Lecture Notes in Mathematics

Edited by A. Dold and B. Eckmann

811

Dag Normann

# Recursion on the Countable Functionals

Springer-Verlag
Berlin Heidelberg New York

# Lecture Notes in Mathematics

Edited by A. Dold and B. Eckmann

811

Dag Normann

# Recursion on the Countable Functionals

Springer-Verlag
Berlin Heidelberg New York 1980

**Author**

Dag Normann
Institute of Mathematics, The University of Oslo
Box 1053 Blindern Oslo 3
Norway

AMS Subject Classifications (1980): 03 D 65

ISBN 3-540-10019-9 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-10019-9 Springer-Verlag New York Heidelberg Berlin

Library of Congress Cataloging in Publication Data. Normann, Dag, 1947- Recursion on the countable functionals. (Lecture notes in mathematics; 811) Bibliography: p. Includes index. 1. Recursion theory. 2. Computable functions. I. Title. II. Series: Lecture notes in mathematics (Berlin); 811.
QA3.L28 no. 811. [QA96] 510s [511.3] 80-19391


Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2141/3140-543210

## Introduction

Generalized recursion theory is an area of mathematical logic which has been rapidly growing over the last twenty years, and it is now recognized as a dicipline of its own.

One reason for generalizing recursion theory is to find analogues to the natural numbers and the recursion theory on them, partly in order to use the intuition about computations on other domains and partly in order to generalize classical results. The consequence is that one quite often looks at theorems requiring a difficult combinatorial proof in a more general situation. Many of the proofs in generalized recursion theory are therefore true generalizations of classical proofs concerning recursion or metarecursion on the natural numbers, and much of their value is that they give a better understanding of the classical proofs.

Another reason for generalizing recursion theory is to look at other domains and 'true' algorithms on such domains. Then the motivation is not just to generalize but to find domains on which the notions of algorithm and computation make sense and to find what the algorithms and computations really are.

One attempt to extend recursion theory was done by Kleene during the fifties. He defined algorithms operating on functionals of arbitrary finite types and it is widely recognized that he gave an important analysis of the notion of a computation in a more general setting. Much of the work that has been done on Kleene's computation theory for higher types has been concerned with computations relative to certain functionals. The theory for recursion in the so-called normal functionals in particular has been a successful subbranch of the general theory. This theory is a generalization of the theory for hyperarithmetic sets and the research follows the patterns of generalized recursion theory described above. In particular computations will be infinite in a very strong sense.

Kleene isolated a subclass of his functionals of higher types, the countable functionals. It is a natural subclass if one wants to preserve some of the finiteness of ordinary recursion theory; a computation in a finite sequence of countable functionals may have an infinite computation tree but the value is decidable from a finite amount of information about the functionals involved. Kleene showed that they are closed under recursion.

Independently Kreisel, being interested in matters of constructivity, found a hierarchy of functionals suitable for arguments about constructive mathematics. He called them continuous functionals. It was clear that modulo unimportant differences these two classes of functionals were equivalent. This makes the countable or continuous functionals an interesting domain for recursion theory; computations are still finite in character (it has been shown that they cannot be extended preserving this property) and the hierarchy itself is as constructive as the real line.

Accepting this and accepting that recursion theory is worthwhile studying for its own sake one is motivated for the material in this book. This book is written out of an interest in the recursion theory of the countable functionals and related structures and it will mainly be concerned with recursion-theoretic problems.

After two introductory chapters we give in chapter 3 a structural analysis of the countable functionals, partly because this material has not been published elsewhere, partly because there is a structural understanding underlying most of the proofs to come later on. The rest of the book will contain various proofs of theorems concerning the recursion theory and the sample of proofs and results should be sufficient to introduce the reader to the main methods and problems in the area up to present research level. The book does not claim to contain all interesting results on the subject, just to enable the reader to understand other research papers.

The book is essentially self-contained. The reader is supposed to know ordinary recursion theory up to a good understanding of the basic notions and a knowledge of the basic terminology.

We will use a result of r.e. degree-theory but not the proof. In one of the proofs we will use a priority-argument and for the understanding of that proof it will be an advantage to have seen a priority-argument before. In Chapters 5 - 7 we need some elementary facts concerning the projective hierarchy but there is no advanced descriptive set theory involved.

