)\vec{u}_3 \end{aligned}$$

then for all $\vec{x},\vec{a},\vec{u}_2,\vec{u}_3$ we obtain

$$\vDash C'(\vec{x};\vec{a},\vec{u}_2,\vec{F}_1\vec{x}\vec{a}\vec{u}_2\vec{u}_3) \vee D'(\vec{x};\vec{a},\vec{u}_3,\vec{F}_2\vec{x}\vec{a}\vec{u}_2\vec{u}_3)$$

Case (Ind). Remembering that induction is only allowed over the input variables, and quantification is only allowed over the output variables. In this case we shall omit the ground-type variables which play no active role here. Assume that $A(x;\,)$ be inferred from $A(0;)$ and $A(x;) \to A(x+1;)$ by the (Ind)-rule, i.e.

$$\frac{A(0;) \qquad A(x;) \to A(x+1;)}{A(x;)}\ (Ind)$$

Let $A^D = \forall\vec{u}\exists\vec{v}A'(x;\vec{u},\vec{v})$. Then by using Definition 3.2.1 we have

$$\begin{aligned} (A(0;))^D &= \forall\vec{u}\exists\vec{v}A'(0;\vec{u},\vec{v}), \\ (A(x+1;))^D &= \forall\vec{u_1}\exists\vec{v_1}A'(x+1;\vec{u_1},\vec{v_1}) \\ (A(x;) \to A(x+1;))^D &= \forall\vec{f}\vec{u_1}\exists\vec{u}\vec{v_1}(A'(x;\vec{u},\vec{f}\vec{u}) \to A'(x+1;\vec{u_1},\vec{v_1})) \end{aligned}$$

By induction hypothesis from the left premise $(A(0;))^D$ is valid. So we have a sequence of $T(:)$-functionals $\vec{G}$ such that for all $\vec{u}$

$$\vDash A'(0;\vec{u},\vec{G}\vec{u}) \tag{3.5}$$

By induction hypothesis from the right premise $(A(x;) \to A(x+1;))^D$ is also valid. So we have a sequence of $T(;)$-functionals $\vec{H}_1, \vec{H}_2$ such that for all $\vec{f}, \vec{u_1}$

$$\vDash A'(x;\vec{H}_1\vec{f}\vec{u_1}, \vec{f}(\vec{H}_1\vec{f}\vec{u_1})) \to A'(x+1;\vec{u_1},\vec{H}_2\vec{f}\vec{u_1})$$

where we can put $\vec{H}_1\vec{f}\vec{u_1}$ for $\vec{u_1}$ and $\vec{u_1} = \vec{u}$, then we get

$$\vDash A'(x;\vec{u},\vec{f}\vec{u}) \to A'(x+1;\vec{u},\vec{H}_2\vec{f}\vec{u}) \tag{3.6}$$

Let $\vec{F}$ have the defining equations

$$\begin{cases} \vec{F}0\vec{u} &= \vec{G}\vec{u} \\ \vec{F}(x+1)\vec{u} &= \vec{H}_2\vec{u}(\vec{F}x\vec{u}) \end{cases}$$

Hence (3.5) becomes

$$\vdash A'(0;\vec{u},\vec{F}0\vec{u})$$

We can put new functionals $\vec{F}\vec{x}$ for $\vec{f}$ in (3.6) to obtain

$$\vDash A'(x;\vec{u},\vec{F}\vec{x}\vec{u}) \to A'(x+1;\vec{u},\vec{H}_2(\vec{F}\vec{x})\vec{u})$$

which becomes by using defining equation of $\vec{F}$

$$\vDash A'(x;\vec{u},\vec{F}\vec{x}\vec{u}) \to A'(x+1;\vec{u},\vec{F}(x+1)\vec{u})$$

We now get, by induction on x,

$$\vDash A'(x;\vec{u},\vec{F}\vec{x}\vec{u})$$

which implies that A^D is valid.

3.4 Elementary Functions Definable in $T(;)$

In this section we will show that every elementary function is definable in $T(;)$. It will be proven by using of two-sorted higher type functionals which are defined by higher type iterations using composition.

Definition 3.4.1 (Pure Type). Starting from ground type o and using $\rightarrow$ we shall describe the *pure types* by $o' = o \rightarrow o$, $o'' = o' \rightarrow o'$, $o''' = o'' \rightarrow o''$, $o^{(4)} = o''' \rightarrow o'''$ and so on. In general, it can be defined recursively as follows

$$\begin{cases} o^{(0)} &= o \\ o^{(k+1)} &= o^{(k)} \rightarrow o^{(k)} \end{cases}$$

for any $k \in \mathbb{N}$.

Definition 3.4.2 (Type Level). The *level* of a type σ, denoted by $L(\sigma)$, is inductively defined as follows

(a) $L(o) = L(\iota) = 0$

(b) $L(\sigma \rightarrow \tau) = max(L(\sigma) + 1, L(\tau))$

Special case that any pure type level can be obtained easily as follows

$$L(o^{(k)}) = k$$

for any $k \in \mathbb{N}$.

Definition 3.4.3 (Composition). The *composition operator*, $\circ$, is only allowed over the O-terms. Let f and g be functionals of the same type. Then the two functionals compose to give a new one h as follows

$$h := f \circ g$$

with defining equation

$$hv := f(gv)$$

In particular, the n-times composition of f with itself,

$$h := \underbrace{f \circ \cdots \circ f}_{n}$$

is given by

$$hv := \underbrace{f(f(...(f}_{n} v)...)) = f^n v.$$

Definition 3.4.4 (Higher Level Iterator). For any $k \in \mathbb{N}$, we can define G of type $o^{(k+2)}$ and H of type $o^{(k+3)}$ by explicit definition as follows:

$$\begin{aligned} G(g)(f) &= f \\ H(h)(g)(f) &= g((h)(g)(f)) \end{aligned}$$

where x, h, g, f are variables of type $\iota, o^{(k+2)}, o^{(k+1)}, o^{(k)}$ respectively. Then we can define the *higher level iterator* I of type $\iota \to o^{(k+2)}$ by primitive recursion:

$$\begin{cases} I0 &= G \\ I(x+1) &= H(Ix) \end{cases}$$

so that

$$Ix(g)(f) = g^x f \tag{3.7}$$

It is easy to see that by induction on x:

- *Base case*, $I0(g)(f) = G(g)(f) = f$.
- *Induction case*, assume that (3.7) is true. Then

$$\begin{aligned} I(x+1)(g)(f) &= H(Ix)(g)(f) \\ &= g((Ix)(g)(f)) \\ &= g((g)^x(f)) \\ &= g^{x+1}(f) \end{aligned}$$

Notations 3.4.5. For sake of brevity and good view, in this section, we adopt the following conventions:

1. We shall use the type level k to denote the corresponding pure type $o^{(k)}$, except that type ι, we use directly ι not its level instead.

2. The type of the iterator is indicated by its superscript, for instance, $I^k n$ is of type k.

3. We shall write I_n for In, then so I_n^k denotes iterator In of type k.

Theorem 3.4.6. *Starting with the successor function S, and using higher type primitive recursion we get the function*

$$f_k(n;a) = a + exp_n^{k+1}(1)$$

for every fixed number k. Such that all of these functions are definable in $T(;)$.

Proof. We know that the higher type iterator I_n^{k+2} is defined by primitive recursion for each type level k. So using these iterations we can define $f_k(n;a)$ by explicit definition.

$$f_k(n;a) = (I_n^{k+2})(I_n^{k+1})...(I_n^2)(S^1)(a^0)$$

where we took I_n^1 to be S and I_n^0 to be a. We now start the proof using the higher type iterators I_n^{k+2} and I_n^{k+1} in order to define function f_0 of type $\iota \to k+1$ by explicit definition

$$f_0(n;) = I_n^{k+2} I_n^{k+1}$$

and using the higher type iterators I_n^k we can define f_1 of type $\iota \to k$ by

$$f_1(n;) = I_n^{k+2} I_n^{k+1} I_n^k = (I_n^{k+1})^n I_n^k$$

and so on. In general, using the iterators I_n^{k+2}, $k = 0, 1, ...$ we can define function f_i of type $\iota \to (k+1-i)$ by explicit definition for each $i \leq k$

$$f_i(n;) = I_n^{k+2} I_n^{k+1} I_n^k ... I_n^{k+1-i}$$

We prove that for each $i \leq k$

$$f_i(n;) = (I_n^{k+2-i})^{exp_n^i(1)} I_n^{k+1-i}$$

It is easy to show by induction on i:

– *Base case,* $f_0(n;) = I_n^{k+2} I_n^{k+1}$.

– *Induction case,* assume that the result of step i holds. Then using the iterator I_n^{k-i} we get

$$\begin{aligned} f_{i+1}(n;) &= f_i(n;)I_n^{k-i} \\ &= (I_n^{k+2-i})^{exp_n^i(1)} I_n^{k+1-i} I_n^{k-i} \\ &= (I_n^{k+1-i})^{\overbrace{n \cdot n \cdot \ldots \cdot n}^{exp_n^i(1)}} I_n^{k-i} \\ &= (I_n^{k+1-i})^{exp_n^{i+1}(1)} I_n^{k-i} \end{aligned}$$

Hence putting $i = k$ we obtain

$$f_k(n;) = (I_n^2)^{exp_n^k(1)} I_n^1$$

where $I_n^1 = S^1$. Then using the output variable a^0, we get

$$\begin{aligned} f_k(n;a) &= (I_n^2)^{exp_n^k(1)}(S)(a) \\ &= S^{exp_n^{k+1}(1)}(a) \\ &= a + exp_n^{k+1}(1) \end{aligned}$$

as required. This completes the proof.

Notice that if we choose different values of $n = n_k$ at each step $k = 0, 1, 2, ...$, we obtain the following function

$$f_k(n_k, n_{k-1}, ..., n_1, n_0; a) = (I_{n_k}^{k+2})(I_{n_{k-1}}^{k+1})...(I_{n_0}^2)(S^1)(a^0) = a + n_0^{n_1^{\cdot^{\cdot^{n_k}}}}$$

Theorem 3.4.7. *Every elementary function is definable in $T(;)$.*

Proof. We know that Kleene's normal form theorem says that (see e.g. Börger [4], Chapter C, Part II) every elementary function $f(x)$ can be defined in form

$$f(x) = U(x, s)$$

where s bounds number of computable steps, U is a function in $\mathcal{E}^2$. We also know, from Chapter 2, $\mathcal{E}^2 = BCu$ which means U is definable in $T(;)$. We proved in the previous theorem that $exp_n^k(x)$ bounds number of computable steps, that is, $s = exp_n^k(x)$ and s is definable in $T(;)$ for each fixed k. Hence

$$f(x) = U(x, exp_n^k(x))$$

using substitution $f(x)$ is definable in $T(;)$. This completes the proof.

Chapter 4

Definable Functions in $T(;)$ are Elementary

In this chapter we wish to prove the converse of the result of the previous chapter, that is: every definable function in $T(;)$ is elementary. This will be proved by using normalisation and transfinite "bounding" by means of infinite terms.

4.1 Infinitary λ-calculus

In order to analyse the complexity of the Gödel system $T(;)$ we are going to embed it in an infinitary λ-calculus, denoted by $T^{\infty}(;)$, due originally to Tait [26], which allows us systematically to unravel the primitive recursions.

The infinite terms of $T^{\infty}(;)$ are obtained from $T(;)$ by three changes:

1. The substitution is replaced by λ-abstraction.
2. The primitive recursion is replaced by "infinite sequencing".
3. The finite lists $\vec{F} = F_1, ..., F_l$ are replaced by cartesian products $(F_1, ..., F_l)$.

Definition 4.1.1 (Types). In our new infinitary λ-calculus $T^{\infty}(;)$, the *types* are as in $T(;)$, except that here we add the *product types*. An object of product type $\sigma \times \tau$ is an ordered pair consisting of an object of type σ and an object of type τ.

Here again, the input type is only ι which is also input ground type as in $T(;)$, and the output types are generated from input ground-types ι and output ground type o using "$\rightarrow$" and "$\times$", according to the following inductive definition:

(a) o is an output type, and ι is the only input type.

(b) If τ and σ are output types, then $\sigma \rightarrow \tau$ and $\iota \rightarrow \tau$ are output types.

(c) If τ and σ are output types, then $\sigma \times \tau$ is an output type.

Definition 4.1.2 (Type Level). The *level* of a type σ, denoted by $L(\sigma)$, is defined as in $T(;)$, but here we have an extra clause for product type. It is inductively defined by

(a) $L(o) = L(\iota) = 0$

(b) $L(\sigma \rightarrow \tau) = max(L(\sigma) + 1, L(\tau))$

(c) $L(\sigma \times \tau) = max(L(\sigma), L(\tau))$

Definition 4.1.3 (Infinite Terms). In $T^{\infty}(;)$, the *variables* and *constants* are the same as in $T(;)$. The terms of the system $T^{\infty}(;)$ are, of course, of two-sorts:

- I^{∞}*-terms.* The *input* terms of $T^{\infty}(;)$ of type ι are called I^{∞}*-terms* and are defined as in $T(;)$.

- O^{∞}*-terms.* The *output* terms of $T^{\infty}(;)$ are called O^{∞}*-terms* and are defined inductively as follows:

(a) Any O^{∞}*-variables* and *constants* are O^{∞}-terms.

