Department of Applied Analysis
and Computer Science

University of Waterloo

Technical Report CS-73-28

INPUT-OUTPUT CONVENTIONS AND THE COMPLEXITY OF TRANSDUCTIONS

by

Lawrence V. Saxton

INPUT-OUTPUT CONVENTIONS

AND

THE COMPLEXITY OF TRANSDUCTIONS

by

Lawrence V. Saxton

A Thesis

Presented to the

University of Waterloo

in Partial Fulfillment of

the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in Mathematics

Department of Applied Analysis and Computer Science

Faculty of Mathematics

September, 1973

ACKNOWLEDGEMENTS

The author wishes to express his very sincere thanks to Professor P.C. Fischer for his guidance, encouragement, criticism and many helpful suggestions which made this thesis possible.

Thanks are also extended to Professor E.L. Robertson for the discussions and suggestions which were invaluable in the development of chapters 3 and 4.

The author is particularly grateful to Mrs. J. MacPherson, Mrs. W. Richardson and Mrs. E. Fitzgerald whose skills were evident in the preparation of this thesis.

The financial support provided by a National Research Council of Canada scholarship is gratefully acknowledged.

ABSTRACT

This thesis is concerned with the effect of input-output behaviour on the computation of functions. We first introduce a model called the sequential Turing machine, which reflects the restriction on the input-output behaviour desired. This model is shown to extend the more commonly used model, the on-line Turing machine, although neither is a model of a universal computer.

We next study whether we can extend the power of sequential Turing machines by allowing input endmarkers. It is shown that for every (computable) function, there is a (computable) sequential function with such an extended domain which has the same output. The function which has such a property is called a 'sequential approximation'. A function which further has the property that it makes the maximal amount of output for any input not containing the endmarker is a 'maximally defined sequential approximation'. Such a function always exists,but need not be computable, even if the original is computable.

The concept of maximally defined sequential approximation leads to the measurement of the sequential nature of a function in terms of the two related functions. It is shown by construction that a dense, infinite hierarchy based on this measure exists.

Classes of functions based on several time functions are also investigated. The time functions are measured in terms of the length of the input, the length of the output, the maximum of these lengths, the length of the profile and the sum of the lengths. It is shown that these classes are contained in one another in the above order, with the exception of the first two which are incomparable.

We are principally interested in the classes defined for time functions t which are small; that is, the case of 'real-time' computable functions (when $t(n) = n$, for all n) and the case of 'linear-time' computable functions (when $t(n) = c.n$, for all n). We show that for the case of real-time computable functions, the containment is proper. Also, by restricting attention to length-preserving functions, we show that the first three classes are equal. For the case of linear-time computable functions, the last three classes are shown to be equal. Furthermore, we demonstrate that there exists classes between these two traditionally studied classes.

Finally, we investigate the effect of using a sequential Turing machine to compute a function in terms of the number of steps used. It is shown that at worst a function will take the length of the input times the time of computation on an unrestricted Turing machine. Moreover, there exists a function for which this bound is almost achieved.

PREFACE

This thesis is largely self-contained, only the most basic definitions concerning finite automata and Turing machines having been omitted. It is organized so that the introductory material for the whole thesis is contained in chapter 1. The remainder falls naturally into two parts, chapter 2, 3, and 4, and chapters 5 and 6. Introductory material to each chapter is contained in the first section.

Chapters are subdivided into sections, which are numbered sequentially within the chapter: e.g., section 2.3 is the third section of chapter 2. Members of each section (definitions, theorems, lemmas, corollaries) are sequentially numbered within sections: e.g., theorem 2.3.3 is the third unit in section 2.3.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

The area of machine-based computational complexity concerns the mathematical study of an abstract computer with one or more of its resources restricted in order to gain insight and knowledge into real-world computer problems. The restrictions most commonly studied are the limiting of the amount of space available for the storage of intermediate data, the limiting of the time (the number of steps) within which a computation must be completed, and the limiting of the manner in which the input and output devices can be operated. The final alternative has been traditionally studied in order to aid in the study of the second alternative.

Two real-world cases for studying the limiting of the manner in which the input and output devices can be operated are the study of 'real-time' processes and 'on-line' (ongoing) processes. In the real-world context, a process is 'real-time' if the computer is providing output as fast as the input is being supplied. An 'on-line' process is one in which the computer provides all of the output defined by the input already supplied before accepting the next input.

Historically, the concept of 'real-time' was one of the foundations of the study of machine-based complexity theory. Yamada [17] introduced the concept of real-time countable functions. Essentially, a function f: $N \to N$ (N is the set of natural numbers) is real-time countable if f is strictly increasing and the sequence

α^f, with $\alpha^f_n=1$ if $n = f(m)$ for some m, and $\alpha^f_n = 0$ otherwise, is computable on some machine such that each α^f_i is produced on the ith step of the machine. In [17], Yamada studied the properties of the class C of real-time countable functions showing that C is closed under the various operations used in standard recursion theory: e.g., limited addition, composition, multiplication and limited multiplication.

In another foundational paper [9], Hartmanis and Stearns introduced the concept of real-time computable sequences, as well as more general topics. Because of the fundamental nature of this paper, we give it a more general discussion.

For a given T: N→N, denote by S_T the set of all sequences α over $\{0,1\}^*$ such that there is some multi-tape Turing machine which produces α_n in no more than T(n) steps for every n. Then, $S_T = S_{\lceil kT}$[1], for any number $k > 0$. This result will be referred to as the constant speedup theorem for obvious reasons.

Hartmanis and Stearns next showed that for any T, S_T is recursively enumerable but not necessarily recursive. They showed when two classes S_U and S_T for U and T real-time countable functions, are not equal. That is, if $0 < \lim_{n\to\infty} T(n)/U(n) < \infty$, then $S_U = S_T$ and if $\lim_{n\to\infty} (\frac{(T(n))^2}{U(n)} = 0)$, then there is a sequence in $S_U - S_T$.

Finally in [9], Hartmanis and Stearns defined the related notion of 'real-time' acceptance by a multitape Turing machine. Essentially, a set A is real-time accepted if there exists a

[1]We denote as $\lceil x$ the smallest integer greater than or equal to x.

multitape Turing machine which reads on every step and produces a 0 or 1 if $x \notin A$ or $x \in A$, respectively, immediately after receiving the last input of x, for every input x. They showed that there exists a context free language which is not real-time accepted by any multitape Turing machine.

P.C. Fischer further studied the classes of sequences generable in time T in [5]. He showed that if a sequence α is generable by a multitape Turing machine in no more than T(n) steps for each α_n, then α is generable by a multitape Turing machine in exactly T(n) steps for each α_n. As a consequence, he concluded that $S_n = S_{\lceil kn}$, for every $k > 0$.

Rabin [13] showed that a Turing machine with two work tapes can compute faster than one with only one work tape. Explicitly, he showed that the set $\{uv\gamma u^R | u\epsilon\{a,b\}^*, v\epsilon\{0,1\}^*, \gamma\epsilon\{\alpha,\beta\}$ and $(a_1\ a_2 \ldots a_n)^R = a_n \ldots a_2\ a_1\}$ can be accepted in real time by a Turing machine with two work tapes but not by any Turing machine with only one work tape. In each case, a separate one-way nonwriting tape is used for input.

In [15], Rosenberg studied the class of sets acceptable in realtime by a multitape Turing machine. He labelled these the 'real-time defineable languages' (RTDL). He showed that RTDL is a Boolean algebra: i.e., RTDL is closed under complementation, intersection, and union. Rosenberg also showed other closure properties (both positive and negative) about RTDL: e.g., RTDL is closed under suffixing with a regular set; RTDL is closed under minimization; and RTDL is not closed under concatenation nor under the operation of closure. Finally, Rosenberg compared RTDL to the traditional classes of formal languages, showing that RTDL is incomparable to both the deterministic context free languages and context free languages and that RTDL is properly contained in the class of context sensitive languages.

In [3], Cole showed for any deterministic context free language L, there exists a one dimensional iterative array that accepts L in realtime. He also showed that the language $L = \{XX^R |$ where $(a_1 a_2 \ldots a_n)^R = a_n \ldots a_2 a_1\}$ over an alphabet of at least two letters, is real-time accepted by an iterative array, but is not a deterministic context free language. In the proof of this, Cole exhibited an iterative array which operated in realtime and simulated a deterministic automaton.

The slightly different notion of performing a real-time simulation appeared in Fischer, Meyer and Rosenberg, [6]. In this paper, the authors showed that a Turing machine having a tape with two independent read-write heads can be simulated in realtime by a Turing machine having ten tapes. In this case, Fischer et al. meant that there exists a Turing machine with ten tapes which acts on a step by step basis exactly as a Turing machine with two independent read-write heads.

Hennie [11] was one of the first to exhibit a computation which must take at least $0(\frac{n^2}{(\log n)^2})$ steps. To do so, he studied the 'on-line Turing machine' as a model: "Briefly, in an off-line computation the entire (finite) string of input symbols must be written on one of the machine's tapes prior to the start of the computation, while in an on-line computation the symbols of the input string are presented to the machine sequentially." Using this notion of an 'on-line' Turing machine, Hennie showed that there is a set which can be accepted in $O(n^2)$ steps but not in less $O(\frac{n^2}{(\log n)^2})$ steps. He also showed in [11] that an on-line Turing machine with two-dimensional tapes is faster than an on-line Turing machine with one-dimensional tapes for a particular computation.

Atrubin [2] showed that iterative arrays can multiply two binary integers in real-time. By this, Atrubin meant that one digit from each operand (the low order digits first) would be read on each step and the resulting output is generated immediately after each input step. Thus, Atrubin was using an on-line model.

Atrubin's result should be compared with the results of Cook and Aanderaa [4] in which they showed that binary multiplication takes at least $O(\frac{n \log n}{\log \log n})$ steps on any on-line multitape Turing machine.[2] By on-line, Cook and Aanderaa meant that the i^{th} input digit is available only when the $i\text{-}1^{st}$ output digit is produced.

Finally, in [1] Arbib introduced the sequential functions and related them to on-line machines. A function f is 'sequential' if f can be applied to the initial segments of inputs and the output so produced will always be useable as initial segments of the respective output strings. Arbib simply states that a machine is on-line if it is computing a sequential function.

The real-world sense of a 'real-time' process implies that it is also an 'on-line' process. For set recognition problems, the Hennie and Cook-Aanderaa model of an on-line Turing machine is sufficient. It is interesting to note that for general

[2] In fact, they showed that any complex function takes at least $O(\frac{n \log n}{\log \log n})$ steps on any bounded activity machine. The given result follows immediately.

transductions that only length-preserving transductions can be 'on-line' computable according to the Cook-Aanderaa model. Atrubin bypasses this problem for binary multiplication, which is not length-preserving, by demanding that sufficient zeroes are placed on the high order part of the input string so that all of the output can be produced. Cook and Aanderaa, on the other hand, restrict their attention to length-preserving binary multiplication: i.e., they truncate the output to force the lengths to be equal.

1.2 Outline of Thesis

The purpose of this thesis is to try to model what is meant by an ongoing process and by a real-time process for general transductions. The main body of the thesis can be divided into two parts. The first part consists of chapters 2, 3 and 4 and deals with the problem of modeling an ongoing process. The second part, chapter 5, considers the problem of modeling a real-time process.

Chapter 2 introduces the concept of a sequential Turing machine and considers its computational power. A sequential Turing machine is a Turing machine which computes a sequential function f in such a way that for each initial substring of the input the output produced before more input is read is exactly f applied to the initial substring. The class of sequential Turing machines includes as special cases the on-line Turing machine (in the Cook-Aanderaa sense) and the 'generalized sequential machines' of Ginsburg and Rose [7].

In chapter 3, we consider the effect of permitting an endmarker ϵ_0 for computation by sequential Turing machines.

A function g whose alphabet is extended to include an endmarker ϵ_0 is called an approximation of f if for every input x, $g(x\epsilon_0) = f(x)$. It is shown that for every (computable) function f, there exists a (computable) sequential approximation of f. Furthermore, if a sequential approximation g of f has the property that for each input x the length of g(x) is maximal, g is called a 'maximally defined sequential approximation' of f (m.d.s.a.). Each function is shown to have a unique m.d.s.a.. However, there are computable functions whose m.d.s.a. is not computable.

Two classes of computable functions with computable m.d.s.a.'s are the sequential functions and the 'truly off-line' functions. A 'truly off-line function' f is one for which the m.d.s.a. of f produces no output for any input without an endmarker. Finally, it is shown that there is a computable function with a computable m.d.s.a. which is neither sequential nor truly off-line.

In chapter 4, we study the difference between a function and its m.d.s.a. We introduce a measure, R_f, which is the ratio of the length of the m.d.s.a. to the length of the function for all inputs not including the endmarker. For any function, except the zero function f_λ, which always yields the empty word λ as output, the ratio R_f lies between 0 and 1. For the classes of sequential and truly off-line functions, R_f takes the extremal values 1 and 0, respectively.

Next, we try to characterize the class of functions for which $R_f = 0$ (resp. $R_f = 1$). We show that there are functions which are not truly off-line (resp. sequential) such that $R_f = 0$ ($R_f = 1$). This leads to the introduction of 'almost truly off-line' functions (resp. 'almost sequential' functions).

A function is 'almost truly off-line' if for any x not including the endmarker ϵ_0, the length of its m.d.s.a. at x is bounded by a constant k independent of x. A function is 'almost sequential' if for all x the difference between the lengths of f(x) and its m.d.s.a. at x is bounded by a constant k, independent of x. The class of almost truly off-line functions (resp. almost sequential functions) is shown to be incomparable to the class of functions with $R_f = 0$ ($R_f = 1$).

The class of bounded functions (functions with finite range) is shown to be the intersection of the class of almost sequential functions and the class of almost truly off-line functions. We show that for any rational p, $0 \leq p \leq 1$, there is a bounded function f such that $R_f = p$. We conclude chapter 4 by showing that the hierarchy based on R_f is infinite and dense.

In chapter 5, we consider the effect of the input-output behaviour on the time needed to compute the function. For any strictly increasing function $t: N_+ \to N_+$ (N_+ is the set of positive integers) we introduce five classes of functions: $I_{t(n)}$, $0_{t(n)}$, $M_{t(n)}$, $P_{t(n)}$ and $T_{t(n)}$. $I_{t(n)}$ is the class of functions which can be computed by a Turing machine M so that for any input of length n, the number of steps M takes is bounded by t(n). Similarly, we define $0_{t(n)}$ where n is the length of the output, $M_{t(n)}$ where n is the maximum of the length of the input and the length of the output, $P_{t(n)}$ where n is the length of the output plus the number of input symbols which do not immediately follow an output operation, and $T_{t(n)}$ where n is the sum of the lengths of the input and output. We show that $I_{t(n)}$ and $0_{t(n)}$ are incomparable and that $I_{t(n)} \subsetneq M_{t(n)}$

and $O_{t(n)} \not\subseteq M_{t(n)} \subseteq P_{t(n)} \subseteq T_{t(n)}$.

We next restrict attention to the case when $t(n) = n$ (i.e., the real-time case). It is shown that the above inclusions are proper. We then select a subclass S_n of P_n as the most realistic representative of the notion of 'realtime'. S_n is the intersection of the class of sequential functions and P_n. S_n contains all of the generalized sequential machine mappings, but M_n does not. The well-known class of length preserving real-time functions is shown to be $I_n \cap O_n$ and a proper subclass of S_n.

For the linear-time case when $t(n) = c \cdot n$, for some constant $c>1$, $T_{cn} = M_{cn}$. It is shown that for length-preserving functions, $T_{cn} = I_{cn} \cap O_{cn}$.

Finally, we show that there are bounded functions f such that $f \in I_{n+c}$ and $f \notin I_{n+c-1}$, for every constant $c>1$.

In chapter 6, we study the difference between the computational speeds of off-line and sequential Turing machines. We exhibit a function f such that f takes at least $O(\frac{n^2}{(\log n)^2})$ steps on a sequential Turing machine and at most $O(n \log n)$ on a off-line Turing machine. We also show that for any computation, a sequential Turing machine will take no longer than the length of the input times the computation time of an equivalent off-line Turing machine. We close chapter 6 and the thesis with some suggestions for further research.

CHAPTER 2
SEQUENTIAL TURING MACHINES

2.1 Introduction

In studying the area of machine-based complexity theory, one discovers two basic classes of multitape Turing machines which are studied. It was noticed by Hennie [11], among others, that by restricting the classes of Turing machines to those which have read-only input tapes and the property that the $k+1^{st}$ input digit is available only when the k^{th} output digit is produced, better time bounds for a particular computation could be proven. This restricted class generally is called the class of 'on-line' Turing machines; the class for which no restrictions are applied is called the class of 'off-line' Turing machines.

The purpose of this chapter is to indicate why this author feels that the class of on-line Turing machines is too restrictive and to extend this class to the more natural class of 'sequential Turing machines' without jeopardizing any previously proven result concerning on-line Turing machines.

We first observe that the definition of an on-line Turing machine restricts it to computing a function in a subclass of the class of computable length-preserving functions. Hence, relatively simple computable functions, such as gsm mappings, are not on-line computable. Also, it is shown that there are computable length-preserving functions which are not on-line computable.