My first contact with the continuous functionals took place in Oxford in Spring 1975. During my stay there I met Robin Gandy, Jan Bergstra, Martin Hyland and Stan Wainer and in the years to follow they inserted an interest for the subject in me. In particular discussions with Wainer and Bergstra concerning precise open problems made me work in the field.

The idea of writing a book on the subject grew out of a seminar I gave in Oslo in the autumn-term 1977. John V. Tucker suggested it

to me and Jens Erik Fenstad encouraged both of us. Tucker also gave some valuable suggestions on the content of the book. Inspired by the visit of Stan Wainer to Oslo in the spring-term '78 the material started to take shape and the actual work with the manuscript took place autumn '78 and early spring '79. The final version gives the status by Easter '79, no later results gave been incorporated.

During my toiling with the manuscript my wife Svanhild has read the various drafts and given valuable suggestions both concerning the English and the way of presenting the material. Her assistance has been most helpful. Stan Wainer was kind enough to read the final version of the manuscript and his comments induced some important changes. I am sorry that I put this burden on these two but the book certainly improved from it.

John Hartley later read parts of the manuscript and discovered several minor errors in the text.

Finally I will express my gratitude towards R. Møller who performed the skilled typing mostly while I was not around to decipher the handwritten manuscript.

Oslo, January 1980
Dag Normann

CONTENTS

# 1. THE MAXIMAL TYPE STRUCTURE

## 1.1 Functionals of higher types

Ordinary recursion theory deals with computable operations on the natural numbers, or in some expositions (e.g. Shoenfield [43]), with computable operations on finite entities. But even then, when we want to compute on finite sequences, words or whatever we are interested in, we normally code our objects as natural numbers and translate the computation to a computation on natural numbers.

Not every interesting operator in mathematics deals with finite arguments and gives finite answers. A typical example is the partial operator

$$I(f,a,b) = \int_a^b f(x)dx$$

where $a,b$ are reals and $f:\mathbb{R} \to \mathbb{R}$ is a function. Two of the arguments, $a,b$, are infinite sequences of finite entities (finite parts of the decimal expansion), while the first argument, $f$, is itself a function operating on infinite arguments and giving infinite answers. Does it make sense to ask whether the Rieman integral operator is computable or not?

Without doubt the numerical analysist will say that for decent $f$ there are good algorithms that may be used by computers computing the integral up to any predecided accuracy, so there must be some notion of computability floating around.

The task of a mathematician is to take some phenomenon, analyze it, build a beautiful model for it and prove a lot of mathematically interesting properties of that model.

We will choose the Rieman integral and related operations.

When a computer computes an integral it is actually given a natural number $n$ and is asked to give the $n$ first decimals in the answer. So we deal in fact with the operator

$$I(f,a,b,n) = \int_a^b f(x)dx \quad \text{given with } n \text{ decimals.}$$

But now we have achieved something, the answers given by the operator are finite entities and may, as in ordinary recursion theory, be coded as natural numbers.

A real number $a$ can be viewed as a function mapping a

natural number $n$ onto the first $n$ decimals of $a$, and the function $f$ can also be viewed as operating on a real $a$ and a natural number $n$, giving the $n$ first decimals of $f(a)$.

Thus all the operators we consider can be regarded as operators with finite or infinite arguments, and giving natural numbers as values.

The discussion above should explain why we are interested in operators giving natural numbers as values and taking finite sequences of natural numbers and operators as arguments. This leads to the following definition of type-symbol $\sigma$ and the type itself, $Tp(\sigma)$, denoted by it.

Definition 1.1

i $0$ is a type-symbol denoting $\omega$ = the set of natural numbers.

ii If $\sigma_1,\ldots,\sigma_n$ are type-symbols denoting $Tp(\sigma_1),\ldots,Tp(\sigma_n)$ resp., then $\sigma = (\sigma_1,\ldots,\sigma_n)$ is a type-symbol denoting

$$Tp(\sigma) = \text{The set of total functions } \psi:Tp(\sigma_1)\times\cdots\times Tp(\sigma_n) \to \omega$$

Remark 1.2

In the literature one will often find the following alternative definition:

i $0$ is a type-symbol denoting $\omega$

ii If $\sigma$ and $\tau$ are type-symbols denoting $Tp(\sigma)$, $Tp(\tau)$ resp., then $(\sigma\to\tau)$ is a type-symbol denoting the set of total functions $\psi:Tp(\sigma) \to Tp(\tau)$.