(b) If t and s are O^{∞}-terms of types $\sigma \rightarrow \tau$ and σ respectively, then the *application* ts is an O^{∞}-term of type τ.

(c) If t is an O^{∞}-term of type τ and v is an O^{∞}- variable of type σ, then the *abstraction* $\lambda v.t$ is an O^{∞}-term of type $\sigma \rightarrow \tau$.

(d) If x is a I^∞-term of type ι and t_i is an O^∞-term of type σ for each $i \in \mathbb{N}$, then the *infinite sequence* $< t_i >_{i\in\mathbb{N}} x$ is an O^∞-term of type σ. (Note the occurrence of the free input-term x here).

(e) If t_1 and t_2 are O^∞-terms of type σ and τ respectively, then the *pair* (t_1, t_2) is an O^∞-term of type $\sigma \times \tau$.

(f) For each σ_1 and σ_2, the new constants *first projection* π_1 of type $\sigma_1 \times \sigma_2 \to \sigma_1$ and *second projection* π_2 of type $\sigma_1 \times \sigma_2 \to \sigma_2$ are O^∞-terms.

Notations 4.1.4. Whenever there is no confusion we shall use the following appreviations.

1. $T^\infty(;)$*-terms* will be used for either I^∞*-terms* or O^∞*-terms*.
2. $< t_i > x$ will be used for $< t_i >_{i\in\mathbb{N}} x$ dropping the subscripts $i \in \mathbb{N}$.
3. $< t_i >_x$ will be used for $< t_i > x$.
4. π_j will be used to denote either π_1 or π_2.

Definition 4.1.5 (Finite Sequence). A *finite sequence* $(t_1, ..., t_l)$ is abbreviated by $\vec{t}$ which can be defined by the *pair* using the following convention of parenthesis

$$(t_1, t_2, ..., t_l) := (((t_1, t_2), t_2)...), t_l)$$

Therefore the first projection π_1 and second projection π_2 applied to a finite sequence give

$$\begin{aligned} \pi_1\vec{t} &:= (t_1, t_2, ..., t_{l-1}) \\ \pi_2\vec{t} &:= t_l \end{aligned}$$

So t_i can be obtained for each $i = 1, 2, ..., l$ by

$$t_1 = \pi_1^{l-1}\vec{t} \quad \text{and} \quad t_i = \pi_2(\pi_1^{l-i}\vec{t}) \quad \text{for } 2 \le i \le l$$

where π_j^l is defined for each $j = 1, 2$ recursively by

$$\begin{cases} \pi_j^0\vec{t} &= \vec{t} \\ \pi_j^{l+1}\vec{t} &= \pi_j(\pi_j^l\vec{t}) \end{cases}$$

Definition 4.1.6 (Valuation Function). Suppose that t is any $T^{\infty}(;)$-term of type σ, containing free variables $u_1, ..., u_k$ of types $\sigma_1, ..., \sigma_k$ respectively. Then given any assignment of functionals $f_1, ..., f_k$ (in the maximal type structure) to the free variables $\vec{u}$, we define the *value* of $t(\vec{u})$

$$\mathcal{V}t[\vec{u} := \vec{f}]$$

which can be defined by induction over the build up of t as follows:

(a) $t = u_i \quad \Rightarrow \quad \mathcal{V}u_i[\vec{u} := \vec{f}] = f_i$

(b) $t = c \quad \Rightarrow \quad \mathcal{V}c[\vec{u} := \vec{f}] = c$ where c is any constant

(c) $t = t_0 t_1 \quad \Rightarrow \quad \mathcal{V}(t_0 t_1)[\vec{u} := \vec{f}] = \mathcal{V}t_0[\vec{u} := \vec{f}](\mathcal{V}t_1[\vec{u} := \vec{f}])$

(d) $t =< t_i >_x \quad \Rightarrow \quad \mathcal{V}(< t_i >_x)[x := n, \vec{u} := \vec{f}] = \mathcal{V}t_n[x := n, \vec{u} := \vec{f}]$

(e) $t = \lambda v.r \quad \Rightarrow \quad \mathcal{V}(\lambda v.r)[\vec{u} := \vec{f}] = F$ where $F(g) := \mathcal{V}r[\vec{u} := \vec{f}, v := g]$.

(f) $t = (t_0, t_1) \quad \Rightarrow \quad \mathcal{V}(t_0, t_1)[\vec{u} := \vec{f}] = (\mathcal{V}t_0[\vec{u} := \vec{f}], \mathcal{V}t_1[\vec{u} := \vec{f}])$

Definition 4.1.7 (Substitution). The formulation $t[s/v]$ denotes the result of substituting s for v in t, assuming that v and s have same output-type. The *substitution* is defined by induction on t as follows.

(a) $v[s/v] = s$

(b) $u[s/v] = u$ for $u \neq v$ or u is a constant

(c) $(tr)[s/v] = (t[s/v])(r[s/v])$

(d) $(\lambda v.t)[s/v] = \lambda v.t$

(e) $(\lambda u.t)[s/v] = \lambda u.t[s/v]$ for $u \neq v$ and $u \notin FV(s)$ or $v \notin FV(t)$

(f) $(\lambda u.t)[s/v] = \lambda w.t[w/u][s/v]$ for $u \neq v$ and $u \in FV(s)$, and the new variable w is not in $FV(s) \cup FV(t)$

(g) $< t_i >_x [s/v] =< t_i[s/v] >_x$

(h) $(t_1, t_1)[s/v] = (t_1[s/v], t_2[s/v])$

where a variable v in a term t is called *bound* if it is in a context $\lambda v.t$, and otherwise is called *free variable*. The set of free variable in t is denoted by $FV(t)$. For further information on λ-calculus, see Hindley and Seldin [12].

4.2 Normalisation

In this section we shall first define a reduction mechanism whereby every redexes of $T^{\infty}(;)$ reduces to another term by reducing the rank of the term by at least one. We then show that every term of finite rank reduces to a normal form with rank 0.

Definition 4.2.1 (Term Level). The *level of a term* t is the level of type of the term t, denoted by $L(t)$, that is

$$t : \sigma \quad \Rightarrow \quad L(t) = L(\sigma)$$

Lemma 4.2.2. *If* $t = \lambda v.r$, *then we have the following*

$$L(v) < L(t)$$

Proof. It is easy to prove that if $v : \sigma$ and $r : \tau$, then $t : \sigma \to \tau$. Hence

$$\begin{aligned} L(v) &= L(\sigma) && \text{by Def. 4.2.1} \\ &< L(\sigma) + 1 \\ &\leq max(L(\sigma) + 1, L(\tau)) \\ &= L(\sigma \to \tau) && \text{by Def. 4.1.2} \\ &= L(t) && \text{by Def. 4.2.1} \end{aligned}$$

Definition 4.2.3 (Redexes). The *redexes* are certain kinds of application terms as defined by

(I) *Abstraction-redex,* $(\lambda v.t)s$

(II) *Sequence-redex,* $< t_i >_x s$

(III) *Projection-redexes,* $\pi_j(t_1, t_2)$ or $\pi_j < t_i >_x$

Definition 4.2.4 (Rank). The *rank* of $T^{\infty}(;)$-term t, denoted by $R(t)$, is defined to be the supremum of type levels of all sub-terms of the form

$$\begin{array}{rlll} \lambda v.r & \text{in context} & (\lambda v.r)s & \text{or} \\ < t_i >_x & \text{in context} & < t_i >_x s & \text{or} \\ \pi_j & \text{in context} & \pi_j(t_1, t_2) \quad \text{or} \quad \pi_j < t_i >_x & \end{array}$$

So the *rank function* R is defined from terms into numbers recursively by the following properties:

(a) $R(t) = 0$ if t is a variable or a constant

(b) $R(\lambda v.r) = R(r)$

(c) $$R(t_1 t_2) = \begin{cases} max(R(t_1), R(t_2), L(t_1)) & \text{if } t_1 t_2 \text{ is a redex} \\ max(R(t_1), R(t_2)) & \text{otherwise} \end{cases}$$

(d) $R(< t_i >_x) = sup(R(t_i))$

(e) $R((t_1, t_2)) = max(R(t_1), R(t_2))$

Definition 4.2.5 (Rank-Reduction). A *rank-reduced* term t' is obtained from t (with a reduction in the rank of the term by at least one, if it is positive to begin with) by the *rank-reduction*. The rank-reduction operator $t \mapsto t'$ is defined recursively as follows:

(a) t is a variable or constant $\Rightarrow$ $t' = t$

(b) $t = \lambda v.r \Rightarrow t' = \lambda v.r'$

(c) $t =< r_i >_x \Rightarrow t' =< r_i' >_x$

(d) $t = rs \Rightarrow t' = (r's')^*$

(e) $t = (r_1, r_2) \Rightarrow t' = (r_1', r_2')$

Definition 4.2.6. The notation $(ts)^*$ which is used above is defined as follows.

(a) $(ts)^* = ts$ if ts is not a redex

(b) $((\lambda v.r)s)^* = r[s/v]$

(c) $(< r_i >_x s)^* =< (r_i s)^* >_x$

(d) $(\pi_j s)^* = \begin{cases} s_j & \text{if } s = (s_1, s_2) \\ < (\pi_j s_i)^* >_x & \text{if } s =< s_i >_x \end{cases}$

Definition 4.2.7 (Normal). A term is said to be *normal* or to be in *normal form*, if it does not contain any kind of redex. Thus if a term t contains no redex its rank is 0.

If a term t reduces to a normal form $t^{\prime\prime\ldots\prime}$ after k-many times application of rank-reduction, then this normal form of t is abbreviated by $t^{[k]}$ and written by

$$t \mapsto t^{[k]}$$

We can define any normal $t^{[n]}$ with rank-reduction operator as follows

$$\begin{cases} t^{[0]} = t \\ t^{[n+1]} = (t^{[n]})' \end{cases}$$

Lemma 4.2.8. *Let t_1, t_2 be any terms with finite rank and let v be a variable in $T^\infty(;)$. Then if $max(R(t_1), R(t_2), L(v)) = k$ we have the following property*

$$R(t_1[t_2/v]) \leq k$$

Proof. The proof is by induction on construction of t_1:

Case 1. If t_1 is a variable or constant, then there are two sub-cases to consider:

1.1. If $t_1 = v$, then

$$\begin{aligned} R(v[t_2/v]) &= R(t_2) && \text{by Def. } 4.1.7 - (a) \\ &\leq k && \text{given} \end{aligned}$$

1.2. If $t_1 \neq v$, then

$$\begin{aligned} R(t_1[t_2/v]) &= R(t_1) && \text{by Def. } 4.1.7 - (b) \\ &\leq k && \text{given} \end{aligned}$$

Case 2. If $t_1 = \lambda u.r$ then there are three sub-cases to consider:

2.1. If $u = v$, then

$$\begin{aligned} R((\lambda v.r)[t_2/v]) &= R(\lambda v.r) && \text{by Def. } 4.1.7-(d) \\ &= R(t_1) && \text{given} \\ &\leq k && \text{given} \end{aligned}$$

2.2. If $u \neq v$ and $u \notin FV(t_2)$ or $v \notin FV(r)$, then

$$\begin{aligned} R((\lambda u.r)[t_2/v]) &= R(\lambda u.r[t_2/v]) && \text{by Def. } 4.1.7-(e) \\ &= R(r[t_2/v]) && \text{by Def. } 4.2.4-(b) \\ &\leq k && \text{by ind. hyp.} \end{aligned}$$

2.3. If $u \neq v$ and $u \in FV(t_2)$ and new variable w is not in $FV(t_2) \cup FV(r)$, then

$$\begin{aligned} R((\lambda u.r)[t_2/v]) &= R(\lambda w.r[w/u][t_2/v]) && \text{by Def. } 4.1.7-(f) \\ &= R((r[w/u])[t_2/v]) && \text{by Def. } 4.2.4-(b) \\ &\leq k && \text{by ind. hyp.} \end{aligned}$$

where the variable w is a fresh one and by induction hypothesis we get $R(r[w/u]) \leq k$ because $R(w) \leq k$ since $R(w) = 0$, $L(u) \leq k$ since $L(u) < L(t_1)$ by Lemma 4.2.2 and $R(r) \leq k$ since $R(t_1) = R(r)$ by Definition 4.2.4-(b).

Case 3. If $t_1 = < r_i >_x$, then

$$\begin{aligned} R(< r_i >_x [t_2/v]) &= R(< r_i[t_2/v] >_x) && \text{by Def. } 4.1.7-(g) \\ &= max(R(r_i[t_2/v])) && \text{by Def. } 4.2.4-(d) \\ &\leq k && \text{by ind. hyp.} \end{aligned}$$

where $R(r_i[t_2/v]) \leq k$ by induction hypothesis because $R(r_i) \leq k$ for every i since $max(R(r_i)) = R(t_1)$ by using Definition 4.2.4.

