21197

National Library of Canada | Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE | THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* Glen Biagioni

TITLE OF THESIS/*TITRE DE LA THÈSE* A Study of Some of the Main Methods of Proof in Abstract Complexity Theory

UNIVERSITY/*UNIVERSITÉ* Simon Fraser University

DEGREE FOR WHICH THESIS WAS PRESENTED/ *GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* Master of Science

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE DEGRÉ* 1974

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Dr. Ronald Harrop

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* June 3, 1974 SIGNED/*SIGNÉ*

PERMANENT ADDRESS/*RÉSIDENCE FIXE*

[redacted]

NL-91 (11-73)

A STUDY OF SOME OF THE MAIN METHODS OF PROOF IN ABSTRACT COMPLEXITY THEORY

by

Glen Biagioni

B.Sc.(Hons), Simon Fraser University, 1972

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

Mathematics

SIMON FRASER UNIVERSITY

May 1974

APPROVAL

Name: Glen Biagioni

Degree: Master of Science

Title of Thesis: A Study of Some of the Main Methods of Proof in Abstract Complexity Theory

Examining Committee:

Chairman: A. R. Freedman

R. Harrop
Senior Supervisor

T. Brown

S. Thomason

A. H. Lachlan
External Examiner

Date Approved: May 30, 1974

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis or dissertation (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this thesis for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis/Dissertation:

A Study of Some of the Main Methods of Proof in Abstract Complexity Theory.

Author: _____

(signature)

Glen Bragioni

(name)

June 3, 1974

(date)

ABSTRACT

A comprehensive development of Blum's axioms for complexity measures is given and the following concepts and some of their uses in abstract complexity theory are discussed: Church's Thesis, almost everywhere, diagonalization, the Halting Problem, dovetailing, the Recursion Theorem. Several results fundamental to abstract complexity theory are investigated in terms of how the above concepts are used in their proofs. The intent of the author is to help the reader develop a strong background in abstract complexity theory without requiring of the reader any previous knowledge in recursive function theory.

for

Elaine

ACKNOWLEDGMENTS

I would like to express my appreciation to my supervisor, Dr. R. Harrop, for all the aid and advice he has given me both for the thesis and throughout my graduate education. I also wish to thank Dr. A. H. Lachlan for several discussions from which I have taken some material for the thesis; Dr. Freedman for reading the manuscript and suggesting some ways of making various parts of the thesis more clear; and Linda Cowan and Nancy Hood for spending many tedious hours typing the thesis.

I am very thankful for the financial assistance received from the National Research Council of Canada.

TABLE OF CONTENTS

CHAPTER 0

INTRODUCTION

§ 0.0 Outline

Most of the literature on abstract complexity theory* is written under the assumption that the reader has a strong background in recursive function theory. This author feels that many mathematicians and computer scientists who might otherwise find abstract complexity theory an interesting and rewarding topic to study, find that to extract from recursive function theory those topics relevant to complexity theory is too much of an undertaking. This paper constitutes an attempt by the author to isolate and demonstrate the use of many of the concepts and tools commonly used in complexity theory proofs and counter-examples. This is done within complexity theory itself, thus making the paper relatively self-contained.

We do assume that the reader has some background in basic mathematics and in the theory of computability. If the reader can understand the topics discussed in § 0.1, has seen a development of Turing machines and the construction of a Universal Turing machine, then his background should be adequate. CHAPTER 1 consists of a review of these latter two topics.

Abstract complexity theory is a study of properties of complexity measures, that is, objects that satisfy certain axioms (from Blum [2]) that will be presented in CHAPTER 4. In this light, abstract complexity theory gives a macroscopic view of specific complexity measures. Several of these

---

* Sometimes we use "complexity theory" for "abstract complexity theory."

measures, such as time and tape measures on Turing machines, are studied in their own right. Although not essential, it might be helpful for the reader to investigate some of these specific measures before studying the more general theory. CHAPTER 2 gives a very brief description of some time and tape measures on the Turing machines developed in CHAPTER 1.

In CHAPTER 3 we give a formal definition of computability. This corresponds to the definition of the partial recursive functions in recursive function theory. However, since this paper appeals to the reader's intuition in computability rather than a knowledge of recursive function theory, we choose to name these partial functions as the computable partial functions rather than the partial recursive functions.

The remainder of the paper deals with specific notions which the author feels form the 'backbone' of abstract complexity theory. In CHAPTER 5 we look at the notion of almost everywhere, why it appears in complexity theory, and at its impact on the relevance of studying complexity theory. CHAPTER 6 deals with diagonalization and the Halting Problem; CHAPTER 7 deals with dovetailing arguments.

In CHAPTER 8 the Recursion Theorem is discussed. Some authors seem very reluctant to make use of the Recursion Theorem in proofs (see the discussion of the Speed Up Theorem in [4]), presumably because they feel that the intuitive drive behind such proofs becomes obscured. This author disagrees and hopes that CHAPTER 8 will help to 'clear the fog' around the Recursion Theorem and enable the reader to fully appreciate the intuition behind various uses of this powerful theorem.

CHAPTER 9 is a brief discussion of further topics that the reader may wish to study.

Although most of the fundamental results of abstract complexity theory are discussed at least briefly in the paper, the author's intention is neither to present a comprehensive study of the results of complexity theory, nor to present all of the notions from recursive function theory which are used in complexity theory. He intends only to present some of these results and notions in such a manner that the reader may acquire a strong background in abstract complexity theory, without requiring that the reader have any previous knowledge in recursive function theory.

§ 0.1 Notation

At times we will use the following symbols in order to abbreviate the given phrases.

$\forall$ - 'for all' or 'for every.'

$\exists$ - 'there is' or 'there exists.'

$\ni$ - 'such that.'

$\in$ - 'is an element of' or 'in' as in set theory.

$\subseteq$ - 'is a subset of' as in set theory.

$\subset$ - 'is a proper subset of' as in set theory.

$\Rightarrow$ - 'implies.' $P \Rightarrow Q$ is 'P implies Q' or 'if P then Q.'

$\Leftarrow$ - 'is implied by.' $P \Leftarrow Q$ is 'P is implied by Q' or 'if Q then P.'

$\Leftrightarrow$ - 'if and only if.'

We use $\underline{N}$ to represent the natural numbers, $\{0,1,2,\ldots\}$. $\underline{N}^k$, for $k \in \{1,2,\ldots\}$, represents the set of all k-tuples of natural numbers. For $k \geq 2$, a particular k-tuple is represented by either an underscored letter or a parenthesized list. For example, we might write $\underline{i}$ for $(i_1,i_2,\ldots,i_k)$. $\underline{N}^1$ and $\underline{N}$ are essentially the same and we do not distinguish between the two.

Partial functions are used extensively in the paper. All partial functions will map from $\underline{N}^k$ to $\underline{N}$ for some k. Let ψ be any partial function on $\underline{N}^k$. For any $(n_1,n_2,\ldots,n_k) \in \underline{N}^k$, $\psi(n_1,n_2,\ldots,n_k)$ may be defined to be some natural number, or else it may be undefined. The domain of ψ, denoted $\mathrm{Dom}(\psi)$, is defined by

$$\mathrm{Dom}(\psi) = \{\underline{n} \in \underline{N}^k \mid \psi\underline{n} \text{ is defined}\}.$$

The image of ψ, denoted $\mathrm{Im}(\psi)$, is defined by

$$\mathrm{Im}(\psi) = \{\psi\underline{n} \mid \underline{n} \in \mathrm{Dom}(\psi)\}.$$

Thus the image of ψ is the set of values taken by ψ.

Suppose we have a set I, and for each $i \in I$, we have one associated partial function ψ_i. We say that I indexes the set of functions $\{\psi_i \mid i \in I\}$. We will denote such an indexing by $(\psi_i)_{i \in I}$. In the paper, the index set will always be $\underline{N}^k$ for some k. An indexing of partial functions, $(\psi_i)_{i \in \underline{N}}$, (that is, where $k = 1$) will be called a sequence of partial functions in order to remain consistent with the literature.

The relational operators $<, \leq, =, \geq, >$ are straightforward except when used with partial functions. Let ψ_1, ψ_2 be partial functions on $\underline{N}^{k_1}, \underline{N}^{k_2}$, respectively. Let $\underline{n}_1 \in \underline{N}^{k_1}$ and $\underline{n}_2 \in \underline{N}^{k_2}$. Let α be any one of the above operators.

If either $\psi_1\underline{n}_1$ or $\psi_2\underline{n}_2$ is undefined then $\psi_1\underline{n}_1 \; \alpha \; \psi_2\underline{n}_2$ is undefined. Otherwise $\psi_1\underline{n}_1 \; \alpha \; \psi_2\underline{n}_2$ is true or false in the obvious manner.

We also use three other relational operators $\lesssim, \simeq, \gtrsim$. $\psi_1\underline{n}_1 \simeq \psi_2\underline{n}_2$ is true if either both $\psi_1\underline{n}_1$ and $\psi_2\underline{n}_2$ are undefined or if $\psi_1\underline{n}_1 = \psi_2\underline{n}_2$, otherwise it is false. $\psi_1\underline{n}_1 \lesssim \psi_2\underline{n}_2$ is true if either $\psi_1\underline{n}_1 \simeq \psi_2\underline{n}_2$ or $\psi_1\underline{n}_1 \leq \psi_2\underline{n}_2$, otherwise it is false. $\psi_1\underline{n}_1 \gtrsim \psi_2\underline{n}_2$ if and only if $\psi_2\underline{n}_2 \lesssim \psi_1\underline{n}_1$.

If ψ_1, ψ_2 both map from $\underline{N}^k$ to $\underline{N}$ and $\forall \underline{n} \in \underline{N}^k$, $\psi_1\underline{n} \simeq \psi_2\underline{n}$, then we write $\psi_1 = \psi_2$.

Let ψ be a partial function on $\underline{N}^k$ and $\underline{n} \in \underline{N}^k$. For $m \in \underline{N}$, we have,

$$\psi\underline{n} = m \text{ is true} \Leftrightarrow \psi\underline{n} \simeq m \text{ is true}$$

$$\text{and } \psi\underline{n} \leq m \text{ is true} \Leftrightarrow \psi\underline{n} \lesssim m \text{ is true.}$$

Notice though that if $\psi\underline{n}$ is undefined then $\psi\underline{n} = m$ and $\psi\underline{n} \leq m$ are undefined, whereas $\psi\underline{n} \simeq m$ and $\psi\underline{n} \lesssim m$ are false.

Partial predicates will map from $\underline{N}^k$ to {TRUE, FALSE}, for some k. Thus, if π is a partial predicate on $\underline{N}^k$ and $\underline{n} \in \underline{N}^k$, then either

$\pi\underline{n}$ is TRUE

or $\pi\underline{n}$ is FALSE

or $\pi\underline{n}$ is undefined.

Often we write '$\pi\underline{n}$ holds' or merely '$\pi\underline{n}$' for '$\pi\underline{n}$ is TRUE.' If $\pi\underline{n}$ is FALSE or undefined we write '$\pi\underline{n}$ does not hold.' We sometimes write '$\neg\pi\underline{n}$' for '$\pi\underline{n}$ is FALSE.'

At times, if an expression is being used to represent a function or a predicate, we will use Church's lambda notation in order to make clear which variables in the expression are arguments (and also to give a fixed order to these variables). For example, if $(\psi_i)_{i\in\underline{N}}$ is a sequence of partial functions on $\underline{N}$ then

$$\lambda i,n[\psi_i(n)]$$

is the partial function ψ on $\underline{N}^2$ defined by

$$\psi(i,n) \simeq_{\text{defn}} \psi_i(n).$$

Given ψ as above, and some fixed $i \in \underline{N}$,

$$\lambda n[\psi(i,n)]$$

is the partial function φ on $\underline{N}$ defined by

$$\varphi(n) \simeq_{\text{defn}} \psi(i,n).$$

Where π is a partial predicate on $\underline{N}$, we use the notation

$$\mu k[\pi(k)]$$

to represent 'the least $k \in \underline{N}$ such that $\pi(k)$ holds.'

The composition of a partial function ψ on $\underline{N}^{k'}$ with partial functions $\psi_1,\psi_2,\ldots,\psi_k$, on $\underline{N}^k$, say

$$\varphi = \lambda n_1, n_2, \ldots, n_k [\psi(\psi_1(n_1, n_2, \ldots, n_k), \ldots, \psi_{k'}(n_1, n_2, \ldots, n_k))]$$

is defined by

$$\varphi(n_1, n_2, \ldots, n_k) =_{defn} \begin{cases} \text{undefined} & \text{if } \exists i \leq k', \ \psi_i(n_1, n_2, \ldots, n_k) \text{ is undefined} \\ \psi(p_1, p_2, \ldots, p_k) & \text{if } \forall i \leq k', \ \psi_i(n_1, n_2, \ldots, n_k) = p_i. \end{cases}$$

The composition of a partial predicate with partial functions is defined in a similar manner. We use $g \circ f$ for $\lambda \underline{n}[g(f(\underline{n}))]$.

A procedure is an intuitive notion with which we assume the reader is familiar. We allow the result of procedures to be undefined on some inputs (possibly because of infinite cycling). If a procedure gives a result on all inputs from some set, then we call it a total procedure on this set of inputs. We use the words check, compute, construct, decide, detect, find, generate, and tell, all with the understanding that they should have connotations of the existence of a procedure which can do the checking, computing, constructing, deciding, detecting, finding, generating, or telling.

CHAPTER 1

TURING MACHINES

§ 1.0 Discussion

In this chapter we look at a specific model for Turing machines (§ 1.1), at a method of numbering these machines (§ 1.2), and at a machine which is universal for these machines (§ 1.3). These notions are treated informally as it is expected that the reader has previously studied them in a more formal setting.

We present the material in this chapter only so that later in the paper we have a fixed development of computability from which to draw examples.

§ 1.1 Turing Machines and Their Partial Functions

In this section we describe a model of Turing machine which will be used to exemplify various concepts presented in the paper. This model is slightly different from models normally used to describe computability in that it has two tapes, one devoted solely to input and output and the other devoted to doing the actual computations. We do this only so that the model can serve better in our examples, not to improve the description of computability in any way.

Our model is a two tape (semi-infinite to the right) deterministic Turing machine with the extra restriction that one of the tape heads, called the I/O tape head, cannot move left, and can only read and write the symbols 0,1. The other tape head will be called the work tape head.

We give a more detailed description. By a Turing machine instruction we mean a quintuple of the following form,

$$((i,t_j),\ q_k,\ q_m,\ (p,t_n),\ (D_1,D_2))$$

where

$$i,p \in \{0,1\}$$

$$D_1,D_2 \in \{L,N,R\},\ \text{with}\ D_1 \neq L$$

$$j,k,m,n \in \underline{N}.$$

$\{q_i \mid i \in \underline{N}\}$ is called the set of states.

$\{t_i \mid i \in \underline{N}\}$ is called the set of tape symbols.

Thus in the above quintuple, q_k, q_m are states, and t_j, t_n are tape symbols. We consider t_0 and 0 as the same tape symbol, and t_1 and 1 as the same tape symbol.

(i,t_j) is called the input pair.

q_k is the initial state of the instruction.

q_m is the final state of the instruction.

(p,t_n) is the output pair.

(D_1,D_2) is the direction pair.

By a Turing machine we mean any finite list of Turing machine instructions such that no two instructions in the list have the same input pair and initial state.

Suppose we have two tapes, infinite only to the right, which are divided into cells, each of which contains precisely one tape symbol. Given a Turing machine, place the I/O tape head on one of the cells of one of the tapes, and the work tape head on one of the cells of the other tape. Place the machine in some state. This machine will now operate in steps as follows. Suppose at the beginning of any step, that the I/O tape head is reading tape symbol i, the work tape head is reading tape symbol t_j, and the machine is in state q_k. If the machine has an instruction in its list of the form,

$$((i,t_j),\ q_k,\ q_m,\ (p,t_n),\ (D_1,D_2))$$

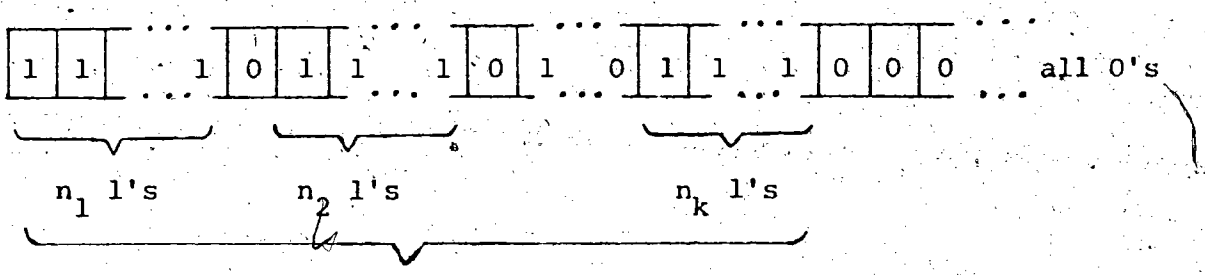
for some q_m,p,t_n,D_1,D_2, then it enters state q_m, replaces i with p using its I/O tape head, replaces t_j with t_n* using its work tape head, moves the I/O tape head no cells or one cell to the right according as D_1 is N or R, and moves its work tape head one cell to the left, no cells, or one cell to the right according as D_2 is L,N or R. If

---

* Henceforth we will abbreviate such phrases as 'replaces t_j with t_n' with 'prints t_n'.

the work tape head moves left off of the work tape, or if there is no instruction with input pair (i,t_j) and initial state q_k, then the machine halts.

Let T be a Turing machine. T defines a partial function τ^k on $\underline{N}^k$ for each k, as follows. The value of $\tau^k(n_1,n_2,\ldots,n_k)$ is determined by starting T in state q_0, with the I/O tape head on the leftmost cell of a tape of the form,



and the work tape head starting on the leftmost cell of a tape filled completely with 0's. If T on these tapes does not halt then $\tau^k(n_1,n_2,\ldots,n_k)$ is undefined, if T halts, then the value of $\tau^k(n_1,n_2,\ldots,n_k)$ is defined to be the number of 1's between the left end of the I/O tape and the final position of the I/O tape head.

Henceforth, when we speak of the function on $\underline{N}^k$ computed by a Turing machine, we will mean the function defined as above.

§ 1.2 A Gödel Numbering

We would like to number the set of all of our Turing machines so that being given the number for a machine and being given a machine will be much the same thing. Our intention is that some Turing machines will be able to have other Turing machines as inputs and outputs by merely inputting and outputting the numbers of those other Turing machines. Such a numbering will be called a Gödel numbering, since our reasons for numbering the Turing machines are similar to Gödel's reasons for numbering formulas in the proof of the famous incompleteness theorem in mathematical logic. (i.e. Gödel intended that some formulas could make statements about other formulas by making statements about the numbers of these formulas). More precisely, by a Gödel numbering of the set of Turing machines we will mean a method of enumerating all of the Turing machines so that there is a procedure which, given a number $i \in N$, constructs (the list of instructions for) the i^{th} machine, and also a procedure which, given (the list of instructions for) the i^{th} machine, computes i. In this section we will describe a specific Gödel numbering for our set of Turing machines.

A list of eight numbers is of <u>instruction format</u> if

- the first number is either 0 or 1
- the second, third, and fourth numbers are natural numbers
- the fifth number is either 0 or 1
- the sixth number is a natural number
- the seventh number is either 1 or 2
- the eighth number is either 0, 1 or 2 .

Such a list of numbers i,j,k,m,p,n,d_1,d_2 corresponds to the Turing

machine instruction

$$((i,t_j),q_k,q_m,(p,t_n),(D_1,D_2))$$

where D_1 and D_2 are L, N or R according as d_1 and d_2 are 0, 1 or 2, respectively. Certainly there is a procedure which checks whether or not a given list of natural numbers is of instruction format.

A list of $8 \cdot n$ natural numbers, for any $n \in N$, is of machine format if, when the list is broken down into consecutive lists of eight numbers, each sub-list is of instruction format, and furthermore, there are no two such sub-lists with the same first three members in the same order. Clearly there is a one to one correspondence between lists of natural numbers of machine format and Turing machines. (Notice that we must consider two lists of Turing machine instructions which are just rearrangements of each other as distinct machines). Also, there is a procedure which checks whether or not a given list of natural numbers is of machine format.

It is also easy to see that there is a procedure which, when given a machine (as a list of instructions), outputs the corresponding list of natural numbers of machine format, and that there is a procedure which, when given a list of natural numbers of machine format, outputs the list of instructions for the corresponding machine.

Suppose we have a procedure which generates, one at a time, each finite list of natural numbers, without repetition. Associate with the number i, the machine corresponding to the $i+1^{th}$ list of machine format which is generated. This constitutes our enumeration or numbering of the Turing machines. (Notice that 0 corresponds to the 1^{st} list of machine format generated, 1 to the 2^{nd}, etc.)

Given a machine we can find its number by generating the lists of natural numbers using the given procedure, incrementing a counter each time a list of machine format is generated until we find the list of machine format which corresponds to the given machine. The machine's number will be the value of the counter minus 1, that is, one less than the number of lists of machine format generated in order to find the correct one. Conversely, given i we can construct the associated machine by generating the lists of natural numbers using the given procedure until the $i+1^{th}$ list of machine format is generated, then outputting (the list of instructions for) the machine corresponding to the $i+1^{th}$ list of machine format.

Thus, given a procedure which generates all finite lists of natural numbers without repetition, we can Gödel number the Turing machines. In § 7.1 we will develop such a procedure. Using that procedure, let us call the resulting Gödel numbering the <u>standard numbering</u> of the Turing machines. Let T_i be the Turing machine associated with the number i, in the standard numbering, and let τ_i^k be the partial function on $\underline{N}^k$ computed by T_i.

Notice that if τ is the partial function on $\underline{N}^k$ computed by a Turing machine, T, then there are infinitely many Turing machines which compute τ. This is because we can keep adding instructions to T whose initial state is neither q_0 nor the final state of any other instruction of T. The resulting new machines clearly work precisely like T on any tapes. Furthermore, all of these machines must have distinct numbers since they are distinct machines. Thus we have infinitely many numbers $i \in \underline{N}$ such that $\tau_i^k = \tau$.

§ 1.3 A Universal Machine

A Turing machine, U, will be called universal for the standard numbering of the Turing machines if, for each k, the function μ_k on $\underline{N}^{k+1}$ defined by U has the property that $\forall i \in \underline{N}$, $\forall n_1,n_2,\ldots,n_k \in \underline{N}$, we have $\mu_k(i,n_1,n_2,\ldots,n_k) \simeq \tau_i^k(n_1,n_2,\ldots,n_k)$. In this section we attempt to indicate how such a machine might operate.

Basically, in order to compute $\mu_k(i,n_1,n_2,\ldots,n_k)$, U will construct the i^{th} Turing machine, T_i, then simulate T_i on input $(n_1,n_2,\ldots,n_k)$. The following gives a general description of how U might accomplish this, but lacks detail. We leave it to the reader to convince himself that such a machine, U, can be constructed.

On input $(i,n_1,n_2,\ldots,n_k)$, U copies i from the I/O tape to the work tape, replacing the i ones on the I/O tape with zeros. Now, holding the I/O tape head stationary, U generates, using the procedure upon which the standard numbering is based, the finite lists of natural numbers until the $i+1^{th}$ list of machine format is generated. The machine associated with this list, T_i, is the machine that must be simulated.

Still holding the I/O tape head stationary, U sets aside an area of work tape in which this $i+1^{th}$ list of machine format is stored, along with an area in which the number of the current state of the simulated machine will be stored. (This current state space must be large enough to contain the number of the largest numbered state of T_i). The remainder of U's work tape will be used as if it were T_i's work tape, using m+1 cells (call each consecutive group of m+1 cells a pseudo-cell)

as if it were one cell, where m is the number of cells required to hold the number of the largest numbered tape symbol of T_i. The left most cell of one of these pseudo-cells is used to mark the current position of the work tape head of the simulated machine. The remaining m cells are used to store the number of the tape symbol which the simulated machine currently has printed there.

Now U can simulate a step of T_i by reading an I/O tape symbol, (initially U's I/O tape head must be positioned to read $(n_1,n_2,\ldots,n_k)$), then comparing this symbol together with the number of the tape symbol in the pseudo-cell marked as currently being read, together with the number in the current state space, to the first three numbers of each 'instruction' in th list of machine format. If it finds no match then U halts. If it finds a match then the number of the new state given by the matched instruction is printed in the current state space, the I/O symbol to be printed is printed by the I/O tape head, the number of the work tape symbol to be printed is written in the pseudo-cell marked as currently being read. Now the I/O tape head and the head marker on the pseudo-work tape are moved as indicated by the 'instruction'. If the head marker is to be moved left off of the pseudo-work tape, then U halts.