From the author's point of view, a more natural restriction of the computing power of Turing machines is to allow computation of only the computable sequential functions discussed in Arbib [1]. Intuitively, the evaluation of a sequential function can proceed in an on-going process:

i.e., f can be applied to initial segments of inputs and the output so produced will always be useable as initial segments of the respective output strings. If f is a sequential function and $f(x) = y$, then for any input string z, $f(x \cdot z) = y \cdot w$, for some output string w. It is shown that all gsm mappings are sequential and that there are some functions which are not sequential.

Finally, sequential functions are related to the 'input-output profile' of a machine computation. Thus, sequential Turing machines are those Turing machines whose input-output profiles define a sequential function. It is shown that for every computable sequential function there exists a sequential Turing machine computing it; also, every on-line Turing machine computes a length-preserving sequential function. Hence, we conclude this chapter by showing the class of on-line Turing machines is a proper subclass of the class of sequentially defined machines.

2.2 On-Line Machines

The word 'machine' will henceforth be used to refer to a Turing machine. However, the concepts introduced in this chapter can be related to any other class of automata which operate on a string input. In this section, our interest is to review the definition of on-line machines, and to indicate why it is felt that definition is too restrictive.

Definition 2.2.1 A machine is called on-line if during its computation each input digit produces exactly one output digit and the $k+1^{st}$ input digit is available only when the k^{th} output digit has been produced.

Notation If a function f is computable on an on-line machine, we say that f is 'on-line computable'.

Remark 2.2.2 On-line computable transductions are length-preserving.

Proof: Obvious. □

The concept of on-line machines has been used by various authors, including Hennie [11], to aid in defining lower bounds for particular computations. Most of these papers were concerned with recognition problems, while we are interested in transductions. As defined by Hennie [11], a machine accepts or rejects the substring already read by producing a 1 or 0, respectively; thus, the machine generates an output string of the same length as the input string. Hence, a recognition problem is a length-preserving transduction which does not require the knowledge of when the end of the input string has been reached. It should easily be seen that every decidable recognition problem is on-line computable. However, the following theorem shows that every transduction is not on-line computable.

Theorem 2.2.3 There exist computable length-preserving transductions which are not on-line computable.

Proof: By definition 2.2.1, an on-line machine requires that for all $n < m$, the n^{th} digit of the output string must be produced before the m^{th} digit of the input string is available. Hence, any computation which demands knowledge of the m^{th} digit of input before producing the n^{th} digit of output, for $n < m$, is not computable on any on-line machine.

Define the string reversal function $Im:\{0,1\}^* \to \{0,1\}^*$ as: $Im(\lambda) = (\lambda)$; $\forall x \in \{0,1\}^*$, $Im(x \cdot 0) = 0 \cdot Im(x)$; $\forall x \in \{0,1\}^*, Im(x \cdot 1) = 1 \cdot Im(x)$.

Im is seen to be computable on no on-line machine since the first output digit cannot be produced until the last input digit is received. □

The common definition of on-line machines yields a proper subclass of the length-preserving computable functions. However, there are other machines which compute in a sufficiently on-going manner to warrant inclusion in any class of functions which purports to characterize the intuitive notion of 'on-line'. One example of such machines, in the opinion of the author, is the class of generalized sequential machines (c.f. Ginsburg and Rose [7]).

<u>Definition 2.2.4</u> A <u>generalized sequential machine</u> (<u>gsm</u>) is a sextuple $S = (Q,\Sigma,\Gamma,\delta,\lambda,q_0)$ where Q,Σ,Γ are finite nonempty sets (the states, input alphabet, and output alphabet, respectively):

$q_0 \in Q$ is the start state;

$\delta : Q \times \Sigma \to Q$ is the next state function;

$\gamma : Q \times \Sigma \to \Gamma^*$ is the output function.

<u>Definition 2.2.5</u> A <u>gsm mapping f</u> is a mapping from Σ^* to Γ^* defined by a gsm $S = (Q,\Sigma,\Gamma,\delta,\lambda,q_0)$ such that:

$$(\forall x \in \Sigma^*)(\forall y \in \Gamma^*)[f(x) = y \text{ iff } \gamma^*(q_0,x) = y].$$

The interpretation of $\delta(q,a) = p$ and $\gamma(q,a) = w$ is that S in state q with input a may enter state p and produce word w. Gsm's are thus simple devices and yet define functions not computable on any on-line machine, as the following theorem shows.

<u>Theorem 2.2.6</u> There exist gsm mappings which are not on-line computable.

<u>Proof</u>: We exhibit a one state gsm which generates a non-length-preserving function.

Define $S = (\{q\},\{0,1\},\{a\},\delta,\lambda,q)$ where:

$\delta(q,0) = q = \delta(q,1)$;

$\gamma(q,0) = \lambda$ (the empty string);

$\gamma(q,1) = a$.

Then, f the gsm mapping defined by S is:

$$f(\lambda) = \lambda,\ (\forall x \in \Sigma^*)[f(x\cdot 0) = f(x)\ \&\ f(x\cdot 1) = f(x)\cdot a].$$

f is obviously not length-preserving and hence is not on-line computable. □

The on-going nature of gsm mappings leads us to reject the technical definition (2.2.1) of on-line computability as not completely capturing the intuitive notion. However, this attribute of gsm's does not depend upon the fact that they have finite state memories. Therefore, definition 2.2.1 seems too restrictive. The next three sections are an attempt to extend the notion of 'on-line' to fit our intuition more closely.

2.3 Sequential Functions

Intuitively, an on-going process is one which produces an output for each input digit without worrying about any input that follows. Thus, previous inputs strongly affect the output generated by a specific input digit, but following inputs should have no effect on the output. This nature is what we are attempting to formalize. We will find it useful to shift primary attention to functions, rather than machines in this section. We consider a class of functions introduced in Arbib [1].

Definition 2.3.1 For any function $f:\Sigma^* \to \Gamma^*$ and for each string $u \in \Sigma^*$, if there exists a unique function $f_u:\Sigma^* \to \Gamma^*$, such that $(\forall v \in \Sigma^*)[f(u \cdot v) = f(u) \cdot f_u(v)]$, then f is called sequential.

Notation We will denote by Δf the function which is formed from f and the related f_u by the following rules: $\Delta f(\lambda) = f(\lambda)$; and

$$(\forall u \in \Sigma^*)(\forall a \in \Sigma)[\Delta f(ua) = f_u(a)].$$

In essence, in the computation of $f(a_1a_2 \ldots a_i a_{i+1} \ldots a_n)$, $\Delta f(a_1a_2 \ldots a_i)$ denotes the output string generated during the time a_i is the most recent symbol which has been read.

From the discussion in section 2.2, it is likely that the class of functions we are seeking will contain the gsm mappings, but need not contain all of the length-preserving functions. The class of sequential functions has these properties as will be shown in the next two theorems.

Theorem 2.3.2 There exist length-preserving functions which are not sequential.

Proof: Let $\Sigma = \Gamma = \{0,1\}$, and define $\mathrm{Im}:\Sigma^* \to \Gamma^*$ as in Theorem 2.2.3. We will show Im is not sequential.

By definition, $\mathrm{Im}(1) = 1$, $\mathrm{Im}(10) = 01$. Hence, $\mathrm{Im}(10) \neq \mathrm{Im}(1)\cdot v$, for any $v \in \Gamma^*$. Hence, there is no way to define $\mathrm{Im}_1(0)$ and Im is not a sequential function. □

Notation: For $u,v \in \Sigma^*$, the left quotient of v by u, $u\backslash v = \begin{cases} w \text{ if } v = uw,\ w \in \Sigma^* \\ \text{undefined otherwise.} \end{cases}$

Theorem 2.3.3 The class of sequential functions properly contains the class of gsm mappings.

Proof: Consider any gsm mapping f and the gsm $S = (Q,\Sigma,\Gamma,\delta,\lambda,q_0)$ defining f. Given any input $u \in \Sigma^*$, we define f_u as the function defined by the gsm $S' = (Q,\Sigma,\Gamma,\delta,\lambda,\delta^*(q_0,u))$. Thus, for any $v \in \Sigma^*$, $f(u\cdot v) = \lambda^*(q_0,u\cdot v) = \lambda^*(q_0,u)\cdot\lambda^*(\delta^*(q_0,u),v) = f(u)\cdot f_u(v)$. Therefore, $f(u\cdot v) = f(u)\cdot f_u(v)$, and f is sequential.

We now must show that there is a sequential function which is not a gsm mapping. First, we note that if a gsm outputs exactly one letter from its output alphabet per step, the machine is a finite automaton. Now consider length-preserving binary multiplication*, reading low-order digits on the left. i.e. $f_\ell^\times:\Sigma^* \to \{0,1\}^*$, where $\Sigma = \{\binom{x}{y} \mid x,y \in \{0,1\}^*\}$ as in [4].

$$\text{e.g.}\quad f_\ell^\times(\tbinom{1011}{1001}) = 1010,\quad f_\ell^\times(\tbinom{10011}{10110}) = 10100.$$

For $u \in \Sigma^*$, we define a function $f_{\ell_u}^\times:\Sigma^* \to \{0,1\}^*$ as $(\forall v \in \Sigma^*)[f_{\ell_u}^\times(v) = f_\ell^\times(u)\backslash f_\ell^\times(uv)]$. As long as $f_\ell^\times(uv) = f_\ell^\times(u)\cdot w$, for some $w \in \{0,1\}^*$, then $f_\ell^\times(uv) = f_\ell^\times(u)\cdot f_{\ell_u}^\times(v)$, by definition of left quotient

*Length-preserving binary multiplication is defined as the n low-order digits of the product of two n digit binary numbers.

and $f_\ell^\times$ will be sequential. We know that in binary multiplication only the n low order digits of the input words affect the n low order digits of the output, and hence $f_\ell^\times$ is sequential. Minsky showed that f is computable on no finite automaton in [12]. We reproduce the proof for completeness.

Suppose a finite automaton $M = (Q,\Sigma,\Gamma,\delta,\lambda,s)$, **(where** $|Q| = k$, $\Sigma = \{\binom{i}{j} | i,j \in \{0,1\}\}$, $\Gamma = \{0,1\}$) computes $f_\ell^\times$.

Consider $(2^i+2^{2i})^2 = 2^{2i} + 2^{3i+1} + 2^{4i}$. Hence $f_\ell^\times(\binom{0^i10^{i-1}1}{0^i10^{i-1}1}) =$ $= 0^{2i}1$ (reading the lower order digits from left to right).

Let $i > k + 1$. After reading $\binom{0^i1}{0^i1}$, M will be in some state q **and will have produced (i+1) zeroes. For each of the next (i-1) pairs of zeroes, M produces a zero output. After receiving these input paris, M will** be in some state q_1. Hence, $\lambda(q_1,\binom{1}{1}) = 1$. However, since $i - 1 > k$, one state must have been entered at least twice. Thus, there exists a $j < i - 1$, such that $q_1 = \delta^*(q,\binom{0^j}{0^j}) = \delta^*(\binom{0^{i-1}}{0^{i-1}})$ (remove the string of pairs of zeros **which causes M to go from the repeated state to the repeated state).**

$$\text{Hence,}\quad \lambda^*(q,\binom{0^j1}{0^j1}) = \lambda^*(q,\binom{0^j}{0^j})\cdot\lambda(q_1,\binom{1}{1})$$
$$= 0^j1.$$
$$\therefore \quad \lambda^*(s,\binom{0^i10^j1}{0^i10^j1}) = 0^{i+1}0^j1 = 0^{i+j+1}1.$$
$$\text{Now,}\quad (2^i+2^{i+j+1})^2 = 2^{2i} + 2^{2i+2j+2} + 2^{2i+2j+2}$$
$$= 2^{2i}(1+2^{j+2}+2^{2j+2})$$
$$\geq 2^{i+j+2}(1+2^{j+2}+2^{2j+2}) \qquad (\text{as } i \geq j + 2).$$
$$\therefore \quad f_\times^\ell(\binom{0^i10^j1}{0^i10^j1}) = 0^{i+j+2} \neq \lambda^*(s,\binom{0^i10^j1}{0^i10^j1}),$$

a contradiction to M computing $f_\ell^\times$.

Hence, $f_\ell^\times$ is a sequential function, but not a gsm mapping. Hence, the class of gsm mappings is a proper subclass of the class of **sequential functions.** □

Although length-preserving binary multiplication is sequential, **the following remark shows that full binary multiplication is not.**

Remark Let $\Sigma = \{\binom{0}{0},\binom{0}{1},\binom{1}{0},\binom{1}{1}\}$, $\Gamma = \{0,1\}$.

Let $\binom{x}{y}$ denote the string $\binom{a_1}{b_1}\binom{a_2}{b_2}\ldots\binom{a_n}{b_n}$, where $a_i, b_i \in \{0,1\}$, and $x = a_1a_2\ldots a_n$, $y = b_1b_2\ldots b_n$.

Define $f^\times:\Sigma^* \to \Gamma^*$ as $f^\times(\binom{x}{y}) = x \times y$ (that is, normal binary multiplication reading the input from lower order on the left to high order on the right).

For example, $f^\times(\binom{011}{001}) = 00011$.

Now consider $f^\times(\binom{01}{11})$ and $f^\times(\binom{011}{111})$.

$$f^\times(\binom{01}{11}) = 011 \text{ and } f^\times(\binom{011}{111}) = 010101.$$

Thus, $f^\times(\binom{011}{111}) \neq f^\times(\binom{01}{11})\cdot u$ for any $u \in \Gamma^*$. Hence full binary multiplication is not a sequential function. $\square$

We now give a simple characterization of a sequential function.

Notation For $u,v \in \Sigma^*$, uBv (u begins v) is written to mean $(\exists w \in \Sigma^*)[uw = v]$.

Theorem 2.3.4 A function $f:\Sigma^* \to \Gamma^*$ is sequential if and only if $(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(u)Bf(uv)]$.

Proof: Suppose f is sequential. By definition 2.3.1, for every $u \in \Sigma^*$, there is a function f_u such that $(\forall v \in \Sigma^*)[f(uv) = f(u)\cdot f_u(v)]$. Hence, we can conclude that $(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(u)Bf(u)\cdot f_u(v) = f(uv)]$.

Now, suppose that $(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(u)Bf(uv)]$. We define $f_u:\Sigma^* \to \Gamma^*$ for any $u \in \Sigma^*$, as $(\forall v \in \Sigma^*)[f_u(v) = f(u)\backslash f(uv)]$. Since $f(u)Bf(uv)$, the left quotient will be defined; i.e. $f(uv) = f(u)\cdot w$ for some $w \in \Gamma^*$. Hence,

$(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(uv) = f(u)\cdot f_u(v)]$, and f is sequential. $\square$

We will generally find it more convenient to use theorem 2.3.4 when proving that a function is sequential rather than definition 2.2.1. We end this section by exhibiting a closure property of sequential functions.

Theorem 2.3.5 Let $g:\Sigma^* \to \Gamma^*$ and $f:\Gamma^* \to \Delta^*$ be sequential functions. Then $h = f \circ g$, $h:\Sigma^* \to \Delta^*$, is a sequential function.

Proof: Since g is sequential, then for any nonempty input words u and v, $g(u \cdot v) = g(u) \cdot g_u(v)$.

Thus, $f \circ g(u \cdot v) = f(g(u) \cdot g_u(v))$.

Let $w_1 = g(u)$, $w_2 = g_u(v)$, $w_1, w_2 \in \Gamma^*$.

Since f is sequential, then, for any input words over Γ^*, $f(w_1 \cdot w_2) = f(w_1) \cdot f_{w_1}(w_2)$.

Therefore, for $u, v \in \Sigma^*$:

$$\begin{aligned} h(u \cdot v) &= f \circ g(u \cdot v) = f(g(u) \cdot g_u(v)) \\ &= f \circ g(u) \cdot f_{g(u)}(g_u(v)) \\ &= h(u) \cdot h_u(v), \quad \text{where } h_u(x) \text{ is defined as } f_{g(u)}(x). \end{aligned}$$

Therefore, h is sequential. □

Having given an indication of the power and the limitation of sequential functions, attention is turned to the construction of machines which compute sequential functions. Arbib [1] defines the class of machines as the class computing sequential functions; instead, we use the relationship of input and output implied in the definition of sequential functions.

2.4 Sequential Profiles and Input-Output Profiles

By studying the structure of the sequential function, we notice that there is an inherent breakup of the output string by the input string. This breakup reminds one of IO ~~profiles,~~ [6].

Definition 2.4.1 For a sequential function $f:\Sigma^* \rightarrow \Gamma^*$ and input word $x = a_1a_2 \ldots a_n$ $(a_i \in \Sigma)$ the sequential profile of f with input x is the string $P_s(f,x) = \Delta f(\lambda)a_1 \; \Delta f(a_1)a_2 \; \Delta f(a_1a_2) \ldots a_n \; \Delta f(a_1a_2 \ldots a_n)$. (See section 2.3 for notation.)

Intuitively, the sequential profile of a function f and input word x is the string formed from the input word and output word, by placing the input letter and the output newly determined by it in order.

Definition 2.4.2 An input-output (IO) profile [6] of a machine computation, given the input word $x = a_1a_2 \ldots a_n$ $(a_i \in \Sigma)$ is the string

$P_{IO}(M,x) = y_0a_1y_1a_2y_2 \ldots a_ny_n$, where $y_i \in \Gamma^*$ and y_i is defined as the string of output generated by machine M between inputs a_i and a_{i+1} $(a_0 = a_{n+1} = \lambda)$. (Note that this means that the last letter of y_i will be written on the same machine step as the reading of a_{i+1}.)