Case 4. If $t_1 = r_1 r_2$, then this case falls into two subcases to consider:

4.1. If $r_1 r_2$ be a redex, then by using Definitions 4.1.7-(c) and 4.2.4-(c) respectively

$$\begin{aligned} R((r_1 r_2)[t_2/v]) &= R((r_1[t_2/v])(r_2[t_2/v])) \\ &= max(R(r_1[t_2/v]), R(r_2[t_2/v]), L(r_1[t_2/v])) \\ &\leq k \qquad \text{by ind. hyp.} \end{aligned}$$

where the application $(r_1[t_2/v])(r_2[t_2/v])$ is a redex since $r_1 r_2$ is a redex, say $r_1 = v$ and t_2 is either an abstraction or an infinite-sequence or projections. We have $R(r_1[t_2/v]) \leq k$, $R(r_2[t_2/v]) \leq k$ by induction hypothesis and $L(r_1[t_2/v]) \leq k$ because $L(r_1) \leq k$, $L(r_2) \leq k$ since $L(r_1) = L(r_2) = L(v)$.

4.2. If $r_1 r_2$ be not a redex, then the application term $(r_1[t_2/v])(r_2[t_2/v])$ may be not a redex. So in this case by using Definition 4.2.4-(c) we get

$$\begin{aligned} R((r_1 r_2)[t_2/v]) &= R((r_1[t_2/v])(r_2[t_2/v])) \\ &= max(R(r_1[t_2/v]), R(r_2[t_2/v])) \\ &\leq k \qquad \text{by ind. hyp.} \end{aligned}$$

Where if the term $(r_1[t_2/v])(r_2[t_2/v])$ may be a redex, then this case is the same as the *Case* 4.1. above.

Case 5. If $t_1 = (r_1, r_2)$, then

$$\begin{aligned} R((r_1, r_2)[t_2/v]) &= R((r_1[t_2/v], r_2[t_2/v])) && \text{by Def. } 4.1.7 - (h) \\ &= max(R(r_1[t_2/v]), R(r_2[t_2/v])) && \text{by Def. } 4.2.4 - (e) \\ &\leq k \qquad \text{by ind. hyp.} \end{aligned}$$

where we used the induction hypothesis because

$$R(r_1[t_2/v]) \leq k, \qquad R(r_2[t_2/v]) \leq k \qquad \text{by ind. hyp.}$$

since $max(R(r_1), R(r_2)) = R(t_1) \leq k$ by Definition 4.2.4-(e).

Lemma 4.2.9. *Let t_1, t_2 be any $T^{\infty}(;)$-terms of finite rank and suppose the application term t_1t_2 is a redex. Then if $R(t_1t_2) = k+1$ where $R(t_1) \leq k$ and $R(t_2) \leq k$ we have*

$$R((t_1t_2)^*) \leq k$$

Proof. Before starting proof we have to point out that since t_1t_2 is a redex we have

$$R(t_1t_2) = max(R(t_1), R(t_2), L(t_1)) \qquad \text{by Def. } 4.2.4-(c)$$

which means $max(R(t_1), R(t_2), L(t_1)) \leq k+1$ and more specifically we get

$$L(t_1) \leq k+1 \tag{4.1}$$

Hence the proof runs by induction on construction of the term t_1:

Case 1. It t_1 is a variable or a constant (except π_j), then t_1t_2 can not be redex. If $t_1 = \pi_j$, then t_1t_2 can be a projection-redex whenever t_2 is either a pair or a sequence. So there are two sub-cases to consider:

1.1. If $t_2 = (r_1, r_2)$, then

$$\begin{aligned} R((\pi_j(r_1,r_2))^*) &= R(r_j) && \text{by Def. } 4.2.6-(d) \\ &\leq max(R(r_1), R(r_2)) \\ &= R(t_2) && \text{by Def. } 4.2.4-(e) \\ &\leq k && \text{given} \end{aligned}$$

1.2. If $t_2 = < r_i >_x$, then

$$\begin{aligned} R((\pi_j < r_i >_x)^*) &= R(< (\pi_j r_i)^* >_x) && \text{by Def. } 4.2.6-(c) \\ &= sup(R((\pi_j r_i)^*)) && \text{by Def. } 4.2.4-(d) \\ &\leq k \end{aligned}$$

since $R((\pi_j r_i)^*) \leq k$ whether the application term $\pi_j r_i$ is a redex or not.

– If the term is a redex, then it is true by induction hypothesis since we have that $L(\pi_j) = L(t_1)$ by Definition 4.2.1,and $R(\pi_j) = 0$ and $sup(R(r_i)) = R(t_2)$ by Definition 4.2.4-(a) and (d) respectively.

– If $\pi_j r_i$ is not a redex, then

$$\begin{aligned} R((\pi_j r_i)^*) &= R(\pi_j r_i) && \text{by Def.4.2.6} \\ &= max(R(\pi_j), R(r_i)) && \text{by Def.4.2.6} - (c) \\ &= sup(R(r_i)) && \text{since } R(\pi_j) = 0 \\ &= R(t_2) && \text{by Def.4.2.6} - (d) \\ &\leq k && \text{given} \end{aligned}$$

Case 2. If $t_1 = \lambda v.r$, then $t_1 t_2$ is an abstract-redex. So

$$\begin{aligned} R(((\lambda v.r)t_2)^*) &= R(r[t_2/v]) && \text{by Def. } 4.2.6 - (b) \\ &\leq k && \text{by Lemma } 4.2.8 \end{aligned}$$

where $max(R(r), R(t_2), L(v)) \leq k$ because $R(r) \leq k$ since $R(r) = R(t_1)$ by Definition 4.2.4-(b), and $L(v) \leq k$ since $L(t_1) \leq k+1$ by (4.1) and $L(v) \leq k$ since $L(v) < L(t_1)$ by Lemma 4.2.2.

Case 3. If $t_1 =< r_i >_x$, then $t_1 t_2$ is a sequence-redex.

$$\begin{aligned} R((< r_i >_x t_2)^*) &= R(< (r_i t_2)^* >_x) && \text{by Def. } 4.2.6 - (b) \\ &= sup(R(r_i t_2)^*)) && \text{by Def. } 4.2.6 - (d) \\ &\leq k \end{aligned}$$

since $R((r_i t_2)^*) \leq k$ whether the application term $r_i t_2$ is a redex or not.

– If the term is a redex, then it is true by induction hypothesis since for every i we have that $L(r_i) = L(t_1)$ by Definition 4.2.1 and $R(r_i) = R(t_1)$ by Definition 4.2.4-(d).

– If $r_i t_2$ is not a redex, say r_i is a variable or constant (except π_j, $j = 1, 2$), then

$$\begin{aligned} R((r_i t_2)^*) &= R(r_i t_2) && \text{by Def.4.2.6} \\ &= max(R(r_i), R(t_2)) && \text{by Def.4.2.6} - (c) \\ &= R(t_2) && \text{since } R(r_i) = 0 \\ &\leq k && \text{given} \end{aligned}$$

Case 4. If $t_1 = (r_1, r_2)$, then $t_1 t_2$ can not be a redex.

Lemma 4.2.10. *Let t be any $T^{\infty}(;)$-term with a positive finite rank. In that case, if t rank-reduces to t' as in Definition 4.2.5, then*

$$R(t') < R(t)$$

Proof. The proof goes by induction on the construction of t:

Case 1. If t is a variable or constant, then this case is vacuously true.

Case 2. If $t = \lambda v.s$, then

$$\begin{aligned} R((\lambda v.s)') &= R(\lambda v.s') && \text{by Def. } 4.2.5-(b) \\ &= R(s') && \text{by Def. } 4.2.4-(b) \\ &< R(s) && \text{by ind. hyp.} \\ &= R(t) && \text{by Def. } 4.2.4-(b) \end{aligned}$$

Case 3. If $t =< r_i >_x$, then

$$\begin{aligned} R((< r_i >_x)') &= R(< r_i' >_x) && \text{by Def. } 4.2.5-(c) \\ &= sup(R(r_i')) && \text{by Def. } 4.2.4-(d) \\ &< sup(R(r_i)) && \text{by ind. hyp.} \\ &= R(t) && \text{by Def. } 4.2.4-(d) \end{aligned}$$

Case 4. If $t = t_1 t_2$, then

$$\begin{aligned} R((t_1 t_2)') &= R((t_1' t_2')^*) && \text{by Def. } 4.2.5-(d) \\ &< R(t_1 t_2) && \text{by Lemma } 4.2.9 \end{aligned}$$

where the application term $t_1' t_2'$ is a redex, if the term is not a redex, then

$$\begin{aligned} R((t_1 t_2)') &= R((t_1' t_2')^*) && \text{by Def. } 4.2.5-(d) \\ &= R((t_1' t_2')) && \text{by Def. } 4.2.6-(c) \\ &= max(R(t_1'), R(t_2')) && \text{by Def. } 4.2.4-(c) \\ &< max(R(t_1), R(t_2)) && \text{by ind. hyp.} \\ &\leq max(R(t_1), R(t_2), L(t_1)) && \text{since } t_1 t_2 \text{ is a redex} \\ &= R(t_1 t_2) && \text{by Def. } 4.2.4-(c) \end{aligned}$$

Case 5. If $t = (t_1, t_2)$, then

$$\begin{aligned} R((t_1,t_2)') &= R((t_1',t_2')) && \text{by Def. } 4.2.5-(e) \\ &= max(R(t_1'),R(t_2')) && \text{by Def. } 4.2.4-(e) \\ &< max(R(t_1),R(t_2)) && \text{by ind. hyp.} \\ &= R(t) && \text{by Def. } 4.2.4-(e) \end{aligned}$$

Theorem 4.2.11 (Normalisation). *Every $T^\infty(;)$-term of finite rank reduces to a normal form.*

Proof. Let t be any $T^\infty(;)$-term. If we continue the procedure of Lemma 4.2.10, then $R(t)$ strictly decreases and then we must eventually obtain a normal form.

4.3 Ordinal Notations

A knowledge of some of the theory of ordinals is required to examine the proof complexity. In this section we wish to introduce some fundamental notions of the theory of *countable tree-ordinals* $< \epsilon_0$, *not set-theoretic ordinals.* We shall use the lower case Greek letters $\alpha, \beta, \gamma, \ldots, \lambda, \ldots$ with or without subscripts to denote the ordinals, and $i, j, k, l, m, n, \ldots$ to denote ordinals $< \omega$ or natural numbers.

For a more thorough analysis of tree ordinals see Fairtlough and Wainer [8].

Definition 4.3.1 (Tree Ordinal). The *set of countable tree-ordinals*, Ω, is generated inductively according to the following rules:

(a) $\quad 0 \in \Omega$

(b) $\quad \alpha \in \Omega \quad \Rightarrow \quad \alpha + 1 := \alpha \cup \{\alpha\} \in \Omega$

(c) $\quad \forall n \in \mathbb{N}.(\lambda_n \in \Omega) \quad \Rightarrow \quad \lambda := \sup_{n \in \mathbb{N}}(\lambda_n) \in \Omega$

Where $\alpha + 1$ is called *successor* of α, and λ is called a *limit* which is function from $\mathbb{N}$ to Ω. An ordinal is a limit if it is neither 0 nor a successor.

Notations 4.3.2. Whenever there is no confusion we shall adopt the following:

1. In the sequel we shall only be dealing with *countable tree ordinals*, and we shall refer to them as *ordinals*.

2. We usually use $sup(\alpha_n)$ for $\sup_{n\in\mathbb{N}}(\alpha_n)$ by dropping off the subscripts $n \in \mathbb{N}$ whenever it is possible.

Definition 4.3.3 (Ordinal Arithmetic). We would like to define arithmetical operations *addition*, *multiplication* and *exponentiation* on the ordinals by recursion, as follows.

Addition

$$\begin{aligned} \alpha + 0 &= \alpha \\ \alpha + (\beta + 1) &= (\alpha + \beta) + 1 \\ \alpha + \lambda &= sup(\alpha + \lambda_n) \end{aligned}$$

Multiplication

$$\begin{aligned} \alpha \cdot 0 &= 0 \\ \alpha \cdot (\beta + 1) &= \alpha \cdot \beta + \alpha \\ \alpha \cdot \lambda &= sup(\alpha \cdot \lambda_n) \end{aligned}$$

Exponentiation

$$\begin{aligned} \alpha^0 &= 1 \\ \alpha^{\beta+1} &= \alpha^\beta \cdot \alpha \\ \alpha^\lambda &= sup(\alpha^{\lambda_n}) \end{aligned}$$

Note that for all the above, it is straightforward to prove that *associativity* and *left-handed distributivity*

$$\alpha \cdot (\beta + \gamma) = \alpha \cdot \beta + \alpha \cdot \gamma$$

for every α, β and γ, holds, but *right-side distributivity* does not necessary holds. Furthermore, they are not commutative.