The reader can see that the action of U's I/O tape head on the $(n_1,n_2,\ldots,n_k)$ portion of the original I/O tape will be precisely the action of T_i's I/O tape on input $(n_1,n_2,\ldots,n_k)$. Furthermore, U on input $(i,n_1,n_2,\ldots,n_k)$ will halt if and only if T_i on input $(n_1,n_2,\ldots,n_k)$ halts. Since the i portion of U's original I/O tape has been set to

0's we have that the number of 1's on the I/O tape which determine the result of U on input $(i,n_1,n_2,\ldots,n_k)$. is precisely the same as those for T_i on input $(n_1,n_2,\ldots,n_k)$. Thus, if U can be constructed, then U is universal for our standard numbering of Turing machines. We leave it to the reader to convince himself that, in fact, U can be constructed.

CHAPTER 2

TIME AND TAPE MEASURES

§ 2.0 Discussion

In chapter 4 we will discuss some axioms for complexity measures. In this chapter we present two measures on our Turing machines which satisfy these axioms. These are time measure and tape measure. Time measure counts the number of steps used by a computation and tape measure counts the number of work tape cells used by a computation. In the literature both of these measures are studied in their own right, as well as being examples of abstract complexity measures. (For various specific results for time and tape measures, as well as some other references see Hopcroft and Ullman [5]). In this paper we make use of these measures only as specific examples of abstract complexity measures.

Another measure that might interest the reader is reversals, a count of the number of times the work tape head changes direction. This author has seen no serious treatment of reversal measures in the literature.

In § 2.1 we look at time measure, in § 2.2 we look at two slightly different tape measures, and in § 2.3 we look at a related topic, the use of table lookups to do efficient computations with respect to time and tape.

§ 2.1 A Time Measure

We can define a time measure on $(\tau_i^1)_{i\in N}$ by

$$\sigma_i(n) \simeq_{\text{defn}} \begin{cases} \text{the number of steps } T_i \text{ takes} \\ \quad \text{to compute } \tau_i^1(n) \text{ if this} \\ \quad \text{number is finite, otherwise} \\ \quad \sigma_i(n) \text{ is undefined.} \end{cases}$$

Since T_i takes a finite number of steps in computing $\tau_i^1(n)$ if and only if $\tau_i^1(n)$ is defined, we have

$$\sigma_i(n) \simeq \begin{cases} \text{number of steps } T_i \text{ takes} & \\ \quad \text{in computing } \tau_i^1(n) & \text{if } \tau_i^1(n) \text{ is defined} \\ \text{undefined} & \text{if } \tau_i^1(n) \text{ is undefined.} \end{cases}$$

Notice that for any i,n and m, we can check whether or not $\sigma_i(n) \simeq m$ as follows. From i construct T_i. Run T_i on n, counting the number of steps taken, until either T_i halts or more than m steps are taken, whichever occurs first. If T_i on n uses more than m steps then answer FALSE, else answer TRUE or FALSE according as T_i on n used exactly m steps or not.

The main reason this argument works is because either T_i on n halts taking no more than m steps, or else T_i on input n uses more than m steps. We will see that the same is not true for tape.

The two important properties of time measure that will be of interest to us in this paper are

(1) $\tau_i^1(n)$ is defined $\Leftrightarrow$ $\sigma_i(n)$ is defined.

(2) It is possible to check, given i,n and m, whether or not $\sigma_i(n) \simeq m$.

§ 2.2 Two Tape Measures

We can define a tape measure on $(\tau_i^1)_{i\in N}$ by

$\lambda_i(n) \simeq_{defn}$ the number of distinct tape cells visited by T_i's work tape head in computing $\tau_i^1(n)$ if this number is finite, otherwise $\lambda_i(n)$ is undefined.

Notice that it is possible for $\lambda_i(n)$ to be defined but $\tau_i^1(n)$ undefined. For example T_i on input n might execute an instruction such as

$$((0,0), q_j, q_j, (0,0), (N,N))$$

thus cycling infinitely on a finite amount of work tape. Thus it is not true, as with time measure, that either T_i on input n halts using no more than m work tape cells, else T_i on input n uses more than m work tape cells. However, we will show that it is still possible to check, given i,n,m, whether or not $\lambda_i(n) \simeq m$. The argument hinges on the fact that we can detect if T_i on input n is cycling on a given finite amount of work tape.

Suppose that the computation done by T_i on input n reaches a point where exactly k distinct work tape cells have been visited and although this computation will not terminate, no previously unvisited work tape cells will be visited. Then, in subsequent steps of computation, the I/O tape head will either

(1) reach an I/O tape cell from which it will never move,

or (2) eventually move across each cell of the I/O tape.

Let q be the number of distinct states and t be the number of distinct tape symbols appearing in T_i's instructions. We can detect case (1) above, for if T_i on input n runs more than $2qkt^k$ steps without visiting a previously unvisited work tape cell and without moving the I/O tape head, then T_i must be cycling. This is because if T_i does this, then it must have been in the same state, with the I/O tape and work tape in the same configuration (a tape configuration is the contents of the tape together with the position of the tape head) at least twice. Thus T_i must be cycling through the same sequence of steps over and over.

We can detect case (2) above, for if T_i's input head moves across more than qkt^k 0's to the right of the input n, without T_i visiting a new work tape cell, then T_i must be cycling. This is because if T_i does this then T_i must have been in the same state, with I/O tape head moving right onto a 0 and having only 0's to the right, and with the work tape head in the same configuration twice. Thus, since the I/O tape head can only move either to the right or else not move at all, the I/O tape is effectively in the same configuration, hence again T_i must be cycling through the same sequence of instructions over and over.

Thus we can detect if T_i on input n cycles on exactly k work tape cells as follows. After T_i uses k cells, check if it ever makes more than $2qkt^k$ steps without moving its I/O tape head or using a new work tape cell, or if T_i ever moves its I/O tape head right across more than qkt^k 0's to the right of input n without using a

new work tape cell. If either of these happen then T_i is cycling. Furthermore, if T_i cycles on k work tape cells then one of the above happens.

Now, given i,n,m we can check whether or not $\lambda_i(n) = m$ as follows. From i construct T_i then run T_i on input n counting distinct work tape cells used, and checking if T_i cycles on k work tape cells for each $k \leq m$. If T_i uses more than m work tape cells then answer FALSE. If T_i cycles on k work tape cells where $k \leq m$, then answer TRUE or FALSE according as k is or is not m. If T_i halts answer TRUE or FALSE according as m distinct work tape cells have been used or not.

We can modify the above tape measure slightly so that if the measure is defined then the computation is defined. Look at the tape measure defined by

$$\lambda^*_i(n) \underset{\text{defn}}{=} \begin{cases} \text{the number of distinct work tape cells used by } T_i \text{ in computing } \tau^1_i(n) & \text{if } \tau^1_i(n) \text{ is defined} \\ \text{undefined} & \text{if } \tau^1_i(n) \text{ is undefined.} \end{cases}$$

We can check whether or not $\lambda^*_i(n) = m$, given i,n,m, by a method similar to the above. The only change is that if we detect that T_i on input n cycles on k work tape cells, we always answer FALSE, even if $k = m$.

The important properties of these tape measures that will interest us are

(1) If the computation terminates then the measure is defined.

(2) Given i,n,m we can check whether or not T_i on input n uses exactly m distinct work tape cells.

Additionally, for the λ^*_i measure we have that if the measure is defined then the computation terminates.

§ 2.3 Table Lookups for Efficient Computations

Suppose a Turing machine T computes a function τ on $\underline{N}$ such that for some given $n \in \underline{N}$, $\tau(n)$ is defined. Suppose further that the computation of $\tau(n)$ is relatively complex in that it uses relatively large amounts of time or tape. Surprisingly, there is another machine that computes τ in such a manner that $\tau(n)$ is computed in an extremely efficient manner in terms of time and tape used. The new machine can do this by making use of a method called a table lookup.

To construct this new machine, first modify T to a new machine T' so that T' operates in basically the same manner as T except that on every input, T' copies the input onto its work tape (setting the I/O tape copy to 0's) before starting any real computations, then does the computations using the work tape copy rather than the now lost I/O tape copy. There are some technicalities to overcome in this modification because of how the original machine T might make use of the I/O tape copy of the input. For instance, T might use some of the 1's composing the input for output as well. However, these technicalities can be overcome. For example, we can make T' do all the computations on the work tape using a section of the work tape as a pseudo I/O tape, then once the result is completely computed, copy it onto the I/O tape. (This may increase the number of work tape cells used.)

Suppose we have T' as above so that it also computes τ. T' can now be modified to make use of a table lookup for an efficient computation when the input i is n. Chose $n+1$ states

$q_{k_0}, q_{k_1}, \ldots, q_{k_n}$ which are not in the instructions of T'. Add instructions to T' so that before copying the input i to the work tape, the first k cells of the I/O tape, where $k = \min\{i,n\}$, are read in such a way that if $i \leq n$ then

(a) the machine is left in state q_{k_i},

(b) the work tape head does not move,

and (c) the number of steps used to read input i is approximately i.

Some modification to the instructions of T' will likely be required but basically the above involves adding instructions of the form

$$((1,0), q_{k_j}, q_{k_{j+1}}, (0,0), (R,N)).$$

Now instructions can be added so that if the input, i, is less than n, (i.e. the machine is in state q_{k_i}, $i < n$, after the input has been read) then i is written on the work tape and the computation continues as with T', if the input is n (i.e. the machine is in state q_{k_n} after the input is read) then m, where m is the value $\tau(n)$, is immediately written on the I/O tape, without moving the work tape head and using as few steps as possible, and if the input is greater than n (i.e. the machine is in state q_{k_n} and the input still is not completely read) then it is copied onto the work tape and processing continues as with T'.

The new machine, call it T", uses a 'table' of instructions

to look up the input. If the input is n, then the table indicates that the output m is to be immediately written without any real computations, otherwise the table indicates that processing should continue as with T'. Since T' computes τ and since $m = \tau(n)$ we have that T'' computes τ. Furthermore, on input n, T'''s work tape head visits only one cell, (i.e. the leftmost cell of the work tape) and uses very few steps (i.e. only enough steps to read the input and print the output, approximately $n+\tau(n)$ steps).

By a slight modification of the above argument or by repeated application of it, we can see that any partial function computed by a Turing machine can be computed quite efficiently, in terms of time and tape, on any finite number of inputs for which the computation terminates.

It is interesting to notice that no matter how efficiently the computation is done, the time taken always gives a bound to the value of the result, since it takes time to write the answer. However, the amount of work tape used does not necessarily reflect the value of the result. In fact table lookups require about $n+\tau(n)$ steps, but only one work tape cell. Furthermore, this one work tape cell is not used for any purpose, it is visited only because the work tape head was initially placed there.

Notice that we seem to have used less work tape on input n at the expense of using more on other inputs, since now the inputs must be copied onto the work tape. This is a result of our choice of a model for Turing machines and of our method of doing table lookups rather than an intrinsic property of table lookups.

CHAPTER 3

THE COMPUTABLE FUNCTIONS AND CHURCH'S THESIS

§ 3.0 Discussion

We would like now to look at a formal definition of the computable functions. As well, we will look at 'Church's Thesis', the claim that this definition does, in fact, capture the intuitive notion of computability.

We also discuss the fact that most of the proofs in this paper hinge on the intuitive notion of computability rather than detailing a more formal proof. In § 3.2 we give some examples of this informal approach to proofs involving computability.

§ 3.1 Church's Thesis

It has been demonstrated that every accepted mathematical formulation of the intuitive notion of computability is in some sense equivalent to the Turing machine formulation. This, together with strong arguments by various people who have developed some of these formulations have convinced most readers of the literature that we have captured the notion of computability by use of a Turing machine formulation. Thus we define the computable partial functions and predicates as follows.

3.1.1 DEFINITION. A partial function ψ on $\underline{N}^k$ is computable if ψ is computed by some Turing machine.

A partial predicate π on $\underline{N}^k$ is computable if there is a computable partial function ψ on $\underline{N}^k$ $\ni \forall \underline{n} \in \underline{N}^k$,

$$\psi(\underline{n}) \simeq \begin{cases} 1 & \text{if } \pi(\underline{n}) \text{ is TRUE} \\ 0 & \text{if } \pi(\underline{n}) \text{ is FALSE} \\ \text{undefined} & \text{if } \pi(\underline{n}) \text{ is undefined.} \end{cases}$$

$(\tau_i^k)_{i \in \underline{N}}$ is called the standard numbering of the computable partial functions on $\underline{N}^k$.

Reasons for using partial functions rather than just total functions in developing computability will be investigated in Chapter 6.

The claim that we have captured the intuitive notion of computability with the above definition has come to be known as 'Church's Thesis'. Church's Thesis has evolved into a justification for lack of formality in many proofs involving computability. Rather than actually

demonstrating the existence of a Turing machine which computes a certain function, we often only demonstrate that this function is intuitively computable. It is left to the reader to convince himself as to the existence of the appropriate Turing machine.

The use of Church's Thesis in this manner can be extremely useful in reducing the length and detail of a proof, thus increasing clarity. However it also introduces many traps and pitfalls to the reader or student with a weak intuition in computability. Any author making use of Church's Thesis in this manner should exercise extreme caution and should be prepared to back up his proof with more rigorous detail if challenged.

In this paper we use Church's Thesis in this manner extensively, as it is the intuitive drive behind the proofs that is of interest to us.

Above, we discussed Church's Thesis as a claim that the intuitively computable functions are computable in the formal sense. Another aspect of Church's Thesis, often called the Converse to Church's Thesis, is the claim that formally computable functions are intuitively computable. Here we must look more closely at precisely what we mean by an intuitively computable function. As an example, look at the function f defined on $\underline{N}$ by

$$f(n) =_{\text{defn}} \begin{cases} 1 & \text{if I have a dime in my pocket as I write this} \\ 0 & \text{otherwise.} \end{cases}$$

f is formally computable, since it is either identically 0 or identically 1, and each of these functions can be computed by a

Turing machine. However, is it intuitively computable? We will agree yes, it is computable, but we may never be able to decide which procedure computes it.

Thus, we must treat the question as to whether a function is computable as a question distinct from whether or not we can decide how to compute it.

§ 3.2 Some Results

We will now look at a few simple results involving computable functions which exemplify the use of Church's Thesis as described in the previous section. We again remind the reader that these proofs should be considered informal. The formalization would involve a demonstration that the appropriate Turing machines exists.

We first show that the composition of computable functions is computable.

3.2.1 PROPOSITION. Let ψ be a computable partial function on $\underline{N}^{k'}$, and $\psi_1,\psi_2,\ldots,\psi_{k'}$ be computable partial functions on $\underline{N}^{k}$. Then

$$\lambda n_1,n_2,\ldots,n_k[\psi(\psi_1(n_1,n_2,\ldots,n_k),\ldots,\psi_{k'}(n_1,n_2,\ldots,n_k))]$$

is computable.

Proof. The required function is computed by the following procedure.

On input $(n_1,n_2,\ldots,n_k)$,

(1) Compute, in turn

$$\psi_1(n_1,n_2,\ldots,n_k)$$
$$\psi_2(n_1,n_2,\ldots,n_k)$$
$$\vdots$$
$$\psi_{k'}(n_1,n_2,\ldots,n_k).$$

If any of these is undefined, then the result of the procedure is undefined, otherwise denote these values by $p_1,p_2,\ldots,p_{k'}$, respectively.

(2) Compute $\psi(p_1,p_2,\ldots,p_{k'})$. If this is defined then output the result and halt, else the result is undefined.

Clearly, this procedure computes the required function hence it is

evident, by Church's Thesis, that the required function is computable.□

Since computable predicates are defined in terms of computable functions we have as a corollary to the above PROPOSITION that composition of a computable partial predicate with computable functions is computable.

3.2.2 COROLLARY. Let π be a computable partial predicate on $\underline{N}^{k'}$. Let $\psi_1,\psi_2,\ldots,\psi_k$, be computable partial functions on $\underline{N}^k$. Then

$$\lambda n_1,n_2,\ldots,n_k[\pi(\psi_1(n_1,n_2,\ldots,n_k),\ldots,\psi_k,(n_1,n_2,\ldots,n_k))]$$

is computable.

Proof. Clear from DEFINITION 3.1.1 (definition of computable predicate) and PROPOSITION 3.2.1.□

A result closely tied to the result in § 2.3 on table lookups is the result that changing the value of a computable function on a finite number of arguments does not change its computability. This result follows from a finite number of applications of the following proposition.

.2.3 PROPOSITION. Let ψ be a computable partial function on $\underline{N}^k$. Let $\underline{i} \in \underline{N}^k$ and $m \in \underline{N}$. Then ψ_1 and ψ_2 defined on $\underline{N}^k$ as follows, are both computable.

$$\psi_1\underline{n} \simeq_{defn} \begin{cases} \text{undefined} & \text{if } \underline{n} = \underline{i} \\ \psi\,\underline{n} & \text{otherwise.} \end{cases}$$

$$\psi_2\underline{n} \simeq_{defn} \begin{cases} m & \text{if } \underline{n} = \underline{i} \\ \psi\,\underline{n} & \text{otherwise.} \end{cases}$$

Proof. Clearly, the following procedure computes ψ_1.

On input $\underline{n}$,

(1) If $\underline{n} = \underline{i}$ then go to (4).

(2) Compute $\psi\,\underline{n}$. If this is undefined then the result of the procedure is undefined.

(3) Output $\psi\,\underline{n}$, halt.

(4) Go to (4).

Evidently ψ_1 is computable. Clearly ψ_2 is computed by the same procedure with (4) replaced by

(4)' Output m, halt.

Thus it is evident that ψ_2 is also computable.□

Statement (1) of the procedure in the above proof corresponds to a table lookup. Statement (4) indicates one method by which we may force a procedure to be undefined. For the sake of clarity of such statements in future proofs, we will use statements which more clearly indicate their function. Thus (4) might be replaced by

(4) Let the result be undefined.

The proof of the following theorem is essentially given by the construction of a universal machine (see § 1.3), thus we do not appeal to Church's Thesis in its proof. It is often called the Universal Machine Theorem.

3.2.4 THEOREM. For each k there is a computable partial function μ_k on $\underline{N}^{k+1}$ $\ni \forall i \in \underline{N}, \forall n_1, n_2, \ldots, n_k \in \underline{N}$,

$$\mu_k(i, n_1, n_2, \ldots, n_k) \simeq \tau_i^k(n_1, n_2, \ldots, n_k).$$

Proof. By construction of a universal machine as outlined in § 1.3.□

However, we do appeal to Church's Thesis in order to prove the following result. This result is closely related to the s_n^m Theorem of recursive function theory. (See Rogers [8]).

3.2.5 THEOREM. For every computable partial function ψ on $\underline{N}^{k+1}$ there is a computable total function s on $\underline{N}^k$ $\ni$ $\forall i_1,i_2,\ldots,i_k \in \underline{N}$, $\forall n \in \underline{N}$,

$$\tau^k_{s(i_1,i_2,\ldots,i_k)}(n) \simeq \psi(i_1,i_2,\ldots,i_k,n).$$

Proof. Let T be a Turing machine which computes ψ. Clearly the following procedure computes s.

On input $(i_1,i_2,\ldots,i_k)$,

(1) Modify the instructions of T to a new machine T' so that on input n, T' writes $(i_1,i_2,\ldots,i_k)$ on its work tape, then mimics the computation of T on $(i_1,i_2,\ldots,i_k,n)$.

(2) Compute and output the Gödel number of T'.

Evidently s is computable.□

Many details have been left out of (1) of the procedure in the above proof for brevity. Since this result is not central to the paper, let it suffice to say that the result can be proven in a more rigorous manner.

CHAPTER 4

BLUM'S AXIOMS

§ 4.0 Discussion

In this chapter we will present the axioms for abstract complexity theory which will be used throughout the remainder of the paper. These axioms tie together two notions. One is the notion of a Gödel numbered list of computing devices. This notion is captured by the definition of an acceptable numbering in § 4.2. In order to intuitively develop this definition, § 4.1 presents a strong background to acceptable numberings.

The second notion is that of measuring complexity by counting resource units used by the computing devices. The type of resource measured must be closely tied to the way in which the devices work so that the number of resource units used does, in fact, reflect the complexity of the computation. The definition of a measured sequence in § 4.3 partially captures this notion. (We have seen particular examples of such resources in CHAPTER 2.)

In § 4.4 we present the axioms of abstract complexity theory. These tie together the above two notions. The reader should be aware, however, that the concepts of acceptable numberings and measured sequences are of independent interest as well as being useful in defining abstract complexity measures.

§ 4.1 Universal Partial Functions

The notion of a universal machine can be extended to an analogous notion for any indexing of partial functions. We will define the notion of an arbitrary universal partial function but will concentrate our attention on only those universal partial functions which are computable.

The following definition defines the universal partial function for any indexing of partial functions on $\underline{N}$, $(\psi_{\underline{i}})_{\underline{i}\in\underline{N}^k}$. The two main theorems of this section indicate that for our purposes we need only consider the notion for sequences, $(\psi_i)_{i\in\underline{N}}$. Indeed, throughout most of the paper we make use of the notion only for such sequences. The more general notion plays a role in some proofs in CHAPTER 8, and is used only in the few places that are essential for those proofs.

In the paper we will deal only with indexings of partial functions on $\underline{N}$. Analogous results hold for indexings of partial functions on $\underline{N}^m$, for each m.

4.1.1 DEFINITION. Let $(\psi_i)_{i\in\underline{N}}$ be a sequence of partial functions on $\underline{N}$. The partial function μ on $\underline{N}^2$ defined by

$$\mu(i,n) \simeq_{\text{defn}} \psi_i(n)$$

is the universal partial function for $(\psi_i)_{i\in\underline{N}}$.

More generally, if $(\psi_{\underline{i}})_{\underline{i}\in\underline{N}^k}$ is an indexing of partial functions on $\underline{N}$, the function μ on $\underline{N}^{k+1}$ defined by

$$\mu(i_1,i_2,\ldots,i_k,n) \simeq_{\text{defn}} \psi_{(i_1,i_2,\ldots,i_k)}(n)$$

is the universal partial function for $(\psi_{\underline{i}})_{\underline{i}\in\underline{N}^k}$.

We will sometimes abbreviate 'universal partial function for $(\psi_i)_{i\in\underline{N}}$' to 'universal for $(\psi_i)_{i\in\underline{N}}$'.

We mentioned earlier that our concern is with computable universal partial functions. Hence we give the following definition.

4.1.2 DEFINITION. Let $(\psi_i)_{i\in\underline{N}}$ be a sequence of partial functions on $\underline{N}$. If the universal partial function for $(\psi_i)_{i\in\underline{N}}$ is computable then $(\psi_i)_{i\in\underline{N}}$ is a computably enumerable sequence of partial functions.

Hence, since the universal partial function for our standard numbering, $(\tau^1_i)_{i\in\underline{N}}$, can be shown to be computable by the construction of a universal machine (THEOREM 3.2.4), $(\tau^1_i)_{i\in\underline{N}}$ is computably enumerable.

Where does the phrase computably enumerable come from? The following theorem helps answer this and also generalizes the following notion.

If f is a computable total function then the sequence of partial functions $(\tau^1_{f(i)})_{i\in\underline{N}}$ has a universal function. That is, there is a computable function μ_f such that $\forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\mu_f(i,n) \simeq \tau^1_{f(i)}(n).$$

Such a function μ_f can be computed by a machine U_f which works as follows. On input (i,n), U_f reads i, computes $f(i)$, then works like U on input $(f(i),n)$, where U computes the universal function for $(\tau^1_i)_{i\in\underline{N}}$. Basically the above is just the result that compositions of computable partial functions are computable.

4.1.3 THEOREM. For any sequence of partial functions on $\underline{N}$, $(\psi_i)_{i\in\underline{N}}$, the following are equivalent.

(1) $\forall$ computable total f on $\underline{N}^k$, the universal partial function ψ on $\underline{N}^{k+1}$ for $(\psi_{f\underline{i}})_{\underline{i}\in\underline{N}^k}$ is computable.

(2) $\forall$ computable total f on $\underline{N}$, the universal partial function ψ on $\underline{N}^2$ for $(\psi_{f(i)})_{i\in\underline{N}}$ is computable.

(3) $(\psi_i)_{i\in\underline{N}}$ is computably enumerable.