The two definitions can be shown equivalent by iterating the following kind of transformations:

If $\psi:Tp(\sigma_1)\times\cdots\times Tp(\sigma_n) \to \omega$

replace it by

$$\psi':Tp(\sigma_1) \to (Tp(\sigma_2)\times\cdots\times Tp(\sigma_n) \to \omega)$$

defined by

$$\psi'(\varphi_1) = \lambda(\varphi_2,\ldots,\varphi_n)\psi(\varphi_1,\ldots,\varphi_n)$$

where $\varphi_1$ varies over $Tp(\sigma_1)$, and for each $\varphi_1$, $\lambda(\varphi_2,\ldots,\varphi_n)\psi(\varphi_1,\ldots,\varphi_n)$ denotes the operator which to arguments $(\varphi_2,\ldots,\varphi_n)$ in $Tp(\sigma_2)\times\cdots\times Tp(\sigma_n)$ gives the value $\psi(\varphi_1,\varphi_2,\ldots,\varphi_n)$.

If $\Phi:Tp(\sigma) \to Tp(\tau)$ where $\tau \neq 0$, then $\tau$ is of the form $\tau_1 \to \tau_2$,

and we replace $\Phi$ by

$$\Phi' : Tp(\sigma) \times Tp(\tau_1) \to Tp(\tau_2)$$

defined by

$$\Phi'(\psi, \varphi_1) = \Phi(\psi)(\varphi_1) .$$

Remark 1.3

The use of $\lambda$ as above is of great notational importance. If $f(x_1,\ldots,x_n,y_1,\ldots,y_m)$ is a function, we will often be interested in the operator that to the arguments $y_1,\ldots,y_m$ gives the function

$$g(x_1,\ldots,x_n) = f(x_1,\ldots,x_n,y_1,\ldots,y_m) .$$

We will denote this function $g$ by

$$\lambda(x_1,\ldots,x_n)f(x_1,\ldots,x_n,y_1,\ldots,y_m) .$$

We will not really be concerned with any of these two notions of functionals of higher types, the purification process will go on a bit further. Therefore we will not give detailed descriptions of the transformations indicated above. If one wants to give a precise definition, the following concept, telling "how far up" a type is, is of value.

Definition 1.4

To any type-symbol, $\sigma$, we associate a natural number, the level of $\sigma$ or the type denoted by $\sigma$ by

i The level of the type-symbol $0$ is $0$

ii Let $\sigma_1,\ldots,\sigma_n$ be type-symbols with levels $k_1,\ldots,k_n$ resp. The level of $(\sigma_1,\ldots,\sigma_n)$ is then $1 + \max\{k_1,\ldots,k_n\}$ .

In order to test the understanding of this definition, we suggest the following exercise:

Prove: Let $\sigma,\tau$ be two type-symbols. Then $\sigma$ and $\tau$ are of the same level if and only if $Tp(\sigma)$ and $Tp(\tau)$ have the same cardinality.

We are now going to define the objects that we really will be working with, the functionals of pure types. The pure types will be hand-picked representatives for each level of types, and we will denote them by natural numbers.

Definition 1.5

i Let $Tp(0) = \omega$

ii Let $Tp(k+1) = {}^{Tp(k)}\omega$

= The set of total functions $\psi : Tp(k) \to \omega$ .

The aim of the last part of this section is to show that, inside $\langle Tp(n)\rangle_{n\in\omega}$ we have enough structure to simulate the broader types from definition 1.1. Since we have not yet developed a computation theory, we cannot prove that our codings are computable, but they are clearly 'effective' in some sense.

It is well known from ordinary recursion theory that there is a recursive pairing function $\langle , \rangle : \omega\times\omega \to \omega$ with recursive projection functions $(\ )_1$ and $(\ )_2$ such that

$$(\langle n,m\rangle)_1 = n \quad \text{and} \quad (\langle n,m\rangle)_2 = m .$$

These definitions are directly lifted to $Tp(n+1)$ by

Definition 1.6

a Let $\psi_1,\psi_2$ be elements in $Tp(n+1)$ .
Let $\langle\psi_1,\psi_2\rangle$ be the element in $Tp(n+1)$ defined by

$$\langle\psi_1,\psi_2\rangle(\varphi) = \langle\psi_1(\varphi),\psi_2(\varphi)\rangle .$$

b) Let $\psi \in Tp(n+1)$ . Define $(\psi)_i$ $(i = 1,2)$ by

$$(\psi)_i(\varphi) = (\psi(\varphi))_i .$$

It follows that $(\langle\psi_1,\psi_2\rangle)_1 = \psi_1$ and $(\langle\psi_1,\psi_2\rangle)_2 = \psi_2$. Given the pairing functions we can map any fixed number of elements $\psi_1,\ldots,\psi_{n+1}$ into one single element by

$$\langle\psi_1,\ldots,\psi_{n+1}\rangle_{n+1} = \langle\langle\psi_1,\ldots,\psi_n\rangle_n,\psi_{n+1}\rangle$$

where $\langle , \rangle_2 = \langle , \rangle$ .