Definition 4.3.4 (Sub-Tree Ordering). For $k \in \mathbb{N}$, the *sub-tree (partial) ordering* $\prec_k$ is the transitive closure of the following two rules for any tree ordinal α

(a) $\alpha \prec_k \alpha + 1$

(b) $\alpha_k \prec_k \alpha$ where $\alpha = sup(\alpha_n)$

Note that we are going to write $\alpha \preccurlyeq_k \beta$ to mean $\alpha = \beta$ or $\alpha \prec_k \beta$. Note also that for all $n, m \in \mathbb{N}$ we have (by identifying n with the tree-ordinal $0+1+1+...+1$ n-times.)

$$m \prec_k n \Longleftrightarrow m < n$$

Definition 4.3.5 (Ordinal Height). The set-theoretical-*height* of tree ordinal α, denoted by $|\alpha|$, is defined inductively by

(a) $|0| = 0$

(b) $|\alpha + 1| = |\alpha| + 1$

(c) $|\lambda| = sup(|\lambda_n| + 1)$

Note that for the ordinal numbers α and β we have that

$$\alpha \prec_k \beta \Rightarrow |\alpha| < |\beta|$$

Definition 4.3.6 (Slow-Growing Function). Let $G_n(\alpha)$ be cardinality of the set $\{\beta \ : \ \beta+1 \preccurlyeq_n \alpha\}$. Then we immediately get the *Slow-Growing function* $G_n : \Omega \to \mathbb{N}$ by the transfinite recursion

$$\begin{cases} G_n(0) &= 0 \\ G_n(\alpha + 1) &= G_n(\alpha) + 1 \\ G_n(\lambda) &= G_n(\lambda_n) \end{cases}$$

for each fixed $n \in \mathbb{N}$.

Lemma 4.3.7. *We have the following arithmetical relations for each fixed* $n \in \mathbb{N}$

$$(i) \qquad G_n(\alpha + \beta) = G_n(\alpha) + G_n(\beta)$$

$$(ii) \qquad G_n(\alpha \cdot \beta) = G_n(\alpha) \cdot G_n(\beta)$$

$$(iii) \qquad G_n(\alpha^\beta) = (G_n(\alpha))^{G_n(\beta)}$$

Proofs. All easily proved by induction on β.

Remark 4.3.8. For each tree-ordinal there are many $\prec_k$-incomparable tree ordinals with the same set-theoretic height. For example, $\omega_0 =< 0, 1, 2, ..., n, ... >$ is an ordinal, $\omega =< 1, 2, ..., n+1, ... >$ is also an ordinal, and so on. In general, for each increasing function $f : \mathbb{N} \to \mathbb{N}$, there exist $\alpha_f =< f(0), f(1), ..., f(n), ... >$ where $\alpha_f = sup(f(n))$, then by using definition of Slow Growing function

$$G_{\alpha_f} = G_{f(n)} = f(n)$$

and hence every such function already crops up at the first limit level in Ω. Of course, each tree ordinal has a set-theoretic ordinal-height. Some of these are more natural then others, the most natural being ω_0 and ω. Set theoretically they are the same because their height are same, but in the context of tree ordinals they are different, and they are also incomparable under each relation $\prec_n$.

In this work we will be concerned with those ordinals which can be constructed from

$$\omega = sup(n + 1)$$

by addition, multiplication and exponentiation. It means that we chose $< n + 1 >$ as the fundamental sequence for ω.

Lemma 4.3.9. *If* β *is "structured tree-ordinal" in the sense that for every limit sub-tree* λ *of* β *there is a property* $\forall m, n.(m < n \Rightarrow \lambda_m + 1 \prec_n \lambda)$, *then*

$$\alpha \prec_k \beta \quad \Rightarrow \quad \forall n > k(G_n(\alpha) < G_n(\beta))$$

Proof. For all $n > k$, we can prove it by induction on β:

Case 1. If $\beta = 0$, it is trivial.

Case 2. If β is a successor, then $\beta = \beta' + 1$. So $\alpha \preccurlyeq_k \beta'$ and we have

$$\begin{aligned} G_n(\alpha) &\leq G_n(\beta') && \text{by ind. hyp.} \\ &< G_n(\beta') + 1 \\ &= G_n(\beta' + 1) && \text{by Def. 4.3.6} \\ &= G_n(\beta) \end{aligned}$$

Case 3. If β is a limit, then $\beta_k \prec_k \beta$. So $\alpha \preccurlyeq_k \beta_k$ and we have

$$\begin{aligned} G_n(\alpha) &\leq G_n(\beta_k) && \text{by ind. hyp.} \\ &< G_n(\beta_k + 1) \\ &\leq G_n(\beta_n) && \text{since} \quad \beta_k + 1 \preccurlyeq_n \beta_n \\ &= G_n(\beta) && \text{by Def. 4.3.6} \end{aligned}$$

Note that from now on all the tree-ordinals we construct will be "*structured tree-ordinal*" as in this Lemma.

Definition 4.3.10 (Step-Down Relation). The *step-down relation* is the relation $\alpha \prec_o \beta$. It is defined inductively by

(a) $\alpha \prec_o \alpha + 1$ for any α

(b) $\alpha_0 \prec_o \alpha$ where $\alpha = sup(\alpha_n)$

(c) $\gamma \prec_o \alpha, \quad \alpha \prec_o \beta \quad \Rightarrow \quad \gamma \prec_o \beta$

Note that our work is based on this special *sub-tree ordering* $\prec_o$, the reason is that from the above Lemma 4.3.9 for all $n \in \mathbb{N}$ we get

$$\begin{aligned} \alpha \prec_o \beta &\Rightarrow G_n(\alpha) < G_n(\beta) \\ \alpha \preccurlyeq_o \beta &\Rightarrow G_n(\alpha) \leq G_n(\beta) \end{aligned}$$

Lemma 4.3.11. *Let α, β, γ be ordinals and let k be a positive natural number. Then the following are true*

(i) $\alpha \preccurlyeq_o \beta \Rightarrow \gamma + \alpha \preccurlyeq_o \gamma + \beta$

(ii) $\alpha \preccurlyeq_o \beta \Rightarrow k \cdot \alpha \preccurlyeq_o k \cdot \beta$

In particular,

(i') $\gamma \preccurlyeq_o \gamma + \beta$ with $\alpha = 0 \preccurlyeq_o \beta$ in (i)

(ii') $k \preccurlyeq_o k \cdot \beta$ with $\alpha = 1 \preccurlyeq_o \beta$ in (ii)

Proofs.

(i). We can prove it by induction on β:

Case 1. If $\beta = 0$, it is trivial.

Case 2. If β is a successor, then $\beta = \beta' + 1$. So $\alpha \preccurlyeq_o \beta'$ and we have

$$\begin{array}{lll} \gamma + \alpha \preccurlyeq_o \gamma + \beta' & & \text{by ind. hyp.} \\ \prec_o \gamma + (\beta' + 1) & & \text{by Def. } 4.3.10 - (a) \\ = \gamma + \beta & & \end{array}$$

Case 3. If β is a limit, then $\beta_0 \prec_o \beta$. So $\alpha \preccurlyeq_o \beta_0$ and we have

$$\begin{array}{lll} \gamma + \alpha \preccurlyeq_o \gamma + \beta_0 & & \text{by ind. hyp.} \\ = (\gamma + \beta)_0 & & \text{by Def. } 4.3.3 \\ \prec_o \gamma + \beta & & \text{by Def. } 4.3.10 - (b) \end{array}$$

(ii). We can prove it by induction on β:

Case 1. If $\beta = 0$, it is trivial.

Case 2. If β is a successor, then $\beta = \beta' + 1$. So $\alpha \preccurlyeq_o \beta'$ and we have

$$\begin{array}{lll} k \cdot \alpha \preccurlyeq_o k \cdot \beta' & & \text{by ind. hyp.} \\ \prec_o k \cdot \beta' + 1 & & \text{by Def. } 4.3.10 - (a) \\ \preccurlyeq_o k \cdot \beta' + k & & \text{by } (i') \\ = k \cdot (\beta' + 1) & & \text{by Def. } 4.3.3 \\ = k \cdot \beta & & \end{array}$$

Case 3. If β is a limit, then $\beta_0 \prec_o \beta$. So $\alpha \preccurlyeq_o \beta_0$ and we have

$$\begin{array}{rcll} k \cdot \alpha & \preccurlyeq_o & k \cdot \beta_0 & \text{by ind. hyp.} \\ & = & (k \cdot \beta)_0 & \text{by Def. 4.3.3} \\ & \prec_o & k \cdot \beta & \text{by Def. 4.3.10} - (b) \end{array}$$

Definition 4.3.12 (Ordinal Function φ). The *ordinal function* φ on the tree-ordinals is defined recursively as follows:

$$\begin{cases} \varphi(0) &= 1 \\ \varphi(\alpha+1) &= \varphi(\alpha) + 2 \cdot \varphi(\alpha) + 2 \cdot \varphi(\alpha) \\ \varphi(\lambda) &= sup\ \varphi(\lambda_i) \end{cases}$$

The reason for this somewhat strange-looking function will become clear in Theorem 4.5.3. It is in fact abstract form of exponentiation.

Lemma 4.3.13. *Let φ be the function above. Then the following are true*

(i) $1 \preccurlyeq_o \varphi(\alpha)$ *for every* α.

(ii) $\varphi(\alpha) + 1 \prec_o \varphi(\alpha + 1)$

(iii) α *is a limit* $\Rightarrow$ $\varphi(\alpha_i) \prec_i \varphi(\alpha)$ *for every* i

(iv) $\alpha \preccurlyeq_o \beta \Rightarrow \varphi(\alpha) \preccurlyeq_o \varphi(\beta)$

Proofs.

(i). It is trivial.

(ii). For any α the proof directly is as follows

$$\begin{array}{rcll} \varphi(\alpha)+1 & \prec_o & \varphi(\alpha)+2 & \text{by Def. 4.3.10} - (a) \\ & \preccurlyeq_o & \varphi(\alpha)+2\cdot\varphi(\alpha) & \text{by Lemmas 4.3.11} - (ii) \\ & \preccurlyeq_o & \varphi(\alpha)+2\cdot\varphi(\alpha)+2\cdot\varphi(\alpha) & \text{by Lemma 4.3.11} - (i) \\ & = & \varphi(\alpha+1) & \text{by Def. 4.3.12} \end{array}$$

(iii). It is trivial.

(iv). If $\alpha = \beta$ the proof is trivial. If $\alpha \prec_o \beta$ it is proved by induction on β:

Case 1. If $\beta = 0$, it is trivial.

Case 2. If β is a successor, then $\beta = \beta' + 1$. So $\alpha \preccurlyeq_o \beta'$ and we have

$$\begin{array}{rcll} \varphi(\alpha) & \preccurlyeq_o & \varphi(\beta') & \text{by ind. hyp.} \\ & \prec_o & \varphi(\beta') + 1 & \text{by Def. } 4.3.10 - (a) \\ & \prec_o & \varphi(\beta' + 1) & \text{by } (ii) \\ & = & \varphi(\beta) & \text{by Def. } 4.3.12 \end{array}$$

Case 3. If β is a limit, then $\beta_0 \prec_o \beta$. So $\alpha \preccurlyeq_o \beta_0$ and we have

$$\begin{array}{rcll} \varphi(\alpha) & \preccurlyeq_o & \varphi(\beta_0) & \text{by ind. hyp.} \\ & \preccurlyeq_o & \varphi(\beta) & \text{by } (iii) \end{array}$$

Lemma 4.3.14. *Let G_n be the slow-growing function, and let φ be as in the Definition 4.3.12. Then for every α we have*

$$G_n(\varphi(\alpha)) = 5^{G_n(\alpha)}$$

Proof. By induction on α:

Case 1. If $\alpha = 0$, it is trivial.

Case 2. If α is a successor, then $\alpha = \alpha' + 1$. So

$$\begin{array}{rll} & G_n(\varphi(\alpha' + 1)) & \\ = & G_n(\varphi(\alpha') + 2 \cdot \varphi(\alpha') + 2 \cdot \varphi(\alpha')) & \text{by Def. } 4.3.12 \\ = & G_n(\varphi(\alpha')) + G_n(2 \cdot \varphi(\alpha')) + G_n(2 \cdot \varphi(\alpha')) & \text{by Lemma } 4.3.7 - (i) \\ = & 5^{G_n(\alpha')} + 2 \cdot 5^{G_n(\alpha')} + 2 \cdot 5^{G_n(\alpha')} & \text{by ind. hyp.} \\ = & 5 \cdot 5^{G_n(\alpha')} & \text{on natural numbers} \\ = & 5^{G_n(\alpha')+1} & \text{on natural numbers} \\ = & 5^{G_n(\alpha'+1)} & \text{by Def. } 4.3.6 \end{array}$$

Case 3. If α is a limit, then $\alpha = sup\ (\alpha_n)$. So

$$\begin{array}{rcll} G_n(\varphi(sup(\alpha_n))) & = & G_n(\varphi(\alpha_n)) & \text{by Def. } 4.3.6 \\ & = & 5^{G_n(\alpha_n)} & \text{by ind. hyp.} \\ & = & 5^{G_n(\alpha)} & \text{by Def. } 4.3.6 \end{array}$$

4.4 Embedding

The aim of this section is to prove that every functional of $T(;)$ is represented by a term of $T^{\infty}(;)$ with an ordinal bound of set-theoretic height below ω^2.

In this section the types will not be written for sake of brevity, though, of course, each term has an appropriate type.