(4) $\exists$ computable total t on $\underline{N}$, $\forall i \in \underline{N}$,

$$\tau^1_{t(i)} = \psi_i .$$

Proof. (1)$\Rightarrow$(2). (2) is a statement of (1) with $k = 1$.

(2)$\Rightarrow$(3). From DEFINITION 4.1.2, it is seen that (3) is a statement of (2) with f defined by

$$f(i) =_{\text{defn}} i .$$

(3)$\Rightarrow$(4). Since $(\psi_i)_{i\in\underline{N}}$ is computably enumerable we have that ψ on $\underline{N}^2$ defined by

$$\psi(i,n) \simeq_{\text{defn}} \psi_i(n) \text{ is computable.}$$

Since ψ is computable we have by THEOREM 3.2.5 that there is a computable total function t on $\underline{N} \ni \forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\tau^1_{t(i)}(n) \simeq \psi(i,n) .$$

Hence, $\forall i \in \underline{N}$,

$$\tau^1_{t(i)} = \psi_i .$$

(4)$\Rightarrow$(1). By THEOREM 3.2.4, choose computable partial μ_1 on $\underline{N}^2$ which is universal for $(\tau^1_i)_{i\in\underline{N}}$. Let t be given by (4). Given f as in (1), define computable partial ψ on $\underline{N}^{k+1}$ by

$$\psi(i_1,i_2,\ldots,i_k,n) \simeq_{\text{defn}} \mu_1(t\circ f(i_1,i_2,\ldots,i_k),n).$$

ψ is universal for $(\psi_{f\underline{i}})_{\underline{i}\in\underline{N}^k}$ since $\forall i_1,i_2,\ldots,i_k \in \underline{N}$, $\forall n \in \underline{N}$,

$$\begin{aligned}\psi(i_1,i_2,\ldots,i_k,n) &\simeq \mu_1(t\circ f(i_1,i_2,\ldots,i_k),n)\\ &\simeq \tau^1_{t\circ f(i_1,i_2,\ldots,i_k)}(n)\\ &\simeq \tau^1_{t(f(i_1,i_2,\ldots,i_k))}(n)\\ &\simeq \psi_{f(i_1,i_2,\ldots,i_k)}(n)\ .\square\end{aligned}$$

We can now see where the name computably enumerable comes from. Given a computably enumerable sequence $(\psi_i)_{i\in\underline{N}}$, the computable total function t, given by property (4) of THEOREM 4.1.3, enumerates indices of partial functions from the standard numbering $(\tau^1_i)_{i\in\underline{N}}$, which are equal to the partial functions in $(\psi_i)_{i\in\underline{N}}$. More precisely, t is computable and, given i, $t(i)$ is an index such that $\tau^1_{t(i)} = \psi_i$. The following corollary is immediate.

4.1.4 COROLLARY. A partial function φ is computable

$\Leftrightarrow \exists$ a computably enumerable sequence $(\psi_i)_{i\in\underline{N}}$ $\ni$

$$\varphi \in \{\psi_i \mid i\in\underline{N}\}.$$

Proof. ($\Rightarrow$) For each $i\in\underline{N}$, define ψ_i on $\underline{N}$ by

$$\psi_i(n) \simeq_{\text{defn}} \varphi(n).$$

Define ψ on $\underline{N}^2$ by

$$\psi(i,n) \simeq_{\text{defn}} \varphi(n).$$

Now, ψ is universal for $(\psi_i)_{i\in\underline{N}}$ and if φ is computable then ψ is

computable, hence $(\psi_i)_{i\in\underline{N}}$ is computably enumerable. Clearly,

$$\varphi \in \{\psi_i \mid i\in\underline{N}\}.$$

($\Leftarrow$) Immediate from the equivalence of (3) and (4) in THEOREM 4.1.3, since $\forall i \in \underline{N}$, $\tau^1_{t(i)}$ is computable.□

In § 6.2 we will see that there are sequences of computable partial functions which are not computably enumerable. Thus if we have a sequence of computable partial functions, $(\psi_i)_{i\in\underline{N}}$, there is no reason to suspect that, from i and n we can compute $\psi_i(n)$. Therefore the the definition of computably enumerable sequences of partial functions is quite significant. The following LEMMA is quite useful in many proofs that rely on the computable enumerability of certain sequences.

4.1.5 LEMMA. Let $(\psi_i)_{i\in\underline{N}}$ be computably enumerable.

Then, $\forall$ computable f on $\underline{N}$, $(\psi_{f(i)})_{i\in\underline{N}}$ is computably enumerable.

Proof. Immediate from (3)$\Rightarrow$(2) in THEOREM 4.1.3, and DEFINITION 4.1.2.□

The properties in THEOREM 4.1.3 can be viewed as various statements for arbitrary sequences of partial functions analagous to the statement of THEOREM 3.2.4. Similarly, the properties in the following theorem correspond to THEOREM 3.2.5.

4.1.6 THEOREM. For any sequence of partial functions on $\underline{N}$, $(\psi_i)_{i\in\underline{N}}$, the following are equivalent.

(1) $\forall$ computable partial ψ on $\underline{N}^{k+1}$, $\exists$ computable total f on $\underline{N}^k$ $\ni$

ψ is universal for $(\psi_{f\underline{i}})_{\underline{i}\in\underline{N}^k}$.

(2) $\forall$ computable partial ψ on $\underline{N}^2$, $\exists$ computable total f on $\underline{N}$ $\ni$

ψ is universal for $(\psi_{f(i)})_{i\in\underline{N}}$.

(3) $\exists$ computable total s on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$,

$$\psi_{s(i)} = \tau^1_i.$$

Proof. (1)$\Rightarrow$(2) (2) is a restatement of (1) with $k = 1$.

(2)$\Rightarrow$(3) Choose computable μ_1 on $\underline{N}^2$ universal for $(\tau^1_i)_{i\in\underline{N}}$. Let s be the f given by (2). Then $\forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\psi_{s(i)}(n) \simeq \mu_1(i,n)$$

$$\simeq \tau^1_i(n).$$

Therefore, $\forall i \in \underline{N}$,

$$\psi_{s(i)} = \tau^1_i.$$

(3)$\Rightarrow$(1) Let ψ on $\underline{N}^{k+1}$ be given as in (1). Choose s as in (3). By the THEOREM 3.2.5 choose computable total s' on $\underline{N}$ $\ni$ $\forall i_1, i_2, \ldots, i_k \in \underline{N}$, $\forall n \in \underline{N}$,

$$\tau^1_{s'(i_1,i_2,\ldots,i_k)}(n) \simeq \psi(i_1,i_2,\ldots,i_k,n).$$

Let $f =_{\text{defn}} s\circ s'$. Then $\forall i_1, i_2, \ldots, i_k \in \underline{N}$, $\forall n \in \underline{N}$,

$$\psi(i_1,i_2,\ldots,i_k,n) \simeq \tau^1_{s'(i_1,i_2,\ldots,i_k)}(n)$$

$$\simeq \psi_{s(s'(i_1,i_2,\ldots,i_k))}(n)$$

$$\simeq \psi_{f(i_1,i_2,\ldots,i_k)}(n). \quad \square$$

Notice that since any sequence of partial functions has exactly one universal function we have

(a) $\forall$ computable total f on $\underline{N}$, the universal partial function ψ on $\underline{N}^2$ for $(\psi_{f(i)})_{i\in\underline{N}}$ is computable,

is equivalent to

(b) $\forall$ computable total f on $\underline{N}$, $\exists$ computable partial ψ on $\underline{N}^2$, $\ni$ ψ is universal for $(\psi_{f(i)})_{i\in\underline{N}}$.

Thus, a beautiful symmetry between THEOREMS 4.1.3 and 4.1.6 is seen. THEOREM 4.1.3 gives us

$\forall$ computable total f on $\underline{N}$, $\exists$ computable partial ψ on $\underline{N}^2$, $\ni$ ψ is universal for $(\psi_{f(i)})_{i\in\underline{N}}$

$\Leftrightarrow$ $\exists$ computable total t on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$, $\tau^1_{t(i)} = \psi_i$,

and THEOREM 4.1.6 gives us

$\forall$ computable partial ψ on $\underline{N}^2$, $\exists$ computable total f on $\underline{N}$, $\ni$ ψ is universal for $(\psi_{f(i)})_{i\in\underline{N}}$

$\Leftrightarrow$ $\exists$ computable total s on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$, $\psi_{s(i)} = \tau^1_i$.

At times we will use statements like (b) above to replace statements like (a) in order to emphasize this symmetry. When this is done, the reader should realize that it is the computability of ψ that is most significant, not the existence of ψ.

§ 4.2 Acceptable Numberings

In Chapter 2 we looked at two resources that seemed appropriate for use in measuring complexity of computations. These resource measures were based on a Turing machine characterization of computability. We mentioned that both of these measures satisfy the definition of a complexity measure.

Abstract complexity theory is blessed with yet another generality. It encompasses not only many measures on a specific characterization of computations, but also measures on any reasonable characterization of computability. In this section we attempt to describe precisely what we mean by a reasonable characterization of computability.

We expect that in any such characterization we should be able to number the computing devices so we can do the following.

(a) Given a number we can construct the associated device.

(b) Given a device we can compute the associated number.

That is, we should be able to Gödel number the set of computing devices satisfying the characterization.

The above two properties are device dependent. The concept of an acceptable numbering is a device independent concept which seems to capture very well the notion of a Gödel numbered set of computing devices satisfying a certain characterization. We will see that the essence of (a) above is captured by one, hence all, of the properties of THEOREM 4.1.3 and the essence of (b) similarily corresponds to THEOREM 4.1.6.

4.2.1 DEFINITION. A sequence of partial functions on $\underline{N}$, $(\varphi_i)_{i\in\underline{N}}$, is an acceptable numbering of the computable partial functions on $\underline{N}$, if $(\varphi_i)_{i\in\underline{N}}$ satisfies

(1) $\forall$ computable total f on $\underline{N}$, $\exists$ computable partial ψ on $\underline{N}^2$,

$$\psi \text{ is universal for } (\varphi_{f(i)})_{i\in\underline{N}}.$$

(2) $\forall$ computable partial ψ on $\underline{N}^2$, $\exists$ computable total f on $\underline{N}$,

$$\psi \text{ is universal for } (\varphi_{f(i)})_{i\in\underline{N}}.$$

If we think of the sequence of functions, $(\varphi_i)_{i\in\underline{N}}$, as a list of computing devices, then (1) of the definition captures (a) above, in that we think of ψ as a procedure which on i and n, computes $f(i)$, constructs $\varphi_{f(i)}$ from $f(i)$, then runs $\varphi_{f(i)}$ on n. As well, (2) of the definition captures (b) above in that we think of f as a procedure which, on input i, modifies a given device which computes ψ to a device φ which computes $\lambda n[\psi(i,n)]$ then computes an index $f(i)$ for φ, so that $\varphi_{f(i)} = \varphi$.

From THEOREMS 4.1.3 and 4.1.6 it is evident that an acceptable numbering forms the set of all the computable partial functions.

4.2.2 PROPOSITION. Let $(\varphi_i)_{i\in\underline{N}}$ be an acceptable numbering and P be the set of all the computable partial functions on $\underline{N}$. Then

$$P = \{\varphi_i \mid i\in\underline{N}\}.$$

Proof: We show that $\{\varphi_i \mid i \in \underline{N}\} = \{\tau^1_i \mid i \in \underline{N}\}$.

Since $(\varphi_i)_{i \in \underline{N}}$ satisfies (1) of DEFINITION 4.2.1 we have, by THEOREM 4.1.3, that there is a total function $t \ni \forall i \in \underline{N}$,

$$\tau^1_{f(i)} = \varphi_i .$$

Hence, $\{\varphi_i \mid i \in \underline{N}\} \subseteq \{\tau^1_i \mid i \in \underline{N}\}$.

By (2) of DEFINITION 4.2.1 and by THEOREM 4.1.6 we have a total function $\_ \ni \forall i \in \underline{N}$,

$$\varphi_{s(i)} = \tau^1_i .$$

Hence, $\{\tau^1_i \mid i \in \underline{N}\} \subseteq \{\varphi_i \mid i \in \underline{N}\}$. □

The computable total functions t and s, given by THEOREMS 4.1.3 and 4.1.6 give a close tie between an acceptable numbering and our standard numbering. This tie allows us to shift our attention away from the standard numbering toward any given acceptable numbering. For instance, we can characterize computably enumerable sequences of partial functions on $\underline{N}$ by the ability to computably enumerate indices in any acceptable numbering rather than enumerate indices in our standard numbering as in (4) of THEOREM 4.1.3.

4.2.3 PROPOSITION. Let $(\varphi_i)_{i \in \underline{N}}$ be any acceptable numbering, and $(\psi_i)_{i \in \underline{N}}$ be any sequence of partial functions on $\underline{N}$. Then,

$(\psi_i)_{i \in \underline{N}}$ is computably enumerable

$\Leftrightarrow \exists$ computable total s on $\underline{N}$, $\forall i \in \underline{N}$,

$$\varphi_{s(i)} = \psi_i .$$

Proof: ($\Rightarrow$). Let $(\psi_i)_{i\in\underline{N}}$ be computably enumerable. Choose computable total t on $\underline{N} \ni \forall i \in \underline{N}$,

$$\tau^1_{t(i)} = \psi_i. \quad \text{(By (3)}\Rightarrow\text{(4) of THEOREM 4.1.3).}$$

Choose computable total f on $\underline{N} \ni \forall i \in \underline{N}, \forall n \in \underline{N}$,

$$\varphi_{f(i)}(n) \simeq \mu_1(i,n)$$

where μ_1 is universal for $(\tau^1_i)_{i\in\underline{N}}$. (By (2) of DEFINITION 4.2.1).

Let $s =_{\text{defn}} f\circ t$. Then $\forall i \in \underline{N}, \forall n \in \underline{N}$,

$$\begin{aligned}\varphi_{s(i)}(n) &\simeq \varphi_{f(t(i))}(n)\\ &\simeq \mu_1(t(i),n)\\ &\simeq \tau^1_{t(i)}(n)\\ &\simeq \psi_i(n).\end{aligned}$$

Thus, s is computable since f and t are, and $\forall i \in \underline{N}$,

$$\varphi_{s(i)} = \psi_i.$$

($\Leftarrow$). Suppose $\exists$ computable total $s \ni \forall i \in \underline{N}$,

$$\varphi_{s(i)} = \psi_i.$$

Choose computable partial ψ on $\underline{N}^2 \ni \forall i \in \underline{N}, \forall n \in \underline{N}$,

$$\psi(i,n) \simeq \varphi_i(n). \quad \text{(by (1) of DEFINITION 4.2.1)}$$

Define computable partial μ on $\underline{N}^2$ by

$$\mu(i,n) \simeq_{\text{defn}} \psi(s(i),n).$$

Then, $\forall i \in \underline{N}, \forall n \in \underline{N}$, $\quad \mu(i,n) \simeq \psi(s(i),n)$

$$\begin{aligned}&\simeq \varphi_{s(i)}(n)\\ &\simeq \psi_i(n).\end{aligned}$$

Thus, μ is universal for $(\psi_i)_{i\in\underline{N}}$. Therefore $(\psi_i)_{i\in\underline{N}}$ is computably enumerable.□

The following Lemma is immediate from (1) of DEFINITION 4.2.1 and (2)⇒(3) of THEOREM 4.1.3.

4.2.4 LEMMA. Every acceptable numbering is computably enumerable.

Proof. Immediate as indicated above.□

The following proposition demonstrates just how closely related acceptable numberings and our standard numbering are.

4.2.5 PROPOSITION. A sequence of partial functions, $(\varphi_i)_{i\in\underline{N}}$, is an acceptable numbering ⇔

(1) $\exists$ computable total t on $\underline{N}$, $\forall i \in \underline{N}$, $\tau^1_{t(i)} = \varphi_i$

and (2) $\exists$ computable total s on $\underline{N}$, $\forall i \in \underline{N}$, $\varphi_{s(i)} = \tau^1_i$.

Proof. Clear from the definition of an acceptable numbering and THEOREMS 4.1.3 and 4.1.6. □

We now turn our attention to measured sequences.

§ 4.3 Measured Sequences

We have seen that each of the three resource measures defined in Chapter 2 has the property that we can compute, given i,n,m whether or not the number of resource units used by the i^{th} machine on input n is m.

The axioms of complexity theory attempt to encompass measures which are obtained by counting resource units used by computations, such as time and tape measures. The type of resource measured should be so closely tied to the computation that we can tell when a computation has reached a point beyond which no more resources will be used. Furthermore, if such a point is reached, we should be able to say something very definite about whether or not a result for the computation is defined. This is intimately tied to the ability to compute, given i,n,m, whether or not the i^{th} machine on input n uses m resource units, as exemplified in the discussion of time and tape measures in Chapter 2.

The above discussion motivates the following definition of measured sequences of partial functions on $\underline{N}$.

4.3.1 DEFINITION. Let $(\psi_i)_{i\in\underline{N}}$ be a sequence of partial functions on $\underline{N}$. $(\psi_i)_{i\in\underline{N}}$ is a measured sequence if

$$\lambda i,n,m[\psi_i(n) \simeq m]$$

is a computable total predicate on $\underline{N}^3$.

A partial function, ψ, on $\underline{N}$ has a computable graph if

$$\lambda n,m[\psi(n) \simeq m]$$

is a computable total predicate on $\underline{N}^2$.

From the definition it is clear that measured sequences of functions and functions with computable graphs are closely related. This relationship between measured sequences and functions with computable graphs seems to parallel the relationship between computably enumerable sequences and computable partial functions. This observation leads us to PROPOSITION 4.3.2 and indirectly to PROPOSITION 4.3.4. Notice the similarity between PROPOSITION 4.3.2 and COROLLARY 4.1.4 .

4.3.2 PROPOSITION. φ partial function has a computable graph $\Leftrightarrow \exists$ a measured sequence $(\psi_i)_{i\in N} \ni \varphi \in \{\psi_i \mid i\in N\}$.

Proof. ($\Rightarrow$). Let φ have a computable graph. For each $i\in N$ define ψ_i by $\psi_i =_{defn} \varphi$. To compute whether or not $\psi_i(n) \simeq m$, compute whether or not $\varphi(n) = m$. Evidently $(\psi_i)_{i\in N}$ is a measured sequence.

($\Leftarrow$). Let $\varphi \in \{\psi_i \mid i\in N\}$ where $(\psi_i)_{i\in N}$ is a measured sequence. Given $i\in N \ni \varphi = \psi_i$, a procedure to compute, given n,m, whether or not $\varphi(n) \simeq m$ is as follows.

On input (n,m).

(1) If $\psi_i(n) = m$ then output TRUE and halt.

(2) Output FALSE and halt.

Since such i exists, the above procedure exists, hence $\lambda n,m[\varphi(n) \simeq m]$ is computable.□

The reader may be uneasy about the ($\Leftarrow$) proof of the above proposition, since given φ we may not be able to compute i such that $\psi_i = \varphi$. However, we only need to show that a procedure for computing $\lambda n,m[\varphi(n) \simeq m]$ exists, we are not required to construct such a procedure.

Hence, we do not need to be able to compute i, but need only know that i exists, for if i exists, then the procedure given in the proof exists.

We now demonstrate that functions with computable graphs are computable.

4.3.3 PROPOSITION. Let ψ be any partial function on $\underline{N}$ with a computable graph. ψ is computable.

Proof. Clearly, the following procedure computes ψ.

On input n.

(1) Set m to 0.

(2) If $\psi(n) = m$, output m and halt.

(3) Set m to $m+1$.

(4) Go to (2).

(Notice that if $\psi(n)$ is undefined then the procedure loops through (2), (3), (4) forever, thus is undefined.) Evidently, ψ is computable.□

Notice that $\forall n \in \underline{N}$,

$$\psi(n) \simeq \mu m[\psi(n) \simeq m].$$

It is clear that, by mimicking the above proof, we can show that any function ψ on $\underline{N}^k$ is computable if $\forall n_1, n_2, \ldots, n_k \in \underline{N}$,

$$\psi(n_1, n_2, \ldots, n_k) \simeq \mu m[P(n_1, n_2, \ldots, n_k, m)]$$

where P is a computable total predicate on $\underline{N}^{k+1}$. Thus, such proofs as the above will, henceforth, be abbreviated to something like the following.

Proof. Since $\forall n \in \underline{N}$,

$$\psi(n) \simeq \mu m[\psi(n) \simeq m]$$

and since $\lambda n,m[\psi(n) \simeq m]$ is a computable total predicate, it is evident that ψ is computable.□

Since functions with computable graphs are computable, we have as an immediate corollary to PROPOSITION 4.3.2 and PROPOSITION 4.3.3 that each function in a measured sequence is computable.

4.3.4 COROLLARY. If $(\psi_i)_{i\in\underline{N}}$ is a measured sequence then each ψ_i is computable.

Proof. Immediate from PROPOSITION 4.3.2 and PROPOSITION 4.3.3.□

The proof of the next proposition also makes use of the μ-operator. The reader should attempt to construct a procedure which actually computes the function ψ in this proof. If any difficulty is encountered, he should refer back to the two proofs of PROPOSITION 4.3.3. This proposition is due to Blum [2].

4.3.5 PROPOSITION. Every measured sequence is computably enumerable.

Proof. Let $(\psi_i)_{i\in\underline{N}}$ be a measured sequence. Define a partial function ψ on $\underline{N}^2$ by

$$\psi(i,n) \simeq_{\text{defn}} \mu m[\psi_i(n) \simeq m].$$

Since $\lambda i,n,m[\psi_i(n) \simeq m]$ is a computable total predicate, ψ is evidently computable. Clearly, $\forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\psi(i,n) \simeq \psi_i(n).$$

Thus ψ is universal for $(\psi_i)_{i\in\underline{N}}$. Since ψ is computable, $(\psi_i)_{i\in\underline{N}}$ is computably enumerable.□

The following lemma will be useful in various proofs later in the paper.

4.3.6 LEMMA. $(\psi_i)_{i\in N}$ is a measured sequence

$\Leftrightarrow$ $\lambda i,n,m[\psi_i(n) \lesssim m]$ is a computable total predicate.

Proof. ($\Rightarrow$). Let $(\psi_i)_{i\in N}$ be a measured sequence. The following procedure tells us, given i,n,m, whether or not $\psi_i(n) \lesssim m$. Thus, it is evident that $\lambda i,n,m[\psi_i(n) \lesssim m]$ is computable.

On input i,n,m,

(1) Set m' to 0.

(2) If $m' > m$, answer FALSE and halt.

(3) If $\psi_i(n) = m'$, answer TRUE and halt.

(4) Set m' to $m'+1$.

(5) Go to (2).

($\Leftarrow$). If $\lambda i,n,m[\psi_i(n) \lesssim m]$ is computable, then the following procedure computes $\lambda i,n,m[\psi_i(n) \eqsim m]$.

On input i,n,m,

(1) If $\psi_i(n) \lesssim m$ is FALSE, then answer FALSE and halt.

(2) If $m = 0$, then answer TRUE and halt.

(3) If $\psi_i(n) \leq m-1$ then answer FALSE and halt.

(4) Answer TRUE and halt.

Evidently $\lambda i,n,m[\psi_i(n) \eqsim m]$ is computable and total hence $(\psi_i)_{i\in N}$ is measured.□

In future proofs we will appeal more and more to the reader's intuition about what is computable. For instance, the above proof might be abbreviated to

Proof. ($\Rightarrow$). Define a predicate P on $\underline{N}^3$ by

$$P(i,n,m) =_{defn} \begin{cases} \text{TRUE} & \exists m' \leq m \ni \psi_i(n) = m' \\ \text{FALSE} & \text{otherwise.} \end{cases}$$

Evidently P is computable and

$$P = \lambda i,n,m[\psi_i(n) \lesssim m].$$

($\Leftarrow$). Define a predicate P on $\underline{N}^3$ by

$$P(i,n,m) =_{defn} \begin{cases} \text{TRUE} & \psi_i(n) \leq m \quad \text{is TRUE} \quad \text{and} \\ & \psi_i(n) \lesssim m-1 \quad \text{is FALSE (or } m = 0) \\ \text{FALSE} & \text{otherwise.} \end{cases}$$

Evidently P is computable and

$$P = \lambda i,n,m[\psi_i(n) \gtrsim m]. \quad \square$$

Also, if $\lambda i,n,m[\psi_i(n) \leq m]$ is computable, then we can compute whether $\psi_i(n) < m$. This can be done since $\psi_i(n) < m$ if and only if $m \neq 0$ and $\psi_i(n) \leq m-1$.