So any finite cartesian product of a fixed type may be identified with the type itself.

In order to jump from one type to another, we need the push up and push down operators given in the following definition.

Definition 1.7

i If $n \in Tp(0)$ , then $n^+ \in Tp(1)$ is defined by

$$n^+(m) = n$$

ii If $f \in Tp(1)$ , then $f^- \in Tp(0)$ is defined by

$$f^- = f(0)$$

iii If $\varphi \in Tp(n)$ and $n > 0$, then $\varphi^+ \in Tp(n+1)$ is defined by

$$\varphi^+(\psi) = \varphi(\psi^-)$$

iv If $\psi \in Tp(n+1)$ and $n > 0$, then $\psi^- \in Tp(n)$ is defined by

$$\psi^-(\varphi) = \psi(\varphi^+)$$

v If $n < m$, then $P_n^m : Tp(n) \to Tp(m)$ is defined by

$$P_n^m(\varphi) = \varphi^{+\cdots+} \text{ where the number of +'s is } m-n$$

If $n > m$, then $P_n^m : Tp(n) \to Tp(m)$ is defined by

$$P_n^m(\varphi) = \varphi^{-\cdots-} \text{ where the number of -'s is } n-m$$

If $n = m$, then $P_n^m$ is the identity on $Tp(n)$.

Remark 1.8

$\varphi^+$ is called the push-up of $\varphi$ and $\varphi^-$ is called the push-down of $\varphi$. When we push down we will loose some information, while we keep all information by pushing up. We will later prove that $P_n^m$ is computable. Now we will show that given the growth of cardinality with the type, the $P_n^m$'s are as faithful as possible.

Lemma 1.9

a If $n < m < k$ or $k < m < n$ then $P_n^k = P_m^k \circ P_n^m$.

b If $n < m$ and $\varphi \in Tp(n)$, then $P_m^n(P_n^m(\varphi)) = \varphi$.

Proof:

a is immediate from the definition. To prove b, it is sufficient to prove that for all $\varphi$ we have that $(\varphi^+)^- = \varphi$. We prove this by induction on the type of $\varphi$.

If $\varphi \in Tp(0)$, then $\varphi$ is a natural number $n$, $\varphi^+$ is the constant function $f(m) = n$ and $f^- = f(0) = n$, so

$$(\varphi^+)^- = \varphi .$$

Now let $\varphi \in Tp(n+1)$ and assume that the claim holds for all functionals of lower types. Then

$$(\varphi^+)^-(\psi)$$

$$= (\varphi^+)(\psi^+) \text{ by definition of } (\varphi^+)^-$$

$= \varphi((\psi^+)^-)$ by definition of $\varphi^+$

$= \varphi(\psi)$ by the induction hypothesis applied to $(\psi^+)^-$.

□

We challenge the reader to use the push-up maps and codings of sequences to give effective embeddings of each $Tp(\sigma)$ from Definition 1.1 into $Tp(k)$, where $k$ is the level of $\sigma$.

From now on we will only work with functionals of pure types, so we do not require technical familiarity with arbitrary types. To us they only serve as an intermediate stage in constructing the pure types and showing that they cover in a coded way the phenomena we want to discuss in this book.

## 1.2 Kleene's Computations

By the introduction of oracles, ordinary recursion theory is easily relativized to functions $f:\omega \to \omega$, so the notion of a recursive or computable functional of type 2 is meaningfull.

Kleene [22] lifted the notion of computability to functionals of arbitrary types. Later in this section we will give the precise definition of Kleene-computations via the schemes S1 - S9. But we will first discuss some of the problems involved.

We are going to define a class of valid computations which takes sequences of functionals as arguments and gives natural numbers as values.

Clearly the successor-operator, the constant functions and the identity operator on the natural numbers are computable (S1-S3). Also, the composition of two valid computations must be a valid computation (S4), and the use of primitive recursion to define new computations must be permitted (S5).