Definition 4.4.1 (Majorization Relation). Let t be a term of $T^{\infty}(;)$ and α be a tree ordinal. The *majorization relation* $t \lhd \alpha$, reading "*t is majorized by α*", is defined inductively as follows:

(a) t is a variable or constant $\Rightarrow$ $t \lhd 0$

(b) $t_1 \lhd \alpha_1,\ t_2 \lhd \alpha_2,\ \alpha_1 \preccurlyeq_o \alpha_2 \Rightarrow t_1 t_2 \lhd \alpha_2 + 1$

(c) $r \lhd \beta \Rightarrow \lambda v.r \lhd \beta + 1$

(d) $\forall i \exists \beta (t_i \lhd \beta \wedge \beta + 1 \preccurlyeq_o \gamma_i) \Rightarrow < t_i > x \lhd \gamma,$ where $\gamma = sup(\gamma_i)$

(e) $t_1 \lhd \alpha,\ t_2 \lhd \alpha \Rightarrow (t_1, t_2) \lhd \alpha$

(f) $t \lhd \beta,\ \beta \prec_o \alpha \Rightarrow t \lhd \alpha$

Note that as one can see easily from clause (e) the following is true

$$\vec{t} \lhd \alpha \iff t_i \lhd \alpha \quad \text{for each } i = 1, ..., l$$

Lemma 4.4.2. *Let α and β be ordinals and let k be a positive natural number. Then for a $T^{\infty}(;)$-term t the following are true*

(i) $t \lhd \alpha \Rightarrow t \lhd \alpha + \beta$

(ii) $t \lhd \alpha \Rightarrow t \lhd \beta + \alpha$

(iii) $t \lhd \alpha \Rightarrow t \lhd k \cdot \alpha$

Proofs.

(i). It is straightforward by Lemma 4.3.11-(i) and Definition 4.4.1-(f).

(ii). By induction on structure of the term t:

Case 1. If t is a variable or constant, then

$$\begin{array}{rcll} t & \lhd & 0 & \text{by Def. } 4.4.1-(a) \\ & \prec_o & \beta & \text{by Def. } 4.3.10-(f) \\ & \preccurlyeq_o & \beta+\alpha & \text{by } (i) \end{array}$$

Case 2. If $t = \lambda v.r$ where $r \lhd \gamma$ and $\gamma + 1 \preccurlyeq_o \alpha$, then

$$r \lhd \beta + \gamma \qquad \text{by ind. hyp.}$$

hence

$$\begin{array}{rcll} \lambda v.r & \lhd & (\beta+\gamma)+1 & \text{by Def. } 4.4.1-(c) \\ & = & \beta+(\gamma+1) & \text{by Def. } 4.3.3 \\ & \preccurlyeq_o & \beta+\alpha & \text{since } \gamma+1 \preccurlyeq_o \alpha \end{array}$$

Case 3. If $t = t_1 t_2$ where $t_1 \lhd \gamma_1$, $t_2 \lhd \gamma_2$ and $\gamma_1 \preccurlyeq_o \gamma_2$ and $\gamma_2 + 1 \preccurlyeq_o \alpha$, then

$$t_1 \lhd \beta + \gamma_1 \quad \text{and} \quad t_2 \lhd \beta + \gamma_2 \qquad \text{by ind. hyp.}$$

and

$$\beta + \gamma_1 \preccurlyeq_o \beta + \gamma_2 \qquad \text{by Lemma } 4.3.11-(i)$$

hence

$$\begin{array}{rcll} t_1 t_2 & \lhd & (\beta+\gamma_2)+1 & \text{by Def. } 4.4.1-(b) \\ & = & \beta+(\gamma_2+1) & \text{by Def. } 4.3.3 \\ & \preccurlyeq_o & \beta+\alpha & \text{since } \gamma_2+1 \preccurlyeq_o \alpha \end{array}$$

Case 4. If $t =< r_i >_x$ where $\forall i \exists \delta (r_i \lhd \delta \wedge \delta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit, then

$$r_i \lhd \beta + \delta \qquad \text{by ind. hyp.}$$

and

$$\begin{array}{lcll} (\beta+\delta)+1 & \preccurlyeq_o & \beta+(\delta+1) & \text{by Def. 4.3.3} \\ & \preccurlyeq_o & \beta+\gamma_i & \text{since } \delta+1 \preccurlyeq_o \gamma_i \\ & = & (\beta+\gamma)_i & \text{by Def. 4.3.3} \end{array}$$

hence

$$\begin{array}{lcll} t =< r_i >_x & \lhd & \beta+\gamma & \text{by Def. 4.4.1} - (d) \\ & \preccurlyeq_o & \beta+\alpha & \text{since } \gamma \preccurlyeq_o \alpha \end{array}$$

Case 5. If $t = (r_1, r_2)$ where $r_j \lhd \gamma$ for each $j = 1, 2$, then

$$r_j \lhd \beta+\gamma \qquad \text{by ind. hyp.}$$

hence

$$\begin{array}{lcll} (r_1, r_2) & \lhd & \beta+\gamma & \text{by Def. 4.4.1} - (d) \\ & \preccurlyeq_o & \beta+\alpha & \text{since } \gamma \preccurlyeq_o \alpha \end{array}$$

(iii). By induction on structure of the term t:

Case 1. If t is a variable or constant, then

$$\begin{array}{lcll} t & \lhd & 0 & \text{by Def. 4.4.1} - (a) \\ & \prec_o & k \cdot \beta & \text{by Def. 4.3.10} - (f) \end{array}$$

Case 2. If $t = \lambda v.r$ where $r \lhd \beta$, then

$$r \lhd k \cdot \beta \qquad \text{by ind. hyp.}$$

hence

$$\begin{array}{lcll} \lambda v.r & \lhd & k \cdot \beta + 1 & \text{by Def. 4.4.1} - (c) \\ & \lhd & k \cdot \beta + k & \text{by Lemma 4.3.11} - (i) \\ & \preccurlyeq_o & k \cdot (\beta+1) & \text{by Def. 4.3.3} \\ & \preccurlyeq_o & k \cdot \alpha & \text{since } \beta+1 \preccurlyeq_o \alpha \end{array}$$

Case 3. If $t = t_1 t_2$ where $t_1 \lhd \beta_1$, $t_2 \lhd \beta_2$ and $\gamma_1 \preccurlyeq_o \gamma_2$, then

$$t_1 \lhd k \cdot \beta_1 \quad \text{and} \quad t_2 \lhd k \cdot \beta_2 \qquad \text{by ind. hyp.}$$

and

$$k \cdot \beta_1 \preccurlyeq_o k \cdot \beta_2 \qquad \text{by Lemma } 4.3.11 - (ii)$$

hence

$$\begin{array}{rcll} t_1 t_2 & \lhd & k \cdot \beta_2 + 1 & \text{by Def. } 4.4.1 - (b) \\ & \preccurlyeq_o & k \cdot \beta_2 + k & \text{by Lemma } 4.3.11 - (i) \\ & = & k \cdot (\beta_2 + 1) & \text{by Def. } 4.3.3 \\ & \preccurlyeq_o & k \cdot \alpha & \text{since } \gamma_2 + 1 \preccurlyeq_o \alpha \end{array}$$

Case 4. If $t =< r_i >_x$ where $\forall i \exists \beta (r_i \lhd \beta \wedge \beta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit, then

$$r_i \lhd k \cdot \beta \qquad \text{by ind. hyp.}$$

and

$$\begin{array}{rcll} (k \cdot \beta) + 1 & \preccurlyeq_o & k \cdot \beta + k & \text{by Lemma } 4.3.11 - (i) \\ & = & k \cdot (\beta) + 1) & \text{by Def. } 4.3.3 \\ & \preccurlyeq_o & k \cdot \gamma_i & \text{since } \beta + 1 \preccurlyeq_o \gamma_i \\ & = & (k \cdot \gamma)_i & \text{by Def. } 4.3.3 \end{array}$$

hence

$$\begin{array}{rcll} t =< r_i >_x & \lhd & k \cdot \gamma & \text{by Def. } 4.4.1 - (d) \\ & \preccurlyeq_o & k \cdot \alpha & \text{since } \gamma \preccurlyeq_o \alpha \end{array}$$

Case 5. If $t = (r_1, r_2)$ where $r_j \lhd \beta$ for each $j = 1, 2$, then

$$r_j \lhd k \cdot \beta \qquad \text{by ind. hyp.}$$

hence

$$\begin{array}{rcll} (r_1, r_2) & \lhd & k \cdot \beta & \text{by Def. } 4.4.1 - (d) \\ & \preccurlyeq_o & k \cdot \alpha & \text{since } \gamma \preccurlyeq_o \alpha \end{array}$$

Theorem 4.4.3. *For every finite list of functionals $\vec{F} = F_1, ..., F_l$ definable in $T(;)$ there is a term $t_{\vec{F}} = (t_{F_1}, ..., t_{F_l})$ of $T^\infty(;)$ such that for each $i = 1, ..., l$, the term t_{F_i} represents functional F_i (the value of t_{F_i} is same as value of F_i) and*

$$t_{\vec{F}} \lhd (k \cdot \omega) \cdot m$$

for some positive integers k and m. Here k is any number greater than or equal to twice the maximum length of any list of functionals defined, and m is the finite number of the different definitions in the program of $T(;)$ defining $\vec{F}$.

Proof. We are going to prove it by induction on the definition of functionals of $T(;)$. When we get the system $T^{\infty}(;)$ from $T(;)$, we replace the substitution and primitive recursion by λ-abstraction application and infinite sequence respectively. So it is enough to show that the *substitution* and *recursion* in $T(;)$ are definable in $T^{\infty}(;)$. The other clauses of definition of $T(;)$ are clearly representable.

Case (Substitution). Let $\vec{r} = (r_1, ..., r_l)$ be a sequence of $T(;)$-terms all of whose variables occur among I-variables $x_1, ..., x_p$ and O-variables $u_1, ..., u_q$. Then a new sequence of constants $\vec{F}$ is defined by *simultaneous explicit definition*

$$\vec{F}(x_1, ..., x_p; u_1, ..., u_q) = \vec{r}$$

i.e. $F_i(x_1, ..., x_p; u_1, ..., u_q) = r_i$ for each $i = 1, 2, ..., l$. If $\vec{r}$ is represented by $t_{\vec{r}}$, then $\vec{F}$ is represented by $T^{\infty}(;)$-term $t_{\vec{F}} = (t_{F_1}, ..., t_{F_l})$ which will be as follows by means of λ-*abstraction*

$$t_{F_i} = \lambda u_1 ... u_q . t_{r_i}$$

for each $i = 1, ..., l$. Where the t_i may be obtained by repeated *application* of previously defined terms, each of which is already assumed to be represented by a $T^{\infty}(;)$-term, hence when translated into $T^{\infty}(;)$ we will have $t_{q_i} \lhd (k \cdot \omega) \cdot m$. Then for each $i = 1, 2, ..., l$

$$\begin{array}{lll} \lambda u_q . t_{r_i} & \lhd & (k \cdot \omega) \cdot m + 1 \qquad \text{by Def. 4.4.1-(c)} \\ & \prec_o & (k \cdot \omega) \cdot m + k \cdot \omega \qquad \text{since } 1 \prec_o k \cdot \omega \end{array}$$

thus using this procedure for $u_{q-1}, ..., u_1, x_p, ..., x_1$ we get that

$$\lambda u_1 ... u_q . t_{r_i} \lhd (k \cdot \omega) \cdot (m + q + p)$$

So if we change m to $m + q + p$ we get the result as it is required

$$t_{F_i} \lhd (k \cdot \omega) \cdot m$$

Case (Recursion). Let I-variable x and $T(;)$-terms $\vec{G}$ and $\vec{H}$ be given. Then a new sequence of constants $\vec{F}$ is defined by *simultaneous primitive recursion*

$$\begin{cases} \vec{F}(0) &= \vec{G} \\ \vec{F}(x+1) &= \vec{H}x(\vec{F}x) \end{cases}$$

i.e. for each $i = 1, ..., l$

$$\begin{cases} F_i(0) &= G_i \\ F_i(x+1) &= H_i x(F_1 x F_2 x ... F_l x) \end{cases}$$

We assume that $\vec{G}$ and $\vec{H}$ are represented by the $T^\infty(;)$-terms $t_{\vec{G}}$ and $t_{\vec{H}}$ respectively with $t_{G_i} \lhd (k \cdot \omega) \cdot m_G$, $t_{H_i} \lhd (k \cdot \omega) \cdot m_H$ where $m = max(m_G, m_H)$.

We now define $T^\infty(;)$-terms $\vec{s}_n = (s_{1,n}, ..., s_{l,n})$ recursively as follows for each $i = 1, ..., l$

$$\begin{cases} s_{i,0} &= t_{G_i} \\ s_{i,n+1} &= t_{H_i} \bar{n}(s_{1,n})(s_{2,n})...(s_{l,n}) \end{cases}$$

The $s_{1,n}, s_{2,n}, ..., s_{l,n}$ can be obtained from $\vec{s}_n$. Hence we get

$$\begin{cases} \vec{s}_0 &= t_{\vec{G}} \\ \vec{s}_{n+1} &= t_{\vec{H}} \bar{n}(\pi_1^{l-1}\vec{s}_n)(\pi_2(\pi_1^{l-2}\vec{s}_n))...(\pi_2(\pi_1^0\vec{s}_n)) \end{cases}$$

Then $\vec{F}x = (F_1 x, ..., F_l x)$ is represented by the infinite term

$$t_{\vec{F}x} =< \vec{s}_n > x$$

by means of *infinite sequencing*. Now we shall find an ordinal which majorizes the $\vec{s}_n$. The proof runs by induction on n.