We now turn our attention to stating the axioms for abstract complexity measures.

§ 4.4 Axioms for Complexity Measures

In [2], Manuel Blum presents the axioms for complexity measures which we will use in this paper. These axioms seem slightly stronger than what is really desirable, since they do not allow complexity measures that let the number of resource units used by a computation be defined while the result of the computation is undefined. In [1], Ausiello presents weaker, more acceptable axioms. We present them here in order that we may compare the two sets of axioms.

4.4.1 DEFINITION. An ordered pair of sequences of partial functions on $\underline{N}$, $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ is a weak complexity measure if

(1) $(\varphi_i)_{i\in\underline{N}}$ is an acceptable numbering.

(2) $(\Phi_i)_{i\in\underline{N}}$ is a measured sequence.

(3a) $\forall i \in \underline{N}, \forall n \in \underline{N},$

$$\varphi_i(n) \text{ defined} \Rightarrow \Phi_i(n) \text{ defined.}$$

(3b) There is a computable partial predicate π on $\underline{N}^2$ ∋

$\forall i \in \underline{N}, \forall n \in \underline{N},$

$$\Phi_i(n) \text{ defined} \Rightarrow \pi(i,n) \text{ defined}$$

$$\text{and } \varphi_i(n) \text{ defined} \Leftrightarrow \pi(i,n) \text{ holds.}$$

Intuitively, the acceptable numbering should be viewed as a Gödel numbered sequence of computing devices, and the measured sequence as functions which count some type of resource units used by the corresponding devices. That is, $\Phi_i(n)$ will be the number of resource units used by the computation $\varphi_i(n)$.

Certainly we should have that if the result of the i^{th} device on input n is defined, then the number of resource units used for that computation should be finite. Hence we have axiom (3a). Axiom (3b) was discussed, to some extent, at the beginning of § 4.3. The type of resource that interests us must be so closely tied to the computations that if the number of resource units used in a computation is finite, then we can compute whether or not the result for the computation is defined.

The following definition gives the axioms as they were orginally presented by Blum. Notice that axiom (3) requires that if the number of resource units used is defined, then the computation must define a result.

4.4.2 DEFINITION. An order pair of sequences of partial functions on $\underline{N}$, $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ is a complexity measure if

(1) $(\varphi_i)_{i\in\underline{N}}$ is an acceptable numbering.

(2) $(\Phi_i)_{i\in\underline{N}}$ is a measured sequence.

(3) $\forall i \in \underline{N}, \forall n \in \underline{N}$,

$$\varphi_i(n) \text{ defined} \Leftrightarrow \Phi_i(n) \text{ defined.}$$

It is interesting that, historically, Blum's axioms appeared first, and they were not weakened to the more 'natural' axioms by Ausiello until several years later. Ausiello's reasons for weakening the Blum's Axioms seem to be these:

Firstly, the weaker axioms demonstrate more clearly how we show that certain measures are, in fact, complexity measures. For example, with the tape measure $(\lambda^*_i)_{i\in\underline{N}}$, in order to show we can tell whether or not $\lambda^*_i(n) = m$, we actually demonstrate the existence of the predicate π of axiom (3b) in DEFINITION 4.4.1.

Secondly, Ausiello's paper, [1], was a study of cycling computations, that is, those computations which never terminate but use only a finite number of resource units. Hence he needed weaker axioms which allowed a broader spectrum of complexity measures.

Thirdly, it is mathematically more tasteful to use the weaker axioms, as virtually all of the important results that are true using Blum's Axioms, remain true (possibly in a slightly modified form) using Ausiello's axioms. This is because any proof which does not use the fact that '$\Phi_i(n)$ defined $\Rightarrow$ $\varphi_i(n)$ defined' will be true in both systems, and proofs that do use this fact can often be modified by replacing uses of '$\Phi_i(n)$ defined $\Rightarrow$ $\varphi_i(n)$ defined' by uses of

'($\Phi_i(n)$ defined and $\pi(i,n)$ holds) $\Rightarrow$ $\varphi_i(n)$ defined.'

This is the essence of part (2) of the following theorem. The proof of part (2) parallels the construction of λ^* from λ in § 2.2.

4.4.3 THEOREM. (1) Every complexity measure is a weak complexity measure.

(2) Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a weak complexity measure. There is a complexity measure $((\varphi_i)_{i\in\underline{N}}, (\Phi^*_i)_{i\in\underline{N}})$ $\forall i \in \underline{N}, \forall n \in \underline{N}, \quad \varphi_i(n)$ defined $\Rightarrow \Phi_i(n) = \Phi^*_i(n)$.

Proof. (1) Clearly, DEFINITION 4.4.1 is satisfied if π is defined on $\underline{N}$ by $\pi(i,n) =_{defn} \begin{cases} \text{TRUE} & \text{if } \varphi_i(n) \text{ is defined} \\ \text{undefined} & \text{if } \varphi_i(n) \text{ is undefined.} \end{cases}$

(2) Define $(\Phi_i^*)_{i \in \underline{N}}$ by

$$\Phi_i^*(n) =_{defn} \begin{cases} \Phi_i(n) & \text{if } \varphi_i(n) \text{ is defined} \\ \text{undefined} & \text{if } \varphi_i(n) \text{ is undefined.} \end{cases}$$

Clearly $(\Phi_i^*)_{i \in \underline{N}}$ is a sequence of partial functions satisfying $\forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\varphi_i(n) \text{ defined} \Leftrightarrow \Phi_i^*(n) \text{ defined}$$

and $\forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\varphi_i(n) \text{ defined} \Rightarrow \Phi_i^*(n) = \Phi_i(n).$$

It remains to show $(\Phi_i^*)_{i \in \underline{N}}$ is a measured sequence.

Clearly, the following procedure, given, i,n,m, computes whether or not $\Phi_i^*(n) = m$.

On input (i,n,m),

(1) Compute whether or not $\Phi_i(n) = m$.

(2) If $\Phi_i(n) = m$ is FALSE, then answer FALSE and halt.

(3) Compute $\pi(i,n)$. (This is defined since $\Phi_i(n)$ is defined.)

(4) If $\pi(i,n)$ holds, then answer TRUE and halt.

(5) Answer FALSE and halt. □

Throughout the remainder of the paper we use only Blum's Axioms. It is felt that the statements of the theorems and proofs are much more clear using the stronger axioms. The reader may take it upon himself to restate the theorems and proofs using Ausiello's axioms.

The following facts are clear from results in this chapter.

4.4.4 FACTS. Let $((\varphi_i)_{i\in\underline{N}},(\Phi_i)_{i\in\underline{N}})$ be a complexity measure. Then the following are true.

(1) $(\varphi_i)_{i\in\underline{N}}$ is computably enumerable.

(2) $(\Phi_i)_{i\in\underline{N}}$ is computably enumerable.

(3) $\exists$ computable total s, $\forall i \in \underline{N}$, $\varphi_{s(i)} = \Phi_i$.

(4) $\forall i \in \underline{N}$, Φ_i is computable.

CHAPTER 5

THE NOTION OF 'ALMOST EVERYWHERE'

§ 5.0 Definitions and Discussion

Let π be any partial predicate on $\underline{N}$. We say that π holds everywhere if $\pi(n)$ is TRUE for every $n \in \underline{N}$. We say π holds almost everywhere if $\pi(n)$ is TRUE for all but finitely many $n \in \underline{N}$. We say that π holds infinitely often if $\pi(n)$ is TRUE for infinitely many $n \in \underline{N}$.

5.0.1 DEFINITION. Let π be a partial predicate on $\underline{N}$.

π holds almost everywhere, written $\overset{\infty}{\forall} n \in \underline{N}, \pi(n)$, if

$$\exists k \in \underline{N}, \forall n \geq k, \pi(n).$$

π holds infinitely often, written $\overset{\infty}{\exists} n \in \underline{N}, \pi(n)$, if

$$\forall k \in \underline{N}, \exists n \geq k, \pi(n).$$

In a sense, almost everywhere and infinitely often are negations of one another. That is, it is false that π holds almost everywhere if and only if π does not hold infinitely often.

5.0.2 PROPOSITION. Let π be a partial predicate on $\underline{N}$.

$(\overset{\infty}{\forall} n \in \underline{N}, \pi(n))$ doesn't hold

$\Leftrightarrow \overset{\infty}{\exists} n \in \underline{N}$, ($\pi(n)$ doesn't hold).

Proof. Intuitively, there is not a $k \in \underline{N}$ for every $n \geq k$ $\pi(n)$ holds, if and only if for every $k \in \underline{N}$ there is some $n \geq k$ for which $\pi(n)$ doesn't hold. □

In this chapter we will encounter two fundamental reasons for the occurrence of the notion of almost everywhere in abstract complexity theory.

The first reason, which we will investigate in § 5.1, is a matter of choice. Complexity theorists have chosen to make a definition which involves the notion of almost everywhere.

The second reason, which will be investigated in § 5.2, is more pressing. We will see several results, both in §5.2 and in the remainder of the paper, which can be proven to hold only almost everywhere. Many of these results give such deep insights into computational complexities that they are considered to be fundamental results of complexity theory.

It is not surprising that both of these reasons can be tied back to the use of table lookups for doing efficient computations on a finite numbers of arguments.

In § 5.3 we attempt to discuss briefly two philosophical questions that can be raised in regard to the prevalence of the notion of almost everywhere in complexity theory. One of these asks about the relevance of complexity theory to real computing environments, the other about the relevance of complexity theory to the computer scientist.

§ 5.1 Complexity Classes

Recall that in §2.3 we discussed the fact that very complex computations on Turing machines can be done very efficiently for a finite number of inputs by using table lookups. That is, given a machine that uses many resources on every input, we can construct a new machine which works like the given machine on almost all inputs, but for the finite number of exceptions it uses relatively few resource units. This is done by having the new machine, on reading one of the finite number of inputs for which we desire an efficient computation, enter some state which causes the output to be printed immediately with no computation other than reading the input and printing the output.

The question arises as to whether we should consider the computation done by the new machine less complex than the computation done by the original machine. It seems to be universally accepted to give a negative answer to this question. This is reflected in the definition of a complexity class.

A partial function φ is in the ψ-complexity class, for another partial function ψ, if there is a way to compute φ using less than ψ resource units almost everywhere. Thus, the definition of complexity classes help the complexity theorist cope with table lookups.

5.1.1 DEFINITION. Let $M = ((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure and ψ be any partial function on $\underline{N}$. The *ψ-complexity class* in M is

$$C_\psi^M = \{\varphi_i \mid \mathrm{Dom}(\varphi_i) = \mathrm{Dom}(\psi) \text{ and } \overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \leq \psi(n)\}.$$

If the underlying measure, M, is understood then we will write C_ψ for C_ψ^M. In complexity theory we are usually interested in ψ-complexity classes for computable total ψ. (eg, [3], [4], [7]). However, some papers do deal with computable partial ψ. (eg, [6]).

Many of the results presented in this paper can be restated in a more concise form using complexity classes. We will discuss this further in § 9.2. However, this author feels that most of these results are intuitively more comprehensible when stated without the use of complexity classes. Therefore the major portion of the paper makes no mention of complexity classes. The reader should be aware, however, that the notion of complexity classes is fundamental to many presentations of complexity theory. In fact, the study of complexity classes is one of the major fields of study within complexity theory.

§ 5.2 Honesty and Related Topics

We now turn our attention to several results that can only be proven to hold almost everywhere. The first result indicates that from the number of resources units used by a computation, we can compute an upper bound to the value of the result of that computation.

5.2.1 THEOREM. Let $((\varphi_i)_{i\in\underline{N}},(\Phi_i)_{i\in\underline{N}})$ be a complexity measure.

There is a computable total h on $\underline{N}^2$ ∋

$$\forall i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, [\varphi_i(n) \lesssim h(n,\Phi_i(n))].$$

Proof. Define computable p on $\underline{N}^3$ by

$$p(i,n,m) =_{defn} \begin{cases} \varphi_i(n) & \Phi_i(n) \approx m \\ 0 & \text{otherwise.} \end{cases}$$

Define computable h on $\underline{N}^2$ by

$$h(n,m) =_{defn} \max\{p(j,n,m) \mid j \leq n\}.$$

Then $\forall i \in \underline{N}$, $\forall n \geq i$, if $\varphi_i(n)$ is defined then

$$\begin{aligned} (*) \qquad h(n,\Phi_i(n)) &= \max\{p(j,n,\Phi_i(n)) \mid j \leq n\} \\ &\geq p(i,n,\Phi_i(n)) \qquad \text{(since } i \leq n\text{)} \\ &= \varphi_i(n) \end{aligned}$$

and if $\varphi_i(n)$ is undefined then $\Phi_i(n)$ is undefined, hence $h(n,\Phi_i(n))$ is undefined. Furthermore, h is total.□

Henceforth we will not bother to separate arguments as (*) into the cases where $\varphi_i(n)$ is defined and $\varphi_i(n)$ is undefined. We will write something more like,

$$\begin{aligned} h(n,\Phi_i(n)) &\approx \max\{p(j,n,\Phi_i(n)) \mid j \leq n\} \\ &\gtrsim p(i,n,\Phi_i(n)) \\ &\approx \varphi_i(n). \end{aligned}$$

Arguments which define functions such as h in terms of functions such as p, as in the above proof, are quite common in complexity theory. It is significant in this type of argument that the almost every $n \in \underline{N}$ for which the result holds depends directly on i. Hence, values n for which the result is known to hold are usually quite large.

The following diagram may help the reader visualize this type of argument more clearly. Let m be fixed.

$$h(1,m) = \max \{p(1,1,m)\}$$

$$h(2,m) = \max \{p(1,2,m),\ p(2,2,m)\}$$

$$h(3,m) = \max \{p(1,3,m),\ p(2,3,m),\ p(3,3,m)\}$$

$$h(4,m) = \max \{p(1,4,m),\ p(2,4,m),\ p(3,4,m),\ p(4,4,m)\}$$

$$\begin{array}{cccccc} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$$

It is clear by looking under the i^{th} column of values for p in the diagram that for all but finitely many n (i.e. for all $n \geq i$) $h(n,m)$ is at least as large as $p(i,n,m)$, and that as i gets larger, the finite number of exceptions gets larger, but is independent of m. We will now look at an example which indicates that we cannot hope to prove a result similar to THEOREM 5.2.1 which holds everywhere.

In § 2.3 we observed that the time measure defined in § 2.1 always reflects the value of the result of a computation even if the computation uses table lookups. This is because the time taken to write the result is a function of the value written. However, our tape measures need not reflect the value of the result since we can

write a result using only one work tape cell by making use of table lookups. Recall, though that table lookups are only good for a finite number of inputs. It is for this reason the function h in THEOREM 5.2.1 can be proven to work almost everywhere but not everywhere.

5.2.2 EXAMPLE. The complexity measure $((\tau_i^1)_{i\in\underline{N}},(\lambda_i^*)_{i\in\underline{N}})$ is such that there is no h on $\underline{N}^2$ $\ni$ $\forall i \in \underline{N}$, $\forall n \in \underline{N}$,

$$\tau_i^1(n) \lesssim h(n,\lambda_i^*(n))$$

Proof. We will give a very rough argument which we leave to the reader to refine.

For each $k\in\underline{N}$ there is a Turing machine which on input 0, outputs k using only 1 work tape cell. It is impossible that there is a total h such that

$$\forall k \in \underline{N}, \quad k \leq h(0,1). \quad \square$$

We have seen that, in a sense, we can bound the values of computations by the complexity of those computations. In § 6.1 we will see that it is not possible to bound complexities of computation by their values in a similar manner. This is because there are arbitrarily complex bounded functions. However, those functions whose complexity can be bounded by their values are important to the complexity theorist. Such functions are called honest functions.

5.2.3 DEFINITION. Let $M = ((\varphi_i)_{i\in\underline{N}},(\Phi_i)_{i\in\underline{N}})$ be a complexity measure. Let Ψ be a set of computable partial functions on $\underline{N}$, and h be a computable total function on $\underline{N}^2$.

Ψ is <u>h-honest</u> in M if $\forall \varphi \in \Psi$, $\exists i \in \underline{N} \ni \varphi_i = \varphi$ and

$$\overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \lesssim h(n,\varphi_i(n)).$$

Ψ is <u>honest</u> in M if $\exists$ computable total g on $\underline{N}^2 \ni \Psi$ is g-honest in M. A computable partial function φ is h-honest or honest in M, if $\{\varphi\}$ is h-honest or honest in M, respectively.

We suppress mention of M if it is understood. Notice that a set of honest functions is not necessarily an honest set of functions.

Honest functions and functions with computable graphs are surprisingly the same. This becomes more clear when we notice that when we are dealing with only one function, we can use everywhere in place of almost everywhere in DEFINITION 5.2.3. The proof of this hinges on the fact that changing a computable function on a finite number of arguments does not affect its computability.

5.2.4 <u>LEMMA</u>. Let $((\varphi_i)_{i\in\underline{N}},(\Phi_i)_{i\in\underline{N}})$ be a complexity measure. Let φ be honest. $\exists$ computable h' on $\underline{N}^2$, $\exists i \in \underline{N} \ni \varphi_i = \varphi$ and

$$\forall n \in \underline{N},\ \Phi_i(n) \lesssim h'(n,\varphi_i(n)).$$

<u>Proof</u>. Choose $i \in \underline{N}$, $k \in \underline{N}$, and computable total h on $\underline{N}^2 \ni \varphi_i = \varphi$ and

$$\forall n \geq k,\ \Phi_i(n) \lesssim h(n,\varphi_i(n)).$$

Define a finite set I by

$$I =_{\text{defn}} \{n \in \underline{N} \mid n < k \text{ and } \varphi_i(n) \text{ is defined}\}$$

Define r on $\underline{N}$ by

$$r(n) =_{\text{defn}} \begin{cases} 0 & \text{if } n \notin I \\ \Phi_i(n) & \text{if } n \in I. \end{cases}$$

Evidently r is computable since r is equal to the computable function which is identically zero, except for a finite number of arguments. Further, r is total since if $n \notin I$ then $r(n) = 0$, and if $n \in I$ then $\varphi_i(n)$ is defined, consequently $r(n)$ is defined.

Define h' on $\underline{N}^2$ by

$$h'(n,m) =_{defn} \begin{cases} h(n,m) & \text{if } n \geq k \\ r(n) & \text{if } n < k. \end{cases}$$

Evidently h' is computable and total.

Also, if $n \geq k$ then

$$h'(n,\varphi_i(n)) \simeq h(n,\varphi_i(n)) \gtrsim \Phi_i(n)$$

and if $n < k$ then

$$h'(n,\varphi_i(n)) \simeq \begin{cases} r(n) & \text{if } \varphi_i(n) \text{ is defined} \\ \text{undefined} & \text{if } \varphi_i(n) \text{ is undefined.} \end{cases} \simeq \Phi_i(n). \ \square$$

We are now in a position to prove that honest functions and functions with computable graphs are the same.

5.2.5 PROPOSITION. Let $((\varphi_i)_{i \in \underline{N}}, (\Phi_i)_{i \in \underline{N}})$ be a complexity measure.

φ is honest $\Leftrightarrow$ φ has a computable graph.

Proof. ($\Rightarrow$). Choose h' on $\underline{N}^2$ and $i \in \underline{N}$ $\ni$

$$\varphi_i = \varphi$$

and $\forall n \in \underline{N}$,

$$\Phi_i(n) \lesssim h(n,\varphi_i(n)).$$

Look at the following procedure.

On input (n,m),

(1) Compute $h'(n,m)$.

(2) Compute whether or not $\Phi_i(n) \lesssim h'(n,m)$. If not, then answer FALSE and halt.

(3) Compute $\varphi_i(n)$.

(4) If $\varphi_i(n) = m$ then answer TRUE and halt, else answer FALSE and halt.

We can see that this procedure computes whether or not $\varphi(n) \simeq m$ as follows.

If $\varphi(n) = m$ then $\varphi_i(n) = m$, hence $\Phi_i(n) \lesssim h'(n,m)$ and (4) results in the answer TRUE.

If $\varphi(n) \neq m$, then either (2) results in the answer FALSE or else $\Phi_i(n) \leq h'(n,m)$. In the latter case we have $\varphi_i(n)$ is defined, hence (3) results in some value for $\varphi_i(n)$. Since $\varphi_i(n) = \varphi(n)$ and $\varphi(n) \neq m$, (4) will result in the answer FALSE.

($\Leftarrow$). Choose $i \ni \varphi_i = \varphi$. Define computable total h on $\underline{N}^2$ by

$$h(n,m) =_{\text{defn}} \begin{cases} \Phi_i(n) & \varphi(n) \simeq m \\ 0 & \text{otherwise.} \end{cases}$$

Now $\forall n \in \underline{N}$,

$$h(n,\varphi_i(n)) \simeq h(n,\varphi(n)) \simeq \Phi_i(n).$$

Thus, φ is h-honest. □

PROPOSITION 4.3.2 tells us that φ has a computable graph if and only if φ is a member of a measured sequence. Hence the following corollary is immediate.

5.2.6 COROLLARY. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure.

$\exists$ a measured sequence $(\psi_i)_{i\in\underline{N}} \ni \varphi \in \{\psi_i \mid i\in\underline{N}\}$

$\Leftrightarrow$ φ is honest.

Proof. Immediate from PROPOSITION 4.3.2 and PROPOSITION 5.2.5. □

Does this indicate that measured sequences and honest sets are essentially the same? Not quite. The following proposition indicates that measured sequences form honest sets. In § 7.2 we will see that the set of all h-honest functions, for any given h, forms a measured sequence. However, in § 6.2 we will construct an honest set which cannot possibly form a measured sequence. This honest set will even be seen to be computably enumerable!

5.2.7 PROPOSITION. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure, $(\psi_i)_{i\in\underline{N}}$ a measured sequence. Then $\{\psi_i \mid i\in\underline{N}\}$ is honest. In fact, for any computable total f on $\underline{N} \ni \forall i \in \underline{N}$, $\varphi_{f(i)} = \psi_i$, we have that $\exists$ computable total h on $\underline{N}^2 \ni \forall i \in \underline{N}$, $\overset{\infty}{\forall} n \in \underline{N}$, $\Phi_{f(i)}(n) \lesssim h(n, \varphi_{f(i)}(n))$.

Proof. Since every measured sequence is computably enumerable $\exists$ computable total f on $\underline{N} \ni \forall i \in \underline{N}$, $\varphi_{f(i)} = \psi_i$, (by PROPOSITION 4.2.3). Thus, it suffices to prove the second part of the proposition.

Let f be any computable total function on $\underline{N} \ni \forall i \in \underline{N}$, $\varphi_{f(i)} = \psi_i$. Since $(\Phi_i)_{i\in\underline{N}}$ is computably enumerable (FACT 4.4.4) we have

$(\Phi_{f(i)})_{i\in\underline{N}}$ is computably enumerable (LEMMA 4.1.5) thus it is evident that p on $\underline{N}^3$ is computable where p is defined by

$$p(i,n,m) =_{defn} \begin{cases} \Phi_{f(i)}(n) & \psi_i(n) \simeq m \\ 0 & \text{otherwise.} \end{cases}$$

p is total since if $\psi_i(n) = m$ we have $\varphi_{f(i)} = m$ thus $\Phi_{f(i)}(n)$ is defined. Define computable total h on $\underline{N}^2$ by

$$h(n,m) =_{defn} \max\{p(j,n,m) \mid j \leq n\}$$

Then $\forall i \in \underline{N}$, $\forall n \geq i$, we have

$$\begin{aligned} h(n,\varphi_{f(i)}(n)) &\simeq h(n,\psi_i(n)) \\ &\simeq \max\{p(j,n,\psi_i(n)) \mid j \leq n\} \\ &\gtrsim p(i,n,\psi_i(n)) \quad \text{(since } i \leq n\text{)} \\ &\simeq \Phi_{f(i)}(n). \ \square \end{aligned}$$

Immediately we have the following corollary.

5.2.8 COROLLARY. For any complexity measure $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ we have that $\{\Phi_i \mid i\in\underline{N}\}$ is honest.

Proof. Immediate from PROPOSITION 5.2.7 and the fact that $(\Phi_i)_{i\in\underline{N}}$ is a measured sequence. $\square$

We turn our attention away from honesty for a moment, to look at how different complexity measures relate to each other. The relationship we will derive is sometimes useful in demonstrating that certain properties are measure invariant. A measure invariant property is a property which is true in all measures if it can be shown to be true in any one measure. We will then look at honesty as a measure invariant property.