Intuitively, the IO profile is the only externally recognizable description of a computation. It should be obvious how the two concepts of IO and sequential profiles are related to form the concept of a 'sequential Turing machine'.

Definition 2.4.3 A (Turing) machine M computing a sequential function $f:\Sigma^* \rightarrow \Gamma^*$ is called a sequential Turing machine if $(\forall x \in \Sigma^*)[P_s(f,x) = P_{IO}(M,x)]$.

Essentially a sequential Turing machine writes all of the output defined by the input received at a given point in time before reading additional inputs (we make the convention that it halts the first time it attempts to read after the input has been exhausted). It should be noted that it is the sequential relationship between the input and output, not the sequential (i.e. step by step) nature of the operation of the control unit, which makes a Turing machine a sequential Turing machine.

We can generalize our definition to any other machine model by replacing the word 'Turing' by the name of the model desired.

For example, a random access machine R computing a sequential function $f:\Sigma^* \to \Gamma^*$ is called a sequential random access machine if $(\forall x \in \Sigma^*)[P_s(f,x) = P_{IO}(R,x)]$.

2.5 Sequential Turing Machines and Sequential Functions

First, we will show formally that every computable sequential function is computable on some sequential Turing machine.

Theorem 2.5.1 If $f:\Sigma^* \to \Gamma^*$ is a computable sequential function, there exists a sequential Turing machine which computes f.

Proof: Since f is a computable function, there exists a Turing machine M with one work tape which computes f. Without loss of generality we may assume that it has an input tape with endmarkers ϵ_3 and ϵ_4 on the left and right, respectively. We construct a sequential Turing machine T with **4 work tapes with M as a submachine.**

At stage 0: **T uses M to compute $f(\lambda)$ using the first tape and writes** the result, y_0, on the output tape. It also writes $\epsilon_1\, y_0\, \epsilon_2$ on the third work tape, where ϵ_1, ϵ_2 are special markers. T also places a marker ϵ_3 on the second tape, and positions the read head of the second tape to the right of this marker. T then clears the first tape.

At stage i: T reads the i^{th} input letter, a_i, and puts it on the second tape, followed on the right by a marker ϵ_4. T rewinds the second tape to the position one to the right of the marker ϵ_3. T lets its submachine, M, compute f on the input on the second tape using the first tape. T writes the result on the fourth tape surrounded by markers ϵ_5 and ϵ_6, on the left and right respectively. After M is finished (by reading the endmarker ϵ_4, and having completed the computation), T rewinds the third and fourth tapes to the markers ϵ_1 and ϵ_5, respectively. T reads along both letter by letter until ϵ_2 on the third tape is reached. T then copies the letters on the fourth tape from the one at the same spot as the ϵ_2, until ϵ_6 is read, onto the output tape and the third tape replacing the ϵ_2 at the end. T next assures that the first tape is clear and the read head of the second tape is over the marker ϵ_4.

T halts when no further input is available.

Now, we show T is the machine we desire. For the empty input, T outputs $f(\lambda) = \Delta f(\lambda)$, by construction. Now, suppose T has output the string $f(a_1a_2...a_{n-1})$ after stage n-1. At stage n, T outputs the string $\Delta f(a_1a_2...a_n)$, as f is sequential and hence $f(a_1a_2...a_n) = f(a_1a_2...a_{n-1})\Delta f(a_1a_2...a_n)$. Hence, by induction T computes f. Also, the IO profile of T for any input $x = a_1a_2...a_n$ is $\Delta f(\lambda)a_1\Delta f(a_1)a_2...a_n\Delta f(a_1a_2...a_n)$, which is the sequential profile of f with input x. Hence, T is a sequential Turing machine which computes f. □

Corollary 2.5.2 The class of generalized sequential machines is a proper subclass of the class of sequential Turing machines.

Proof: Immediate from theorems 2.3.3 and 2.5.1. □

We finally reach the point where we can prove easily that the class of sequential Turing machines extends the class of on-line machines without jeopardizing any previously proven result.

Theorem 2.5.3 The class of on-line machines is exactly the class of sequential Turing machines which compute length-preserving sequential functions.

Proof: **Let M be a sequential** Turing machine which computes a length-preserving sequential function $f:\Sigma^* \to \Gamma^*$. Since f is a length-preserving sequential function, then the sequential profile of f with input $x = a_1a_2\ldots a_n$, $a_i \in \Sigma$, is $P_s(f,x) = \Delta f(\lambda)a_1\Delta f(a_1)a_2\ldots a_n\Delta f(a_1a_2\ldots a_n)$ where $\Delta f(\lambda) = \lambda$ and $\Delta f(a_1a_2\ldots a_n) = b_i$, $b_i \in \Gamma$ for all $1 \le i \le n$. Since M is a sequential Turing machine, the IO profile of M with input x must be $P_{IO}(M,x) = a_1b_1a_2b_2\ldots a_nb_n$, where a_i and b_i are as in $P_s(f,x)$. Hence, M has the property that each input digit produces exactly one output digit and the $k+1^{st}$ input digit is read only when the k^{th} output digit has been produced. Hence, by definition, M is an on-line machine.

Now, suppose M is an on-line machine computing a function $f:\Sigma^* \to \Gamma^*$. By definition, f must be a length-preserving function. Suppose **f is not a sequential function. Hence, there exists a $u \in \Sigma^*$, such that for all functions g, there exists a $v \in \Sigma^+$ such that $f(u\cdot v) \neq f(u)\cdot g(v)$.** That is, $(\exists u \in \Sigma^*)(\forall w \in \Gamma^*)(\exists v \in \Sigma^+)[f(u\cdot v) \neq f(u)\cdot w]$; otherwise we could define a function g such that $g(v) = w$. Let $u = a_1a_2\ldots a_n$, $v = a_{n+1}a_{n+2}\ldots a_m$. The IO profile of M with input $u\cdot v$ is

$$P_{IO}(M,u\cdot v) = a_1b_1a_2b_2\ldots a_nb_na_{n+1}b_{n+1}\ldots a_mb_m,$$

where $b_i \in \Gamma$, $1 \le i \le m$, since M is an on-line machine. Thus, the output generated for input uv is the string $b_1b_2\ldots b_nb_{n+1}b_{n+2}\ldots b_m$. The IO profile of M with input u is $P_{IO}(M,u) = a_1b_1a_2b_2\ldots a_nb_n$. Thus, the output generated for input u is the string $b_1b_2\ldots b_n$. Since M computes f, $f(u) = b_1b_2\ldots b_n$ and $f(u\cdot v) = b_1b_2\ldots b_nb_{n+1}\ldots b_m$. But therefore, $f(u\cdot v) = f(u)\cdot b_{n+1}\ldots b_m$, a contradiction. Hence f must be sequential.

Now, we must show that for any input x the IO profile of M with input x and the sequential profile of f with input x are the same. Suppose $x = \lambda$. Since f is length-preserving, $f(\lambda) = \lambda$. Since

M computes f, M outputs the empty string when no input is available. Hence, $P_{IO}(M,\lambda) = \lambda = P_s(f,\lambda)$. Now, suppose $P_{IO}(M,a_1a_2\cdots a_j) = P_s(f,a_1a_2\cdots a_j) = a_1b_1a_2b_2\cdots a_jb_j$, where $b_i = \Delta f(a_1a_2\cdots a_i)$ and $b_i \in \Gamma$, $1 \leq i \leq j$. Since f is sequential, $f(a_1a_2\cdots a_ja_{j+1}) = f(a_1a_2\cdots a_j)\Delta f(a_1a_2\cdots a_ja_{j+1}) = b_1b_2b_3\cdots b_jb_{j+1}$. Thus, $P_s(f,a_1a_2\cdots a_ja_{j+1}) = a_1b_1a_2b_2\cdots a_jb_ja_{j+1}b_{j+1}$. Since M computes f, then it must output $b_1b_2\cdots b_jb_{j+1}$ for the input $a_1a_2\cdots a_ja_{j+1}$. However, M has outputted the string $b_1b_2\cdots b_j$ before reading the input a_{j+1}. Hence, b_{j+1} is the only output following the reading of a_{j+1}. Hence, $P_{IO}(M,a_1a_2\cdots a_ja_{j+1}) = a_1b_1\cdots a_jb_ja_{j+1}b_{j+1} = P_s(f,a_1a_2\cdots a_ja_{j+1})$. By induction, $(\forall x \in \Sigma^*)[P_{IO}(M,x) = P_s(f,x)]$. **Therefore, if M is an on-line machine, M is a sequential Turing machine which computes a length-preserving function.** □

<u>Corollary 2.5.4</u> The class of on-line machines is a proper subclass of the class of sequential Turing machines.

<u>Proof</u>: Immediate from theorem 2.5.3, corollary 2.5.2 and the fact that some gsm mappings are not length-preserving. □

Notice that for any on-line computable function f, the class of on line machines computing f is the class of sequential Turing machines computing f. It should also be noted that the class of seuqential Turing machines can be directly associated with the class of computing sequential functions. Hence, facts concerning the class of sequential Turing machines can be couched in terms of the class of computable sequential functions and be proven using functions instead of machines. This approach is used in the next chapter to study the problem of extending the power of sequential Turing machines.

CHAPTER 3

SEQUENTIAL APPROXIMATIONS

3.1 Introduction

In chapter 2, it was shown that there are some computable functions which are not sequential and hence cannot be computed on any sequential Turing machine. The purpose of this chapter is to introduce the concept of the approximation of one function by another and to study the effects of this concept.

An endmarker ϵ_0 is added to the input alphabet and a function g is said to be an 'approximation' of a function f, if for any input x, $g(x\epsilon_0) = f(x)$. We will show that for any function f, one can find a sequential function g which is an approximation of f. Furthermore, for any computable function, there exists a sequential Turing machine which computes an approximation of f. Thus, by restricting the class of machines to the class of sequential Turing machines little computational power is lost.

The obvious sequential approximation defined for each functions turns out to be 'worst' in the sense that the output which is generated for any input before the receipt of the input endmarker, is the empty string. We next formalize what is meant by a 'best' approximation of a function f as a 'maximally defined sequential approximation', abbreviated m.d.s.a., of f. Intuitively, this is a function which generates at least as much output for any input as any other sequential approximation of f. Alternatively, it is shown that a m.d.s.a. of f is one for which the amount of output generated for any input x is the longest substring of $f(x)$ which begins every string $f(x\cdot v)$, for any nonempty input v. It is then shown that for any function f, there exists a unique m.d.s.a. of f. However, it is shown that there is no effective procedure for determining what the unique m.d.s.a. of a function is; indeed, there is a computable function having a non-computable m.d.s.a.

The concept of maximally defined sequential approximations leads to the consideration of two interesting classes of functions: the class of sequential functions and the class of 'truly off-line' functions. Denote the m.d.s.a. of a function f as $\hat{f}$. A function f is sequential if and only if for all input strings x, not containing the endmarker ϵ_0, $\hat{f}(x) = f(x)$. The opposite class of functions, when $\hat{f}(x) = \lambda$ for all inputs x not containing ϵ_0, is the class of 'truly off-line' functions. The string reversal function is an example of a truly off-line function.

A function may be neither sequential nor truly off-line and still have a computable m.d.s.a. We are thus led to investigate methods of measuring a function in relation to its m.d.s.a. in the next chapter.

3.2 Approximations of Functions

At times, it has been suggested that by adding an endmarker to the input alphabet and demanding output only when the endmarker is read, we can show that every computable function is on-line computable. In view of definition 2.2.1, this stratagem is not applicable. However, by replacing the phrase 'on-line computable' by 'a computable sequential function', this intuitive approach will be valid.

We first must formalize what we mean by adding an endmarker in function notation.

Definition 3.2.1 A function $f:\Sigma^* \to \Gamma^*$ is said to be final w.r.t. ϵ_0

if $\epsilon_0 \in \Sigma$ and $(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(u\epsilon_0) = f(u\epsilon_0 v)]$.

In essence, ϵ_0 is an input endmarker: once it has been reached, no further output is generated. What we would like to show is that for every computable f, there exists a sequential Turing machine which uses an input endmarker and computes f. However, a machine using endmarkers is not computing a function which has no endmarkers in its domain; the machine is computing a similar function over an extended domain. We will call this function an 'approximation' of the original function.

Definition 3.2.2 A function $g:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ is an approximation of $f:\Sigma^* \to \Gamma^*$ if $\epsilon_0 \notin \Sigma$, and $(\forall u \in \Sigma^*)(\forall v \in (\Sigma \cup \{\epsilon_0\})^*)[g(u\epsilon_0 v) = f(u)]$.

Remark Clearly, if g is an approximation of f, g is final w.r.t. ϵ_0.

We now show that the suggestion made in the introduction of this section is valid.

Theorem 3.2.3 For any function $f:\Sigma^* \to \Gamma^*$ and $\epsilon_0 \notin \Sigma$, there exists a sequential function $g:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$, which is an approximation of f.

Proof: We define g by cases depending on whether ϵ_0 is in the input string or not:

1) for $x \in \Sigma^*$, $g(x) = \lambda$;

2) for $x = u\epsilon_0 w$, where $u \in \Sigma^*$, $w \in (\Sigma \cup \{\epsilon_0\})^*$, $g(x) = f(u)$.

By (2), $(\forall u \in \Sigma^*)(\forall w \in (\Sigma \cup \{\epsilon_0\})^*)[g(u\epsilon_0 w) = f(u)]$. Hence, g is an approximation of f. Thus, we need only show that g is sequential.

We proceed by cases to show that $(\forall u \in (\Sigma \cup \{\epsilon_0\})^*)$ $(\forall v \in (\Sigma \cup \{\epsilon_0\})^*)[g(u)Bg(uv)]$ and hence by lemma 2.3.4, g is sequential.

a) For $u \in \Sigma^*$, $g(u) = \lambda$ by (1).

Hence, $g(u)Bg(uv)$, since $(\forall w \in \Gamma^*)[\lambda Bw]$.

b) For $u = x\epsilon_0 w$, $x \in \Sigma^*$, $w \in (\Sigma \cup \{\epsilon_0\})^*$, $g(u) = f(x)$, by (2).

Now, $(\forall v \in (\Sigma \cup \{\epsilon_0\})^*)[uv = x\epsilon_0 wv]$; thus, $g(uv) = g(x\epsilon_0 wv) =$
$= f(x) = g(u)$.

Clearly, $g(u)Bg(u)$; thus $g(uv)Bg(uv) = g(u)$. Hence, by definition g is a sequential approximation of f. □

<u>Corollary 3.2.4</u> For every computable function $f:\Sigma^* \to \Gamma^*$, there exists a sequential Turing machine which computes an approximation of f.

<u>Proof</u>: By theorem 3.2.3, for every function f, there exists a sequential function g which is an approximation of f. Notice that if f is computable, then g is also computable. Hence, by theorem 2.5.1, there exists a sequential Turing machine which computes g. □

3.3 Maximally Defined Sequential Approximations

The preliminary goal of this chapter has been achieved by theorem 3.2.3. However, upon studying the approximation constructed, one discovers that it is the 'worst' sequential approximation possible. For example, consider any sequential function $f:\Sigma^* \to \Gamma^*$, $\epsilon_0 \notin \Sigma$. Define $g:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ as $(\forall x \in \Sigma^*)[g(x) = f(x)]$ and $(\forall x \in \Sigma^*)(\forall w \in (\Sigma \cup \{\epsilon_0\})^*)[g(x\epsilon_0 w) = g(x)]$. The function g is a sequential approximation of f. Also, for any input string not containing the endmarker, the amount of output for g is always greater than or equal to the amount for the function guaranteed by the construction in the proof of theorem 3.2.3. Hence, g is a 'better' approximation of f. The following definition formalizes what is meant by a 'best' approximation.

<u>Definition 3.3.1</u> A sequential function $g:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ is a <u>maximally defined sequential approximation</u> (m.d.s.a.) of $f:\Sigma^* \to \Gamma^*$ if

g is an approximation of f and for all other sequential approximations g' of f, $(\forall x \in \Sigma^*)[g'(x) B g(x)]$.

Thus, a sequential function g is a m.d.s.a. of f if the length of $g(x)$ for all input strings x is greater than or equal to the length of any other sequential approximation of f on the same input string x.

Using definition 3.3.1 to prove that a function is a m.d.s.a., one would have to test whether any other sequential approximation of the same function has an output string of greater length. This would be tedious, if not impossible. The following theorem offers a more useful test of whether a function is a m.d.s.a., as only the function being approximated is needed.

Theorem 3.3.2 A sequential function $g:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ is a maximally defined sequential approximation of $f:\Sigma^* \to \Gamma^*$ if and only if g is an approximation of f and $(\forall u \in \Sigma^*)(\forall b \in \Gamma)(\exists v \in \Sigma^*)[g(u)\cdot b \not B f(u \cdot v)]$.