- *Base case*, $n = 0$. Then $\vec{s}_0 \lhd (k \cdot \omega) \cdot m$ since $\vec{s}_0 = t_{\vec{G}}$ and $m_G \leq m$.
- *Induction case*, assume that $s_{i,n} \lhd (k \cdot \omega) \cdot m + 2ln$. Then we first get the following

$\vec{s}_n \lhd (k \cdot \omega) \cdot m + 2ln$	by Def. 4.4.1-(e)
$\pi_i^l \vec{s}_n \lhd (k \cdot \omega) \cdot m + 2ln + l$	by Def. 4.4.1-(b)
$S \lhd 0, \quad 0 \lhd 0$	by Def. 4.4.1-(a)
$\bar{n} = \underbrace{SS...S}_{n} 0 \lhd n \preccurlyeq_o (k \cdot \omega) \cdot m + 2ln$	by Def. 4.4.1-(b)

Hence we now can majorize $\vec{s}_{n+1}$ as follows

$$t_{\vec{H}}\bar{n} \lhd (k\cdot\omega)\cdot m + 2ln + 1 \qquad \text{by Def. 4.4.1-(b)}$$

and then by Definition 4.4.1-(b)

$$\begin{aligned} t_{\vec{H}}\bar{n}(\pi_1^{l-1}\vec{s}_n) &\lhd (k\cdot\omega)\cdot m + 2ln + l + 1 \\ t_{\vec{H}}\bar{n}(\pi_1^{l-1}\vec{s}_n)(\pi_2(\pi_1^{l-2}\vec{s}_n)) &\lhd (k\cdot\omega)\cdot m + 2ln + l + 2 \\ &\vdots \\ t_{\vec{H}}\bar{n}(\pi_1^{l-1}\vec{s}_n)(\pi_2(\pi_1^{l-2}\vec{s}_n))...(\pi_2(\pi_1^{0}\vec{s}_n)) &\lhd (k\cdot\omega)\cdot m + 2ln + l + l \end{aligned}$$

that is, we have got $\vec{s}_{n+1} \lhd (k\cdot\omega)\cdot m + 2l(n+1)$.

Therefore by induction on n we have $\vec{s}_n \lhd (k\cdot\omega)\cdot m + 2ln$ where by the Definition 4.3.10

$$\begin{aligned} (k\cdot\omega)\cdot m + 2ln &\preccurlyeq_o (k\cdot\omega)\cdot m + 2ln + 1 \\ &\preccurlyeq_o (k\cdot\omega)\cdot m + 2ln + 2l \end{aligned}$$

we chose that $\omega = sup(n+1)$, then

$$sup((k\cdot\omega)\cdot m + 2l(n+1)) = (k\cdot\omega)\cdot m + 2l\cdot\omega$$

If we change k to $2l$ we get

$$< \vec{s}_n > x \lhd (k\cdot\omega)\cdot(m+1) \qquad \text{by Def. } 4.4.1-(d)$$

So if we also change m to $m+1$ we get the result as it is required

$$t_{\vec{F}_x} \lhd (k\cdot\omega)\cdot m$$

which completes the proof.

4.5 Ordinal Bounding

In this section we shall prove that the complexity of normalisation is at most super-exponential.

Lemma 4.5.1. *Let t_1, t_2 be $T^\infty(;)$-terms such that $t_1 \lhd \alpha_1$ and $t_2 \lhd \alpha_2$. Then*

$$t_1[t_2/v] \lhd \alpha_2 + \alpha_1$$

Proof. We shall prove it by induction on the construction of t_1:

Case 1. If $t_1 = u$, then this case falls into one of the following subcases according to u:

1.1. If $u = v$, then

$$\begin{array}{rcll} v[t_2/v] & = & t_2 & \text{by Def. } 4.1.7 - (a) \\ & \lhd & \alpha_2 & \text{given} \\ & \preccurlyeq_o & \alpha_2 + \alpha_1 & \text{by Lemma } 4.3.11 - (i) \end{array}$$

1.2. If $u \neq v$, then

$$\begin{array}{rcll} u[t_2/v] & = & u & \text{by Def. } 4.1.7 - (b) \\ & \lhd & 0 & \text{by Def. } 4.4.1 - (a) \\ & \preccurlyeq_o & \alpha_2 + \alpha_1 & \text{by Remark } 4.3.8 - 2. \end{array}$$

Case 2. If $t_1 = \lambda u.r$ and $r \lhd \beta$ where $\beta + 1 \preccurlyeq_o \alpha_1$, then this case also falls into one of the following subcases according to u:

2.1. If $u = v$, then

$$(\lambda v.r)[t_2/v] = \lambda v.r \qquad \text{by Def. } 4.1.7 - (d)$$

Hence

$$\begin{array}{rcll} \lambda u.r & \lhd & \beta + 1 & \text{by Def. } 4.4.1 - (c) \\ \lambda u.r & \lhd & \alpha_2 + \beta + 1 & \\ & \preccurlyeq_o & \alpha_2 + \alpha_1 & \text{by Lemma } 4.3.11 - (i) \end{array}$$

2.2. If $u \neq v$ and $u \notin FV(t_2)$ or $u \notin FV(r)$, then

$$(\lambda u.r)[t_2/v] = \lambda u.r[t_2/v] \qquad \text{by Def. } 4.1.7 - (e)$$

where

$$r[t_2/v] \lhd \alpha_2 + \beta \qquad \text{by ind. hyp.}$$

Hence

$$\begin{aligned} \lambda u.r[t_2/v] &\lhd \alpha_2 + \beta + 1 && \text{by Def. } 4.4.1 - (c) \\ &\preccurlyeq_o \alpha_2 + \alpha_1 && \text{since } \beta + 1 \preccurlyeq_o \alpha_1 \end{aligned}$$

2.3. If $u \neq v$ and $u \in FV(t_2)$ and a new variable $w \notin FV(t_2) \cup FV(r)$, then

$$(\lambda u.r)[t_2/v] = \lambda w.r[w/u][t_2/v] \qquad \text{by Def. } 4.1.7 - (f)$$

where

$$r[w/u] \lhd \beta \qquad \text{by ind. hyp. and } w \lhd 0$$

Hence

$$(r[w/u])[t_2/v] \lhd \alpha_2 + \beta \qquad \text{by ind. hyp.}$$

and then

$$\begin{aligned} \lambda w.r[w/u][t_2/v] &\lhd \alpha_2 + \beta + 1 && \text{by Def. } 4.4.1 - (c) \\ &\preccurlyeq_o \alpha_2 + \alpha_1 && \text{since } \beta + 1 \preccurlyeq_o \alpha_1 \end{aligned}$$

Case 3. If $t_1 =< r_i >_x$ and $\forall i \exists \beta (r_i \lhd \beta \wedge \beta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit and $\gamma \preccurlyeq_o \alpha_1$, then

$$< r_i >_x [t_2/v] =< r_i[t_2/v] >_x \qquad \text{by Def. } 4.1.7 - (g)$$

where fro each i there is a β such that

$$\begin{aligned} r_i[t_2/v] &\lhd \alpha_2 + \beta && \text{by ind. hyp.} \\ &\prec_o \alpha_2 + \beta + 1 && \text{by Def. } 4.3.10 - (a) \\ &\preccurlyeq_o \alpha_2 + \gamma_i && \text{since } \beta + 1 \preccurlyeq_o \gamma_i \\ &= (\alpha_2 + \gamma)_i && \text{by Def. } 4.3.3 \end{aligned}$$

where $sup(\alpha_2 + \gamma)_i = \alpha_2 + \gamma$. Hence

$$\begin{aligned} < r_i[t_1/v] > x &\lhd \alpha_2 + \gamma && \text{by Def. } 4.4.1 - (d) \\ &\preccurlyeq_o \alpha_2 + \alpha_1 && \text{since } \gamma \preccurlyeq_o \alpha_1 \end{aligned}$$

Case 4. If $t_1 = rs$ and $r \lhd \beta_1$, $s \lhd \beta_2$, $\beta_1 \preccurlyeq_o \beta_2$ where $\beta_2 + 1 \preccurlyeq_o \alpha_1$, then

$$(rs)[t_2/v] = (r[t_2/v])(s[t_2/v]) \qquad \text{by Def. } 4.1.7 - (c)$$

where

$$r[t_2/v] \lhd \alpha_2 + \beta_1, \qquad s[t_2/v] \lhd \alpha_2 + \beta_2 \qquad \text{by ind. hyp.}$$

and

$$\alpha_2 + \beta_1 \preccurlyeq_o \alpha_2 + \beta_2 \qquad \text{by Lemma } 4.3.11 - (i)$$

Hence

$$\begin{array}{lll} (r[t_2/v])(s[t_2/v]) & \lhd & \alpha_2 + \beta_2 + 1 \qquad \text{by Def. } 4.4.1 - (b) \\ & \preccurlyeq_o & \alpha_2 + \alpha_1 \qquad \text{since } \beta_2 + 1 \preccurlyeq_o \alpha_1 \end{array}$$

Case 5. If $t_1 = (r_1, r_2)$ and $r_1 \lhd \beta$, $r_2 \lhd \beta$ where $\beta \preccurlyeq_o \alpha_1$, then

$$(r_1, r_2)[t_2/v] = (r_1[t_2/v], r_2[t_2/v]) \qquad \text{by Def. } 4.1.7 - (h)$$

where

$$r_1[t_2/v] \lhd \alpha_2 + \beta, \qquad r_2[t_2/v] \lhd \alpha_2 + \beta \qquad \text{by ind. hyp.}$$

Hence

$$\begin{array}{lll} (r_1[t_2/v], r_2[t_2/v]) & \lhd & \alpha_2 + \beta \qquad \text{by Def. } 4.4.1 - (f) \\ & \preccurlyeq_o & \alpha_2 + \alpha_1 \qquad \text{since } \beta \preccurlyeq_o \alpha_1 \end{array}$$

Lemma 4.5.2. *Let t_1, t_2 be $T^\infty(;)$-terms with $t_1 \lhd \alpha_1$ and $t_2 \lhd \alpha_2$. Then if $t_1 t_2$ is a redex we have*

$$(t_1 t_2)^* \lhd 2 \cdot \alpha_2 + 2 \cdot \alpha_1$$

Proof. We shall prove it by induction on the construction of t_1. Since $t_1 t_2$ is a redex, there are just three main cases:

Case 1. If $t_1 = \pi_j$ where $j = 1, 2$, then $t_1 t_2$ is a projection-redex. So there are two subcases to consider, and we deal with them by sub-induction on t_2:

1.1. If $t_2 = (r_1, r_2)$ with $r_j \lhd \beta$ where $j = 1, 2$ and $\beta \preccurlyeq_o \alpha_2$, then

$$\begin{array}{lll} (\pi_j(r_1, r_2))^* & = & r_j \qquad \text{by Def. } 4.2.6 - (d) \\ & \lhd & \beta \qquad \text{given} \\ (\pi_j(r_1, r_2))^* & \lhd & \alpha_2 \qquad \text{since } \beta \preccurlyeq_o \alpha_2 \\ (\pi_j(r_1, r_2))^* & \lhd & 2 \cdot \alpha_2 \qquad \text{Lemma } 4.4.2 - (iii) \\ & \preccurlyeq_o & 2 \cdot \alpha_2 + 2 \cdot \alpha_1 \qquad \text{by Def. } 4.4.1 - (f) \end{array}$$

1.2. If $t_2 =< r_i >_x$ with $\forall i \exists \beta (r_i \lhd \beta \wedge \beta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit and $\gamma \preccurlyeq_o \alpha_2$, then

$$(\pi_j < r_i >_x)^* =< (\pi_j r_i)^* >_x \qquad \text{by Def. } 4.2.6 - (c)$$

where the term $\pi_j r_i$ is a redex or not, for $j = 1, 2$ and for every $i \in \mathbb{N}$.