If $\psi(\varphi_1,\ldots,\varphi_n)$ is a computable function and $\sigma$ is a permutation on $\{1,\ldots,n\}$, then

$$\psi'(\varphi_1,\ldots,\varphi_n) = \psi(\varphi_{\sigma(1)},\ldots,\varphi_{\sigma(n)})$$

must be regarded as computable (S6).

If $f$ is of type 1 and $x \in \omega$, then $f(x)$ is clearly uniformly computable in $f,x$ (S7).

Combining this with composition we see that if we have computed $x$ by some algorithm, we compute $f(x)$ by using an oracle for $f$ combined with the algorithm for $x$. Then we may use $f(x)$ further in some larger computation. Thus, regarding point-evaluation $f(x)$ on numbers as com-

putable permits us to use functions in intermediate computations.

The most natural way to generalize S7 to higher types might seem to let the evaluation-operator $E_V(\psi,\varphi) = \psi(\varphi)$ be computable. If we would permit computations to take functionals as values we could use functional-application in intermediate computations. But life will be much easier if we can keep the natural numbers as the only possible values of our computations, and the need to evaluate $\psi$ on an intermediately computed functional $\varphi$ is our only reason for introducing functionals as values. Thus we introduce a new scheme which to a functional $\psi$ and an algorithmic description of a functional $\varphi$ gives us $\psi(\varphi)$ whenever this has meaning (S8).

Our algorithms will be indexed by numbers, and like most recursion theorists we believe that going from an index for an algorithm directly to the algorithm itself is effective (S9).

The exact notion of a Kleene-computation is defined by an inductive definition with nine clauses called schemes (S1-S9). To each scheme we associate a natural number, an index, which will give perfect coding of the actual algorithm. The index will be a coded sequence $\langle i,\ldots,\sigma\rangle$ where $i$ is the number of the clause used, ... will contain special information (e.g. in the case of composition the indices for the two algorithms composed) and $\sigma$ will be a coded sequence $\langle k_1,\ldots,k_n\rangle$ saying that arguments accepted by this algorithm should be of types $k_1,\ldots,k_n$ in that order.

The expression $\underline{\{e\}(\vec{\psi})}$ means the algorithm with index $e$ applied on $\vec{\psi}$, and if the algorithm works, $\{e\}(\vec{\psi})$ will also denote the value of the computation. This ambiguity is not greater than the one used in calculus, where $\sum_{n=1}^{\infty} a_n$ both means the sequence of finite partial sums and the limit whenever the limit exists.

We will now give the definition we are going to work with. It is not symbol by symbol as in Kleene [22], but we clearly define the same notion of computability.

Definition 1.10

S1. If $e = \langle 1,\sigma\rangle$, $x \in \omega$ and $\sigma$ is the sequence (number) of the types of $(x,\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(x,\varphi_1,\ldots,\varphi_k) = x+1$$

S2. If $e = \langle 2,q,\sigma\rangle$, $q \in \omega$ and $\sigma$ is the sequence of the types of $(\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(\varphi_1,\ldots,\varphi_k) = q$$

S3. If $e = \langle 3,\sigma\rangle$, $x \in \omega$ and $\sigma$ is the sequence of the types of $(x,\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(x,\varphi_1,\ldots,\varphi_k) = x$$

S4. If $e = \langle 4,e_1,e_2,\sigma\rangle$ and $\sigma$ is the sequence of the types of $(\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(\varphi_1,\ldots,\varphi_k) \simeq \{e_1\}(\{e_2\}(\varphi_1,\ldots,\varphi_k),\varphi_1,\ldots,\varphi_k)$$

where $a \simeq b$ means 'both a and b are defined and are equal, or they are both undefined'.

S5. If $e = \langle 5,e_1,e_2,\sigma\rangle$ and $\sigma$ is the sequence of the types of $(x,\varphi_1,\ldots,\varphi_k)$, then

i $\{e\}(x,\varphi_1,\ldots,\varphi_k) \simeq \{e_1\}(\varphi_1,\ldots,\varphi_k)$ if $x = 0$

ii $\{e\}(x,\varphi_1,\ldots,\varphi_k) \simeq \{e_2\}(\{e\}(x-1,\varphi_1,\ldots,\varphi_k),\varphi_1,\ldots,\varphi_k)$ if $x > 0$