Proof: Suppose g is a m.d.s.a. of f; by definition, g is sequential, final w.r.t. ϵ_0 and an approximation of f. Suppose that $(\exists u \in \Sigma^*)(\exists b \in \Gamma)(\forall v \in \Sigma^*)[g(u)\cdot b B f(u \cdot v)]$. Define a new function $g':(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ from g as follows:

1) for $y \in (\Sigma \cup \{\epsilon_0\})^*$ such that $u \not B y, g'(y) = g(y)$;

2) for $y \in \Sigma \cup \{\epsilon_0\})^*$ such that $u B y, g'(y) = \begin{cases} g(y) & \text{if } |g(y)| > |g(u)| \\ g(u)\cdot b, & \text{otherwise.} \end{cases}$

We will show that g' is an approximation of f by showing that $(\forall v \in \Sigma^*)(\forall w \in (\Sigma \cup \{\epsilon_0\})^*)[g'(v\epsilon_0 w) = g(v\epsilon_0 w)]$. Since g is an approximation of f, we know $g(v\epsilon_0 w) = f(v)$ and hence we will have that g' is an approximation of f. If $u \not B v\epsilon_0 w$, then $g'(v\epsilon_0 w) =$

$= g(v\epsilon_0 w)$, by (1). Now suppose $uBv\epsilon_0 w$; since $u \in \Sigma^*$, there must exist an $x \in \Sigma^*$ such that $v = ux$. Now, since g is an approximation of f, $g(v\epsilon_0 w) = f(v) = f(ux)$. Hence, $|g(v\epsilon_0 w)| = |f(v)| > |g(u)|$, by the assumption that $g(u)\cdot bBf(ux)$, for any x. Thus, by (2), $g'(v\epsilon_0 w) = g(v\epsilon_0 w)$. Hence, g' is an approximation of f.

We now show that g' is sequential by considering the two cases for $y \in (\Sigma \cup \{\epsilon_0\})^*$.

a) $u\not{B}y$. Thus, $g'(y) = g(y)$, by (1). Now, for $v \in (\Sigma \cup \{\epsilon_0\})^*$, $g'(yv)$ could be either of two values depending on whether $uByv$ or not.

i) If $u\not{B}yv$, $g'(yv) = g(yv)$ by (1). Since g is sequential, $g(y)Bg(yv)$; hence $g'(y) = g(y)Bg(yv) = g'(yv)$.

ii) If $uByv$, then yBu since $u\not{B}y$. Since g is sequential, $g(y)Bg(u)$ and $g(u)Bg(yv)$. By (2), $g'(yv)$ is either $g(yv)$ or $g(u)\cdot b$. But $g'(y) = g(y)Bg(u)Bg(yv)$. Hence, $g'(y)Bg(u)\cdot b$ and $g'(y)Bg(yv)$. Thus, in either case, $g'(y)Bg'(yv)$.

b) uBy and for $v \in (\Sigma \cup \{\epsilon_0\})^*$, we obtain three subcases.

i) $|g(y)| > |g(u)|$. Hence, by (2), $g'(y) = g(y)$. But since g is sequential, $g(y)Bg(yv)$ and $|g(yv)| > |g(u)|$. Thus, by (2), $g'(yv) = g(yv)$. Hence, $g'(y) = g(y)Bg(yv) = g'(yv)$.

ii) $|g(y)| \leq |g(u)|$ and $|g(yv)| \leq |g(u)|$. Hence, $g'(y) = g(u)\cdot b$ and $g'(yv) = g(u)\cdot b$, by (2). Therefore, $g'(y)Bg'(yv)$.

iii) $|g(y)| \leq |g(u)|$ and $|g(yv)| > |g(u)|$. Hence, $g'(y) = g(u)\cdot b$ and $g'(yv) = g(yv)$. Since g is a sequential approximation of f, $g(yv)Bg(yv\epsilon_0) = f(yv)$. Since uBy, for some $w \in \Sigma^*$, $y = uw$. Thus, $g(u)\cdot bBf(yv) = f(uwv)$, using the assumption. Summarizing, $g(yv)Bf(yv)$ and $g(u)\cdot bBf(yv)$. But, $|g(yv)| > |g(u)|$, so $|g(yv)| \geq |g(u)\cdot b|$, and we may conclude that $g(u)\cdot bBg(yv)$, yielding $g'(y) = g(u)\cdot bBg(yv) = g'(yv)$.

Hence, g' is a sequential approximation of f; however, $g'(u) = g(u)\cdot b\not{B}g(u)$, a contradiction, as g was assumed to be a m.d.s.a. of f. This concludes the only if half of the proof.

Now suppose a sequential function $g:(\Sigma \cup \{\epsilon_0\}^* \to \Gamma^*$ is not a m.d.s.a. of $f:\Sigma^* \to \Gamma^*$, but g is a sequential approximation of f.

Since g is not a m.d.s.a. of f, there exists a sequential approximation g' of f for which $(\exists u \in \Sigma^*)[g'(u)\not{B}g(u)]$.

Because g' is an approximation of f, $g'(u\epsilon_0) = f(u)$. Since g' is a sequential approximation of f, $(\forall x \in \Sigma^*)[g'(x)Bg'(x\epsilon_0) = f(x)]$. Hence, $g'(u)Bf(u)$ and $(\forall v \in \Sigma^*)[g'(uv)Bf(uv)]$. Also, since g' is sequential, $(\forall v \in \Sigma^*)[g'(u)Bg'(uv)]$. Thus, $(\forall v \in \Sigma^*)[g'(u)Bf(uv)]$.

Now, since g is a sequential approximation of f, $g(u)Bf(u)$. But because $g'(u)Bf(u)$, either $g'(u)Bg(u)$ or $g(u)Bg'(u)$. By assumption, $g'(u)\not{B}g(u)$; thus, $g'(u) \neq g(u)$ and $g(u)Bg'(u)$. Hence, $(\exists b \in \Gamma)(\exists y \in \Gamma^*)[g'(u) = g(u)\cdot b\cdot y]$. It follows that $g(u)\cdot bBg'(u)$. And since $g'(u)Bf(uv)$, for all $v \in \Sigma^*$, we obtain $(\forall v \in \Sigma^*)[g(u)\cdot bBf(uv)]$. Thus, if g is a sequential approximation of f, but not a m.d.s.a. of f, then $(\exists u \in \Sigma^*)(\exists b \in \Gamma)(\forall v \in \Sigma^*)[g(u)\cdot bBf(uv)]$, concluding the proof of the theorem. □

Theorem 3.3.2 gives a useful characterization of a m.d.s.a. However, it does not indicate when or under what circumstances a m.d.s.a. is obtainable. Theorem 3.3.3 asserts the existence of a m.d.s.a. for every function, while theorem 3.3.4 shows that there can be no uniform algorithm for constructing it.

<u>Theorem 3.3.3</u> For every function $f:\Sigma^* \to \Gamma^*$, there exists a maximally defined sequential approximation of f.

<u>Proof</u>: For any $f:\Sigma^* \to \Gamma^*$, define a function $g:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ as follows:

1) For $x = u\epsilon_0 v$, where $u \in \Sigma^*$, $v \in (\Sigma \cup \{\epsilon_0\})^*$, $g(x) = f(u)$;

2) For $x \in \Sigma^*$, $g(x) = y$, where y is the longest initial substring of $f(x)$ such that $(\forall w \in \Sigma^*)[yBf(xw)]$. Such a y must exist since at least one substring of $f(x)$, namely λ, satisfies the requirement that $yBf(xw)$. Note that we claim only the existence of the string, not an effective method of finding it.

By (1), g is an approximation of f. By (2) and theorem 3.3.2, g will be a m.d.s.a. of f provided that g is sequential. We proceed by cases to show this.

a) If $u = w\epsilon_0 z$, where $w \in \Sigma^*$, $z \in (\Sigma \cup \{\epsilon_0\})^*$, $g(u) = f(w)$, by (1). Now, for $v \in (\Sigma \cup \{\epsilon_0\})^*$, $g(uv) = g(w\epsilon_0 zu) = f(w)$, by (1). Thus, $g(u) = g(uv)$ and $g(u)Bg(uv)$.

b) If $u \in \Sigma^*$, $g(u) = y$ such that $(\forall w \in \Sigma^*)[yBf(uw)]$. For $v \in (\Sigma \cup \{\epsilon_0\})^*$, we have two cases.

i) For $v \in \Sigma^*$, $g(uv)Bf(uv)$, by (2) and $g(uv)$ is the longest initial substring of $f(uv)$. From (b), $g(u)Bf(uv)$, and $|g(u)| \leq |g(uv)|$. Thus, $g(u)Bg(uv)$.

ii) For $v = w\epsilon_0 z$, where $w \in \Sigma^*$, $z \in (\Sigma \cup \{\epsilon_0\})^*$, $g(uv) = g(uw\epsilon_0 z) = f(uw)$, by (1). But from (b), $g(u)Bf(uw)$ and thus $g(u)Bg(uv)$.

Thus, by lemma 2.3.4, g is sequential. $\square$

As was noted in the proof, theorem 3.3.3 provides no algorithm for constructing a m.d.s.a. of any given function. Theorem 3.3.4 shows that no such algorithm can exist even if information about the function is available via an 'oracle' or a method of evaluating f.

Theorem 3.3.4 There exists a computable function $f:\{0,1\}^* \to \{0,1\}^*$, for which there exists no computable maximally defined sequential approximation.

Proof: Let $\{M_i\}$ represent a canonical ordering of Turing machines. Define f as follows:

$$f(\lambda) = 0;$$

$$\text{for } x \in \{0,1\}^*,\ f(1x) = 0;$$

$$\text{for any } n > 0,\ x \in \{0,1\}^*, f(0^n 1x) = \begin{cases} 0 & \text{if } M_n \text{ halts started with a blank input tape in } |x| \text{ steps} \\ 1 & \text{otherwise} \end{cases}$$

Clearly, f is computable. Suppose g is a m.d.s.a. of f. Then, $g(0^n 1) = 1$ if and only if M_n does not halt when started with a blank input tape. Hence, g is not computable, else the blank tape halting problem would be solvable. $\square$

3.4 Sequential and Truly Off-line Functions

In this section, we will study two classes of functions which have 'natural' maximally defined sequential approximations. First, however, we show that every m.d.s.a. is unique.

Lemma 3.4.1 For any function $f:\Sigma^* \to \Gamma^*$, the maximally defined sequential approximation of f is unique up to the choice of endmarker.

Proof: Suppose that $g_1:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ and $g_2:(\Sigma \cup \{\epsilon_0\})^* \to \Gamma^*$ are both m.d.s.a.'s of f. We consider two cases.

1) $x = y\epsilon_0 w$, where $y \in \Sigma^*$, $w \in (\Sigma \cup \{\epsilon_0\})^*$. Since g_1 is an approximation of f, $g_1(x) = f(y)$. Since g_2 is an approximation of f, $g_2(x) = f(y) = g_1(x)$.

2) $x \in \Sigma^*$. Since g_1 is a m.d.s.a. of f and g_2 is a sequential approximation of f, by definition 3.3.1, $g_2(x) B g_1(x)$. Similarly, $g_1(x) B g_2(x)$. Hence, $g_1(x) = g_2(x)$. Therefore,

$(\forall x \in (\Sigma \cup \{\epsilon_0\})^*)[g_1(x) = g_2(x)]$. □

Notation Denote as $\hat{f}$ the maximally defined sequential approximation of f.

In introducing the concept of a m.d.s.a. at the beginning of section 3.3, it was noted that if f is a sequential function, the m.d.s.a. of f agrees with f for all inputs not containing the end-marker.

Theorem 3.4.2 Let $f:\Sigma^* \to \Gamma^*$. f is sequential if and only if $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)]$.

Proof: Suppose f is sequential. We know then that $(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(u) B f(uv)]$. We define a function g which is shown to be the m.d.s.a. of f.

1) For $x \in \Sigma^*$, $g(x) = f(x)$.

2) For $x = y\epsilon_0 w$, where $y \in \Sigma^*$, $w \in (\Sigma \cup \{\epsilon\})^*$, $g(x) = f(y)$.

By (2), g is an approximation of f. We must show that g is sequential.

a) For $u \in \Sigma^*$, $g(u) = f(u)$, by (1). We get two subcases for $v \in (\Sigma \cup \{\epsilon_0\})^*$.

i) For $v \in \Sigma^*$, $g(uv) = f(uv)$. Hence $g(u) = f(u)Bf(uv) =$ $= g(uv)$.

ii) For $v = y\epsilon_0 w$, where $y \in \Sigma^*$, $w \in (\Sigma \cup \{\epsilon_0\})^*$, $g(uv) =$ $= g(uy\epsilon_0 w) = f(uy)$, by (2). Thus, $g(u) = f(u)Bf(uy) =$ $= g(uv)$.

b) For $u = y\epsilon_0 w$, where $y \in \Sigma^*$, $w \in (\Sigma \cup \{\epsilon_0\})^*$, $g(u) = f(y)$, by (2). Also, for $v \in (\Sigma \cup \{\epsilon_0\})^*$, $g(uv) = g(y\epsilon_0 wv) = f(y)$, by (2). Hence, $g(u)Bg(uv)$.

We have so far shown that g is a sequential approximation of f. Furthermore, $(\forall u \in \Sigma^*)[g(u) = f(u)]$; thus, $(\forall u \in \Sigma^*)(\forall b \in \Gamma)$ $[g(u)\cdot b \not B f(u\cdot\lambda)]$. Therefore, by theorem 3.3.2, g is the m.d.s.a. of f. This concludes the only if part of the theorem.

Now, suppose that $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)]$. Since $\hat{f}$ is sequential, $(\forall u \in (\Sigma \cup \{\epsilon_0\})^*)(\forall v \in (\Sigma \cup \{\epsilon_0\})^*)[\hat{f}(u)B\hat{f}(uv)]$ and hence, $(\forall u \in \Sigma^*)(\forall v \in \Sigma^*)[f(u) = \hat{f}(u)B\hat{f}(uv) = f(uv)]$ and f is sequential. □

<u>Corollary 3.4.3</u> Let $f:\Sigma^* \to \Gamma^*$. f is computable and sequential if and only if $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)]$ and $\hat{f}$ is computable.

<u>Proof</u>: From theorem 3.4.2, f is sequential if and only if $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)]$. The construction of $g = \hat{f}$ from f is clearly effective. Therefore, if f is computable, $\hat{f}$ is computable. Conversely, if $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)]$ and $\hat{f}$ is computable, then f is obviously computable. □

<u>Remark</u> There are two cases if $f:\Sigma^* \to \Gamma^*$ is either not sequential or not computable to compare to corollary 3.4.3. First, if f is computable but not sequential, $\hat{f}$ need not be computable and $(\exists x \in \Sigma^*)[\hat{f}(x) \neq f(x)]$. Next, if f is sequential but not computable, then $\hat{f}$ is not computable and $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)]$.

<u>Corollary 3.4.4</u> Let $f:\Sigma^* \to \Gamma^*$. f is on-line computable if and only if $(\forall x \in \Sigma^*)[\hat{f}(x) = f(x)$ & $|x| = |f(x)|]$ and f is computable.

<u>Proof:</u> Immediate from theorem 2.5.3 and corollary 3.4.3. $\square$

Theorem 3.4.2 gives an alternate characterization of the class of sequential functions in terms of the class of their m.d.s.a.'s. We now isolate a second class of computable functions having computable m.d.s.a.'s.

<u>Definition 3.4.5</u> A function $f:\Sigma^* \to \Gamma^*$ is said to be <u>truly off-line</u> if $(\forall x \in \Sigma^*)[\hat{f}(x) = \lambda]$.

Essentially a function f is truly off-line if for any machine computing f, no output can be made until all of the input has been read. That is, f is truly off-line if the sequential Turing machine computing $\hat{f}$ produces its output only when the input endmarker has been read. This should be compared to the case for sequential functions. That is, a function f is sequential if the sequential Turing machine computing $\hat{f}$ produces no output after the input endmarker is reached.

<u>Remark</u> Clearly, the zero function, f_λ, $((\forall x \in \Sigma^*)[f_\lambda(x) = \lambda])$, is both truly off-line and sequential. This anomaly will lead to minor complications in the next chapter.

<u>Lemma 3.4.6</u> There exists a length-preserving function $f:\{0,1\}^* \to \{0,1\}^*$ which is truly off-line.

<u>Proof:</u> Let $f:\{0,1\}^* \to \{0,1\}^*$ be the string reversal function I_m defined in theorem 2.2.3. That is, $f(\lambda) = \lambda$, $(\forall x \in \{0,1\}^*)[f(x0) =$ $= 0\cdot f(x)]$ and $(\forall x \in \{0,1\}^*)[f(x1) = 1\cdot f(x)]$.

Suppose f were not truly off-line. Therefore, $(\exists x \in \Sigma^*)[\hat{f}(x) \neq \lambda]$. Let $\hat{f}(x) = y$, $y \in \Gamma^+$. Since $\hat{f}$ is sequential,

$(\forall b \in \{0,1\})[\hat{f}(x) B f(xb)]$. Hence, $yB0 \cdot f(x)$ and $yB1 \cdot f(x)$. Since $y \neq \lambda$ by assumption, y begins with both a 0 and a 1, a contradiction. $\square$

Lemma 3.4.7 There exists a length-preserving function $f:\{0,1\}^* \to \{0,1\}^*$ which is neither sequential nor truly off-line.

Proof: Consider $f:\{0,1\}^* \to \{0,1\}^*$ defined as:

1) $f(\lambda) = \lambda$;

2) For $x \in \{0,1\}^*$, $f(0x) = 0x$;

3) For $x \in \{0,1\}^*$, $f(1x) = I_m(1x)$.