– If it is a redex, then

$$\begin{array}{lll} (\pi_j r_i)^* & \lhd & 2 \cdot \beta + 2 \cdot 0 \qquad \text{by ind. hyp.} \\ & \prec_o & 2 \cdot \beta + 1 \qquad \text{by Def. } 4.3.10 - (a) \end{array}$$

– If it is not a redex, then we get

$$(\pi_j r_i)^* = \pi_j r_i \qquad \text{by Def. 4.2.6}$$

where

$$\begin{aligned} \pi_j &\lhd 0 && \text{by Def. 4.3.10} - (a) \\ \pi_j &\lhd 2 \cdot \beta && \text{by Lemma 4.4.2} - (iii) \end{aligned}$$

and

$$\begin{aligned} r_i &\lhd \beta && \text{given} \\ r_i &\lhd 2 \cdot \beta && \text{by Lemma 4.4.2} - (iii) \end{aligned}$$

then

$$\pi_j r_i \lhd 2 \cdot \beta + 1 \qquad \text{by Def. 4.3.10} - (b)$$

So for the both sub-cases, we get that $(\pi_j r_i)^* \lhd 2 \cdot \beta + 1$. Here $2 \cdot \beta + 1 + 1 = 2 \cdot (\beta + 1) \preccurlyeq_o 2 \cdot \gamma_i$, and $sup(2 \cdot \gamma_i) = 2 \cdot \gamma$. Then

$$\begin{aligned} < (\pi_j r_i)^* >_x &\lhd 2 \cdot \gamma && \text{by Def. 4.3.10} - (b) \\ &\preccurlyeq_o 2 \cdot \alpha_2 && \text{since } \gamma \preccurlyeq_o \alpha_2 \\ &\preccurlyeq_o 2 \cdot \alpha_2 + 2 \cdot \alpha_1 && \text{Lemma 4.3.11} - (i) \end{aligned}$$

Case 2. If $t_1 = \lambda v.r$ with $r \lhd \beta$ where $\beta + 1 \preccurlyeq_o \alpha_1$, then $t_1 t_2$ is an abstraction-redex.

$$\begin{aligned} ((\lambda v.r)t_2)^* &= r[t_2/v] && \text{by Def. 4.2.6} - (b) \\ &\lhd \alpha_2 + \beta && \text{by Lemma 4.5.1} \\ ((\lambda v.r)t_2)^* &\lhd 2 \cdot \alpha_2 + 2 \cdot \beta && \text{by Lemma 4.3.11} - (ii) \\ &\prec_o 2 \cdot \alpha_2 + 2 \cdot \beta + 2 && \text{by Def. 4.3.10} - (a) \\ &= 2 \cdot \alpha_2 + 2 \cdot (\beta + 1) \\ &\preccurlyeq_o 2 \cdot \alpha_2 + 2 \cdot \alpha_1 && \text{since} \quad \beta + 1 \preccurlyeq_o \alpha_1 \end{aligned}$$

Case 3. If $t_1 =< r_i >_x$ with $\forall i \exists \beta (r_i \lhd \beta \wedge \beta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit and $\gamma \preccurlyeq_o \alpha_1$, then $t_1 t_2$ is a sequence-redex.

$$(< r_i >_x t_2)^* =< (r_i t_2)^* >_x \qquad \text{by Def. 4.2.6} - (c)$$

In this case, we shall look at whether the term $r_i t_2$ is a redex or not, for $j = 1, 2$ and for every $i \in \mathbb{N}$.

– If it is a redex, then

$$\begin{array}{lll} (r_i t_2)^* & \lhd & 2\cdot\alpha_2 + 2\cdot\beta \qquad \text{by ind. hyp.} \\ & \prec_o & 2\cdot\alpha_2 + 2\cdot\beta + 1 \qquad \text{by Def.4.3.10} - (a) \end{array}$$

– If it is not a redex, then

$$(r_i t_2)^* \;=\; r_i t_2 \qquad \text{by Def. 4.2.6}$$

where

$$\begin{array}{lll} r_i & \lhd & \beta \qquad \text{by Def. 4.3.10} - (a) \\ r_i & \lhd & 2\cdot\beta \qquad \text{by Lemma 4.4.2} - (iii) \\ r_i & \lhd & 2\cdot\alpha_2 + 2\cdot\beta \qquad \text{by Lemma 4.4.2} - (ii) \end{array}$$

and

$$\begin{array}{lll} t_2 & \lhd & \alpha_2 \qquad \text{given} \\ t_2 & \lhd & 2\cdot\alpha_2 \qquad \text{by Lemma 4.4.2} - (iii) \\ & \preccurlyeq_o & 2\cdot\alpha_2 + 2\cdot\beta \qquad \text{by Lemma 4.3.11} - (ii) \end{array}$$

then

$$r_i t_2 \lhd 2\cdot\alpha_2 + 2\cdot\beta + 1 \qquad \text{by Def. 4.3.10} - (b)$$

So for the both sub-cases, we get that $(r_i t_2)^* \lhd 2\cdot\alpha_2 + 2\cdot\beta + 1$. Here $2\cdot\alpha_2+2\cdot\beta+1+1 = 2\cdot\alpha_2+2\cdot(\beta+1) \preccurlyeq_o 2\cdot\alpha_2+2\cdot\gamma_i$, and $sup(2\cdot\alpha_2+2\cdot\gamma_i) = 2\cdot\alpha_2 + 2\cdot\gamma$. Then

$$\begin{array}{lll} < (r_i t_2)^* >_x & \lhd & 2\cdot\alpha_2 + 2\cdot\gamma \qquad \text{by Def. 4.3.10} - (b) \\ & \preccurlyeq_o & 2\cdot\alpha_2 + 2\cdot\alpha_1 \qquad \text{since } \gamma \preccurlyeq_o \alpha_1 \end{array}$$

Theorem 4.5.3 (Ordinal Bounding). *Let t be any $T^\infty(;)$-term with positive finite rank such that $t \lhd \alpha$. Then $t \mapsto t'$ where the rank of t' is at least one less then the rank of t, and the following result holds*

$$t' \lhd \varphi(\alpha)$$

where φ is the function defined in Definition 4.3.12.

Proof. We can prove it by induction over the construction of t:

Case 1. If t is a variable or constant where $0 \preccurlyeq_o \alpha$, then

$$\begin{array}{rcll} t' & = & t & \text{by Def. } 4.2.5-(a) \\ & \lhd & 0 & \text{by Def. } 4.4.1-(a) \\ & \prec_o & \varphi(\alpha) & \text{by Remark } 4.3.8-2 \end{array}$$

Case 2. If $t = \lambda v.s$ with $s \lhd \beta$ where $\beta + 1 \preccurlyeq_o \alpha$, then

$$t' = \lambda v.s' \qquad \text{by Def. } 4.2.5-(b)$$

where

$$s' \lhd \varphi(\beta) \qquad \text{by ind. hyp.}$$

Hence

$$\begin{array}{rcll} \lambda v.s' & \lhd & \varphi(\beta)+1 & \text{by Def. } 4.4.1-(c) \\ & \prec_o & \varphi(\beta+1) & \text{by Lemma } 4.3.13-(ii) \\ & \preccurlyeq_o & \varphi(\alpha) & \text{by Lemma } 4.3.13-(iv) \end{array}$$

Case 3. If $t =< r_i >_x$ with $\forall i \exists \beta (r_i \lhd \beta \wedge \beta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit and $\gamma \preccurlyeq_o \alpha$, then for every i

$$t' =< r_i' > x \qquad \text{by Def. } 4.2.5-(c)$$

where

$$r_i' \lhd \varphi(\beta) \qquad \text{by ind. hyp.}$$

and

$$\begin{array}{rcll} \varphi(\beta)+1 & \prec_o & \varphi(\beta+1) & \text{by Lemma } 4.3.13-(ii) \\ & \preccurlyeq_o & \varphi(\gamma_i) & \text{by Lemma } 4.3.13-(iv) \end{array}$$

and

$$sup\ \varphi(\gamma_i) \ = \ \varphi(\gamma) \qquad \text{by Def. } 4.3.3$$

Hence

$$\begin{array}{rcll} < r_i' > x & \lhd & \varphi(\gamma) & \text{by Def. } 4.4.1-(d) \\ & \preccurlyeq_o & \varphi(\alpha) & \text{by Lemma } 4.3.13-(iv) \end{array}$$

Case 4. If $t = t_1 t_2$ with $t_1 \lhd \beta_1$, $t_2 \lhd \beta_2$, $\beta_1 \preccurlyeq_o \beta_2$ where $\beta_2 + 1 \preccurlyeq_o \alpha$, then

$$t' = (t'_1 t'_2)^* \qquad \text{by Def. } 4.2.5 - (d)$$

where

$$t'_1 \lhd \varphi(\beta_1), \quad t'_2 \lhd \varphi(\beta_2) \qquad \text{by ind. hyp.}$$

Hence

$$\begin{array}{lll} (t'_1 t'_2)^* \lhd 2 \cdot \varphi(\beta_2) + 2 \cdot \varphi(\beta_1) & & \text{by Lemma } 4.5.2 \\ \preccurlyeq_o \; 2 \cdot \varphi(\beta_2) + 2 \cdot \varphi(\beta_2) & & \text{since} \quad \varphi(\beta_1) \preccurlyeq_o \varphi(\beta_2) \\ (t'_1 t'_2)^* \lhd \varphi(\beta_2) + 2 \cdot \varphi(\beta_2) + 2 \cdot \varphi(\beta_2) & & \text{by Lemma } 4.3.11 - (ii) \\ = \; \varphi(\beta_2 + 1) & & \text{by Def. } 4.3.12 \\ \preccurlyeq_o \; \varphi(\alpha) & & \text{by Lemma } 4.3.13 - (iv) \end{array}$$

Case 5. If $t = (t_1, t_2)$ with $t_i \lhd \beta$ for each $i = 1, 2$, where $\beta \preccurlyeq_o \alpha$, then

$$t' = (t'_1, t'_2) \qquad \text{by Def. } 4.2.5 - (d)$$

where for each $i = 1, 2$

$$t'_i \lhd \varphi(\beta) \qquad \text{by ind. hyp.}$$

Hence

$$\begin{array}{lll} (t'_1, t'_2) \lhd \varphi(\beta) & & \text{by Def. } 4.4.1 - (e) \\ \preccurlyeq_o \; \varphi(\alpha) & & \text{by Lemma } 4.3.13 - (iv) \end{array}$$

Theorem 4.5.4. *Assume that t be any $T^\infty(;)$-term which has a positive finite rank k and majorized by ordinal α. If the term t reduces to a normal form $t^{[k]}$ after k-many times application of rank-reduction, then the complexity of normalisation is majorized by the k-th iterate of φ, thus*

$$t^{[k]} \lhd \varphi^k(\alpha)$$

Proof. By the definition of rank-reduction, each reduction reduces the rank by at least one, that is,

$$R(t^{[n+1]}) < R(t^{[n]}) \qquad \text{by Lemma } 4.2.10$$

So the proof runs by induction on rank k:

– *Base case*, if $k = 1$, then $(t^{[0]})' = t'$ which gives that

$$t' \lhd \varphi(\alpha) \qquad \text{by Lemma 4.5.3}$$

– *Induction case*, assume that the reduction $(t^{[k-1]})' = t^{[k]}$ with $t^{[k-1]} \lhd \varphi^{k-1}(\alpha)$. Then

$$\begin{aligned} t^{[k]} &\lhd \varphi(\varphi^{k-1}(\alpha)) \qquad \text{by ind. hyp.} \\ &= \varphi^k(\alpha) \end{aligned}$$

This completes the proof as required.

4.6 Complexity

In this final section of this chapter we shall prove that every $T(;)$-definable function is bounded by a super-exponential function exp_5^k for some fixed k. We then get the result that every $T(;)$-definable function is elementary. This follows from the embedding of $T(;)$ into $T^\infty(;)$ and the normalisation of $T^\infty(;)$-terms.

Lemma 4.6.1. *Suppose that $t(\vec{x};\vec{a})$ is a normal term of type o in $T^\infty(;)$ which contains only free variables $\vec{x}$ of ground type ι and $\vec{a}$ of ground type o. Then the term t must have one of the following forms:*

(I) *A variable a or the constant 0*

(II) *Applications: St_0 or Pt_0 or $Ct_0t_1t_2$*

(III) *An infinite sequence $< t_i >_x$*

where $t_0, t_1, ..., t_i, ...$ are themselves normal of type o.

Proof. By induction on construction of term t of type o.

Case 1. If t is a variable a of type o or the constant 0 of type o, then we obtain the form (I). But t can not be variable x of type ι and can not be the constants S or P or C or π_j because their types are wrong.

Case 2. The term t can not be an abstraction $\lambda v.s$ because the type is wrong.

Case 3. If t be an infinite sequence $< t_i >_x$ of type o, then we obtain the form (III).

Case 4. The term t can not be a pair (t_1, t_2) because the type is wrong.

Case 5. If t is an application $t_0t_1t_2...t_n$ of type o where t_0 not an application, then we shall look at the following *sub-cases* by induction on the construction of term t_0:

5.1. The term t_0 must have function-type. So t_0 can not be a variable because t contains no variables of function-type. However t_0 might be S or P or C. So we obtain the terms St_1, Pt_1 and $Ct_1t_2t_3$ which gives the form (II). The term t contains also the constants π_j of function-type but it can not be t_0 see *Case* 5.5.

5.2. The term t_0 can not be an abstraction $\lambda v.s$ because then t would not be normal.

5.3. The term t_0 can not be an infinite sequence $< t_i >_x$ because then t would not be normal.

5.4. The term t_0 can not be a pair (r_1, r_2) because the type is wrong.

5.5. It remains to show that t_0 can not be a constant π. Here the π stands for either π_1 or π_2. If $t_0 = \pi$, then we have to look at t_1.

The term t_1 can not be a pair or a sequence because then t would not be normal; and can not be a constant or a variable or λ-abstraction because the type would be wrong. The only possibility is that t_1 is also an application term $s_0s_1s_3...s_m$ of product-type where $m > 0$ as before and s_0 is not an application. We have to continue the procedure by induction on the construction of term s_0.

Again, there is only one possibility, namely $s_0 = \pi$. Then by the same argument, s_1 must be an application beginning with a π. We can thus continue indefinitely, contradicting the well foundedness of the term t.

Remark 4.6.2. Any (even non-recursive) function $f : \mathbb{N} \to \mathbb{N}$ is representable in $T^\infty(;)$ by a normal term $< n_i >_x$ where $n_i = f(i)$. However, the uniformity of the embedding of $T(;)$ into $T^\infty(;)$ produces a term t_F representing each $T(;)$-definable functional F, which is clearly intuitively "*computable*". In what follows we shall not make this notion of "computable" any more precise, but we shall show how the foregoing results provide slow-growing bounds on their computation-steps.