S6. If $e = \langle 6,e_1,\bar{\tau},\sigma\rangle$, $\bar{\tau}$ codes a permutation $\tau$ of $k$ elements and $\sigma$ is the sequence of the types of $(\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(\varphi_1,\ldots,\varphi_k) \simeq \{e_1\}(\varphi_{\tau(1)},\ldots,\varphi_{\tau(k)})$$

S7. If $e = \langle 7,\sigma\rangle$, $x \in \omega$, $f \in Tp(1)$ and $\sigma$ is the sequence of the types of $(x,f,\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(x,f,\varphi_1,\ldots,\varphi_k) = f(x)$$

S8. If $e = \langle 8,e_1,\sigma\rangle$ and $\sigma$ is the sequence of the types of $(\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(\varphi_1,\ldots,\varphi_k) \simeq \varphi_1(\lambda\psi\{e_1\}(\psi,\varphi_1,\ldots,\varphi_k))$$

where the $\lambda$-notation is interpreted as in remark 1.3.

S9. If $e = \langle 9,t,\sigma\rangle$, $e_1 \in \omega$, $t \leq k$ and $\sigma$ is the sequence of the types of $(e_1,\varphi_1,\ldots,\varphi_k)$, then

$$\{e\}(e_1,\varphi_1,\ldots,\varphi_k) \simeq \{e_1\}(\varphi_1,\ldots,\varphi_t)$$

Remark 1.11

In S4 it is understood that $\{e\}(\varphi_1,\ldots,\varphi_k)$ is defined only if $\{e_2\}(\varphi_1,\ldots,\varphi_k)$ is defined with some value $s$ and $\{e_1\}(s,\varphi_1,\ldots,\varphi_k)$ is defined.

The same remark is valid for S5 and S6.

In S8 it is understood that the variable $\psi$ varies over functionals

of two types less than the type of $\varphi_1$. The computation is defined if all computations $\{e_1\}(\psi,\varphi_1,\ldots,\varphi_k)$ are defined for $\psi$ of the appropriate type, i.e. if $\lambda\psi\{e_1\}(\psi,\varphi_1,\ldots,\varphi_k)$ is a total functional of type one less than the type of $\varphi_1$.

The definition of the relation $\{e\}(\varphi_1,\ldots,\varphi_k) \simeq n$ may be regarded as an inductive definition $\Gamma$, as we will describe below. In discussing computations we will without mentioning it assume that the arguments are in the appropriate form for the index, and normally we discuss just a few cases covering the methods and ideas involved.

Although $\Gamma$ is defined by 9 clauses corresponding to S1 - S9, we only give a sample of them here:

Definition 1.12

Define the operator $\Gamma(S)$ generating the computation-tuples of Kleene-recursion by

1. $\langle e,x,\varphi_1,\ldots,\varphi_k,x+1\rangle \in \Gamma(S)$ when $e = \langle 1,\sigma\rangle$

2., 3. and 7. are analogous.

4. If $\langle e_2,\varphi_1,\ldots,\varphi_k,x\rangle$ and $\langle e_1,x,\varphi_1,\ldots,\varphi_k,y\rangle$ are both in $S$, then $\langle e,\varphi_1,\ldots,\varphi_k,y\rangle \in \Gamma(S)$, where $e = \langle 4,e_1,e_2,\sigma\rangle$

5., 6. and 9. are analogous.

8. If $e = \langle 8,e_1,\sigma\rangle$ and for some functional $\xi$,

$$\forall\psi\langle e_1,\psi,\varphi_1,\ldots,\varphi_k,\xi(\psi)\rangle \in S,$$

then $\langle e,\varphi_1,\ldots,\varphi_k,\varphi_1(\xi)\rangle \in \Gamma(S)$.

A sequence is in $\Gamma(S)$ if it is in $S$ or if it is in $\Gamma(S)$ by one of the nine conditions above.

Remark 1.13

Let

$$\Gamma^0 = \emptyset$$

$$\Gamma^\beta = \bigcup_{\gamma<\beta} \Gamma(\Gamma^\gamma)$$

and

$$\Gamma^\infty = \bigcup_{\alpha \in \text{Ordinals}} \Gamma^\alpha$$

We then see that $\{e\}(\varphi_1,\ldots,\varphi_k) \simeq n$ if and only if $\langle e,\varphi_1,\ldots,\varphi_k,n\rangle \in \Gamma^\infty$.

Definitions 1.11 and 1.13 are various ways of formulating the same notion of computations, and the only thing we need from the definition