By (3), $f(1) = I_m(1) = 1$ and $f(10) = I_m(10) = 01$. Hence, $f(1) \not B f(10)$ and f is not sequential. By (2), $f(0x) = 0x = f(0)x$, for any $x \in \{0,1\}^*$. Thus, $(\forall x \in \Sigma^*)[0Bf(0x)]$. Suppose $\hat{f}(0) = \lambda$. Then, by theorem 3.3.2, $(\exists v \in \{0,1\}^*)[\hat{f}(0) \not B f(0v)]$, a contradiction. So, f is not truly off-line. $\square$

The function f defined in lemma 3.4.7 has the property that for one half of the inputs, f has a sequential nature, while for the other half, f has a truly off-line nature. There are functions for which this is not true and which still satisfy lemma 3.4.7. In the next chapter, we will show that there exist functions such that for any rational constant p between 0 and 1, and for all inputs, the portions of the output before and after the endmarker are p and $1-p$, respectively.

Remark For the function f defined in lemma 3.4.7, $\hat{f}:\{0,1,\epsilon_0\}^* \to \{0,1\}^*$ is defined as:

a) $\hat{f}(\lambda) = \lambda$;

b) for $x \in \{0,1,\epsilon_0\}^*$, $\hat{f}(\epsilon_0 x) = \lambda$;

c) for $x \in \{0,1\}^*$, $\hat{f}(0x) = 0x$;

d) for $x = u\epsilon_0 v$, where $u \in \{0,1\}^*$, $v \in \{0,1,\epsilon_0\}^*$, $\hat{f}(0x) =$

$\hat{f}(0u\epsilon_0 v) = \hat{f}(0u) = 0u$;

e) for $x \in \{0,1\}^*$, $\hat{f}(1x) = \lambda$;

f) for $x = u\epsilon_0 v$, where $u \in \{0,1\}^*$, $v \in \{0,1,\epsilon_0\}^*$, $\hat{f}(1x) =$

$= \hat{f}(1u\epsilon_0 v) = f(1u) = \mathrm{Im}(1u)$. $\square$

CHAPTER 4

A CLASSIFICATION OF FUNCTIONS BY THEIR MAXIMALLY DEFINED SEQUENTIAL APPROXIMATIONS

4.1 Introduction

In chapter 3, we showed that the concept of a m.d.s.a leads to the classification of functions as sequential, truly off-line, or neither. In this chapter, we study the problem of measuring a function in terms of its m.d.s.a. and studying the results of this measurement.

We first define a measure R_f based on the function f and its m.d.s.a. $\hat{f}$. Intuitively, R_f is the average ratio of the length of $\hat{f}(x)$ to the length of $f(x)$ over all inputs x. It is shown that for any function f other than f_λ (the zero function), $0 \le R_f \le 1$.

For truly off-line and sequential functions, R_f has the extremal values 0 and 1 , respectively. Unfortunately, R_f may also be 0 for a function which is not truly off-line, or 1 for a function which is not sequential. This phenomenon leads to the introduction of the class of 'almost truly off-line' functions and the class of 'almost sequential' functions.

A function f is almost truly off-line if there exists a constant k such that the amount of output generated by $\hat{f}$ before the receipt of an endmarker is always bounded by k . As was the case for off-line functions, there is a function f which is not almost truly off-line but for which $R_f = 0$. Furthermore, there is an almost truly off-line function f for which $R_f \neq 0$.

A parallel class of almost sequential functions is introduced next and analogous results are obtained. Indeed, the function f which is almost truly off-line but for which $R_f \neq 0$ is also almost sequential. We show that 'bounded functions' (i.e. the functions with finite range) are both almost sequential and almost truly off-line, and conversely.

Finally, for $0 \leq p \leq 1$, we define C_p as the class of functions for which $R_f \leq p$. By construction, we can show that for any rational constant p between 0 and 1, there is a function f such that $R_f = p$. Hence, the hierarchy of classes defined by C_p is infinite and dense.

4.2 Measuring the Sequential Nature of a Function

In this section, we consider the problem of measuring a function in relation to its m.d.s.a. and in particular, the classes at each end of the scale.

The most obvious method of measuring the sequential nature of a function p is to compare for each input the length of $f(x)$ and $\hat{f}(x)$. By definition 3.4.5, if $h: \Sigma^* \to \Gamma^*$ is truly off-line, $(\forall x \in \Sigma^*)[\hat{h}(x) = \lambda]$, and hence $(\forall x \in \Sigma^*)[|\hat{h}(x)| = 0]$. Thus, if $h: \Sigma^* \to \Gamma^*$ is truly off-line, $(\forall x \in \Sigma^*)[h(x) \neq \lambda \Rightarrow (|\hat{h}(x)|/|h(x)|) = 0]$. Now let $g: \Sigma^* \to \Gamma^*$ be sequential. By theorem 3.4.2, $(\forall x \in \Sigma^*)[g(x) = \hat{g}(x)]$. Hence, if $g: \Sigma^* \to \Gamma^*$ is sequential, $(\forall x \in \Sigma^*)[g(x) \neq \lambda \Rightarrow (|\hat{g}(x)|/|g(x)|) = 1]$.

Finally, consider the function $f: \{0, 1\}^* \to \{0, 1\}^*$ defined in lemma 3.4.7. $(\forall x \in 0\cdot\{0, 1\}^*)[f(x) = \hat{f}(x)]$; thus, $(\forall x \in 0\cdot\{0, 1\}^*)[(|\hat{f}(x)|/|f(x)|) = 1]$. Now, $(\forall x \in 1\cdot\{0, 1\}^*)[\hat{f}(x) = \lambda]$; thus, $(\forall x \in 1\cdot\{0, 1\}^*)[(|\hat{f}(x)|/|f(x)|) = 0]$. Thus, for any meaningful measure of the sequential nature of a function, all of the input words x for which $f(x) \neq \lambda$ must be considered in the ratio. Definition 4.2.1 reflects all of these considerations in formalizing the measure.

Definition 4.2.1 Let $f\colon \Sigma^* \to \Gamma^*$. Let $M_n = \{x \mid |x| \le n$ and $f(x) \ne \lambda\}$ and m_n be the total number of words in M_n. For each n, let $R_f(n) = \frac{1}{m_n} \sum_{x \in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$. For f, define a real number $R_f = \lim_{n\to\infty} R_f(n)$.

Remark For the zero function, f_λ, R_{f_λ} is undefined, as $m_n = 0$ for all n. This is not a surprising occurrence since f_λ is both sequential and truly off-line.

Remark it should be noted that R_f may not be defined for some functions, although it is in all cases that follow.

Remark In the case that m_n is bounded independently of n, then $f\colon \Sigma^* \to \Gamma^*$ is truly off-line.

Proof Since m_n is bounded independently of n, then for some n_0, $(\forall n \ge n_0)[M_n = M_{n_0}]$. Therefore, $(\forall x \notin M_{n_0})[f(x) = \lambda]$. Hence, $(\forall x \notin M_{n_0})[\hat{f}(x) = \lambda]$. Suppose $x \in M_{n_0}$. Since $\hat{f}$ is sequential, $(\forall y \in \Sigma^*)[xBy \Rightarrow \hat{f}(x)B\,\hat{f}(y)]$. Choose a $y \in \Sigma^*$ such that $|y| > n_0$. Then, $\hat{f}(y) = \lambda$, as $y \notin M_{n_0}$ and consequently, $\hat{f}(x) = \lambda$. Thus $(\forall x \in \Sigma^*)[\hat{f}(x) = \lambda]$ and f is truly off-line. $\square$

As a consequence of the preceding two remarks, the nontrivial cases occur when $f(x) \ne \lambda$ infinitely often, i.e. when m_n is unbounded. Lemma 4.2.2 shows that for any f other than f_λ, R_f lies in a well-defined range.

Lemma 4.2.2 For every function $f\colon \Sigma^* \to \Gamma^*$, $f \ne f_\lambda$, $0 \le R_f \le 1$.

Proof Since $\hat{f}$ is the m.d.s.a. of f and the length of a string is nonnegative, $0 \le \frac{|\hat{f}(x)|}{|f(x)|} \le 1$ for all $x \in M_n$ (as $|f(x)| > 0$ if $x \in M_n$). Therefore, $0 \le \sum_{x \in M_n} \frac{|\hat{f}(x)|}{|f(x)|} \le m_n$ (as

$|M_n| = m_n$). Thus, $0 \le R_f(n) = \frac{1}{m_n} \sum_{x \in M_n} \frac{|\hat{f}(x)|}{|f(x)|} \le 1$. We thus conclude that $0 \le R_f = \lim_{n\to\infty} R_f(n) \le 1$. $\square$

As has been noted, the class of truly off-line functions should lie at the low end of the scale formed at R_f, while the class of sequential functions should lie at the high end. Theorems 4.2.3 and 4.2.5 with their corollaries will prove these facts formally.

Theorem 4.2.3 Let $f\colon \Sigma^* \to \Gamma^*$ be any function. f is truly off-line if and only if $f = f_\lambda$ or for some number n_0, $(\forall n \ge n_0)\ [R_f(n) = 0]$.

Proof: Suppose f is truly off-line. If $f = f_\lambda$, we are done. Otherwise, by definition 4.2.1, $(\forall x \in \Sigma^*)\ [\hat{f}(x) = \lambda]$. Since $f \ne f_\lambda$, $(\exists x_0 \in \Sigma^*)\ [f(x_0) \ne \lambda]$. Hence, M_n is not empty for $n \ge |x_0|$. Choosing $n_0 = |x_0|$ we obtain for any $n > n_0$, $R_f(n) = \frac{1}{m_n} \sum_{x \in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$

$$= \frac{1}{m_n} \sum_{x \in M_n} \frac{0}{|f(x)|} \quad \text{(as f is truly off-line)}$$

$$= \frac{1}{m_n} \sum_{x \in M_n} (0) \quad \text{(as } |f(x)| > 0 \text{ if } x \in M_n)$$

$$= 0 \quad \text{(as } m_n > 0).$$

Now suppose f is not truly off-line. Then $f \ne f_\lambda$, and $(\exists x \in \Sigma^*)\ [\hat{f}(x) \ne \lambda]$. Choose a word $y \in \Sigma^*$ such that xBy and $|y| \ge n_0$. Since $\hat{f}$ is the m.d.s.a. of f, $\hat{f}(x)\ B\hat{f}(y) B\ f(y)$ and hence, $\hat{f}(y) \ne \lambda$ and $f(y) \ne \lambda$. Let $n_1 = |y|$. Hence, $y \in M_{n_1}$, $m_{n_1} > 0$ and $n_1 \ge n_0$.

$$\text{But, } R_f(n) = \frac{1}{m_{n_1}} \sum_{x \in M_{n_1}} \frac{|\hat{f}(x)|}{|f(x)|}$$

$$\geq \frac{1}{m_{n_1}} \frac{|\hat{f}(y)|}{|f(y)|}$$

$$> 0. \quad \square$$

Corollary 4.2.4 If $f\colon \Sigma^* \to \Gamma^*$ is truly off-line, either $f = f_\lambda$ or $R_f = 0$.

Proof: Immediate. $\square$

Two observations about the above characterizations of truly off-line functions can be made before proving the parallel results about sequential functions. For theorem 4.2.3, it is not necessarily true that $(\forall n \geq 0)\ [R_f(n) = 0]$ as $R_f(n)$ may be undefined because M_n is empty for some values of n. In Corollary 4.2.4, even if $R_f = 0$, f need not be truly off-line. In this case, the limit may approach 0, although for no n is $R_f(n) = 0$. This case and its parallel for sequential functions will be investigated more fully later.

Theorem 4.2.5 Let $f\colon \Sigma^* \to \Gamma^*$ be any function. f is sequential if and only if $f = f_\lambda$ or for some number n_0, $(\forall n \geq n_0)[R_f(n) = 1]$.

Proof: Suppose f is a sequential function. If $f = f_\lambda$, we are done. Hence we assume that $(\exists x_0 \in \Sigma^*)\ [f(x_0) \neq \lambda]$. Let $|x_0| = n_0$. Now by theorem 3.4.2, $(\forall x \in \Sigma^*)\ [\hat{f}(x) = f(x)]$ and hence, $(\forall n \geq n_0)\ (\forall x \in M_n)$ $[|\hat{f}(x)| = |f(x)|]$ and $m_n > 0$.

Hence, for $n \geq n_0$, $R_f(n) = \frac{1}{m_n} \sum_{x \in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$

$$= \frac{1}{m_n} \sum_{x \in M_n} (1) \quad \text{(as } |f(x)| \neq 0 \text{ if } x \in M_n\text{)}$$

$$= \frac{m_n}{m_n} \quad \text{(by definition of } M_n\text{)}$$

$$= 1 \quad \text{(as } m_n \neq 0\text{).}$$

Now suppose f is not sequential. Then, $f \neq f_\lambda$ and $(\exists x_1 \in \Sigma^*)\ [\hat{f}(x_1) \neq f(x_1)]$. Since $(\forall x \in \Sigma^*)\ [\hat{f}(x)\ B\ f(x)]$, then, $f(x_1) \neq \lambda$. Let $n_1 = \max(|x_1|, n_0)$; thus $x_1 \in M_{n_1}$. Let $M = M_{n_1} - \{x_1\}$.

Hence, $R_f(n) = \frac{1}{m_{n_1}} \sum_{x \in M} \frac{|\hat{f}(x)|}{|f(x)|} + \frac{1}{m_{n_1}} \frac{|\hat{f}(x_1)|}{|f(x_1)|}$

$$\leq \frac{1}{m_{n_1}} \sum_{x \in M} \frac{|f(x)|}{|f(x)|} + \frac{1}{m_{n_1}} \frac{|\hat{f}(x_1)|}{|f(x_1)|} \quad \text{(as } |\hat{f}(x)| \leq |f(x)|\text{)}$$

$$= \frac{m_{n_1} - 1}{m_{n_1}} + \frac{1}{m_{n_1}} \frac{|\hat{f}(x_1)|}{|f(x_1)|} \quad \text{(as } |M| = m_{n_1} - 1\text{)}$$

$$< \frac{m_{n_1} - 1}{m_{n_1}} + \frac{1}{m_{n_1}} \quad \text{(as } |\hat{f}(x_1)| < |f(x_1)|\text{)}$$

$$= 1\ . \quad \square$$

<u>Corollary 4.2.6</u> If $f: \Sigma^* \to \Gamma^*$ is sequential either $f = f_\lambda$ or $R_f = 1$.

<u>Proof</u>: Immediate. $\square$

We notice that, as was the case for truly off-line functions, we cannot use an if and only if condition in the statement of corollary 4.2.6. That is, there is a function f which is not sequential and for which $R_f = 1$. Since the two ends of the scale formed by R_f are exactly parallel in nature, we will prove results about each in parallel, rather than investigating each individually.

4.3 Almost Truly Off-line Functions

Lemma 4.3.1 proves formally the existence of functions which show that knowing the value of R_f is not sufficient for deciding whether f is truly off-line.

Lemma 4.3.1 There exists a function $f: \{0,1\}^* \to \{0,1\}^*$ which is not truly off-line but for which $R_f = 0$.

Proof: Let $Im: \{0,1\}^* \to \{0,1\}^*$ be the string reversal function as defined in theorem 2.2.3. Consider $f: \{0,1\}^* \to \{0,1\}^*$ defined by $(\forall x \in \{0,1\}^*)$ $[f(x)=1.Im(x)]$. For example, $f(\lambda) = 1$, $f(0001) = 11000$, and $f(1010110) = 10110101$.

Now we define a function $g: \{0,1,\epsilon_0\}^* \to \{0,1\}^*$ which will be shown to be the m.d.s.a. of f.

1) For $x \in \{0,1\}^*$, $g(x) = 1$.

2) For $x = w\epsilon_0 y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(w\epsilon_0 y) = 1.Im(w)$.

We first show that g is sequential.

a) For $x \in \{0,1\}^*$, $g(x) = 1$. We get two cases:

i) For $v \in \{0,1\}^*$, $g(xv) = 1 = g(x)$. Hence $g(x)$ B $g(xv)$

ii) For $v = w\epsilon_0 y$, where $w \in \{0,1\}^*$ and $y \in \{0,1,\epsilon_0\}^*$, $g(xv) = g(xw\epsilon_0 y) = 1.Im(xw)$, by (2). Hence, $g(x) = 1$ B $1.Im(xw) = g(xv)$.

b) For $x = w\epsilon_0 y$ where $w \in \{0,1\}^*$ and $y \in \{0,1,\epsilon_0\}^*$, $g(x) = 1.\mathrm{Im}(w)$ by (2). For $v \in \{0,1,\epsilon_0\}^*$, by (2), $g(xv) = g(w\epsilon_0 yv) = 1.\mathrm{Im}(w) = g(x)$. Again, $g(x)\,\mathrm{B}\,g(xv)$ and g must be sequential.

Now by (2), $(\forall x \in \{0,1\}^*)\ [g(x\epsilon_0) = 1.\mathrm{Im}(x) = f(x)]$. Thus, g is a sequential approximation of f.