Definition 4.6.3 (Computation Function). Let $t(\vec{x};\vec{a})$ be a computable term, for example one which arises under the embedding F to t_F. Then the *valuation function* $\mathcal{V}t[\vec{x} := \vec{n}; \vec{a} := \vec{m}]$ needs a certain number of steps to compute, say by a register machine. This number of steps, denoted $\#t(\vec{n};\vec{m})$ can be defined by induction over the constraction of the term.

(a) $t = 0$ or $a_i \quad \Rightarrow \quad \#t(\vec{n};\vec{m}) = 1$

(b) $t = St_0$ or $Pt_0 \Rightarrow \quad \#t(\vec{n};\vec{m}) = \#t_0(\vec{n};\vec{m}) + 1$

(c) $t = Ct_0t_1t_2 \quad \Rightarrow \quad \#t(\vec{n};\vec{m}) = \#t_0(\vec{n};\vec{m}) + max(\#t_1(\vec{n};\vec{m}), \#t_2(\vec{n};\vec{m})) + 1$

(d) $t =< t_i >_{x_j} \quad \Rightarrow \quad \#t(\vec{n};\vec{m}) = \#t_{n_j}(\vec{n};\vec{m}) + 1$

Definition 4.6.4 (φ-Majorization Relation). Let t be a term of $T^{\infty}(;)$ and let φ be as in Definition 4.3.12. Then the *φ-majorization relation* $t \lhd_\varphi \varphi(\alpha)$ is defined inductively as follows:

(a) t is a variable or constant $\quad \Rightarrow \quad t \lhd_\varphi \varphi(\alpha)$ for any α.

(b) $r \lhd_\varphi \varphi(\beta), \quad \beta + 1 \preccurlyeq_o \alpha \quad \Rightarrow \quad \lambda v.r \lhd_\varphi \varphi(\alpha)$

(c) $t_1 \lhd_\varphi \varphi(\alpha_1), \quad t_2 \lhd_\varphi \varphi(\alpha_2), \quad \alpha_1 \preccurlyeq_o \alpha_2 \prec_o \alpha_2 + 1 \preccurlyeq_o \alpha \quad \Rightarrow \quad t_1t_2 \lhd_\varphi \varphi(\alpha)$

(d) $\forall i \exists \beta(t_i \lhd_\varphi \varphi(\beta) \wedge \beta + 1 \preccurlyeq_o \gamma_i), \quad \gamma \preccurlyeq_o \alpha \quad \Rightarrow \quad < t_i > x \lhd_\varphi \varphi(\alpha)$

(e) $t_k \lhd_\varphi \varphi(\alpha), \quad k = 1, 2 \quad \Rightarrow \quad (t_1, t_2) \lhd_\varphi \varphi(\alpha)$

Note 4.6.5. The proof of the Theorem 4.5.3 actually gives that

$$t \lhd \alpha \quad \Rightarrow \quad t' \lhd_\varphi \varphi(\alpha)$$

Theorem 4.6.6 (Complexity Theorem). *Let $t(\vec{x};\vec{a})$ be a normal computable term of type o, let G_n be the slow-growing function, and let $n = max(\vec{n})$. Then*

$$t \lhd_\varphi \varphi(\alpha) \quad \Rightarrow \quad \#t(\vec{n};\vec{m}) \leq G_n(\varphi(\alpha))$$

Proof. By the Lemma 4.6.1 the term t is built up from one of the following forms: $a, \quad 0, \quad St_0, \quad Pt_0, \quad Ct_0t_1t_2, \quad < t_i >_x$. So it is enough to consider these forms:

Case 1. If $t = 0$ or a_i, then

$$\begin{aligned} \#t(\vec{n};\vec{m}) &= 1 && \text{by Def. 4.6.3-(a)} \\ &\leq G_n(\varphi(\alpha)) \end{aligned}$$

Case 2. If $t = St_0$ or Pt_0 where $t_0 \lhd_\varphi \varphi(\beta)$ and $\beta + 1 \preccurlyeq_o \alpha$, then

$$\begin{aligned} \#t(\vec{n};\vec{m}) &= \#t_0(\vec{n};\vec{m}) + 1 && \text{by Def. } 4.6.3-(b) \\ &\leq G_n(\varphi(\beta)) + 1 && \text{by ind. hyp.} \\ &= G_n(\varphi(\beta) + 1) && \text{by Def. 4.3.6} \\ &\leq G_n(\varphi(\beta + 1)) && \text{by Lemma 4.3.13} - (ii) \\ &\leq G_n(\varphi(\alpha)) && \text{since} \quad \beta + 1 \preccurlyeq_o \alpha \end{aligned}$$

Case 3. If $t = Ct_0t_1t_2$ where $t_k \lhd_\varphi \varphi(\beta)$ for each $k = 0, 1, 2$ and $\beta + 1 \preccurlyeq_o \alpha$, then

$$\begin{aligned} & \#t(\vec{n};\vec{m}) \\ = \ & \#t_0(\vec{n};\vec{m}) + max(\#t_1(\vec{n};\vec{m}), \#t_2(\vec{n};\vec{m})) + 1 && \text{by Def. } 4.6.3-(c) \\ \leq \ & G_n(\varphi(\beta)) + G_n(\varphi(\beta)) + 1 && \text{by ind. hyp.} \\ \leq \ & G_n(\varphi(\beta)) + 2 \cdot G_n(\varphi(\beta)) + 2 \cdot G_n(\varphi(\beta)) \\ = \ & G_n(\varphi(\beta) + 2 \cdot \varphi(\beta) + 2 \cdot \varphi(\beta)) && \text{by Def. 4.3.6} \\ = \ & G_n(\varphi(\beta + 1)) && \text{by Def. 4.3.12} \\ \leq \ & G_n(\varphi(\alpha)) && \text{since} \quad \beta + 1 \preccurlyeq_o \alpha \end{aligned}$$

Case 4. Let $t =< t_i >_{x_j}$ where $\forall i \exists \beta (t_i \lhd_\varphi \varphi(\beta) \wedge \beta + 1 \preccurlyeq_o \gamma_i)$ where γ is a limit and $\gamma \preccurlyeq_o \alpha$, then for $i = n_j$,

$$\begin{aligned} \#t(\vec{n};\vec{m}) &= \#t_{n_j}(\vec{n};\vec{m}) + 1 && \text{by Def.4.6.3} - (d) \\ &\leq G_n(\varphi(\beta)) + 1 && \text{by ind. hyp.} \\ &= G_n(\varphi(\beta) + 1) && \text{by Def. 4.3.6} \\ &\leq G_n(\varphi(\beta + 1)) && \text{by Lemma 4.3.13} - (ii) \\ &\leq G_n(\varphi(\gamma_{n_j})) && \text{since} \quad \beta + 1 \preccurlyeq_o \gamma_{n_j} \\ &\leq G_n(\varphi(\gamma)) && \text{since} \quad n_j \leq n \\ &\leq G_n(\varphi(\alpha)) && \text{since} \quad \varphi(\gamma) \preccurlyeq_o \varphi(\alpha) \end{aligned}$$

Corollary 4.6.7. *Assume that $t(\vec{x};\vec{a})$ be a computable term and let $n = max(\vec{n})$ and $m = max(\vec{m})$. Then*

$$t \lhd_\varphi \varphi(\alpha) \quad \Rightarrow \quad \mathcal{V}t[\vec{x} := \vec{n}; \vec{a} := \vec{m}] \leq m + G_n(\varphi(\alpha))$$

Proof. By induction on the term t same style proof as in provious Theorem 4.6.6.

Theorem 4.6.8. *Every number-theoretic function f definable in $T(;)$ is elementary.*

Proof. Putting all the foregoing work together, we obtain a $T^\infty(;)$-term t and then a normal $t^{[k]}$ representing $f(\vec{n},\vec{m})$ such that

Step 1. $t \lhd (p \cdot \omega) \cdot q$ for some constant p and q, by Embedding Theorem 4.4.3.

Step 2. $t \mapsto t' \quad \Rightarrow \quad t' \lhd \varphi((p \cdot \omega) \cdot q)$ by Ordinal Bounding Theorem 4.5.3.

Step 3. $t \mapsto t^{[k]}$, $t^{[k]}$ is a normal term, by Normalisation Theorem 4.2.11.

Step 4. $t^{[k]} \lhd \varphi^k((p \cdot \omega) \cdot q)$ by Theorem 4.5.4, where $k = R(t)$.

Step 5. $\#t^{[k]}(\vec{n};\vec{m}) \leq G_n(\varphi^k(\alpha))$ by Complexity Theorem 4.6.6,
where $\alpha = (p \cdot \omega) \cdot q$.

Then $f(\vec{n},\vec{m})$ is computable with in a number of steps bounded by

$$G_n(\varphi^k((p \cdot \omega) \cdot q)) = 5^{5^{\cdot^{\cdot^{5(p(n+1)q)}}}} \qquad (k \text{ many } 5)$$

which is an elementary function of $\vec{n}$. Since every function computable in an elementary-bounded number of steps is itself elementary, we therefore have

$$f \in \mathcal{E}^3.$$

Bibliography

[1] BECKMANN, A., AND WEIERMANN, A. Characterizing the elementary recursive functions by a fragment of Gödel's T. Wilhelms-Universität Munster, Germany. to appear.

[2] BELLANTONI, S., AND COOK, S. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity 2* (1992), 97–110.

[3] BELLANTONI, S. J. *Predicative recursion and computational complexity.* PhD thesis, Department of Computer Science, University of Toronto, 1992.

[4] BÖRGER, E. *Computability, Complexity, Logic.* Elsevier, Volum 128, 1989.

[5] BUSS, S. *Bounded Arithmetic.* Bibliopolis, Naples, 1986.

[6] COBHAM, A. The intrinsic computational difficulty of functions. In *Proceedings of the Internatinal Conference on Logic* (1962), Y. Bar Hillel, Ed., North-Holland, pp. 24–30.

[7] CSILLAG, P. Eine Bemerkung zur Auflösung der eingeschachtelten Rekursion. *Acta Sci. Math. Szeged 11* (1947), 169–173.

[8] FAIRTLOUGH, M., AND WAINER, S. S. Hierarchies of provably recursive function. In *Handbook of Proof Theory* (1998), S. R. Buss, Ed., Elsevier Science, pp. 149–207.

[9] GÖDEL, K. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes (also appears, with English translation, in Feferman, S. et al Eds.

Kurt Gödel: Collected works, Vol 2, oxford university press, 1990). *Dialectica*, 12 (1958), 280–287.

[10] GRZEGORCZYK, A. *Some classes of recursive functions.* Rozprawy Mate, No. IV, Warsaw, 1953.

[11] HANDLEY, W. G., AND WAINER, S. S. Complexity of primitive recursion. In *Computational Logic* (NATO ISI Series F, 1999), U. Berger and H. Schwichtenberg, Eds., vol. 165, Springer, pp. 273–300.

[12] HINDLEY, J. R., AND SELDIN, J. P. *Introduction to Combinators and λ-calculus.* Cambridge University Press, Cambridge, 1986.

[13] KALMAR, L. Egyszerü példa eldönthetetlen aritmetikai problémára. *Mates és Fizikai Lapok 50* (1943), 1–23.

[14] LEIVANT, D. Subrecursion and lambda representation over free algebras. In *Feasible Mathematics* (1990), S. Buss and P. Scott, Eds., Birkhauser-Boston, pp. 281–291.

[15] LEIVANT, D. Intrinsic theories and computational complexity. In *Logic and Computational Complexity* (1995), D. Leivant, Ed., vol. 960, Springer, LNCS, pp. 177–194.

[16] MENDELSON, E. *Introduction to Mathematical Logic.* Wadsworth, California, 1987.

[17] MINTS, G. Quantifier-free and one-quantifier systems. *J. Soviet Math. 1* (1973), 71–84.

[18] OSTRIN, G. E. *Proof Theories of Low Sub-recursive Classes.* PhD thesis, Department of Pure Mathematics, University of Leeds, 1999.

[19] PARSONS, C. On n-quantifier induction. *J. Symbolic Logic 37* (1972), 466–482.

[20] PÉTER, R. *Recursive Functions.* Academic Press, New York, 1967.

[21] ROSE, H. E. *Subrecursion.* Clarendon Press, Oxford, 1984.

[22] SCHWICHTENBERG, H. Elimination of higher type levels in definitions of primitive recursive functionals by means of transfinite recursion. In *Logic Colloquiume '73* (1975), H. E. Rose and J. C. Shepherdson, Eds., North Holland, pp. 279–304.

[23] SHOENFIELD, J. R. *Mathematical Logic.* Addison-Wesley Publishing Company, 1967.

[24] SIMMONS, H. The realm of primitive recursion. *Archive for Mathematical Logic 27* (1988), 117–188.

[25] SKOLEM, T. *Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich.* Skrifter Utgit av Videnskapsselskapet, I. Mate. Klasse. No. 6, Oslo, 1923.

[26] TAIT, W. W. Infinitely long terms of transfinite type. In *Formal Systems and Recursive Functions* (1965), J. N. Crossley and M. Dummett, Eds., North Holland, pp. 176–185.

[27] TAIT, W. W. Normal derivability in classical logic. *In Lecture Notes in Mathematics 72, Ed: J. Barwise* (1968), 204–236.

Index