5.2.9 LEMMA. Let $((\varphi_i)_{i\in\underline{N}},(\Phi_i)_{i\in\underline{N}}),((\hat{\varphi}_i)_{i\in\underline{N}},(\hat{\Phi}_i)_{i\in\underline{N}})$ be any two complexity measures. There is a computable total f on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$,

$$\varphi_{f(i)} \approx \hat{\varphi}_i .$$

Furthermore, $\forall$ computable total f on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$, $\varphi_{f(i)} \approx \hat{\varphi}_i$ $\exists$ computable total h on $\underline{N}^2$ $\ni$ $\forall i \in \underline{N}$, $\overset{\infty}{\forall} n \in \underline{N}$,

$$\Phi_{f(i)}(n) \lesssim h(n,\hat{\Phi}_i(n))$$

and $\forall m_1,m_2 \in \underline{N}$, $\forall n \in \underline{N}$,

$$m_1 \leq m_2 \Leftrightarrow h(n,m_1) \leq h(n,m_2) .$$

Proof. $(\hat{\varphi}_i)_{i\in\underline{N}}$ is an acceptable numbering, hence is computably enumerable by LEMMA 4.2.4. Thus by PROPOSITION 4.2.3 $\exists$ computable total f on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$,

$$\varphi_{f(i)} \approx \hat{\Phi}_i .$$

Let f be any such function.

Define p on $\underline{N}^3$ by

$$p(i,n,m) =_{\text{defn}} \begin{cases} \Phi_{f(i)}(n) & \hat{\Phi}_i(n) \simeq m \\ 0 & \text{otherwise} . \end{cases}$$

Evidently, p is computable since $(\Phi_{f(i)})_{i\in\underline{N}}$ is computably enumerable (FACT 4.4.4) and since $(\hat{\Phi}_i)_{i\in\underline{N}}$ is a measured sequence (DEFINITION 4.4.1).

p is total since $\hat{\Phi}_i(n) = m \Rightarrow \hat{\varphi}_i(n)$ is defined

$$\Rightarrow \varphi_{f(i)}(n) \text{ is defined}$$

$$\Rightarrow \Phi_{f(i)}(n) \text{ is defined.}$$

Define $h'(n,m) =_{\text{defn}} \max\{p(j,n,m) \mid j \leq n\}$.

Define $h(n,m) =_{\text{defn}} \max\{h'(n,m') \mid m' \leq m\}$.

Evidently h' and h are computable and total. Clearly, $m_1 \geq m_2 \Leftrightarrow h(n,m_1) \geq h(n,m_2)$ by definition of h.

Also $\forall i \in \underline{N}$, $\forall n \geq i$,

$$\begin{aligned} h(n,\hat{\Phi}_i(n)) &= \max\{h'(n,m') \mid m' \leq \hat{\Phi}_i(n)\} \\ &\geq h'(n,\hat{\Phi}_i(n)) \\ &= \max\{p(j,n,\hat{\Phi}_i(n)) \mid j \leq n\} \\ &\geq p(i,n,\hat{\Phi}_i(n)) \\ &= \Phi_{f(i)}(n). \ \square \end{aligned}$$

Although the previous LEMMA is more useful, the following more symmetric result seems more impressive. The result is from Blum [2].

5.2.10 PROPOSITION. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$, $((\varphi_i)_{i\in\underline{N}}, (\hat{\Phi})_{i\in\underline{N}})$ be two complexity measures.

$\exists$ computable total h on $\underline{N} \ni \forall i \in \underline{N}$, $\overset{\infty}{\forall} n \in \underline{N}$,

$$\Phi_i(n) \leq h(n,\hat{\Phi}_i(n))$$

and $$\hat{\Phi}_i(n) \leq h(n,\Phi_i(n)).$$

Proof. Define $f(i) =_{\text{defn}} i$. Choose h_1,h_2 on $\underline{N}^2$, $k_1,k_2 \in \underline{N}$, $\ni \forall i \in \underline{N}$, we have,

$\forall n \geq k_1$, $\Phi_i(n) \leq h_1(n,\hat{\Phi}_i(n))$

$\forall n \geq k_2$, $\hat{\Phi}_i(n) \leq h_2(n,\Phi_i(n))$, by the above LEMMA.

Define computable total h on $\underline{N}^2$ by

$$h(n,m) =_{\text{defn}} \max\{h_1(n,m), h_2(n,m)\}.$$

Then $\forall i \in \underline{N}$, $\forall n \geq \max\{k_1,k_2\}$,

$$\begin{aligned} h(n,\hat{\Phi}_i(n)) &\geq h_1(n,\hat{\Phi}_i(n)) \\ &\geq \Phi_i(n) \end{aligned}$$

and $$\begin{aligned} h(n,\Phi_i(n)) &\geq h_2(n,\Phi_i(n)) \\ &\geq \hat{\Phi}_i(n). \ \square \end{aligned}$$

The following proposition uses LEMMA 5.2.9 to show that honesty is measure invariant.

5.2.11 PROPOSITION. Let $M = ((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ and $\hat{M} = ((\hat{\varphi}_i)_{i\in\underline{N}}, (\hat{\Phi}_i)_{i\in\underline{N}})$ be any two complexity measures. Let Ψ be a set of computable partial functions on $\underline{N}$. Ψ is honest in $\hat{M} \Leftrightarrow \Psi$ is honest in M.

Proof. By symmetry we need only prove $\Rightarrow$. Choose $g \ni \Psi$ is g-honest in $\hat{M}$.

Choose f and h by LEMMA 5.2.9 $\ni$

$\forall i \in \underline{N}, \varphi_{f(i)} = \hat{\varphi}_i$,

$\forall i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, \Phi_{f(i)}(n) \leq h(n,\hat{\Phi}_i(n))$,

and $\forall m_1,m_2 \in \underline{N}, \forall n \in \underline{N}, \; m_1 \geq m_2 \Rightarrow h(n,m_1) \geq h(n,m_2)$.

Define h' on $\underline{N}^2$ by

$$h'(n,m) =_{\text{defn}} h(n,g(n,m)).$$

Evidently h' is computable and total. We will show that Ψ is h'-honest in M. Let $\varphi \in \Psi$. Choose $i \in \underline{N} \ni$

$\hat{\varphi}_i = \varphi$ and $\overset{\infty}{\forall} n \in \underline{N}, \hat{\Phi}_i(n) \leq g(n,\hat{\varphi}_i(n))$.

Then we have

$\varphi_{f(i)} = \varphi$ (since $\varphi_{f(i)} = \hat{\varphi}_i$) and

$\forall i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}$,

$$\begin{aligned}(*)\quad h'(n,\varphi_{f(i)}(n)) &= h(n,g(n,\varphi_{f(i)}(n))) \\ &= h(n,g(n,\hat{\varphi}_i(n))) \\ &\geq h(n,\hat{\Phi}_i(n)) \quad (\text{since } \overset{\infty}{\forall} n \in \underline{N},\ g(n,\hat{\varphi}_i(n)) \geq \hat{\Phi}_i(n)) \\ &\geq \Phi_{f(i)}(n)\end{aligned}$$

Thus Ψ is h'-honest in M. □

Notice that we should choose $k \in \underline{N} \ni$ (*) holds for all $n \geq k$, to be the maximum of k_1 and k_2 where,

$$\forall n \geq k_1,\ \Phi_{f(i)}(n) \lesssim h(n, \hat{\Phi}_i(n))$$

$$\forall n \geq k_2,\ \hat{\Phi}_i(n) \lesssim g(n, \hat{\varphi}_i(n)).$$

In future proofs we will not bother to indicate when we are using such 'tricks' with 'almost everywhere.'

§ 5.3 Relevance of Abstract Complexity Theory

In this section we wish to take a brief philosophical look at abstract complexity theory. We will look at how complexity theory relates to real computing environments, and at how worthwhile the study of complexity theory might be to a computer scientist. This author maintains that if any doubt at all arises as to whether or not abstract complexity theory relates closely to real computing environment, or as to whether or not the study of complexity theory is worthwhile to the computer scientist, it arises because of the prevalence of the notion of almost everywhere in complexity theory. Hence the seeming misplacement of this section in a chapter about the notion of almost everywhere.

Let us look first at the question about computing environments. We will see that a computer system does, in a sense, fit the axioms of a complexity measure.

We will make a few assumptions of the machine language of a computer system. We assume that the instructions have an upper bound to their length, that for each computable function we can write a programme to compute it (i.e., we should be able to simulate Turing machines using the language), and that we can Gödel number the programmes which can be written in the language. These seem to be true of every machine language. The set of Gödel numbered programmes will form an acceptable numbering.

We will encounter a minor difficulty in trying to run these programmes in our systems since almost all programmes are too large for the system. However, since the system cannot execute more than a fixed number of instructions at a time (usually one) let us store the programme on tape and give the system only those instructions it requires at any one

time.

The reader can see we have already moved away from an ideal computing system in order to create an environment suitable to discussing a complexity measure.

Suppose we feed inputs into the system on tape. Then our system can work on arbitrarily large inputs. Now, since this is supposedly a real computing environment, it may have available to it limited quantities of some resources and all it requires of others. For example, it may have all the time it needs but only a fixed amount of memory. It is possible that some programmes require more units of some resource than are available to it.

Let us make another assumption. Suppose any time the system attempts to use more units of resources than are available to it, that we can give it more of any type of resource it requires.

We have moved another step away from any real system. However, we are not being too unrealistic in that we have the technology, if not the resources, to build such a system.

Suppose that we define a complexity measure on the programmes for this system by counting certain resources used by the programmes.

Now any result from complexity theory can be applied to programmes on this system. However, are these results at all useful within this computing environment?

Suppose we have a programme to which we would like to apply some result from complexity theory, and that this result holds only almost everywhere. In any real application, we will only be interested in the computations of the programme on some finite set of inputs, if for no other

reason than because the size of inputs is limited by how long we are willing to spend writing them on the tape. Since we will only use the programme for a finite number of inputs, of what use to us is a result which holds for all but a finite number of inputs?

We can see then, that although a computer system may satisfy, in a sense, the axioms for a complexity measure, many of the results of complexity theory (i.e., those which hold only almost everywhere) have no real significance.

Does this mean that complexity theory itself is not significant? This author feels that, in spite of the above, complexity theory is an important field for the computer scientist to study. Many results, although they may hold only almost everywhere, do give deep insights into the complexity of computations. The computer scientist can greatly increase his intuition in computability, and his understanding of computations and computational complexities by studying abstract complexity theory.

CHAPTER 6

DIAGONALIZATION AND THE HALTING PROBLEM

§ 6.0 Discussion

Suppose we wish to define a function so that it is not in a given sequence of functions. If the sequence, say $(f_i)_{i \in \underline{N}}$, contains only total functions on $\underline{N}$, then we can define total f on $\underline{N}$ by

$$f(n) = _{\text{defn}} f_n(n) + 1$$

It is clear that $f \notin \{f_i \mid i \in \underline{N}\}$ since for each $i \in \underline{N}$, $f(i) \neq f_i(i)$, hence $f \neq f_i$.

This type of process is called diagonalization since f is made not equal to the diagonal of the following table.

$$\begin{array}{cccccc}
f_0(0) & f_0(1) & f_0(2) & f_0(3) & \ldots \\
f_1(0) & f_1(1) & f_1(2) & f_1(3) & \ldots \\
f_2(0) & f_2(1) & f_2(2) & f_2(3) & \ldots \\
f_3(0) & f_3(1) & \ldots & \ldots & \ldots \\
\vdots & \vdots & \vdots & \vdots &
\end{array}$$

In cases where f must be computable and the functions in the sequence are partial, various complications arise. This is because $\varphi_n(n) \simeq \varphi_n(n) + 1$, if $\varphi_n(n)$ is undefined. Many times it is still possible to define computable f, usually at the expense of adding complications to the argument.

We will use the term 'diagonalization' to describe any process, no matter how complex, of defining some function such that it is not in a set of other functions.

In § 6.1 we look at some uses of diagonalization. One of these

is the proof of the Halting Problem. The Halting Problem indicates that there is no procedure which tells us, given i and n, whether or not the ith computing device halts on input n. In § 6.2 the Halting Problem is used to construct various examples of functions which are badly behaved in some respects.

§ 6.1 Some Applications - The Halting Problem

Let us look first at a simple application of diagonalization. The proof of the following proposition consists of defining a computable total function f so that it is not in a given measured sequence, $(\psi_i)_{i\in\underline{N}}$. This is done by defining $f(n)$ so that it is not equal to $\psi_n(n)$.

6.1.1 PROPOSITION. Let $(\psi_i)_{i\in\underline{N}}$ be a measured sequence. There is a computable total f on $\underline{N}$ ∋

$$f \notin \{\psi_i \mid i \in \underline{N}\} .$$

Furthermore $Im(f) \subsetneq \{0,1\}$.

Proof. Define f on $\underline{N}$ by

$$f(n) = \underset{\text{defn}}{} \begin{cases} 1 & \psi_n(n) \simeq 0 \\ 0 & \text{otherwise} . \end{cases}$$

Evidently f is computable and total. Also, $\forall i \in \underline{N}$,

$$f(i) \neq \psi_i(i) ,$$

$$\text{hence} \quad f \neq \psi_i . \quad \square$$

Thus, since acceptable numberings contain all the computable functions, they cannot be measured.

6.1.2 COROLLARY. No acceptable numbering is a measured sequence.

Proof. Immediate from PROPOSITION 6.1.1 and PROPOSITION 4.2.2.□

It follows, then, that we cannot compute, given i,n, whether or not $\varphi_i(n)$ is defined, for any acceptable numbering $(\varphi_i)_{i\in\underline{N}}$. This is clear, for if we could then $(\varphi_i)_{i\in\underline{N}}$ would certainly be measured.

The question, 'Given an acceptable numbering $(\varphi_i)_{i\in\underline{N}}$, is there

a way to compute, for each i,n, whether or not $\varphi_i(n)$ is defined?' has been called, for obvious reasons, the Halting Problem. The above argument indicates that the answer to the Halting Problem is negative.

6.1.3 THEOREM. Let $(\varphi_i)_{i\in\underline{N}}$ be an acceptable numbering. The total predicate, P, on $\underline{N}^2$ defined by

$$P =_{\text{defn}} \lambda i,n\ [\varphi_i(n) \text{ is defined}]$$

is not computable.

Proof. Suppose it is computable. We will show $(\varphi_i)_{i\in\underline{N}}$ is measured, thus contradicting COROLLARY 6.1.2.

Given i,n,m, it is clear that the following procedure computes whether or not $\varphi_i(n) = m$.

On input (i,n,m).

(1) Compute whether or not $\varphi_i(n)$ is defined.

(2) If not then answer FALSE and halt.

(3) Compute $\varphi_i(n)$.

(4) If $\varphi_i(n) = m$ then answer TRUE and halt,

else answer FALSE and halt.

Thus $(\varphi_i)_{i\in\underline{N}}$ is measured, contradicting COROLLARY 6.1.2. Therefore $\lambda i,n\ [\varphi_i(n)$ is defined$]$ is not computable.□

The reader may wonder why we define computability in terms of partial functions since in any real situation we are interested only in total computable functions and since in dealing with partial functions we get a negative answer to the Halting Problem. We can easily answer this

query using a simple diagonal argument.

Suppose we have some workable characterization of all the intuitively computable total functions. We expect that these functions could be Gödel numbered so that given i we can construct a device that computes the ith function. Thus we could construct a device which on input n, constructs the nth device, runs on n, then outputs one plus the result of the nth device on input n. Certainly the function defined by this new device would be intuitively computable. However, by its definition it would not be in the original set of computable functions. (It should be clear that the above argument can be used to prove that the set of all computable total functions cannot form a computably enumerable sequence.)

Thus it seems that any characterization of computability which allows only total functions is insufficient. If our characterization allows partial functions, then the argument which shows total functions are insufficient, can be modified to show that the answer to the Halting Problem is negative. This can be seen more clearly in the proof that the following special form of the Halting Problem has a negative answer. 'Given an acceptable numbering $(\varphi_i)_{i\in\underline{N}}$, is there a way to compute for each n, whether or not $\varphi_n(n)$ is defined?' The following lemma gives a negative answer to this Special Halting Problem.

6.1.4 LEMMA. Let $(\varphi_i)_{i\in\underline{N}}$ be an acceptable numbering. Then the total predicate P on $\underline{N}$ defined by

$$P =_{\text{defn}} [\varphi_n(n) \text{ is defined}]$$

is not computable.

Proof. Suppose it is computable. We obtain a contradiction by defining a computable function φ which is not in $(\varphi_i)_{i\in\underline{N}}$. Define φ on $\underline{N}$ by

$$\varphi(n) =_{\text{defn}} \begin{cases} \varphi_n(n) + 1 & \text{if } \varphi_n(n) \text{ is defined} \\ 0 & \text{otherwise.} \end{cases}$$

Evidently φ is computable. However, $\forall i \in \underline{N}$, $\varphi \neq \varphi_i$ since

if $\varphi_i(i)$ is defined then $\varphi(i) = \varphi_i(i) + 1$

and if $\varphi_i(i)$ is undefined then $\varphi(i) = 0$.

This contradicts PROPOSITION 4.2.2. Thus $\lambda n\,[\varphi_n(n)$ is defined$]$ is not computable.□

The above LEMMA proves more useful in constructing counterexamples than THEOREM 6.1.3. In fact, most discussions of the Halting Problem use LEMMA 6.1.4 to prove PROPOSITION 6.1.3. We leave it to the reader to see how this might be done. We will look at some uses of LEMMA 6.1.4 in § 6.2 of this chapter.

The next result uses a slightly more complicated diagonal argument. The result indicates that there are arbitrarily complex functions in the sense that for any computable ψ there is a computable function φ with the same domain as ψ such that every device which computes φ almost always uses more than ψ resource units. φ is constructed by diagonalizing over the set of functions that can be computed by using less than ψ resource units infinitely often. Thus if φ_i is computed using less than ψ resource units infinitely often then $\varphi \neq \varphi_i$. Taking the contrapositive we have that if $\varphi_i = \varphi$ then φ_i is computed using more than ψ resources almost everywhere.

6.1.5 PROPOSITION. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure.

Let ψ be any computable partial function on $\underline{N}$. There is a computable partial function φ on $\underline{N}$ $\ni$ $\mathrm{Dom}(\varphi) = \mathrm{Dom}(\psi)$ and $\forall i \in \underline{N}$,

$$\varphi_i = \varphi \Rightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \gtrsim \psi(n) .$$

Furthermore, φ can be defined $\ni$ $\forall n \in \underline{N}$, if $\varphi(n)$ is defined then $\varphi(n) \leq n + 2$.

Proof. Let φ be defined by the following procedure.

On input n.

(1) Compute $\psi(n)$. If $\psi(n)$ is undefined then $\varphi(n)$ is undefined.

(2) (In case $\psi(n)$ is defined.)

Define $X_n = \{\varphi_i(n) \mid i \leq n \text{ and } \Phi_i(n) < \psi(n)\}$

(Evidently X_n is finite, having at most the $n + 1$ members $\varphi_0(n), \varphi_1(n), \ldots, \varphi_n(n)$, and can be computed from n.)

(3) Define $\varphi(n) = \mu k\ [k \notin X_n]$. Halt.

(Evidently such k can be computed and $k \leq n + 2$, since X_n has at most $n + 1$ elements. Hence $\varphi(n) \leq n + 2$.)

Suppose we have $i \in \underline{N}$ $\ni$ $\overset{\infty}{\exists} n \in \underline{N}$, $\Phi_i(n) < \psi(n)$. Choose any $n \geq i$ $\ni$ $\Phi_i(n) < \psi(n)$. Then $\psi(n)$ is defined and $\varphi_i(n) \in X_n$. Thus $\varphi(n) \neq \varphi_i(n)$ hence $\varphi \neq \varphi_i$. We have, then, that $\forall i \in \underline{N}$,

$$(\overset{\infty}{\exists} n \in \underline{N},\ \Phi_i(n) < \psi(n)) \Rightarrow \varphi_i \neq \varphi .$$

Clearly $\mathrm{Dom}(\varphi) = \mathrm{Dom}(\psi)$ by (1), hence

$$\varphi_i = \varphi \Rightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \gtrsim \psi(n). \ \square$$

Since $\varphi(n) \leq n + 2$ we have as a corollary that the set of all computable functions is not honest.

6.1.6 COROLLARY. $\forall$ computable total h on $\underline{N}^2$, $\exists$ computable total f on $\underline{N}$ $\ni$ f is not h-honest.

Proof. Let h be given. Define r on $\underline{N}$ by

$$r(n) = \underset{\text{defn}}{\max\{h(n,m) \mid m \leq n + 2\}} + 1.$$

Evidently r is computable and total. Choose computable total f on $\underline{N} \ni f(n) \leq n + 2$ and $\forall i \in \underline{N}$,

$$\varphi_i = f \Rightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \geq r(n)$$

as in the above PROPOSITION. Thus, $\forall i \in \underline{N}$, if $\varphi_i = f$ then $\overset{\infty}{\forall} n \in \underline{N}$,

$$(*) \qquad \begin{aligned} \Phi_i(n) &\geq r(n) \\ &> \max\{h(n,m) \mid m \leq n + 2\} \\ &\geq h(n,f(n)) \\ &= h(n,\varphi_i(n)) \end{aligned}$$

Thus, f is not h-honest. □

In fact, (*) of the above proof is stronger than what is needed since in order to prove f is not h-honest we only need prove that $\overset{\infty}{\exists} n \in \underline{N}$,

$$\Phi_i(n) > h(n,\varphi_i(n)).$$

In [2], Blum has shown that the function φ of PROPOSITION 6.1.5 can be defined to take on only values 0 or 1. The following is the essence of the argument.

Given an input n, if $\psi(n)$ is undefined then let $\varphi(n)$ be undefined. If $\psi(n)$ is defined construct a list, called the list of indices of cancelled functions, as follows. Place in the list each $i \in \underline{N}$ such that for some $k < n$ $\varphi(k)$ has been previously defined to be not equal to $\varphi_i(k)$. Now find the least i in $\{i \leq n \mid \Phi_i(n) \lesssim \psi(n)\} \ni i$ is not in the list of indices of cancelled functions. Define $\varphi(n)$ to be 0 or 1 so that $\varphi(n)$ is not equal to $\varphi_i(n)$. (Notice that for any larger n, i will now be in the list of indices of cancelled functions.)

The proof follows by showing two things. Firstly, that if i is such that $\overset{\infty}{\exists} n \in \underline{N}$, $\Phi_i(n) < \psi(n)$, then for some n, $\varphi(n) \neq \varphi_i(n)$. Secondly, that φ is computable.

The first of these follows from the construction of φ. The second follows if we can show that it is possible to construct the list of indices of cancelled functions. A problem arises here if there is some $k \in \underline{N} \ni \psi(k)$ is undefined. For if this is true then $\varphi(k)$ is undefined, and it may be impossible, for $n > k$, to construct the list in a straightforward fashion. Blum's construction of φ makes use of "dovetailing" to overcome this difficulty as we will see in § 7.2. For the moment let us prove the result for total ψ.

6.1.7 PROPOSITION. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure. Let r be a computable total function on $\underline{N}$. There is a computable total f on $\underline{N} \ni \mathrm{Im}(f) \subseteq \{0,1\}$ and $\forall i \in \underline{N}$,

$$\varphi_i = f \Rightarrow \overset{\infty}{\forall} n \in \underline{N}, \ \Phi_i(n) \geq r(n) .$$

Proof. Let f be defined by the following procedure. On input n,

(1) Do the first n stages in the construction of the list L, where L is constructed in stages $k = 0,1,2,\ldots$ as follows.

Stage k. Find the least $i' \leq k \ni$

(a) $\Phi_{i'}(k) < r(k)$, [$r(k)$ can be computed, thus we can compute whether $\Phi_{i'}(k) < r(k)$ holds.]

and (b) i' is not yet in L. [At each stage the list L will be only finite since we add at most one element to L at each stage.]

If such i' exists place it in L, then go to the next stage, else go directly to the next stage adding nothing to L.

(2) If i is added to L at stage n then define $f(n)$ by

$$f(n) =_{\text{defn}} \begin{cases} 1 & \varphi_i(n) = 0 \\ 0 & \text{otherwise} . \end{cases}$$

(Since $\Phi_i(n) < r(n)$, $\varphi_i(n)$ is defined and can be computed.)