Now by (1) $(\forall x \in \{0,1\}^*)\ [g(x) = 1]$. Consider $g(x).0$. $f(x1) = 11\mathrm{Im}(x)$, by the definition of Im. Hence $g(x).0 \not\mathrm{B}\ f(x1)$. Now consider $g(x).1$. $f(x0) = 10\ \mathrm{Im}(x)$, by definition of Im. Hence, $g(x).1 \not\mathrm{B}\ f(x0)$. Hence, $(\forall x \in \{0,1\}^*)(\forall b \in \{0,1\})(\exists v \in \{0,1\}^*)\ [g(x){\cdot}b \not\mathrm{B}\ f(xv)]$. Thus, by theorem 3.3.2, g is the m.d.s.a. of f, i.e. $g = \hat{f}$. Since $\hat{f}(w) = g(1) = 1$, $(\exists x \in \{0,1\}^*)\ [\hat{f}(x) \neq \lambda]$ and f is not truly off-line.

Now consider $R_f = \lim\limits_{n\to\infty} \frac{1}{m_n} \sum\limits_{x\in M_n} \left|\frac{\hat{f}(x)}{f(x)}\right|$.

Note that for any n, $M_n = \{x \mid |x| \leq n\}$ and $m_n = 2^{n+1} - 1$. Also, $(\forall x \in M_n)\ [|\hat{f}(x)|=1]$ and $|f(x)| = 1 + |x|$. Furthermore, there are 2^i words of length i in $\{0,1\}^*$.

Hence, $R_f = \lim\limits_{n\to\infty} \frac{1}{m_n} \sum\limits_{x\in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} \frac{2^i}{(1+i)} \qquad \text{(by above)}$$

$$\leq \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \quad \frac{3 \cdot 2^n}{1+n} \qquad \left(\begin{array}{l}\text{as can be verified}\\ \text{by the reader.}\end{array}\right)$$

$$\leq \lim_{n\to\infty} \frac{3}{2(1+n)} = 0.$$

By lemma 4.2.2, $R_f \geq 0$; hence, $R_f = 0$, although f is not truly off-line. □

Note that the function f in lemma 4.3.1 has the property that $|\hat{f}(x)| = 1$, for all inputs $x \in \{0,1\}^*$. Any function having the properties that $|\hat{f}(x)| \leq k$, for all inputs x, and that as the length of the input x increases, the length of f(x) increases, could have been used in the proof. This observation motivates the following definition of almost truly off-line functions.

<u>Definition 4.3.2</u> Let $f: \Sigma^* \to \Gamma^*$. f is said to be <u>almost truly off-line</u> if $(\forall x \in \Sigma^*)\ [|\hat{f}(x)| \leq k]$, for some integer constant $k \geq 0$.

Remember that a truly off-line function can be described by the sequential Turing machine M which computes its m.d.s.a.; that is, f is truly off-line if M outputs nothing until the receipt of the input endmarker. We can make a similar description for almost truly off-line functions. For any $f: \Sigma^* \to \Gamma^*$, let M_f be the sequential Turing machine computing $\hat{f}$. Then, f is almost truly off-line if M_f outputs at most k digits of f(x) for any input x before the receipt of the input endmarker.

It should be noted that every truly off-line function is an almost truly off-line function and we have just seen that the converse does not hold.

It is now natural to consider the class of almost truly off-line functions as a possible characterization of the class of functions for which $R_f = 0$. Unfortunately, as we will see in lemma 4.3.3 and corollary 4.4.5 below, the characterization fails in both directions.

Lemma 4.3.3 There exists a function $f:\{0,1\}^* \to \{0,1\}^*$ which is not almost truly off-line but for which $R_f = 0$.

Proof: Define $f:\{0,1\}^* \to \{0,1\}^*$ as $(\forall x \in \{0,1\}^*)$ $[f(x) = 1^{|x|} 0^{2^{|x|}}$, where $a^n = \underbrace{a\ a \ldots a}_{n \text{ times}}$. That is, f is the function which outputs a 1 for every digit read followed by 2^n 0's, if the whole input word is of length n.

For example, $f(\lambda) = 1^0 0^{2^0} = 0$; $f(10101101) = 1^8 0^{2^8} = 1^8 0^{256}$.

Define $g: \{0,1,\epsilon_0\}^* \to \{0,1\}^*$ by cases as:

1) For $x \in \{0,1\}^*$, $g(x) = 1^{|x|}$;

2) For $x = w\ \epsilon_0\ y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = f(w)$.

By (2), g is an approximation of f. We now show that g is sequential by cases.

a) For $x \in \{0,1\}^*$, $g(x) = 1^{|x|}$. We get two cases for $v \in \{0,1,\epsilon_0\}^*$:

i) $v \in \{0,1\}^*$, $g(xv) = 1^{|xv|} = 1^{|x|}\ 1^{|v|} = g(x)\cdot 1^{|v|}$.

Hence, g(x) B g(xv).

ii) $v = w\ \epsilon_0\ y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(xv) = g(xw\epsilon_0 y) = f(xw) = 1^{|xw|}\ 0^{2^{|xw|}} = 1^{|x|}\ 1^{|w|}\ 0^{2^{|xw|}} = g(x)\cdot 1^{|w|}\ 0^{2^{|xw|}}$. Hence, g(x) B g(xv).

b) For $x = w\ \epsilon_0\ y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = f(w)$. For $v \in \{0,1,\epsilon_0\}^*$ $g(xv) = g(w\epsilon_0 yv) = f(w) = g(x)$. Hence, g(x) B g(xv) and g is a sequential approximation of f. Now $(\forall x \in \{0,1\}^*)$ $[g(x) = 1^{|x|}]$. Consider $g(x)\cdot 0$. Then $f(x0) = 1^{|x0|}\ 0^{2^{|x0|}} = 1^{|x|}\ 1\ 0^{2^{|x0|}}$. Hence $g(x)\cdot 0 \not B f(x0)$. Consider $g(x)\cdot 1 = 1^{|x|}1$. Then $f(x) = 1^{|x|}\ 0^{2^{|x|}}$ and, $g(x)\cdot 1 \not B f(x)$. Again, by theorem 3.3.2, g is the m.d.s.a. of f. i.e., $g = \hat{f}$.

Now consider R_f. As before, $M_n = \{x \mid |x| \leq n\}$. Hence, $m_n = 2^{n+1}-1$. Also, $\forall x \in \{0,1\}^*$, $|g(x)| = |x|$ and $|f(x)| = |x| + 2^{|x|}$. Note again that there 2^i words of length i in M_n. Hence

$$R_f = \lim_{n\to\infty} \frac{1}{m_n} \sum_{x\in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} \frac{2^i(i)}{i+2^i}$$

$$\leq \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} i$$

$$= \lim_{n\to\infty} \frac{n^2+n}{(2^{n+1}-1)\cdot 2} = 0.$$

Since $(\forall x \in \{0,1\}^*)\ [|\hat{f}(x)| = |x|]$, there can be no bound on the length of $\hat{f}(x)$. Hence f is not almost truly off-line, although $R_f = 0$. □

Thus, it seems that there is no simple characterization of the functions f for which $R_f = 0$. However, the class of almost truly off-line functions will be useful in establishing the hierarchy of functions defined in section 4.5.

4.4 Almost Sequential Functions

We now turn our attention to the other end of the scale formed by R_f. We will parallel the lemmas and definition for the case $R_f = 0$ by symmetric results when $R_f = 1$. Our first result shows that there are non-sequential functions for which $R_f = 1$.

Lemma 4.4.1 There exists a function $f\colon \{0,1\}^* \to \{0,1\}^*$ such that f is not sequential and $R_f = 1$.

Proof: Consider the function f defined as $(\forall x \in \{0,1\}^*)$ $[f(x) = x\cdot 1]$.

We define a function $g\colon \{0,1,\epsilon_0\}^* \to \{0,1\}^*$, which is shown to be the m.d.s.a. of f.

1) For $x \in \{0,1\}^*$, $g(x) = x$;

2) for $x = w\ \epsilon_0\ y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = w\cdot 1$.

By (2), $g(w\ \epsilon_0) = g(w\ \epsilon_0\ y) = f(w)$. Hence, g is an approximation of f. We show g is sequential by cases.

a) For $x \in \{0,1\}^*$, by (1), $g(x) = x$. We consider two cases for $v \in \{0, 1, \epsilon_0\}^*$.

i) For $v \in \{0,1\}^*$, $g(xv) = xv$, by (1). Hence, $g(x) = x\ B\ xv = g(xv)$.

ii) For $v = w\ \epsilon_0\ y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(xv) = g(xw\ \epsilon_0\ y) = x\ w\ 1$, by (2). Hence, $g(x) B g(xv)$.

b) For $x = w\ \epsilon_0 y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = w.1$, by (2). Hence, for $v \in \{0,1,\epsilon_0\}^*$, $g(xv) = g(w\ \epsilon_0\ y\ V) = w.1 = g(x)$, by (2). Hence, $g(x)\ B\ g(xv)$.

Thus, $(\forall x \in \{0,1,\epsilon_0\}^*)(\forall v \in \{0,1,\epsilon_0\}^*)$ $[g(x)\ B\ g(xv)]$ and g is a sequential approximation of f.

Now, by (1) for $x \in \{0,1\}^*$, $g(x) = x$. Consider $g(x)\cdot 0 = x0$. $f(x1) = x11$, by definition. Hence $g(x)\cdot 0 \not B f(x1)$. Now consider $g(x)\cdot 1 = x1$. $f(x0) = x01$, by definition. Hence, $g(x)\cdot 1 \not B f(x0)$.

Therefore, by theorem 3.3.2, g is the m.d.s.a. of f, i.e., $g = \hat{f}$. Since $\hat{f}(1) = g(1) = 1$, and $f(1) = 11$, $(\exists x \in \{0,1\}^*)$ $[\hat{f}(x) \neq f(x)]$. Hence, f is not sequential.

Note that for any n, $M_n = \{x | \; |x| \leq n\}$ and $m_n = 2^{n+1}-1$. Also, $(\forall x \in M_n)[|\hat{f}(x)| = |x| \; \& \; |f(x)| = |x|+1]$.

Therefore, $R_f = \lim\limits_{n\to\infty} \frac{1}{m_n} \sum\limits_{x\in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} \frac{2^i \cdot i}{i+1} \qquad \text{(by above)}$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} (2^i - \frac{2^i}{i+1})$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} (\sum_{i=0}^{n} 2^i - \sum_{i=0}^{n} \frac{2^i}{i+1})$$

$$\geq \lim_{n\to\infty} \frac{1}{2^{n+1}-1} (2^{n+1}-1 - \frac{3\ 2^n}{1+n}) \qquad \text{(as in lemma 4.3.1)}$$

$$= \lim_{n\to\infty} (1 - \frac{3\ 2^n}{(2^{n+1}-1)(1+n)})$$

$$= 1 - \lim_{n\to\infty} \frac{3\ 2^n}{(2^{n+1}-1)(1+n)} = 1. \quad \square$$

Note that for f as defined in lemma 4.4.1, $(\forall x \in \{0,1\}^*)$ $[|f(x)| - |\hat{f}(x)| = 1]$. We could replace f by any function which had the properties that $(\forall x \in \Sigma^*)$ $[|f(x)|-|\hat{f}(x)|\leq k$, for some $k]$ and that $|f(x)|$ increases as $|x|$ increases. An interesting example of such a function is full binary addition.

<u>Definition 4.4.2</u> Let $f\colon \Sigma^* \to \Gamma^*$. f is said to be an <u>almost sequential</u> function if $(\forall x \in \Sigma^*)$ $[|f(x)| - |\hat{f}(x)| \leq k]$, for some constant k.

As was the case for almost truly off-line functions, we can describe an almost sequential function in terms of the sequential Turing machine which computes its m.d.s.a. Let M_f be the sequential Turing machine which computes $\hat{f}$. An almost sequential function is a function for which M_f produces all but at most the last k digits of the output for any input before receiving the endmarker. In lemma 4.4.3, we show that the class of almost sequential functions does not contain the class of functions for which $R_f = 1$.

<u>Lemma 4.4.3</u> There exists a function $f\colon \{0,1\}^* \to \{0,1\}^*$ which is not almost sequential but for which $R_f = 1$.

<u>Proof:</u> Consider $f\colon \{0,1\}^* \to \{0,1\}^*$ defined as $(\forall x \in \{0,1\}^*)$, $[f(x) = 0^{2^{|x|}} 1^{|x|}]$.

We now define a function $g\colon\{0,1,\epsilon_0\}^* \to \{0,1\}^*$ which is shown to be the m.d.s.a. of f.

1) For $x \in \{0,1\}^*$, $g(x) = 0^{2^{|x|}}$;

2) For $x = w\, \epsilon_0 y$, where $w \in \{0,1\}^*$, $y \in\{0,1,\epsilon_0\}^*$, $g(x) = 0^{2^{|w|}} 1^{|w|}$.

By (2), g is an approximation of f. We now show g is sequential by cases.

a) For $x \in \{0,1\}^*$, by (1), $g(x) = 0^{2^{|x|}}$. We consider two cases for $v \in \{0,1,\epsilon_0\}^*$:

i) $v \in \{0,1\}^*$, $g(xv) = 0^{2^{|xv|}} = 0^{2^{|x|}} 0^{2^{|x|}(2^{|v|}-1)} = g(x)\, 0^{2^{|x|}(2^{|v|}-1)}$. Hence, $g(x) B\, g(xv)$.

ii) $v = w\epsilon_0 y$, where $w \in\{0,1\}^*$, $y \in\{0,1,\epsilon_0\}^*$, $g(xv)$

$= g(x\ w\ \epsilon_0\ y) = 0^{2^{|xw|}}\ 1^{|w|} = 0^{2^{|x|}} (0^{2^{|x|}\ (2^{|w|}-1)})\ 1^{|w|}$

$= g(x)\ (0^{2^{|x|}(2^{|w|}-1)})\ 1^{|w|}$. Hence, $g(x)\ B\ g(xv)$.

b) For $x = w\ \epsilon_0\ y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = 0^{2^{|w|}} 1^{|w|}$. For $v \in \{0,1,\epsilon_0\}^*$, $g(xv) = g(w\epsilon_0 yv) = g(x)$. Hence, $g(x)\ B\ g(xv)$.

Thus, $(\forall x \in \{0,1,\epsilon_0\}^*)(\forall v \in\{0,1,\epsilon_0\}^*)$ $[g(x)\ B\ g(xv)]$ and g is a sequential approximation of f.

Now consider $g(x)\cdot 0 = 0^{2^{|x|}} \cdot 0$. Then, $f(x) = 0^{2^{|x|}} \cdot 1^{|x|} \neq 0^{2^{|x|}} \cdot 0\ v$, for any $v \in \{0,1\}^*$. Thus, $g(x).0 \not B\ f(x)$. Consider $g(x)\cdot 1 = 0^{2^{|x|}} \cdot 1$. Then, $f(x\cdot 0) = 0^{2^{|x|+1}}\ 1^{|x|+1} \neq 0^{2^{|x|}} \cdot 1 \cdot v$, for any $v \in \{0,1\}^*$. Thus, $g(x)\cdot \not B\ f(x\cdot 0)$ and $(\forall x \in \{0,1\}^*)$ $(\forall b \in \{0,1\})$ $(\exists v \in \{0,1\}^*)$ $[g(x)\cdot b \not B\ f(xv)]$. Again, using theorem 3.3.2, g is the m.d.s.a. of f. i.e., $g = \hat{f}$.

Now, $(\forall x \in\{0,1\}^*)$ $[|f(x)| - |\hat{f}(x)|) = 2^{|x|} + |x| - 2^{|x|} = |x|]$. Hence, f is not almost sequential.

As before, $M_n = \{x|\ |x| \leq n\ \}$, $m_n = 2^{n+1} - 1$, $|\hat{f}(x)| = 2^{|x|}$ and $|f(x)| = 2^{|x|} + |x|$.

Hence, $R_f = \lim\limits_{n\to\infty} \frac{1}{m_n} \sum\limits_{x\in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} \frac{2^i \cdot 2^i}{2^i+i}$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \sum_{i=0}^{n} (2^i - \frac{2^i\ i}{2^i+1}\ )$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \quad \sum_{i=0}^{n} 2^i - \sum_{i=0}^{n} \frac{2^i\, i}{2^i+i}$$

$$\geq \lim_{n\to\infty} \frac{1}{2^{n+1}-1} \left(2^{n+1}-1 - \frac{n^2+n}{2}\right) \quad \text{(as in lemma 4.3.3)}$$

$$= \lim_{n\to\infty} \left(1 - \frac{n^2+n}{2^{n+1}-1)\ 2} \right)$$

$$= 1 - \lim_{n\to\infty} \frac{n^2+n}{(2^{n+1}-1)\cdot 2}$$

$$= 1. \quad \square$$

We have shown by lemmas 4.3.3 and 4.4.3 that if f is a function such that $R_f = 0$ ($R_f = 1$), it is not necessarily true that f is almost truly off-line (almost sequential). We wish to know if the respective converses are true.

Lemma 4.4.4 There is an almost sequential function $f: \{0,1\}^* \to \{0,1\}^*$ such that $R_f \neq 1$.

Proof: Consider $f: \{0,1\}^* \to \{0,1\}^*$ defined as:

I) For $x \in \{0,1,\lambda\}$, $f(x) = \lambda$;

II) For $x = i\ v\ j$, where $i \in \{0,1\}$, $j \in \{0,1\}$, $v \in \{0,1\}^*$, $f(x) = ij$.

Intuitively, $f(x)$ equals the first and last letters of x if $|x| \geq 2$; $f(x)$ equals λ otherwise. We define a function $g:\{0,1,\epsilon_0\} \to \{0,1\}^*$ which is shown to be the m.d.s.a. of f.

1) For $x \in \{0,1,\lambda\}$, $g(x) = \lambda$

2) For $x = w\ \epsilon_0\ y$, where $w \in \{0,1,\lambda\}$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = \lambda$;

3) For $x = ivj$, where $i,j \in \{0,1\}$, $v \in \{0,1\}^*$, $g(x) = i$;

4) For $x = ivj\epsilon_0 w$, where $i,j\epsilon\{0,1\}$, $v\epsilon\{0,1\}^*$ $w\epsilon\{0,1\epsilon_0\}^*$ $g(x) = i\ j$.

By 2) and 4), g is an approximation of f. We show g is sequential by cases.

a) For $x \in \{0,1,\lambda\}$, $g(x) = \lambda$. For $v \in\{0,1,\epsilon_0\}^*$, $\lambda B g(xv)$. Hence $g(x)\ B\ g(xv)$.

b) For $x = w\ \epsilon_0\ y$, where $w \in \{0,1,\lambda\}$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = \lambda$, and hence $(\forall x \in \{0,1,\epsilon_0\}^*)$ $[g(x)\ B\ g(xv)]$, as in (a).

c) For $x = iwj$, where $i,j \in\{0,1\}$, $w \in \{0,1\}^*$, $g(x) = i$. We get two cases for $v \in \{0,1,\epsilon_0\}^*$.

i) $v \in \{0,1\}^*$, $g(xv) = i$, by (3). Hence $g(x)\ B\ g(xv)$

ii) $v = z\ m\ \epsilon_0\ y$, where $m \in \{0,1\}$, $z \in \{0,1\}^*$ $y \in \{0,1,\epsilon_0\}^*$ $g(xv) = g(iwjzm\epsilon_0 y) = i\ m$, by (4). Hence, $g(x) = iBim = g(xv)$.

d) For $x = iwj\epsilon_0 y$, where $i,j\epsilon\{0,1\}$, $w\epsilon\{0,1\}^*$, $y\ \epsilon\{0,1,\epsilon_0\}^*$, $g(x) = ij$. For $v\epsilon\{0,1,\epsilon_0\}^*$, $g(xv) = ij = g(x)$. Hence, $g(x)\ B\ g(xv)$ and g is a sequential approximation of f.

Now, if $x \epsilon\{\lambda,0,1\}$, $g(x)\cdot 0 = 0 \not B f(x) = \lambda$ and $g(x)\cdot 1 = 1 \not B f(x)$. If $x = i\ u\ j$, where $i,j\epsilon\{0,1\}$ and $u\epsilon\{0,1\}^*$, then $g(x)\cdot 0 = i0$. Now $f(x1) = i1$ and $g(x)\cdot 0 \not B f(x1)$. Similarly, $g(x)\cdot 1 = i1 \not B f(x0) = i0$. Again using theorem 3.3.2, g is the m.d.s.a. of f. i.e., $g = \hat{f}$

Now, $(\forall x \in \{0,1\}^*)$ $[|f(x)| - |\hat{f}(x)| \le 1]$. Hence, f is almost sequential. Now, $M_n = \{x|2\le|x|\le n\}$, so $m_n = 2^{n+1} - 4$ if $n \ge 2$; $m_0 = m_1 = 0$. Also, $(\forall x \in M_n)\left[\frac{|\hat{f}(x)|}{|f(x)|}\right] = \frac{1}{2}$.

$$\text{Thus, } R_f = \lim_{n\to\infty} \frac{1}{m_n} \sum_{x\epsilon M_n} \frac{|\hat{f}(x)|}{|f(x)|}$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-4} \sum_{i=1}^{2^{n+1}-4} (\tfrac{1}{2}) = \tfrac{1}{2} \qquad \square$$

Notice that the function f defined in lemma 4.4.4 is also almost truly off-line, because for all x, $|\hat{f}(x)| \leq 1$. Hence, corollary 4.4.5 follows immediately from lemma 4.4.4.

Corollary 4.4.5 There is an almost truly off-line function f: $\{0,1\}^* \to \{0,1\}^*$ such that $R_f \neq 0$.

We note that any function f: $\Sigma^* \to \Gamma^*$ such that $(\exists k \geq 0)(\forall x \in \Sigma^*)[|f(x)| \leq k]$ will be both almost truly off-line and almost sequential. Since we desire that $R_f = 0$ if f is almost truly off-line and $R_f = 1$ if f is almost sequential, it is not surprising that the R_f measure does not fully characterize these notions. We are thus led to consider removing all functions $f: \Sigma^* \to \Gamma^*$ such that $(\exists k \geq 0)(\forall x \in \Sigma^*)[|f(x)| \leq k]$ from the classes of almost truly off-line and almost sequential functions by altering the appropriate definitions. Note that in doing so, however, that the class of almost truly off-line (almost sequential) functions would no longer contain the class of truly off-line (sequential) functions. e.g., f_λ, would then be both truly off-line and sequential but neither almost truly off-line nor almost sequential.

4.5 A Dense Hierarchy of Functions

The function f defined in lemma 4.4.4 had the property that $R_f = \frac{1}{2}$. It is natural to wonder if for any rational constant p/q, such that $0 \leq p \leq q$, there is a function $f_{p/q}$ such that $R_{f_{p/q}} = \frac{p}{q}$. Definition 4.5.1 formalizes the concept of a bounded function, which is shown in theorem 4.5.2 to be exactly the intersection of the class of almost truly off-line functions and the class of almost sequential functions.

Definition 4.5.1 A function $f:\Sigma^* \to \Gamma^*$ is said to be bounded if there exists a constant k such that $(\forall x \in\Sigma^*)$ $[|f(x)|\le k]$.

Remark A function f is bounded if and only if f has finite range.

Theorem 4.5.2 A function $f: \Sigma^* \to \Gamma^*$ is bounded if and only if f is both almost sequential and almost truly off-line.

Proof: Suppose f is bounded. By definition 4.5.1 $(\forall x \in \Sigma^*)$ $[|f(x)|\le k]$. Since $\hat{f}$ is the m.d.s.a. of f, $(\forall x \in \Sigma^*)$ $[0 \le|\hat{f}(x)| \le |f(x)|]$. Thus, $(\forall x \in\Sigma^*)$ $[|\hat{f}(x)|\le k]$ and f is almost truly off-line. Also, $(\forall x \in \Sigma^*)$ $[|f(x)|-|\hat{f}(x)|\le k]$ and f is almost sequential.

Now, suppose f is almost sequential and almost truly off-line; thus, $(\forall x \in \Sigma^*)$ $[|\hat{f}(x)|\le k_0]$ and $(\forall x \in \Sigma^*)$ $[|f(x)| - |\hat{f}(x)|\le k_1]$. Thus, $(\forall x \in \Sigma^*)$ $[|f(x)|\le k_0+k_1]$ and f is bounded. □

Lemma 4.5.3 Let p and q be integers such that $0\le p\le q$ and $q>0$. Then, there exists a bounded function $f_{p/q}:\{0,1\}^* \to \{0,1\}^*$ such that $R_{f_{p/q}} = \frac{p}{q}$.

Proof: We let $q = p+r$, $r\ge 0$. We define a function $f_{p/q}$: $\{0,1\}^* \to \{0,1\}^*$ as follows:

I) For $x \in \{0,1,\lambda\}$, $f(x) = \lambda$;

II) For $x = iwj$, where $i \in \{0,1\}$, $w\in\{0,1\}^*$, $j \in\{0,1\}$, $f_{p/q}(x) = i^p j^r$, where i^n is $\underbrace{i\ i\ \ldots\ i}_{n \text{ times}}$

That is, for $p = 1$, $q = 2$, $f_{\frac{1}{2}}$ is the function defined in lemma 4.4.4.

For example, if $p = 2$, $q = 5$, $f_{2/5}(01101) = 00111$, $f_{2/5}(011010) = 00000$, $f_{2/5}(011011) = 00111$, $f_{2/5}(10) = 11000$.

We now define a function $g: \{0,1,\epsilon_0\}^* \rightarrow \{0,1\}^*$, which is the m.d.s.a. of $f_{p/q}$. i.e., $g = \hat{f}_{p/q}$

1) For $x \in \{0,1,\lambda\}$, $g(x) = \lambda$;

2) For $x = w\, \epsilon_0\, y$, where $w \in \{0,1,\lambda\}$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = \lambda$;

3) For $x = iw$, where $i \in \{0,1\}$, $w \in \{0,1\}^+$, $g(x) = i^p$;

4) For $x = iwj\epsilon_0 y$, where, $i,j \in \{0,1\}$, $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = i^p j^r$.

Since the proof that g is the m.d.s.a. of $f_{p/q}$ completely parallels the proof in lemma 4.4.4, it will be omitted.

Note that $(\forall x \in \{0,1\}^*)\ [|f_{p/q}(x)| \leq q]$; hence, f is a bounded function.

As before $M_n = \{x \mid 2 \leq |x| \leq n\}$, hence, $m_n = 2^{n+1} - 4$ if $n \geq 2$; $m_0 = m_1 = 0$. Also $(\forall x \in M_n)\ [|\hat{f}_{p/q}(x)| = p\ \&\ |f_{p/q}(x)| = q]$.

Hence,
$$R_{f_{p/q}} = \lim_{n\to\infty} \frac{1}{m_n} \sum_{x \in M_n} \frac{|\hat{f}_{p/q}(x)|}{|f_{p/q}(x)|}$$
$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-4} \sum_{i=1}^{2^{n+1}-4} \frac{p}{q}$$
$$= \frac{p}{q}\ . \qquad \Box$$

Lemma 4.5.2 forms the foundation of the proof of theorem 4.5.4 which shows the existence of a dense, infinite hierarchy of classes of functions based on the measure R_f.

Notation We denote by C_t, for t a real number in the open interval $(0,1)$, the class of functions f such that $R_f \leq t$.

Theorem 4.5.4 The hierarchy defined by C_t is dense and infinite.

Proof: For all p and q, as in lemma 4.5.3, $f_{p/q} \in C_t$ if and only if $t \geq \frac{p}{q}$. □

We observe that it was the study of the bounded functions which led to the proof of the existence of the dense, infinite hierarchy. It is natural to wonder if functions with infinite range (hence not bounded) could have been used in place of the $f_{p/q}$'s.

Lemma 4.5.5 Let p and q be integers such that $o \leq p \leq q$ and $q > o$. Then, there exists a function $h_{p/q}: \{0,1\}^* \to \{0,1\}^*$ such that $h_{p/q}$ is not bounded and $R_{h_{p/q}} = \frac{p}{q}$.

Proof: We will define functions f and g which will be shown to be the desired $h_{p/q}$ and $\hat{h}_{p/q}$. Let $r = q-p$ and define $f: \{0,1\}^* \to \{0,1\}^*$ as: for all $x \in \{0,1\}^*$, where $x = a_1 a_2 \ldots a_n$, and $a_i \in \Sigma$, $f(x) = a_1^p a_2^p \ldots a_n^p a_1^r a_2^r \ldots a_n^r$. Note that f is not a bounded function since for any x, the length of $f(x)$ is q times the length of x.

We now define a function $g: \{0,1,\epsilon_0\}^* \to \{0,1\}^*$ which is shown to be the m.d.s.a. of f.

1) For $x \in \{0,1\}^*$, where $x = a_1 a_2 \ldots a_n$ and $a_i \in \Sigma$, $g(x) = a_1^p a_2^p \ldots a_n^p$.

2) For $x = w\epsilon_0 y$, where $w \in \{0,1\}^*$, $y \in \{0,1,\epsilon_0\}^*$, $g(x) = f(w)$.

By (2), g is an approximation of f. We show g is sequential by cases.

a) For $x \in \{0,1\}^*$, where $x = a_1a_2 \ldots a_n$, $a_i \in \Sigma$, $g(x) = a_1{}^p\, a_2{}^p \ldots a_n{}^p$, by (1). We get two cases for $v\in\{0,1,\epsilon_0\}^*$.

i) For $v \in\{0,1\}^*$, where $v=b_1b_2 \ldots b_m$, $b_i\in\Sigma$, $g(xv) = a_1{}^pa_2{}^p \ldots a_n{}^pb_1{}^pb_2{}^p \ldots b_m{}^p$, by (1). Hence $g(x)\ Bg(xv)$.

ii) For $v = w\epsilon_0y$, where $w = b_1b_2 \ldots b_m$, $b_i\in\Sigma$ and $y \in \{0,1,\epsilon_0\}^*$, $g(xv) = f(xw) = a_1{}^pa_2{}^p \ldots a_n{}^pb_1{}^pb_2{}^p \ldots b_m{}^p\cdot a_1{}^ra_2{}^r \ldots a_n{}^rb_1{}^rb_2{}^r \ldots b_m{}^r$, by (2). Hence, $g(x)\ Bg(xv)$.

b) For $x = y\epsilon_0y$, where $w\in\{0,1,\epsilon_0\}^*$, $g(xv) = f(w)$, by (2). Hence $g(x)\ Bg(xv)$.

Hence, g is a sequential approximation of f. For $x\in\{0,1\}^*$ and $x=a_1a_2 \ldots a_n$, $a_i\in\Sigma$, $g(x) = a_1{}^pa_2{}^p \ldots a_n{}^p$. Consider $g(x)\cdot 0$ and $f(x.1)$. $f(x1) = a_1{}^pa_2{}^p \ldots a_n{}^p1^pa_1{}^ra_2{}^r \ldots a_n{}^r1^r$, and hence $g(x)\cdot 0 \not B f(x1)$.

Similarly, $g(x)\cdot 1 \not B f(x0)$. Again, by theorem 3.3.2, g is the m.d.s.a. of f. i.e. $g=\hat{f}$.

Now, $M_n = \{x|1\le|x|\le n\ \&\ x \in\{0,1\}^*\}$. Hence, $m_n = 2^{n+1}-2$, $\forall n$, and $(\forall x \in M_n)\ [|\hat{f}(x)|= p + |f(x)|=q)$.

$$\text{Hence, } R_f = \lim_{n\to\infty} \frac{1}{m_n} \sum_{x\in M_n} \frac{|\hat{f}(x)|}{|f(x)|}$$

$$= \lim_{n\to\infty} \frac{1}{2^{n+1}-2} \sum_{i=1}^{2^{n+1}-2} \frac{p}{q}$$

$$= \frac{p}{q}\ . \quad \square$$

We conclude this chapter by some observations concerning the hierarchy defined by C_t. It is not known whether for every real t in the open interval (0,1) there is a function in c_t but in no c_t, for $t_1 < t$. Secondly, it is not known whether the hierarchy C_t bears any meaningful relationship to other known hierarchies. Finally, we note that theorem 4.5.4 can be stated intuitively as: if a sequential Turing machine is used to compute functions, for any rational p, there are functions for which a fraction p of the output cannot be made until after receipt of the endmarker. This result should be contrasted with theorem 3.2.3 which shows that all computable functions can be 'computed' on some sequential Turing machine.

CHAPTER 5

CLASSES OF TIME-BOUNDED FUNCTIONS

5.1 Introduction

We now turn our attention to the study of time-bounded computation of functions. Most authors who have investigated this area have restricted their attention to the class of length-preserving functions, which includes the class of recognition problems. This restriction has led to two basic problems.

First, this restriction has led to the study of a function's complexity mainly in terms of the storage and retrieval of the data needed in the computation of the function. There are two other factors that contribute to the difficulty of computing a function: the size of the output and the off-line nature of the function. In [8], Hartmanis has approached the study of the effect of the size of the output of a function on its complexity; however, no instances are known of the study of the contribution of the off-line nature of a function to its complexity. In this chapter, we will be more interested in the effect on the complexity of a function of these two factors than in the effect of the need for storage and retrieval of data.

A second consequence of the common restriction to length-preserving functions is that the measure of the complexity of a function is related only to the length of the input. It seems natural when discussing the complexity of functions in general to take into account the nature of the output as well.

These measures are especially interesting for the case of a 'real-time' computation. Intuitively, a function is real-time computable if the input is read as fast as possible and the output is produced as fast as possible. This intuitive meaning of real-time is the motivation for finding a more general means of measuring the complexity of a function.

The primary purpose of this chapter is to introduce and to study several classes of functions whose complexity is measured in different ways. Five methods of measurement are used: the length of the input, the length of the output, the maximum of the length of the input and the length of the output, the length of sequential profile of the m.d.s.a. of the function (counting overlapped input and output letters as 1 in the length), and the sum of the length of the input and the length of the output.

We will say that a function f is 'input-bounded $t(n)$ - computable', if there exists a machine M which computes f in at most $t(n)$ steps, where n is the length of the input. We denote the class of input-bounded $t(n)$ - computable functions as $I_{t(n)}$.

We make similar definitions for output-bounded $t(n)$ - computable functions, maximum-bounded $t(n)$ - computable functions, profile-bounded $t(n)$ - computable functions, and total-bounded $t(n)$ - computable functions. We denote the corresponding classes as $O_{t(n)}$, $M_{t(n)}$, $P_{t(n)}$, and $T_{t(n)}$ respectively.