If no index is added to L at stage n let $f(n)$ be defined by

$$f(n) =_{\text{defn}} 0 .$$

Notice that if i' is placed in L at stage k then $\varphi_{i'}(k) \neq f(k)$, by definition of f. Evidently f can be computed. We now show that $\forall i \in \underline{N}$, if $\overset{\infty}{\exists} n \in \underline{N}$, $\Phi_i(n) < r(n)$, then $\varphi_i \neq f$. Suppose we have $i \in \underline{N} \ni \overset{\infty}{\exists} n \in \underline{N}$, $\Phi_i(n) < r(n)$. Choose $n' \geq i \ni \Phi_i(n') < r(n')$ and n' is greater than every element in the following finite set,

$$\{n_{i'} \mid i' < i,\ i' \text{ is added to } L \text{ at stage } n_{i'}\} .$$

(This set is finite since an index is added to L in at most one stage of the construction of L.) Such n' exists since, $\overset{\infty}{\exists} n \in \underline{N}$, $\Phi_i(n) < r(n)$. Now let us look at the computation of $f(n')$.

CASE 1. If i is placed in L before stage n', then $\exists k < n'$ such that i is added to L at stage k. Hence $f(k) \neq \varphi_i(k)$ and $f \neq \varphi_i$.

CASE 2. If i is not placed in L before stage n', then look at stage n' of the construction of L. We are to find the least $i' \leq n'$ $\ni \Phi_{i'}(n') < r(n')$, and i' is not yet in L. Certainly i is a candidate for such i', but is it the least candidate? Any

other $i' < i \ni \Phi_{i'}(n') < r(n')$, is added to L at stage $n_{i'}$, and $n_{i'} < n'$ by choice of n'. Hence any other candidate is already in the list. Thus i is the least i' satisfying the conditions. Therefore i is added to L at stage n', and $f(n') \neq \varphi_i(n')$. Hence $f \neq \varphi_i$.

Thus we have $\forall i \in \underline{N}$,

$$(\overset{\infty}{\exists} n \in \underline{N},\ \Phi_i(n) < r(n)) \Rightarrow \varphi_i \neq f.$$

Taking the contrapositive we have $\forall i \in \underline{N}$,

$$\varphi_i = f \Rightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \geq r(n).$$

By definition of f it is clear that $\mathrm{Im}(f) \subseteq \{0,1\}$ and that f is total.□

§ 6.2 Using the Halting Problem

In the previous section we saw that there is a negative answer to both the Halting Problem and the Special Halting Problem, that is, there is no way to compute, for each i,n, whether or not $\varphi_i(n)$ is defined, and there is no way to compute, for each n, whether or not $\varphi_n(n)$ is defined.

The negative answers to the Halting Problem and the Special Halting Problem have proven extremely useful in constructing various counter-examples. The method in which these are used usually follows these lines. Suppose we would like to find some type of object that does not have a certain property. We attempt to define such an object so that if it did have this property, then we could compute whether or not $\varphi_n(n)$ is defined (or $\varphi_i(n)$ is defined) for each n (or each i,n), thus contradicting the known negative result to the Special Halting Problem (or Halting Problem). Hence the object does not have that property.

In this section we will look at three different counter-examples and use the negative answer to the Special Halting Problem (i.e., LEMMA 6.1.4) to prove that they are, in fact, counter-examples.

We first show that there are computable partial functions whose graphs are not computable. Notice that any such function cannot be total since every computable total function has a computable graph. Our approach is to define a computable function $\varphi \ni \forall n \in \underline{N}$,

$$\varphi(n) = 0 \Leftrightarrow \varphi_n(n) \text{ is defined.}$$

Then, if this function has a computable graph we can compute whether or not $\varphi_n(n)$ is defined by computing whether or not $\varphi(n) = 0$. Thus φ cannot have a computable graph.

Since functions with computable graphs and honest functions are the same, by PROPOSITION 5.2.5, we have that there are computable functions which are not honest. This is stronger than COROLLARY 6.1.6 which indicates that for each h there is a function which is not h-honest. However, COROLLARY 6.1.6 is still of interest in that the function which is constructed is total, whereas φ as above is partial.

6.2.1. EXAMPLE. There is a computable function φ whose graph is not computable. Hence, there is a computable function φ which is not honest.

Proof. The second result follows directly from the first and PROPOSITION 5.2.5.

In order to prove the first result, define φ on $\underline{N}$ by

$$\varphi(n) \simeq_{\text{defn}} \varphi_n(n) - \varphi_n(n).$$

Evidently, φ is computable, since $(\varphi_i)_{i\in\underline{N}}$ is computably enumerable, and

$$\varphi(n) \simeq \begin{cases} 0 & \text{if } \varphi_n(n) \text{ is defined} \\ \text{undefined} & \text{if } \varphi_n(n) \text{ is undefined.} \end{cases}$$

If φ has a computable graph then the predicate $\lambda n,m[\varphi(n) \simeq m]$ is computable, hence $\lambda n[\varphi(n) \simeq 0]$ is computable. Clearly $\lambda n[\varphi(n) \simeq 0]$ is the same as the predicate $\lambda n[\varphi_n(n)$ is defined$]$. Hence $\lambda n[\varphi_n(n)$ is defined$]$ is computable contradicting LEMMA 6.1.4. Thus, φ cannot have a computable graph. □

An argument much more sophisticated than the above, but using the same basic notion, aids us in constructing an honest set of functions, Ψ, which cannot be formed into a measured sequence. Ψ is defined to be $\{\psi'_i \mid i \in \underline{N}\}$ where $(\psi'_i)_{i\in\underline{N}}$ is defined so that $\varphi_n(n)$ is defined $\Leftrightarrow$

$\psi'_n(n) = 0$. If $(\psi_i)_{i\in\underline{N}}$ is a measured sequence such that $\Psi = \{\psi_i \mid i \in N\}$ then we can compute whether or not $\varphi_n(n)$ is defined by searching for $i \ni \psi_i = \psi'_n$ then checking whether or not $\psi_i(n) = 0$.

6.2.2 EXAMPLE. There is an honest set of functions, Ψ, $\ni$ there is no measured sequence $(\psi_i)_{i\ni\underline{N}}$ with $\Psi = \{\psi_i \mid i \in \underline{N}\}$.

Proof. For any given complexity measure $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$, define ψ'_i on $\underline{N}$ by

$$\psi'_i(n) \simeq_{\text{defn}} \begin{cases} \varphi_n(n) - \varphi_n(n) & \text{if } n = i \\ 1 & \text{otherwise} \end{cases}, \text{ for every } i \in \underline{N}.$$

Define ψ on $\underline{N}^2$ by

$$\psi(i,n) =_{\text{defn}} \begin{cases} \varphi_n(n) - \varphi_n(n) & \text{if } n = i \\ 1 & \text{otherwise.} \end{cases}$$

Evidently, ψ is computable and universal for $(\psi'_i)_{i\in\underline{N}}$, hence by DEFINITION 4.1.2, $(\psi'_i)_{i\in\underline{N}}$ is computably enumerable. Choose, by THEOREM 4.2.3, a computable total function f on $\underline{N}$ $\ni$

$$\varphi_{f(i)} = \psi'_i .$$

Define h on $\underline{N}^2$ by

$$h(n,m) =_{\text{defn}} \begin{cases} \max\{\Phi_{f(i)}(n) \mid i < n\} & \text{if } m = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Evidently h is computable since $(\Phi_{f(i)})_{i\in\underline{N}}$ is computably enumerable, by LEMMA 4.1.5. Also h is total, by the following.

If $m \neq 1$ then $h(n,m) = 0$. For $m = 1$,

notice that for each $i < n$, $\Phi_{f(i)}(n)$ is defined since

$$\varphi_{f(i)}(n) \simeq \psi'_i(n)$$
$$= 1 ,$$

hence, $h(n,m) = \max\{\Phi_{f(i)}(n) \mid i < n\}$ is defined.

Now, $\forall i \in \underline{N}$, $\varphi_{f(i)} = \psi'_i$ and $\forall n > i$,

$$\begin{aligned} h(n,\varphi_{f(i)}(n)) &= h(n,\psi'_i(n)) \\ &= h(n,1) \\ &= \max\{\Phi_{f(j)}(n) \mid j < n\} \\ &\geq \Phi_{f(i)}(n). \end{aligned}$$

Hence $\Psi =_{\text{defn}} \{\psi'_i \mid i \in \underline{N}\}$ is h-honest. Now suppose $(\psi_i)_{i\in\underline{N}}$ is a measured sequence with $\Psi = \{\psi_i \mid i \in \underline{N}\}$. The following procedure computes, given n, whether or not $\varphi_n(n)$ is defined.

On input n.

(1) For each $i = 0,1,2,\ldots$ compute whether or not $\psi_i(n) \simeq 1$ until i is found $\ni \psi_i(n) \not\simeq 1$. (This will be an $i \ni \psi_i = \psi'_n$.)

(2) Compute whether or not $\psi_i(n) \simeq 0$. If $\psi_i(n) \simeq 0$ then answer TRUE and halt, else answer FALSE and halt.

It becomes clear that the above does answer whether or not $\varphi_n(n)$ is defined, when we notice that $\varphi_n(n)$ is defined $\Leftrightarrow \psi'_n(n) \simeq 0$, and that $\psi_i(n) \not\simeq 1 \Rightarrow \psi_i = \psi'_n$. This contradicts LEMMA 6.1.4. Thus no such $(\psi_i)_{i\in\underline{N}}$ exists.□

In the discussion just prior to LEMMA 6.1.4 it was indicated that a simple diagonal argument can be used to prove the set of all computable total functions cannot form a computably enumerable sequence. The next counter-example demonstrates, using LEMMA 6.1.4, the existence of another set of computable total functions which is not computably enumerable.

6.2.3 EXAMPLE. There is a set, Ψ, of computable total functions on $\underline{N}$ $\ni$ there is no computably enumerable sequence (ψ_i)

$\Psi = \{\psi_i \mid i \in \underline{N}\}$.

Proof. For each $i \in \underline{N}$ define ψ'_i on $\underline{N}$ by

$$\psi'_i(n) =_{\text{defn}} \begin{cases} 0 & \text{if } n \neq i \\ 1 & \text{if } n = i \text{ and } \varphi_i(i) \text{ is defined} \\ 2 & \text{if } n = i \text{ and } \varphi_i(i) \text{ is undefined.} \end{cases}$$

Notice that any function which is 0 everywhere but at one argument is computable. Hence each ψ'_i is computable. Let $\Psi = \{\psi'_i \mid i \in \underline{N}\}$. Suppose there is a computably enumerable sequence $(\psi_i)_{i \in \underline{N}}$ such that $\Psi = \{\psi_i \mid i \in \underline{N}\}$. The following procedure computes, given n, whether or not $\varphi_n(n)$ is defined.

On input n.

(1) For each $i = 0,1,2,\ldots$ compute $\psi_i(n)$ until we find $i \ni \psi_i(n) \neq 0$. (This is an $i \ni \psi_i = \psi'_n$.)

(2) If $\psi_i(n) = 1$ then answer TRUE and halt, else answer FALSE and halt.

Evidently $\lambda n[\varphi_n(n)$ is defined$]$ is computable. This contradicts LEMMA 6.1.4. Hence no such $(\psi_i)_{i \in \underline{N}}$ exists. □

CHAPTER 7

COMPUTABLE BIJECTIONS AND DOVETAILING

§ 7.0 Discussion

The negative answer to the Halting Problem presents a great barrier in proofs that would otherwise seem quite natural. However, many proofs that, at first glance, seem to require that we have a positive answer to the Halting Problem, do not need such a strong result after all. They may only require that we find those n for which a computable function φ is defined. Dovetailing is any orderly process for generating such values n.

The reader may not see the distinction between computing, given n, whether or not $\varphi(n)$ is defined, and generating those n for which $\varphi(n)$ is defined. Let us look a little more closely at this. We will attempt to show that being able to generate those n for which $\varphi(n)$ is defined does not imply that we can compute, given n, whether or not $\varphi(n)$ is defined. Suppose we can generate those n for which $\varphi(n)$ is defined. Given n, we can check if $\varphi(n)$ is defined by seeing if n is generated, then answering TRUE if it is. However, if n is not generated we will wait forever. That is, we may never be able to say definitely that $\varphi(n)$ is not defined.

In § 7.1 we present a discussion of computable bijections, which aid us in defining and making use of some dovetailing procedures. In § 7.2 we discuss three dovetailing procedures. We also look at a few proofs using what this author feels is the best of the three dovetailing procedures presented.

§ 7.1 Computable Bijections

A total function on $\underline{N}^k$ is surjective if for every $n \in \underline{N}$ there is an element of $\underline{N}^k$ which the function maps to n. It is injective if no two elements of $\underline{N}^k$ map to the same element of $\underline{N}$. A total function is bijective if it is both surjective and injective.

7.1.1 DEFINITION. Let f be a total function on $\underline{N}^k$. f is bijective if

(1) $\forall n \in \underline{N}, \exists \underline{n} \in \underline{N}^k, \quad f\underline{n} = n$

and (2) $\forall \underline{n}_1, \underline{n}_2 \in \underline{N}^k, \quad \underline{n}_1 \neq \underline{n}_2 \Rightarrow f\underline{n}_1 \neq f\underline{n}_2$.

We will show how to construct a computable bijection on $\underline{N}^k$ for each k. If k = 1 then the identity function suffices. The following lemma shows the result for k = 2.

7.1.2 LEMMA. There is a computable bijection on $\underline{N}^2$.

Proof. Let f be the function defined by the following procedure.

On input (n_1, n_2),

(1) Set m to 0.

(2) Set k to 0.

(3) Set i to k.

(4) Set j to 0.

(5) If $n_1 = i$ and $n_2 = j$, output m and halt.

(6) Set m to m + 1.

(7) Set i to i - 1. (At times i may be set to -1.)

(8) Set j to j + 1.

(9) If $j \leq k$ then go to (5).

(10) Set k to k + 1.

(11) Go to (3).

Essentially the above procedure enumerates the elements of $\underline{N}^2$ in the order indicated by the following diagram,

$$\begin{array}{llll} (0,0)^0 & (0,1)^2 & (0,2)^5 & (0,3)^9 \quad \ldots \\ (1,0)^1 & (1,1)^4 & (1,2)^8 & (1,3)^{13} \quad \ldots \\ (2,0)^3 & (2,1)^7 & (2,2)^{12} & (2,3)^{18} \quad \ldots \\ (3,0)^6 & (3,1)^{11} & (3,2)^{17} & (3,3) \quad \ldots \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

until (n_1,n_2) is encountered. It then outputs one less than the number of ordered pairs enumerated so far. Evidently f is a computable bijection on $\underline{N}^2$.□

From this bijection on $\underline{N}^2$ we can define a bijection on $\underline{N}^k$ for each $k > 2$ by a straightforward induction argument.

7.1.3 PROPOSITION. For each $k \in \underline{N}$ there is a computable bijection on $\underline{N}^k$.

Proof. For $k = 1$ the identity function suffices. For $k = 2$, f defined in the above LEMMA suffices. For $k > 2$ define f_k on $\underline{N}^k$ by

$$f_k(n_1,\ldots,n_k) =_{\text{defn}} f(f_{k-1}(n_1,\ldots,n_{k-1}),n_k)$$

where $f_2 =_{\text{defn}} f$, with f as in the above LEMMA. Now $f_2 = f$ is computable. Assuming that f_{k-1} is computable we have f_k is computable since it is a composition of two computable functions. Hence by induction f_k is computable for all k.

Similar induction arguments show that f_k is a bijection. (Hint: Treat injectivity and surjectivity separately.) □

7.1.4 DEFINITION. For each k, let

$$\lambda n_1,n_2,\ldots,n_k\,[\langle n_1,n_2,\ldots,n_k\rangle]$$

be some fixed computable bijection on $\underline{N}^k$.

The notation introduced in the above definition will be used in several proofs in the remainder of this paper. Given $n_1,n_2,\ldots,n_k$ the k-tuple $(n_1,n_2,\ldots,n_k)$ is mapped, by the above defined bijection, to the natural number $\langle n_1,n_2,\ldots,n_k\rangle$. Since the particular bijection to be used is left arbitrary in the definition, any computable bijection can be used in these proofs. The definition is really given to fix a notation, not to fix a bijection.

There are two important properties of computable bijection which will be used repeatedly in proofs. The first is that given n, there are only a finite number of k-tuples, $(n_1,n_2,\ldots,n_k)$ such that $\langle n_1,n_2,\ldots n_k\rangle \leq n$. This is because there are only a finite number of natural numbers m with $m \leq n$.

The second property is that given $n \in \underline{N}$, we can compute $(n_1,n_2,\ldots,n_k)$ such that $\langle n_1,n_2,\ldots n_k\rangle = n$. This is because we can enumerate, in some computable fashion, all of the k-tuples (similar to enumerating all ordered pairs as in LEMMA 7.1.2). As each k-tuple $(n_1,\ldots,n_k)$ is enumerated we can compute $\langle n_1,n_2,\ldots,n_k\rangle$ and check if it equals n. If it does, output $(n_1,n_2,\ldots,n_k)$, else enumerate the next k-tuple.

The above argument will be used implicitly in several proofs. If n is given we may write a statement such as, "compute $n_1,n_2,\ldots,n_k$ $\ni$ $\langle n_1,n_2,\ldots,n_k\rangle = n$," and leave it to the reader to see that $n_1,n_2,\ldots,n_k$ can in fact be computed.

In § 1.2 we mentioned that there is a procedure which generates, without repetition, all of the lists of natural numbers. We can now see this as follows.

Firstly, notice that given k, and $n_1,n_2,\ldots n_k \in \underline{N}$, it is possible to compute $f_k(n_1,n_2,\ldots,n_k)$ where f_k is defined as in the proof of PROPOSITION 7.1.3. All we do is compute

$$f(f(\ldots f(f(n_1,n_2),n_3)\ldots,n_{k-1}),n_k) ,$$

where f is defined as in the proof of LEMMA 7.1.2. Hence, given n and k, it will be possible to compute $(n_1,n_2,\ldots,n_k)$ such that $n = f_k(n_1,n_2,\ldots,n_k)$.

We can now see that the following procedure generates all of the lists of natural numbers as required.

In order to output the mth list, for $m \in \underline{N}$,

(1) Compute (n,k) such that $m = f(n,k)$.

(2) Compute $(n_1,n_2,\ldots,n_k)$ such that $n = f_k(n_1,n_2,\ldots,n_k)$.

(3) Output the list $n_1,n_2,\ldots,n_k$ and halt.

This procedure gives all the lists, for if $n_1,n_2,\ldots,n_k$ is a list of natural numbers, then it will be the $f(f_k(n_1,n_2,\ldots,n_k),k)$-th list outputted by the procedure. Furthermore, there are no repetitions since f and f_k are injective.

We now look at some methods of dovetailing and at some proofs that make use of dovetailing.

§ 7.2 Dovetailing

Let us look at a simple example of dovetailing, using a given complexity measure $((\varphi_i)_{i\in N},(\Phi_i)_{i\in N})$. We would like to define a procedure which, on input i, enumerates those values n for which $\varphi_i(n)$ is defined. The following procedure does this.

On input i, enumerate values n in stages $m = 0,1,2,\ldots$, as follows. At stage m,

(1) Output each value $n \leq m \ni \Phi_i(n) \leq m$.

(2) Go to the next stage.

(Recall that $\lambda i,n,m[\Phi_i(n) \leq m]$ is computable.)

Certainly this procedure works, for if $\varphi_i(n)$ is defined then n is enumerated at stage $m = \max\{n,\Phi_i(n)\}$. Also, if n is enumerated then it must be enumerated at some stage m where $\Phi_i(n) \leq m$. Hence $\Phi_i(n)$ is defined, thus $\varphi_i(n)$ is defined. Notice though that values n will be enumerated infinitely often.

The following, slightly more sophisticated dovetailing procedure enumerates a value n at most once and enumerates at most one value at each stage.

On input i, enumerate values n and build a list L, in stages $m = 0,1,2,\ldots$, as follows. At the start, L is empty.

At stage m,

(1) Find the least $n \leq m \ni \Phi_i(n) \leq m$ and n has not yet been placed in L.

(2) If such n exists then (a) Place n in L.

(b) Output n.

(3) Go to the next stage.

We leave it to the reader to see that the above procedure works, mentioning only that L is used in order to keep track of the values enumerated so far.

One remaining drawback of the above procedure is that there is no strong tie between n and m, where n is enumerated at stage m, other than m is greater than or equal to both n and $\Phi_i(n)$. This becomes more apparent when we look at the following procedure, for which each value enumerated is more closely tied to the stage at which it is enumerated.

On input i, enumerate values n in stages $x = 0,1,2,\ldots$, as follows. (Recall that $\lambda n,m[\langle n,m\rangle]$ is a computable bijection.)

At stage x,

(1) Compute $n,m \ni \langle n,m\rangle = x$.

(2) If $\Phi_i(n) = m$ output n.

(3) Go to the next stage.

Notice that n is enumerated at stage x if and only if $x = \langle n,\Phi_i(n)\rangle$.

The procedure works since if $\varphi_i(n)$ is defined then n is enumerated at stage $x = \langle n,\Phi_i(n)\rangle$. Also, if n is enumerated then it is enumerated at stage $x = \langle n,m\rangle$ where $\Phi_i(n) = m$. Since $\Phi_i(n)$ is defined we have that $\varphi_i(n)$ is defined.

We will make use of only the latter method of dovetailing, since it has all of the strengths of the other two methods plus a strength of its own. Both PROPOSITION 7.2.1 and PROPOSITION 7.2.2 can be done using either of the first two methods. PROPOSITION 7.2.4, however, requires the use of the latter method.

Let us first look at a fairly simple proof which uses dovetailing.

Let $\pi(i,n)$ be a partial predicate which is defined precisely on those values i,n for which $\varphi_i(n)$ is defined. Think of $\pi(i,n)$ as saying something about the computation, $\varphi_i(n)$. Suppose we would like to enumerate indices for those functions $\varphi_i \ni \overset{\infty}{\forall} n \in \underline{N}$, $\pi(i,n)$ does not hold. Notice that if φ_i is undefined almost everywhere then we should enumerate an index for φ_i, thus we can proceed as follows. For each i,k define a function $\psi_{i,k}$ to be φ_i if, by using dovetailing, we never find an $n \geq k \ni \pi(i,n)$ is TRUE, and to be undefined on each n' enumerated after n if we find an $n \geq k \ni \pi(i,n)$ is TRUE. Then we have the following.

If φ_i is such that $\forall n \geq k$, $\pi(i,n)$ does not hold, then $\psi_{i,k} = \varphi_i$. If φ_i is such that $\exists n \geq k$, $\pi(i,n)$ holds, then $\psi_{i,k}$ is undefined almost everywhere. Now by enumerating indices in $(\varphi_i)_{i \in \underline{N}}$ for $\{\psi_{i,k} \mid i,k \in \underline{N}\}$ we enumerate the required set. Here is the formalized proof.

7.2.1 PROPOSITION. Let $((\varphi_i)_{i \in \underline{N}}, (\Phi_i)_{i \in \underline{N}})$ be a complexity measure.

Let π be a predicate on $\underline{N}^2 \ni$

$\forall i \in \underline{N}$, $\forall n \in \underline{N}$, $\pi(i,n)$ is defined $\Leftrightarrow \varphi_i(n)$ is defined.

Then $\exists$ computably enumerable $(\psi_j)_{j \in \underline{N}} \ni$

$$\{\psi_j \mid j \in \underline{N}\} = \{\varphi_i \mid i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, \pi(i,n) \text{ does not hold}\}.$$

Proof. Define computable partial ψ on $\underline{N}^2$ by the following procedure.

On input (j,n),

(1) Compute $i,k \ni \langle i,k \rangle = j$.

(2) Compute $\varphi_i(n)$. If $\varphi_i(n)$ is undefined then let $\psi(j,n)$ be undefined.

(3) (In case $\varphi_i(n)$ is defined.)

For each $x \leq \langle n, \Phi_i(n) \rangle$ compute

$n', m \ni x = \langle n', m \rangle$.

If $\Phi_i(n') = m$ and $n' \geq k$ and $\pi(i,n')$ is TRUE then let $\psi(j,n)$ be undefined. (Since $\Phi_i(n')$ is defined, $\varphi_i(n')$ is defined, hence $\pi(i,n')$ is defined and can be computed.)

(4) If (3) does not result in $\psi(j,n)$ being undefined, then output $\varphi_i(n)$ and halt.

Evidently ψ is computable and we have

$$\psi(j,n) = \psi(\langle i,k \rangle, n)$$

$$= \begin{cases} \text{undefined if } \varphi_i(n) \text{ is undefined.} \\ \text{undefined if } \exists \langle n', m \rangle \leq \langle n, \Phi_i(n) \rangle \ni \Phi_i(n') = m, n' \geq k, \\ \qquad \text{and } \pi(i,n') \text{ holds.} \\ \varphi_i(n) \quad \text{if } \forall \langle n', m \rangle \leq \langle n, \Phi_i(n) \rangle \ni \Phi_i(n') = m, n' \geq k, \\ \qquad \text{we have } \pi(i,n') \text{ does not hold.} \end{cases} \Bigg\} \text{if } \varphi_i(n) \text{ is defined}$$

Define $(\psi_j)_{j \in \underline{N}}$ by, $\psi_j(n) = \text{defn } \psi(j,n)$. Then $(\psi_j)_{j \in \underline{N}}$ is computably enumerable since ψ is universal for $(\psi_j)_{j \in \underline{N}}$. We show that each ψ_j is in $\{\varphi_i \mid i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, \pi(i,n) \text{ does not hold}\}$. Choose $i,k \ni \langle i,k \rangle = j$.

If $\forall n \geq k$, $\pi(i,n)$ does not hold, then

$\psi_j = \varphi_i$ and $\varphi_i \in \{\varphi_i \mid i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, \pi(i,n) \text{ does not hold}\}$.

If $\exists n' \geq k \ni \pi(i,n')$ holds, then for all but those finitely many $n \ni \langle n, \Phi_i(n) \rangle < \langle n', \Phi_i(n') \rangle$ we have either

(a) $\varphi_i(n)$ is undefined hence $\psi_j(n)$ is undefined

or (b) $\varphi_i(n)$ is defined and $\langle n', \Phi_i(n') \rangle \leq \langle n, \Phi_i(n) \rangle$ hence $\psi_j(n)$ is undefined by step (3) of the procedure for computing $\psi(j,n)$.

Thus, ψ_j is undefined almost everywhere. Therefore

$\psi_j \in \{\varphi_i \mid i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}[\pi(i,n) \text{ does not hold}]\}$.

Furthermore, if $\varphi_i \notin \{\varphi_i \mid i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, \pi(i,n)$ does not hold$\}$ then $\psi_j = \varphi_i$ where $j = \langle i,k \rangle$ and k is such that $\forall n \geq k, \pi(i,n)$ does not hold. Thus $\{\psi_j \mid j \in \underline{N}\} = \{\varphi_i \mid i \in \underline{N}, \overset{\infty}{\forall} n \in \underline{N}, \pi(i,n)$ does not hold$\}$.□

The above PROPOSITION shows that the set of all h-honest functions for given h is computably enumerable, by choosing $\pi(i,n)$ to be the predicate $\lambda i,n[\Phi_i(n) > h(n,\varphi_i(n))]$.

By slightly modifying the above proof we can get an even stronger result. We show that the set of all h-honest functions is measured. This result was stated as a fact, without proof in [7].

7.2.2 PROPOSITION. Let $((\varphi_i)_{i \in \underline{N}}, (\Phi_i)_{i \in \underline{N}})$ be a complexity measure. Let h be a computable total function on $\underline{N}^2$. There is a measured sequence $(\psi_j)_{j \in \underline{N}} \ni$

$$\{\psi_j \mid j \in \underline{N}\} = \{\varphi \mid \varphi \text{ is h-honest}\}.$$

Proof. Define computable partial ψ on $\underline{N}^2$ by the following procedure.

On input (j,n).

(1) Compute $i,k,m \ni \langle i,k,m \rangle = j$.

(2) Compute $\varphi_i(n)$. If $\varphi_i(n)$ is undefined then let $\psi(j,n)$ be undefined.

(3) (In case $\varphi_i(n)$ is defined.) For each $x \leq \langle n, \Phi_i(n) \rangle$ do the following. Compute $n',m' \ni \langle n',m' \rangle = x$. If $\Phi_i(n') = m'$ then if either $n' < k$ and $m' > m$ or $n' \geq k$ and $m' > h(n',\varphi_i(n'))$ then let $\psi(j,n)$ be undefined.

(4) (In case $\psi(j,n)$ was not seen to be undefined in (3).) Output $\varphi_i(n)$ and halt.

Define $(\psi_j)_{j\in\underline{N}}$ by $\psi_j =_{\text{defn}} \lambda n[\psi(j,n)]$. Evidently ψ is computable hence $(\psi_j)_{j\in\underline{N}}$ is computably enumerable. We show that $(\psi_j)_{j\in\underline{N}}$ is a measured sequence.

By referring to the procedure for computing ψ, particularly at step (3) with $x = \langle n,\Phi_i(n)\rangle$, it is easy to see that the following procedure computes, for given j,n,p, whether or not $\psi_j(n) \simeq p$.

On input (j,n,p).

(1) Compute $i,k,m \ni \langle i,k,m\rangle = j$.

(2) If $n < k$ check if $\Phi_i(n) \leq m$ and
if $n \geq k$ check if $\Phi_i(n) \leq h(n,m)$.
If not then answer FALSE and halt.

(3) (In case $n < k$ and $\Phi_i(n) \leq m$, or $n \geq k$ and $\Phi_i(n) \leq h(n,m)$.) Compute $\varphi_i(n)$. (This must be defined since $\Phi_i(n)$ is defined.) If $\varphi_i(n) \neq p$ then answer FALSE and halt.

(4) (In case $\varphi_i(n) = p$.) Work through the procedure for computing $\psi(j,n)$. If step (3) ever requires that $\psi(j,n)$ be undefined then answer FALSE and halt, otherwise answer TRUE and halt.

Evidently $(\psi_j)_{j\in\underline{N}}$ is a measured sequence. Each ψ_j can be seen to be h-honest as follows. (Let $\langle i,k,m\rangle = j$.)

CASE 1. $\exists n' \in \underline{N}$, $\psi(j,n')$ is undefined by (3) of the procedure for calculating ψ. Then for almost all n, $\psi(j,n)$ is undefined. This is because for almost all n either $\varphi_i(n)$ is undefined or $\langle n,\Phi_i(n)\rangle \geq \langle n',\Phi_i(n')\rangle$. If $\varphi_i(n)$ is undefined then $\psi(j,n)$ is undefined by (2) of the procedure, and if

$\langle n,\Phi_i(n)\rangle \geq \langle n',\Phi_i(n')\rangle$ then $\psi(j,n)$ is undefined by (3) of the procedure. Therefore, ψ_j is undefined almost everywhere, hence is h-Honest.

CASE 2. $\forall n \in \underline{N}$, $\psi(j,n)$ is never undefined by (3) of the procedure. Then $\forall n \in \underline{N}$,

$$\begin{aligned}\psi_j(n) &\simeq \psi(j,n)\\ &\simeq \varphi_i(n).\end{aligned}$$

Furthermore $\forall n \geq k$, either $\varphi_i(n)$ is undefined or

$$\begin{aligned}\Phi_i(n) &= m\\ &\leq h(n,\varphi_i(n)).\end{aligned}$$ (By (3) of the procedure.)

Thus $\psi_j = \varphi_i$ and $\forall n \in \underline{N}$,

$$\Phi_i(n) \leq h(n,\varphi_i(n)).$$

Hence ψ_j is h-honest.

It remains to show that if φ is h-honest then $\exists j \in N \ni \psi_j = \varphi$. Let φ be h-honest. Choose $i,k \in \underline{N} \ni \varphi = \varphi_i$ and $\forall n \geq k$,

$$\Phi_i(n) \leq h(n,\varphi_i(n)).$$

Let $m = \max\{\Phi_i(n) \mid n < k, \varphi_i(n) \text{ is defined}\}$. One can easily see, by following through the procedure for computing $\psi(j,n)$, that

$$\begin{aligned}\psi_j &= \varphi_i\\ &= \varphi,\end{aligned}$$

where $j = \langle i,k,m\rangle$.□

7.2.3 COROLLARY. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure and h be a computable total function on $\underline{N}^2$. There is a computably enumerable sequence $(\psi_j)_{j\in\underline{N}} \ni$

$$\{\psi_j \mid j \in \underline{N}\} = \{\varphi \mid \varphi \text{ is h-honest}\}.$$

Proof. Clear from the previous PROPOSITION and PROPOSITION 4.3.5.□

The following theorem extends PROPOSITION 6.1.7 to computable partial functions. The proof makes use of dovetailing together with a diagonal argument similar to that appearing in the proof of PROPOSITION 6.1.7. The result and the basic method of proof are from Blum [2].

7.2.4 THEOREM Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure. There is a computable total function $f \ni \forall i \in \underline{N}, \forall j \in \underline{N}, \ \mathrm{Dom}(\varphi_{f(i)}) = \mathrm{Dom}(\varphi_i)$ and $\forall j\in\underline{N}, \ \varphi_j = \varphi_{f(i)} \Rightarrow \overset{\infty}{\forall} n \in \underline{N}, \ \Phi_j(n) \gtrsim \varphi_i(n)$.

Furthermore $\forall i \in \underline{N}, \ \mathrm{Im}(\varphi_{f(i)}) \subseteq \{0,1\}$.

Proof. Let f be a computable total function $\ni \forall i \in \underline{N}$, $\varphi_{f(i)} = \lambda n[\psi(i,n)]$, where ψ is the computable partial function on $\underline{N}^2$ defined by the following procedure.

On input (i,n).

(1) Compute $\varphi_i(n)$. If $\varphi_i(n)$ is undefined then let $\psi(i,n)$ be undefined.

(2) (In case $\varphi_i(n)$ is defined.) Do the first $\langle n, \Phi_i(n)\rangle$ stages of the construction of the list L_i, where L_i is constructed in stages $x = 0,1,2,\ldots$ as follows. At the start L_i is empty. Stage $x = \langle k,m\rangle$

(a) Compute whether or not $\Phi_i(k) \simeq m$. If not go to the next stage in the construction of L_i, adding nothing to L_i.

(b) (In case $\Phi_i(k) = m$.) Find the least $i' \leq k \ni \Phi_{i'}(k) < \varphi_i(k)$ and i' is not yet in L_i. If no such i' exists then go on to the next stage of the construction of L_i, adding nothing to L_i.

(c) (In case such i' exists.) Add i' to L_i and go

to the next stage of the construction of L_i.

(3) If i' is added to L_i at stage $\langle n,\Phi_i(n)\rangle$ then define $\psi(i,n)$ by

$$\psi(i,n) =_{\text{defn}} \begin{cases} 1 & \text{if } \varphi_{i'}(n) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

(Notice that if i' is added to L_i at stage $\langle n,\Phi_i(n)\rangle$ then $\Phi_{i'}(n) < \varphi_i(n)$ and $\varphi_i(n)$ is defined hence $\varphi_{i'}(n)$ is defined and can be computed. Also, at most one element is added to L_i at any one stage.) If nothing is added to L_i at stage $\langle n,\Phi_i(n)\rangle$ then define $\psi(i,n)$ by

$$\psi(i,n) =_{\text{defn}} 0.$$

Clearly, $\text{Im}(\psi) \subseteq \{0,1\}$ hence $\forall i \in \underline{N}$, $\text{Im}(\varphi_{f(i)}) = \{0,1\}$. It is easy to see that $\text{Dom}(\varphi_i) = \text{Dom}(\varphi_{f(i)})$. Thus, if we can show that $\forall i \in \underline{N}$, $\forall j \in \underline{N}$,

$$(\overset{\infty}{\exists} n \in \underline{N},\ \Phi_j(n) < \varphi_i(n)) \Rightarrow \varphi_j \neq \varphi_{f(i)}$$

then we have by contraposition and the fact that $\text{Dom}(\varphi_i) = \text{Dom}(\varphi_{f(i)})$, that $\forall i \in \underline{N}$, $\forall j \in \underline{N}$,

$$\varphi_j = \varphi_{f(i)} \Rightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_j(n) \geq \varphi_i(n).$$

Suppose that $\Phi_j(n) < \varphi_i(n)$ for infinitely many n. Choose $n' \geq j \ni \Phi_j(n') < \varphi_i(n')$ (this requires that $\varphi_i(n')$ be defined) and $\langle n',\Phi_i(n')\rangle$ is greater than the maximum of the finite set

$I = \{\langle n_{i'},\Phi_i(n_{i'})\rangle \mid i' < j,\ i' \text{ is added to } L_i \text{ at stage } \langle n_{i'},\Phi_i(n_{i'})\rangle.\}$

Let us look at the value of $\varphi_{f(i)}(n')$.

CASE 1. j is placed in L_i before stage $\langle n',\Phi_i(n')\rangle$ in the construction of L_i. Then by looking at the way in which L_i is constructed we can see that there must be some $k \ni$ $\langle k,\Phi_i(k)\rangle < \langle n',\Phi_i(n')\rangle$ and j is placed in L_i at stage

$\langle k, \Phi_i(k) \rangle$. Hence

$$\varphi_{f(i)}(k) = \psi(i,k)$$
$$\neq \varphi_j(k),$$

by the definition of $\psi(i,k)$.

CASE 2. j is not placed in L_i before stage $\langle n', \Phi_i(n') \rangle$. Look at stage $\langle n', \Phi_i(n') \rangle$ of the construction of L_i. We are to find the least $i' \leq n' \ni \Phi_{i'}(n') < \varphi_i(n')$ and i' is not yet in L_i. Certainly j is a candidate for such i' by the choice of n', but is it the smallest candidate? Yes, it must be, since any candidate, i', smaller than j is in the set I, hence must have been added to L_i at stage $\langle n_{i'}, \Phi_i(n_{i'}) \rangle$ where n' was chosen so that $\langle n_{i'}, \Phi_i(n_{i'}) \rangle < \langle n', \Phi_i(n') \rangle$. Hence j is added to L_i at stage $\langle n', \Phi_i(n') \rangle$. By the definition of f and ψ, then

$$\varphi_{f(i)}(n') = \psi(i,n')$$
$$\neq \varphi_j(n').$$

Thus we have the desired result that $\forall i \in \underline{N}$, $\forall j \in \underline{N}$,

$$(\exists^{\infty} n \in \underline{N},\ \Phi_j(n) < \varphi_i(n)) \Rightarrow \varphi_j \neq \varphi_{f(i)}. \square$$

CHAPTER 8

THE RECURSION THEOREM

§ 8.0 Discussion

In this chapter we look at one of the fundamental results of recursive function theory, the Recursion Theorem. We will not investigate the far-reaching impact of the Recursion Theorem, but only look at it in relation to complexity theory.

In § 8.1 we introduce and prove the Recursion Theorem. We investigate the self-referential aspect of the Recursion Theorem, which entails investigating precisely how the proof works. This investigation of the proof of the Recursion Theorem leads us to see how one might expect to apply the Recursion Theorem to obtain various results in complexity theory. In § 8.2 we look at two results provable by a general form of the Recursion Theorem.

§ 8.1 The Self-Referential Aspect of the Recursion Theorem

Suppose we subdivide the natural numbers into disjoint sets according to a certain property that the numbers have. We think of a function g on $\underline{N}$ as having a fixed point, $x \in \underline{N}$, with respect to this subdivision or this property if x and $g(x)$ are in the same member of the subdivision. That is, x and $g(x)$ are essentially the same in terms of the given property.

For instance, suppose that the property of value of a number is of interest. Each number is placed in its own unique member of the subdivision. Thus, for x to be a fixed point of g with respect to value we must have x and $g(x)$ in the same member of the subdivision, hence $g(x) = x$.

Let us look at a less-trivial example. Suppose the property that interests us is the property of being an index to a function in some acceptable numbering, $(\varphi_i)_{i \in \underline{N}}$. Those numbers which index the same function will be in the same member of the subdivision. Then x is a fixed point for g with respect to indexing if x and $g(x)$ are in the same subdivision, i.e., if $\varphi_{g(x)} = \varphi_x$. The recursion theorem states that every computable total function has such a fixed point.

8.1.1 THEOREM (The Recursion Theorem) Let $(\varphi_i)_{i \in \underline{N}}$ be an acceptable numbering.

$\forall$ computable total g on $\underline{N}$, $\exists x \in \underline{N}$ $\ni$

$$\varphi_x = \varphi_{g(x)}.$$

Proof. Choose μ universal for $(\varphi_i)_{i \in \underline{N}}$ by (1) of DEFINITION 4.2.1 (definition of acceptable numbering). Define ψ on $\underline{N}^2$ by

$$\psi(i,n) \simeq_{\text{defn}} \mu(\mu(i,i),n).$$

Choose computable s by (2) of DEFINITION 4.2.1 such that $\forall i \in \underline{N}, \forall n \in \underline{N}, \varphi_{s(i)}(n) \simeq \psi(i,n)$. Then $\forall i \in \underline{N}, \forall n \in \underline{N}$,

$$\varphi_{s(i)}(n) \simeq \psi(i,n)$$
$$\simeq \mu(\mu(i,i),n)$$
$$\simeq \begin{cases} \varphi_{\varphi_i(i)}(n) & \text{if } \varphi_i(i) \text{ is defined} \\ \text{undefined} & \text{if } \varphi_i(i) \text{ is undefined.} \end{cases}$$

Choose $\nu \ni \varphi_\nu = g \circ s$. This is possible since g and s are computable, hence $g \circ s$ is computable. Let $x = s(\nu)$. Then $\forall n \in \underline{N}$,

$$\varphi_x(n) \simeq \varphi_{s(\nu)}(n)$$
$$\simeq \varphi_{\varphi_\nu(\nu)}(n) \qquad \left[\text{since } \varphi_\nu(\nu) = g \circ s(\nu) \text{ is defined.}\right]$$
$$\simeq \varphi_{g\circ s(\nu)}(n)$$
$$\simeq \varphi_{g(x)}(n).$$

Hence $\varphi_x = \varphi_{g(x)}$. □

An aspect of the recursion theorem which is much more interesting than the fixed-point aspect is the self-referential aspect (see Rogers [8]). In order to investigate this self-referential aspect, let us look more closely at the computation of $\varphi_x(n)$ in the theorem.

In order to compute $\varphi_{s(\nu)}(n)$ (recall $x = s(\nu)$), we compute $\varphi_\nu(\nu)$, then compute and output $\varphi_{\varphi_\nu(\nu)}(n)$. Since $\varphi_\nu = g \circ s$, we can compute $\varphi_{s(\nu)}(n)$ by computing $g \circ s(\nu)$, then computing and outputting $\varphi_{g\circ s(\nu)}(n)$. Since $x = s(\nu)$ we have that $\varphi_x(n)$ is computed by computing $g(x)$, then computing and outputting $\varphi_{g(x)}(n)$. Hence, the computation of $\varphi_x(n)$ is a roundabout computation of $\varphi_{g(x)}(n)$.

In this author's opinion the 'essence' of the self-referential aspect of the recursion theorem is embodied in the following statement.

'The device φ_x computes $\varphi_x(n)$ by computing another index for itself, $g(x)$, then computing and outputting $\varphi_{g(x)}(n)$.'

The above may constitute the essence, but it certainly does not constitute the 'substance' or 'usefulness' of the self-referential aspect of the recursion theorem. We get at the 'substance' by looking at possible choices for the function g.

Suppose g is a computable function defined so that for each i, the device $\varphi_{g(i)}$ computes $\varphi_{g(i)}(n)$ by investigating various computations of φ_i. Now, the device φ_x of the theorem, computes $\varphi_x(n)$ by computing $g(x)$, then computing and outputting $\varphi_{g(x)}(n)$. Hence the computation of $\varphi_x(n)$ will look at various computations of φ_x. Thus, 'by appropriate choice of g, we can define a device φ_x which investigates various computations done by itself.'

This author considers the above to be the 'substance' of the self-referential aspect of the recursion theorem. Looked at in the light of the above discussion, the proof of the following 'resource waste' theorem becomes almost trivial. The theorem indicates that any function can be computed in arbitrarily complex ways. Notice the result holds everywhere.

8.1.2 PROPOSITION. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure. Let r, f be any two computable total functions on $\underline{N}$. Then,

$$\exists x \in \underline{N}, \ni \varphi_x = f \text{ and}$$

$$\forall n \in \underline{N}, \Phi_x(n) \geq r(n).$$

Proof. Define ψ on $\underline{N}^2$ by

$$\psi(i,n) \simeq \begin{cases} \text{undefined} & \text{if } \Phi_i(n) < r(n) \\ f(n) & \text{otherwise.} \end{cases}$$

Evidently ψ is computable. Choose computable total g on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$, $\forall n \in \underline{N}$, $\varphi_{g(i)}(n) = \psi(i,n)$. This is possible since $(\varphi_i)_{i\in\underline{N}}$ is an acceptable numbering. Choose $x \ni \varphi_x = \varphi_{g(x)}$ by the Recursion Theorem. Then, $\forall n \in \underline{N}$,

$$\begin{aligned}\varphi_x(n) &\simeq \varphi_{g(x)}(n)\\ &\simeq \psi(x,n)\\ &\simeq \begin{cases} \text{undefined} & \text{if } \Phi_x(n) < r(n)\\ f(n) & \text{otherwise.}\end{cases}\end{aligned}$$

Now, if $\Phi_x(n) < r(n)$ then $\varphi_x(n)$ is defined. Thus, it is impossible, for any $n \in \underline{N}$, that $\varphi_x(n)$ is undefined and $\Phi_x(n) < r(n)$. Therefore, $\forall n \in \underline{N}$,

$$\varphi_x(n) = f(n) \quad \text{and} \quad \Phi_x(n) \geq r(n). \square$$

The recursion theorem proves to be more useful in the following more general form.

8.1.3 LEMMA. Let $(\varphi_i)_{i\in\underline{N}}$ be an acceptable numbering. Let h be any computable total function on $\underline{N}^{k+1}$. Then, $\exists$ computable total f on $\underline{N}^k$, $\forall i_1, i_2, \ldots, i_k \in \underline{N}$,

$$\varphi_{f(i_1,i_2,\ldots,i_k)} = \varphi_{h(i_1,i_2,\ldots,i_k,\, f(i_1,i_2,\ldots,i_k))}$$

Proof. Define ψ on $\underline{N}^{k+2}$ by

$$\psi(i_1,i_2,\ldots,i_k,i,n) = \mu(\mu_{k+1}(i,i_1,\ldots,i_k,i),n)$$

where μ is universal for $(\varphi_i)_{i\in\underline{N}}$ and μ_{k+1} is universal for $(\tau^{k+1}_i)_{i\in\underline{N}}$. Evidently ψ is computable. Choose g on $\underline{N}^{k+1}$, by (2) $\Rightarrow$ (1) of THEOREM 4.1.6 and DEFINITION 4.2.1 (definition of acceptable numbering), such that ψ is universal for $(\varphi_{g(i_1,\ldots,i_k,i)})_{(i_1,\ldots,i_k,i)\in\underline{N}^{k+1}}$. Choose $\nu \ni \forall i_1, i_2, \ldots, i_k, i \in \underline{N}$,

$$\tau_{\nu}^{k+1}(i_1,i_2,\ldots,i_k,i) = h(i_1,i_2,\ldots,i_k,\ g(i_1,i_2,\ldots,i_k,i)).$$

Define f by

$$f(i_1,i_2,\ldots,i_k) =_{\text{defn}} g(i_1,i_2,\ldots,i_k,\nu).$$

Then $\forall i_1,i_2,\ldots,i_k \in \underline{N}$, $\forall n \in \underline{N}$,

$$\begin{aligned}
\varphi_{f(i_1,\ldots,i_k)}(n) &\simeq \varphi_{g(i_1,i_2,\ldots,i_k,\nu)}(n) \\
&\simeq \psi(i_1,i_2,\ldots,i_k,\nu,n) \\
&\simeq \mu(\mu_{k+1}(\nu,i_1,i_2,\ldots,i_k,\nu),n) \\
&\simeq \varphi_{\mu_{k+1}(\nu,i_1,i_2,\ldots,i_k,\nu)}(n) \\
&\simeq \varphi_{\tau_{\nu}^{k+1}(i_1,\ldots,i_k,\nu)}(n) \\
&\simeq \varphi_{h(i_1,\ldots,i_k,\ g(i_1,\ldots,i_k,\nu))}(n) \\
&\simeq \varphi_{h(i_1,\ldots i_k,\ f(i_1,\ldots,i_k))}(n).
\end{aligned}$$

Evidently, f is computable and total.□

In the next section we look at two results whose proofs make use of this more general result.

§ 8.2 Some Uses of the Recursion Theorem

It is not surprising that many results that are very trivial to prove using a specific model of computation become quite difficult to prove when we abstract to acceptable numberings. In some cases this is because of the fact that in dealing with acceptable numberings we cannot stipulate precisely how a computation is to be carried out nor can we look at how the index of a function is arrived at.

As an example of the first situation, where we cannot stipulate how a computation is to be carried out, recall the Resource Waste Theorem of the previous section. In the case of Turing machines, we know precisely how Turing machines operate and can see that it is possible to design one that wastes $r(n)$ work tape cells in computing $f(n)$. We simply compute $r(n)$, run the work tape head along $r(n)$ tape cells, then proceed with the computation of $f(n)$. However, there is no straightforward way to generalize this proof to arbitrary acceptable numberings. Instead, we rely on the recursion theorem to tell us that in any acceptable numbering it is possible to define functions which investigate their own computations.

The following proposition is a generalization of the Resource Waste Theorem which shows that the result works for partial functions as well as total functions. Furthermore, it indicates that an index for a wasteful function can be computed from the indices of the given functions, thus the wasteful function can be 'constructed' from the given functions.

8.2.1 PROPOSITION. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure.

$\exists$ computable total w on $\underline{N}^2 \ni \forall i,j \in \underline{N}$, $\mathrm{Dom}(\varphi_i) = \mathrm{Dom}(\varphi_j)$

$\Rightarrow \varphi_{w(i,j)} = \varphi_i$ and $\forall n \in \underline{N}$, $\Phi_{w(i,j)}(n) \geq \varphi_j(n)$.

Proof. Define ψ on $\underline{N}^4$ by

$$\psi(i,j,k,n) \simeq_{defn} \begin{cases} \text{undefined} & \text{if } \varphi_j(n) \text{ is undefined} \\ \text{undefined} & \text{if } \Phi_k(n) < \varphi_j(n) \\ \varphi_i(n) & \text{otherwise} \end{cases} \Bigg\} \text{if } \varphi_j(n) \text{ is defined.}$$

Evidently ψ is computable. Choose computable h on $\underline{N}^3$ $\ni$ $\forall i,j,k \in \underline{N}$, $\forall n \in \underline{N}$,

$$\varphi_{h(i,j,k)}(n) \simeq \psi(i,j,k,n)$$

by (2) $\Rightarrow$ (1) of THEOREM 4.1.6 and by DEFINITION 4.2.1 (the definition of an acceptable numbering). Choose w by THEOREM 8.1.2 $\ni$ $\forall i,j \in \underline{N}$,

$$\varphi_{w(i,j)} = \varphi_{h(i,j,w(i,j))}.$$

Then, $\forall i,j \in \underline{N}$, $\forall n \in \underline{N}$,

$$\varphi_{w(i,j)}(n) \simeq \varphi_{h(i,j,w(i,j))}(n)$$

$$(*) \qquad \simeq \begin{cases} \text{undefined} & \text{if } \varphi_j(n) \text{ is undefined} \\ \text{undefined} & \Phi_{w(i,j)}(n) < \varphi_j(n) \\ \varphi_i(n) & \text{otherwise} \end{cases} \Bigg\} \text{if } \varphi_j(n) \text{ is defined.}$$

Suppose $\varphi_j(n)$ is defined and $\Phi_{w(i,j)}(n) < \varphi_j(n)$. Then $\Phi_{w(i,j)}(n)$ is defined, hence $\varphi_{w(i,j)}(n)$ is defined. But in this case $\varphi_{w(i,j)}(n)$ is undefined by (*) above. Therefore, this case is impossible. Thus, $\forall i,j \in \underline{N}$, $\forall n \in \underline{N}$,

$$\varphi_{w(i,j)}(n) \simeq \begin{cases} \text{undefined} & \text{if } \varphi_j(n) \text{ is undefined} \\ \varphi_i(n) & \text{if } \varphi_j(n) \text{ is defined,} \end{cases}$$

and $\varphi_j(n)$ defined $\Rightarrow$ ($\Phi_{w(i,j)}(n) \geq \varphi_j(n)$ or $\Phi_{w(i,j)}(n)$ is undefined).

Hence, $\forall i,j \in \underline{N}$, if $Dom(\varphi_i) = Dom(\varphi_j)$ then,

$$\varphi_{w(i,j)} = \varphi_i \text{ and}$$

$$\forall n \in \underline{N},\ \Phi_{w(i,j)}(n) \gtrsim \varphi_j(n). \square$$

As an example of the problems that can arise because of the fact that we do not know how an index of a function is arrived at in an arbitrary acceptable numbering, we look at the result that each computable function has infinitely many indices. This is easy to prove using Turing machines because we know that adding more instructions to a machine gives it a larger index. Thus we can get successively larger indices for the same machine by successively adding "useless" instructions.

For arbitrary acceptable numberings we can prove the result by making use of the recursion theorem. Here we use the recursion theorem to construct functions that investigate their own indices.

8.2.2 PROPOSITION. Let $(\varphi_i)_{i\in\underline{N}}$ be an acceptable numbering. $\exists$ computable total s on $\underline{N}$, $\forall i \in \underline{N}$,

$$s(i) > i \text{ and } \varphi_{s(i)} = \varphi_i.$$

Proof. Define $\psi(i,j,k,n)$ on $\underline{N}^4$ by

$$\psi(i,j,k,n) \simeq_{\text{defn}} \begin{cases} \varphi_i(n) & k > i \\ j & k \leq i. \end{cases}$$

Choose computable h on $\underline{N}^3 \ni \forall i,j,k \in \underline{N}$, $\forall n \in \underline{N}$

$$\varphi_{h(i,j,k)}(n) \simeq \psi(i,j,k,n).$$

Choose computable w on $\underline{N}^2 \ni \forall i,j \in \underline{N}$,

$$\varphi_{w(i,j)} = \varphi_{h(i,j,w(i,j))}.$$

Then $\forall i,j \in \underline{N}$, $\forall n \in \underline{N}$,

$$\varphi_{w(i,j)}(n) = \begin{cases} \varphi_i(n) & w(i,j) > i \\ j & w(i,j) \leq i. \end{cases}$$

Now, we wish to show that

$$\forall i \in \underline{N}, \exists j \in \underline{N} \ni w(i,j) > i.$$

Suppose not. Choose $i \ni$

$$\forall j \in \underline{N},\ w(i,j) \leq i.$$

Then there is $j_1, j_2 \in \underline{N}$ ∋

(*) $$j_1 \neq j_2$$

and $$w(i,j_1) = w(i,j_2)$$
$$\leq i.$$

Thus $\forall n \in \underline{N}$, $$\varphi_{w(i,j_1)}(n) = j_1,$$
$$\varphi_{w(i,j_2)}(n) = j_2,$$

and since $w(i,j_1) = w(i,j_2)$ we have

$$\varphi_{w(i,j_1)} = \varphi_{w(i,j_2)}.$$

Hence, $$j_1 = j_2.$$

This contradicts (*), thus $\forall i \in \underline{N}, \exists j \in \underline{N}, \ni w(i,j) > i$. Define s on $\underline{N}$ by

$$s(i) = \underset{\text{defn}}{} w(i, \mu j[w(i,j) > i]).$$

Evidently, s is computable and

$$s(i) > i.$$

Furthermore, $\forall i \in \underline{N}$, $\exists j \in \underline{N}$ ∋

$$s(i) = w(i,j) > i$$

and $\forall n \in \underline{N}$,

$$\varphi_{s(i)}(n) \simeq \varphi_{w(i,j)}(n)$$
$$\simeq \varphi_i(n),$$

since $w(i,j) > i$. Thus, $\forall i \in \underline{N}$, $\varphi_{s(i)} = \varphi_i$ and $s(i) > i$.□

Repeated applications of the above PROPOSI ION give us the result that every computable function has infinitely many indices in any acceptable numbering.

CHAPTER 9

FURTHER TOPICS

§ 9.0 Discussion

A technique of proof that may gain increasing popularity in complexity theory is the use of measure invariance (see § 5.2). If a property can be shown to be measure invariant and it can be demonstrated that at least one specific measure has this property, then we can conclude that all complexity measures have this property. In § 9.1 we look at the Speed Up Theorem as an example of this technique.

In § 5.1 it was mentioned that the study of complexity classes has become one of the main topics of study in complexity theory. This is due in part to the fact that most of the basic complexity theory results can be stated in terms of complexity classes. In § 9.2 we will look at the Compression Theorem and the Gap Theorem in this light.

The proof of this Gap Theorem resulted in a flurry of activity in an attempt to find a better method of naming the complexity classes. From this activity came the Naming Theorem. This theorem was proven using a technique which is called a priority argument. It is felt that a study of priority arguments is beyond the scope of this paper. However, in § 9.3 we will take a brief look at the proof of the Naming Theorem.

§ 9.1 Measure Invariance

If a property can be shown to be measure invariant and if it can be demonstrated that some particular complexity measure has that property, then we can conclude that all complexity measures have that property. LEMMA 5.2.9 proves to be quite useful in showing that certain properties are measure invariant. This can be nicely exemplified by a proof of the Speed Up Theorem given by Hartmanis and Hopcroft in [4].

The Speed Up Theorem was originally stated and proven by Blum (see [2]) in a straightforward but difficult manner using some techniques similar to those used in the proof of THEOREM 7.2.4. In [4], Hartmanis and Hopcroft demonstrated that Speed Up is measure invariant and also gave an example of a particular measure in which Speed Up was shown to hold. This two-stage proof is relatively easy to follow.

Essentially, the Speed Up Theorem indicates that in every complexity measure there are functions so complex that for each method of computing such a function, there is always a more efficient method. However, the computation is more efficient only on almost every argument, and furthermore, the faster we "speed up" the computation by successive applications of the theorem, the greater the number of exceptions.

In the following proof of the Speed Up Theorem we demonstrate only that Speed Up is measure invariant. The reader can refer to the work by Hartmanis and Hopcroft in [4] in order to see how to construct a specific measure for which the Speed Up Theorem can be easily shown to hold.

9.1.1 THEOREM. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure and r be any computable total function on $\underline{N}^2$. There is a computable total function g on $\underline{N}$ $\ni$ $Im(g) \subseteq \{0,1\}$ and $\forall i \in \underline{N}$,

$$\varphi_i = g \Rightarrow \exists j \in \underline{N},\ \varphi_j = g \text{ and } \overset{\infty}{\forall} n \in \underline{N},\ r(n,\Phi_j(n)) \leq \Phi_i(n).$$

Proof. We only show that if a complexity measure $((\hat{\varphi}_i)_{i\in\underline{N}}, (\hat{\Phi}_i)_{i\in\underline{N}})$ satisfies the THEOREM then so does any other complexity measure $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$.

By LEMMA 5.2.9 choose computable total $f,\hat{f}$ on $\underline{N}$ and $h,\hat{h}$ on $\underline{N}^2$ $\ni \forall i \in \underline{N}$,

$$\varphi_{f(i)} = \hat{\varphi}_i$$

$$\hat{\varphi}_{\hat{f}(i)} = \varphi_i$$

(**) $\quad \forall m_1,m_2 \in \underline{N},\ \forall n \in \underline{N},\ \hat{h}(n,m_1) \geq \hat{h}(n,m_2) \Rightarrow m_1 \geq m_2$

(*) $\quad \overset{\infty}{\forall} n \in \underline{N},\ \Phi_{f(i)}(n) \leq h(n,\hat{\Phi}_i(n))$

and $\overset{\infty}{\forall} n \in \underline{N},\ \hat{\Phi}_{\hat{f}(i)}(n) \leq \hat{h}(n,\Phi_i(n))$.

Given computable total r on $\underline{N}^2$ define r' on $\underline{N}^2$ by

$$r'(n,m) =_{\text{defn}} \hat{h}(n,\max\{r(n,m') \mid m' \leq h(n,m)\}).$$

Since $((\hat{\varphi}_i)_{i\in\underline{N}}, (\hat{\Phi}_i)_{i\in\underline{N}})$ satisfies the THEOREM, choose computable total g on $\underline{N}$ $\ni$ $\mathrm{Im}(g) \subseteq \{0,1\}$ and $\forall i \in \underline{N}$,

(***) $\quad \hat{\varphi}_i = g \Rightarrow \exists j' \in \underline{N},\ \hat{\varphi}_{j'} = g$ and $\overset{\infty}{\forall} n \in \underline{N},\ r'(n,\hat{\Phi}_{j'}(n)) \leq \hat{\Phi}_i(n)$.

Let $i \in \underline{N}$ be given $\ni \varphi_i = g$. We will attempt to find $j \in \underline{N}$ $\ni$

$$\varphi_j = g \text{ and } \overset{\infty}{\forall} n \in \underline{N},\ r(n,\Phi_j(n)) \leq \Phi_i(n),$$

thus proving the desired result.

From the definition of r' and the fact that

$$\hat{\varphi}_{\hat{f}(i)} = \varphi_i$$

$$= g$$

we have, by (***), some $j' \in \underline{N}$ $\ni \hat{\varphi}_{j'} = g$ and

$$\overset{\infty}{\forall} n \in \underline{N},\ \hat{h}(n,\max\{r(n,m') \mid m' \leq h(n,\hat{\Phi}_{j'}(n))\}) \leq \hat{\Phi}_{f(i)}(n).$$

By (*) and the fact that φ_i is total, we have that

$$\overset{\infty}{\forall} n \in \underline{N},\ \hat{\Phi}_{\hat{f}(i)}(n) \leq \hat{h}(n,\Phi_i(n)).$$

Thus,

$$\forall^{\infty} n \in \underline{N},\ \hat{h}(n,\max\{r(n,m') \mid m' \leq h(n,\hat{\Phi}_{j'}(n))\}) \leq \hat{h}(n,\Phi_i(n)).$$

Hence, by (**), we have

$$\forall^{\infty} n \in \underline{N},\ \max\{r(n,m') \mid m' \leq h(n,\hat{\Phi}_{j'}(n))\}) \leq \Phi_i(n).$$

Now, by (*), we have

$$\forall^{\infty} n \in \underline{N},\ \Phi_{f(j')}(n) \leq h(n,\hat{\Phi}_{j'}(n)),$$

thus,

$$\forall^{\infty} n \in \underline{N},\ r(n,\Phi_{f(j')}(n)) \leq \max\{r(n,m') \mid m' \leq h(n,\hat{\Phi}_{j'}(n))\}.$$

Therefore, letting $j = f(j')$, we have

$$\begin{aligned} \forall^{\infty} n \in \underline{N},\ r(n,\Phi_j(n)) &= r(n,\Phi_{f(j')}(n)) \\ &\leq \max\{r(n,m') \mid m' \leq h(n,\hat{\Phi}_{j'}(n))\} \\ &\leq \Phi_i(n). \end{aligned}$$

Also,

$$\begin{aligned} \varphi_j &= \varphi_{f(j')} \\ &= \hat{\varphi}_{j'} \\ &= g.\square \end{aligned}$$

§ 9.2 Complexity Classes

In § 5.1 we indicated that one of the main fields of study in complexity theory is the study of properties of complexity classes. (For example, see [6] and [7].) One reason for this is because many of the fundamental results of complexity theory can be stated very concisely in terms of complexity classes.

For example, THEOREM 7.2.4 can be reformulated (with a slight modification to the proof) as (*) $\forall$ complexity measure $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ $\exists$ computable total f on $\underline{N}$, $\forall i \in \underline{N}$, $\mathrm{Dom}(\varphi_i) = \mathrm{Dom}(\varphi_{f(i)})$ and if φ_i is total then

$$\varphi_{f(i)} \notin C_{\varphi_i}.$$

Thus no complexity class contains all of the computable total functions. This result can be carried slightly further to give a more striking result often called the Compression Theorem (Blum [2]).

9.2.1 THEOREM. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure. There is computable total function h on $\underline{N}^2$ $\ni$ $\forall i \in \underline{N}$, if φ_i is total then

$$C_{\Phi_i} \subsetneq C_{\lambda n[h(n,\Phi_i(n))]}.$$

Proof. Choose f as in (*) above. By PROPOSITIONS 4.2.3 and 4.3.5, choose computable total s on $\underline{N}$ $\ni$ $\forall i \in \underline{N}$,

$$\varphi_{s(i)} = \Phi_i.$$

Define computable total p on $\underline{N}^3$ by

$$p(i,n,m) = \text{defn} \begin{cases} \max\{\Phi_i(n), \Phi_{f(s(i))}(n)\} & \text{if } \Phi_i(n) = m \\ 0 & \text{otherwise.} \end{cases}$$

Define computable total h on $\underline{N}^2$ by

$$h(n,m) =_{defn} \max\{p(j,n,m) \mid j \leq n\}.$$

Then, $\forall i \in \underline{N}$, $\forall n \geq i$,

$$h(n,\Phi_i(n)) \geq \Phi_{f(s(i))}(n)$$

hence,

$$\varphi_{f(s(i))} \in C_{\lambda n[h(n,\Phi_i(n))]}.$$

Also, $\forall i \in \underline{N}$, $\forall n \geq i$,

$$h(n,\Phi_i(n)) \geq \Phi_i(n)$$

hence

$$C_{\Phi_i} \subseteq C_{\lambda n[h(n,\Phi_i(n))]}.$$

However, if φ_i is total, then

$$\varphi_{f(s(i))} \notin C_{\varphi_{s(i)}} = C_{\Phi_i}.$$

Thus

$$C_{\Phi_i} \subsetneq C_{\lambda n[h(n,\Phi_i(n))]}.\square$$

The Compression Theorem is often contrasted with the Gap Theorem (Borodin [3]). This result indicates that there are arbitrarily large gaps in complexities through which no new functions are computed.

9.2.2 THEOREM. Let $((\varphi_i)_{i\in\underline{N}}, (\Phi_i)_{i\in\underline{N}})$ be a complexity measure.

$\forall$ computable total g on $\underline{N}^2$ $\ni$ $\forall n,m \in \underline{N}$, $g(n,m) \geq m$, $\forall$ computable total r on $\underline{N}$, $\exists$ computable total increasing t on $\underline{N}$ $\ni$ $\forall n \in \underline{N}$, $t(n) \geq r(n)$ and

$$C_t = C_{\lambda n[g(n,t(n))]}$$

Proof. Given g and r define t inductively by the following procedure.

On input n.

(1) If $n = 0$ then output $r(0)$ and halt.

(2) Compute $t(n - 1)$.

(3) Set k to $\max\{t(n - 1), r(n)\}$.

(4) If, for each $i \leq n$, either

$$\Phi_i(n) \lesssim k$$

or $$\Phi_i(n) \not\lesssim g(n,k)$$

then output k and halt.

(5) Set k to $k + 1$.

(6) Go to (4).

Evidently, t is computable. t is total since if $\Phi_i(n)$ is undefined then $\Phi_i(n) \not\lesssim g(n,k)$ and if $\Phi_i(n)$ is defined then $\Phi_i(n) \lesssim k$ for every $k \geq \Phi_i(n)$. t is increasing and $\forall n \in \underline{N}$, $t(n) \geq r(n)$, by (3). We now prove t is the required function by proving that $\forall i \in \underline{N}$,

$$\overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \leq t(n) \Leftrightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \leq g(n,t(n)).$$

Let $i \in \underline{N}$ be given. We have $(\Rightarrow)$ since $\forall n \in \underline{N}$,

$$g(n,t(n)) \geq t(n).$$

Suppose $\overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \leq g(n,t(n))$. For any $n \geq i \ni \Phi_i(n) \leq g(n,t(n))$ we have $\Phi_i(n) \leq t(n)$, by (4) in the procedure for computing t. Thus, $\overset{\infty}{\forall} n \in \underline{N},\ \Phi_i(n) \leq t(n)$.□

The Compression Theorem indicates that if a complexity class is named by a total resource counting function then we have a method of naming a strictly larger complexity class. However, the Gap Theorem indicates that this method will not work for every computable total function used to name a complexity class.

§ 9.3 Conclusion

In light of the Gap Theorem, much work was done in an attempt to rename the complexity classes of a complexity measure with a smaller set of functions than the set of all computable functions, so that various gap phenomena would not appear. In [3], Borodin shows that this is to some extent, impossible. However, one result of these attempts is the Naming Theorem.

The Naming Theorem states that for every complexity measure there is a measured sequence which names all of the complexity classes. The result is due to McCreight and Meyer [7] and is proven using a technique called a priority argument.

The author feels that a discussion of priority arguments is beyond the scope of this paper. Suffice it to say that these types of arguments can sometimes be used in defining functions which must satisfy a set of near conflicting conditions. (See Rogers [8] for a more detailed discussion of priority arguments.)

In McCreight and Meyer's proof of the Naming Theorem a priority argument is used to define a computable total function g on $\underline{N}$ $\ni$ $\mathrm{Dom}(\varphi_i) = \mathrm{Dom}(\varphi_{g(i)})$ and

(1) Given $i,j \in \underline{N}$, we have for each $n \in \underline{N}$ $\ni$ $\varphi_i(n) < \Phi_j(n)$, some associated $m_n \in \underline{N}$ $\ni$

(a) $\varphi_{g(i)}(m_n) < \Phi_j(m_n)$

and (b) $n_1 \neq n_2 \Rightarrow m_{n_1} \neq m_{n_2}$.

Furthermore (c) $\varphi_{g(i)}(m) < \Phi_j(m) \Rightarrow m = m_n$ for some n $\ni$ $\varphi_i(n) < \Phi_j(n)$.

(2) Each $\varphi_{g(i)}$ has value large enough (relative to its complexity) so

that $\{\varphi_{g(i)} \mid i \in \underline{N}\}$ is honest.

These two conditions conflict to the extent that (1) asks that certain values of $\varphi_{g(i)}$ be small whereas (2) asks that values of $\varphi_{g(i)}$ be large. The intention of condition (1) is to give us that $\forall i,j \in \underline{N}$,

$$\overset{\infty}{\exists} n \in \underline{N},\ \varphi_i(n) < \Phi_j(n) \Leftrightarrow \overset{\infty}{\exists} n \in \underline{N},\ \varphi_{g(i)}(n) < \Phi_j(n).$$

Hence, by contraposition, and the fact that $\mathrm{Dom}(\varphi_i) = \mathrm{Dom}(\varphi_{g(i)})$, we have $\forall i,j \in \underline{N},\ \overset{\infty}{\forall} n \in \underline{N},\ \Phi_j(n) \leq \varphi_{g(i)}(n) \Leftrightarrow \overset{\infty}{\forall} n \in \underline{N},\ \Phi_j(n) \leq \varphi_i(n)$, or $\forall i \in \underline{N},\ C_{\varphi_i} = C_{\varphi_{g(i)}}$.

The intention of condition (2) is to give us that $\{\varphi_{g(i)} \mid i \in \underline{N}\}$ is contained in some measured sequence, by PROPOSITION 7.2.2.

The impact of the Naming Theorem becomes clear when we notice that, in the Compression Theorem, we can replace the resource counting functions, $(\Phi_i)_{i \in \underline{N}}$, by any other measured sequence. Hence, the resource counting functions can be replaced by the measured sequence given in the Naming Theorem.

Currently much of the work done in complexity theory is a investigation of the recursive properties of compexity classes. There is some work being done in relating complexity theory to concepts such as simulation and parallelism (see [4]).

It is hoped that this paper has provided a strong basis of knowledge in complexity theory from which the reader may easily begin to study these more advanced topics.

BIBLIOGRAPHY

[1] Ausiello, Georgio. Abstract computational complexity and cycling computations. Journal of Computer and System Sciences, 5 (1971), 118-128.

[2] Blum, Manuel. A machine-independent theory of the complexity of computations. Journal of the Association for Computing Machinery, 14, No. 2 (April 1967), 322-336.

[3] Borodin, A. Computational complexity and the existence of complexity gaps. Journal of the Association for Computing Machinery, 19, No. 1 (January 1972), 158-174.

[4] Hartmanis, J. and Hopcroft, J. E. An overview of the theory of computational complexity. Journal of the Association for Computing Machinery, 18, No. 3 (July 1971), 444-475.

[5] Hopcroft, J. E. and Ullman, J. D. Formal languages and their relation to Automata. Addison-Wesley Publishing Company, Don Mills, Ontario, 1969.

[6] Landweber, L. H. and Robertson, E. R. Recursive properties of abstract complexity classes. Journal of the Association for Computing Machinery, 19, No. 2 (April 1972), 296-308.

[7] McCreight, E. M. and Meyer, A. R. Classes of computable functions defined by bounds on computation: preliminary report. Association for Computing Machinery Symposium on the Theory of Computing, May 1969, pp. 79-88.

[8] Rogers, Hartley, Jr. Theory of recursive functions and effective computability. McGraw-Hill Book Company, Toronto, Ontario, 1967.