We consider the lattice structure of these classes, along with $I_{t(n)} \cap O_{t(n)}$ and $I_{t(n)} \cup O_{t(n)}$, under set-theoretic containment. For any monotonically strictly increasing function t, $I_{t(n)} \cap O_{t(n)} \subseteq I_{t(n)}$, $I_{t(n)} \cap O_{t(n)} \subseteq O_{t(n)}$, $I_{t(n)} \not\subseteq O_{t(n)}$, $O_{t(n)} \not\subseteq I_{t(n)}$, $I_{t(n)} \subseteq I_{t(n)} \cup O_{t(n)}$, $O_{t(n)} \subseteq I_{t(n)} \cup O_{t(n)} \subseteq M_{t(n)} \subseteq P_{t(n)} \subseteq T_{t(n)}$.

We next investigate the case when $t(n) = n$, (i.e., the "real-time" case). For this case, the inclusions stated above are shown to be proper.

Furthermore, there is a gsm mapping $f: \{0,1\}^* \rightarrow \{0,1\}^*$ such that $f \notin M_n$. However, if g is a gsm mapping, $g \in P_n$. Since the author is of the opinion that gsm mappings satisfy the intuitive notion of a real-time computable function, we are led to reject the classes I_n, O_n, and M_n as representatives of the concept of real-time computable functions and focus attention on P_n.

The class of real-time functions should include only sequential functions. However, P_n includes nonsequential functions while I_n does not. We are thus led to consider restricting P_n to the class of functions which are both sequential and profile bounded. We denote this class as S_n.

We next consider our classes of real-time functions in comparison to the more common definition, which is limited to the 'essentially length-preserving' functions. We show that the class of essentially length-preserving functions which are sequential and profile-bounded real-time computable (denoted S_n^{ℓ}) is exactly the class $I_n \cap O_n$, which is the more common class investigated. From these considerations, the author is of the opinion that S_n fully characterizes the class of real-time computable functions.

We next investigate the classes of 'linear-time computable' functions: (i.e., $t(n) = c.n$). For this case, $T_{c.n} = M_{c.n}$ and the lattice collapses to five classes, $I_{c.n} \cap O_{c.n}$, $I_{c.n}$, $O_{c.n}$, $I_{c.n} \cup O_{c.n}$ and $M_{c.n}$.

Finally, we study the problem of how the off-line nature of a computation affects the complexity classes. It is shown that for any constant $c > 0$, there exists a function f such that $f \in M_{n+c}$, but $f \notin M_{n+c-1}$. This result, which holds for all classes in the lattice contained in M_n and M_{n+c}, shows that there are infinitely many distinct classes between real-time and linear-time.

5.2 Time-bounded Computations

We now turn out attention to the problem of measuring the difficulty or 'complexity' of a function in terms of the number of steps a machine computing the function takes to achieve the result. Most authors who have previously studied this area have been interested in recognition problems, generation problems or length-preserving transductions. Note that both recognition problems and generation problems can be stated as length-preserving transductions in the following sense. A recognition problem over some finite alphabet Σ is a mapping from Σ^* to $\{0,1\}^*$ with a 0 or 1 produced for each new input letter read depending upon whether the substring already read is rejected or accepted, respectively. For a generation problem, the input letter is ignored in the computation and is used only as a counter of how many digits of output are to be produced. Thus, a generation problem is mapping from Σ^* to Γ^* with an input letter read before each output letter is produced.

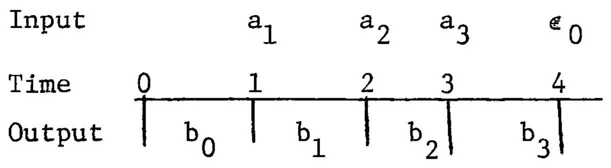
In order to measure the complexity of a function f, we associate a step counting function $t_M(x)$ with a machine M computing f. Because of the bias towards length-preserving transductions, the variable in which $t_M(x)$ is expressed usually is considered to be the length of the input. However, we are interested in general transductions and we are led to consider new ways of measuring the complexity in terms of x and $f(x)$. Since it is not clear exactly in which way the measure should be made, we will introduce five alternatives.

Before doing so, we first clarify our concept of the operation of a machine and of the step counting function, $t_M(x)$. We consider the machine to have a read-only one-way input tape and a write-only one-way output tape, as well as work tapes. The machine begins operation with the input tape positioned so that the read-head is located one square to the left of the first input letter and with the output tape positioned so that the write-head is over a blank square. We make the convention that the machine acts as though the square to the left of the start of the input word always contains an endmarker ϵ_F, regardless of what is really present. To the right of the input word is an endmarker ϵ_0 which acts as an end-of-file marker. For the case of a sequential Turing machine, ϵ_0 may be regarded as an input causing the machine to halt immediately upon reading it. For the off-line Turing machine, ϵ_0 must be in the machine's alphabet to allow for additional computation after encountering ϵ_0.

We make the convention that a machine halts if there is no transition specified for the current state-symbol(s) configuration and that the machine must read all of the input string, including ϵ_0. For IO-profiles (see section 2.4), we make the convention that if ϵ_0 appears on the right end of the profile, we do not record it.

The number of steps $t_M(x)$, taken in the computation of a machine M with input x, is the number of state transitions M goes through before halting. Hence, $t_M(x) \geq |x| + 1$, by the above discussion. We assume that an input letter and the last letter of the preceding output string can be overlapped in the same step, with the input available on the time step and the output produced between time steps.

For example, if the IO-profile of a machine M with input x is $P_{IO}(M,x) = b_0\ a_1\ b_1\ a_2\ b_2\ a_3\ b_3$, we would have the following timing if output occurred at every possible occasion.



In this case, $t_M(a_1\ a_2\ a_3) = 4$. In a sense, our machine model is a Moore device. In order to simplify our definitions, we will use the following notation to indicate the synthetic lengths of strings.

Notation For an input string x, we let $\ell_I(x) = |x| + 1$. We add 1 to the length of the string in order to reflect the fact that the end-marker ϵ_0 must also be read by the machine.

Notation For an output string $f(x)$ produced by x, we let

$$\ell_0(x) = \begin{cases} |f(x)|+1, & \text{if } \hat{f}(\lambda) = \lambda \\ |f(x)| & \text{otherwise} \end{cases}$$

If $\hat{f}(\lambda) = \lambda$, it is not possible to produce an output before the first input has been read. Since such an output could have been overlapped with the first input (between times 0 and 1) and could not be produced, the timing would be off and we compensate by adding 1.

Notation The synthetic length, $\ell_p(x)$, of the sequential profile of the m.d.s.a. of f with input $x\epsilon_0$ will be the sum of the number of output digits and the number of input digits (including ϵ_0) not overlapped (i.e., do not immediately follow an output in the profile).

For example, if $P_s\ (\hat{f}, x\epsilon_0) = a_1\ a_2\ b_1\ b_2\ b_3\ a_3\ b_4\ \epsilon_0\ b_5$, $1_p(x) = 7$, (where $f:\Sigma^* \to \Gamma^*$, $a_i \in \Sigma$, $b_i \in \Gamma$). If $P_s\ (\hat{f}, x\epsilon_0) = b_0\ a_1\ b_1\ a_2\ b_2\ \epsilon_0$, $1_p(x) = 3$, (where $f:\Sigma^* \to \Gamma^*$, $a_i \in \Sigma$, $b_i \in \Gamma$). Note that in the last example all of the inputs including the final ϵ_0 are overlapped by an output digit.

Definition 5.2.1 A function $t:N_+ \to N_+$ is a limiting function if t is monotonically strictly increasing and $(\forall n \in N_+)\ [t(n) \geq n]$.

Definition 5.2.2 For a limiting function $t:N_+ \to N_+$, the class $I_{t(n)}$ of input-bounded t(n)-computable functions, is defined as $I_{t(n)} = \{f \mid$ there exists a machine M which computes $f:\Sigma^* \to \Gamma^*$ such that $(\forall n \in N_+)\ (\forall x \in \Sigma^*)[\ell_I(x) \le n \Rightarrow t_M(x) \le t(n)]\}$.

The class $I_{t(n)}$ is the complexity class which is normally defined in previous works.

Definition 5.2.3 For a limiting function $t:N_+ \to N_+$, the class $O_{t(n)}$ of output-bounded t(n)-computable functions is defined as $O_{t(n)} = \{f \mid$ there exists a machine M which computes f such that $(\forall n \in N_+)$ $(\forall x \in \Sigma^*)[\ell_O(x) \le n \Rightarrow t_M(x) \le t(n)]\}$.

$O_{t(n)}$ is obviously the class of t-honest functions [14]. Since we are measuring the complexity of a function in terms of the length of its output, the off-line nature of the function and the storage-retrieval of data are the remaining factors which make one function more complex than another.

Theorem 5.2.4 For any limiting function $t:N_+ \to N_+$, $O_{t(n)} \not\subseteq I_{t(n)}$.

Proof: For any limiting function $t:N_+ \to N_+$, there exists a real-time countable function (cf [17]) $g:N_+ \to N_+$ such that $(\forall n \in N_+)[t(n) < g(n)]$[1]. We define $f:\Sigma^* \to \{1\}^*$ as $(\forall x \in \Sigma^*)[f(x) = 1^{g(\ell_I(x))}]$. That is, produce $g(|x| + 1)$ 1's for an input x. Thus, $\ell_O(x) = g(\ell_I(x))$, since $\hat{1}Bf(\lambda)$. Since g is real-time countable, there exists a machine M computing f such that $(\forall x \in \Sigma^*)[t_M(x) = g(\ell_I(x))]$.

[1]The class of real-time countable functions as introduced by Yamada [17], map from N to N. In order to change to N_+, we make the obvious changes. That is, if $g':N \to N$ is real-time countable, we define $g:N_+ \to N_+$ as $g(n) = g'(n-1) + 1$.

Hence, if we choose any $x \in \Sigma^*$ and let $n_0 = |x| + 1$, we have $[\ell_I(x) \le n_0 \;\&\; t(n_0) < g(n_0) = g(\ell_I(x)) = t_M(x)]$, and it follows that $f \notin I_{t(n)}$. Furthermore, $(\forall n_1 \in N_+)(\forall x \in \Sigma^*)$ $[\ell_0(x) \le n_1 \Rightarrow t_M(x) = g(\ell_I(x)) = \ell_0(x) \le n_1]$. Thus, $f \in 0_n$, and hence, $f \in 0_{u(n)}$ for <u>any</u> limiting function $u:N_+ \to N_+$; therefore, $f \in 0_{t(n)}$. □

Before showing that it is also true that $I_{t(n)} \not\subseteq 0_{t(n)}$, we indicate a problem with the class $0_{t(n)}$.

<u>Lemma 5.2.5</u> Let $f:\Sigma^* \to \Gamma^*$ be a bounded function. There exists no limiting function $t:N_+ \to N_+$ such that $f \in 0_{t(n)}$.

<u>Proof</u>: Since f is a bounded function , there exists a nonnegative k such that $(\forall x \in \Sigma^*)[\,|f(x)| \le k]$. Let M be any machine computing f. Now, $(\forall x \in \Sigma^*)[t_M(x) > |x|\,]$ since M must read all of its input. Hence, if $f \in 0_{t(n)}$, $(\forall x \in \Sigma^*)[\,|x| < t_M(x) \le t(k)]$, a contradiction. □.

<u>Theorem 5.2.6</u> For any limiting function $t:N_+ \to N_+$, $I_{t(n)} \not\subseteq 0_{t(n)}$.

<u>Proof</u>: Since f_λ, the zero function, is a bounded function, then $f_\lambda \notin 0_{t(n)}$ for any limiting function $t:N_+ \to N_+$, by lemma 5.2.5. However, we can construct a machine M which simply reads the input and halts on encountering ϵ_0. Hence, $(\forall n \in N_+)(\forall x \in \Sigma^*)$ $[\ell_I(x) \le n \Rightarrow t_M(x) \le n]$, and for any limiting function $t:N_+ \to N_+$, $f_\lambda \in I_{t(n)}$. □

The next theorem is a stronger version of theorem 5.2.6.

<u>Theorem 5.2.7</u> For any limiting function $t:N_+ \to N_+$, there exists an unbounded function $f:\{1\}^* \to \{1\}^*$ such that $f \notin 0_{t(n)}$, $f \in I_{t(n)}$.

<u>Proof</u>: We proceed as in theorem 5.2.4 to find a function $f:N_+ \to N_+$ and a machine M which computes f such that $t(\ell_0(x)) < t_M(x) \le t(\ell_I(x))$.

Stage i, i $\geq$ 1 On each step, M reads one input symbol while simultaneously computing $t(i+1)$. When the computation of $t(i+1)$ is finished, M writes exactly $t(i+1)$ 1's on a work tape. After doing this, M outputs a 1 and goes to stage $i+1$. M halts when ϵ_0 is read.

Note that while M is computing $t(i + 1)$ and writing the $t(i + 1)$ 1's at least $t(i + 1)$ inputs will be read. Furthermore, $t_M(x) = \ell_I(x)$, for all x. Also, note that M will produce arbitrarily long output strings for long enough input strings and f will be unbounded.

M produces i output letters in more than $t(i + 1)$ steps, by the above observation. Hence, if we choose x sufficiently long so that stage 1 is completed and choose n_1 to be $|f(x)| + 1$ we obtain $[\ell_0(x) \leq n_1 = i + 1 \; \& \; t(n_1) = t(i + 1) < t_M(x)]$ and $f \notin O_{t(n)}$. Furthermore, $(\forall n \in N_+)(\forall x \in \Sigma^*) [\ell_I(x) \leq n \Rightarrow t_M(x) = \ell_I(x) = n \leq t(n)$, for any limiting function $t:N_+ \to N_+]$ and $f \in I_{t(n)}$. □

Remark From theorems 5.2.4 and 5.2.6, we know that $I_{t(n)} \cap O_{t(n)} \nsubseteq I_{t(n)} \nsubseteq I_{t(n)} \cup O_{t(n)}$ and $I_{t(n)} \cap O_{t(n)} \nsubseteq O_{t(n)} \nsubseteq I_{t(n)} \cup O_{t(n)}$.

We have noted that the class $I_{t(n)}$ seems too restrictive and by lemma 5.2.5, there are functions which are not in $O_{t(n)}$, for any t . Hence, we now consider three classes based on a combination of the length of the input and the length of the output. We will leave the discussion of the intuitive meanings to the next section.

Definition 5.2.8 For a limiting function $t:N_+ \to N_+$, the class $M_{t(n)}$ of maximum-bounded $t(n)$-computable functions is defined as $M_{t(n)} = \{f|$ there exists a machine M which computes $f:\Sigma^* \to \Gamma^*$ such that $(\forall n \in N_+)(\forall x \in \Sigma^*) [\max(\ell_I(x), \ell_0(x)) \leq n \Rightarrow t_M(x) \leq t(n)]\}$.

Equivalently, we could define $M_{t(n)}$ as $M_{t(n)} = \{f|$ there exists a machine M which computes $f:\Sigma^* \to \Gamma^*$ such that $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) \le n \;\&\; \ell_0(x) \le n) \Rightarrow t_M(x) \le t(n)]\}$.

Theorem 5.2.9 For any limiting function $t:N_+ \to N_+$, $I_{t(n)} \cup O_{t(n)} \subseteq M_{t(n)}$.

Proof: Let $f:\Sigma^* \to \Gamma^*$. Suppose $f \in I_{t(n)}$, then there exists a machine M such that $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) \le n \Rightarrow t_M(x) \le t(n)]$. Hence, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) \le n \;\&\; \ell_0(x) \le n \Rightarrow \ell_I(x) \le n \Rightarrow t_M(x) \le t(n)]$. and $f \in M_{t(n)}$. Similarly, if $f \in O_{t(n)}$, $f \in M_{t(n)}$.□

When the classes $M_{t(n)}$ and $I_{t(n)} \cup O_{t(n)}$ are equal is discussed later in this chapter.

Definition 5.2.10 For any limiting function $t:N_+ \to N_+$, the class $P_{t(n)}$ of profile-bounded t(n)-computable functions, is defined as $P_{t(n)} = \{f|$ there exists a machine M which computes $f:\Sigma^* \to \Gamma^*$, such that $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_p(x) \le n \Rightarrow t_M(x) \le t(n)]\}$.

Theorem 5.2.11 For any limiting function $t:N_+ \to N_+$, $M_{t(n)} \subseteq P_{t(n)}$.

Proof: By definition, if $\ell_p(x) \le n$, then $\ell_0(x) \le n$. Suppose $\ell_I(x) = m + k > n$ and $\ell_0(x) = k \le n$. Then, at least m input letters could not be preceded by any output letters in $P_s(\hat{f},x)$. Hence, $\ell_p(x) \ge m + k > n$. Hence, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_p(x) \le n \Rightarrow \ell_0(x) \le n \;\&\; \ell_I(x) \le n)]$. But if $f \in M_{t(n)}$, then $(\forall n \in N_+)(\forall x \in \Sigma^*)$ $[\ell_0(x) \le n \;\&\; \ell_I(x) \le n \Rightarrow t_M(x) \le t(n)]$. Hence, $(\forall n \in N_+)(\forall x \in \Sigma^*)$ $[\ell_p(x) \le n \Rightarrow (\ell_0(x) \le n \;\&\; \ell_I(x) \le n) \Rightarrow t_M(x) \le t(n)]$ and $f \in P_{t(n)}$. □.

We will again leave the discussion of when $P_{t(n)} = M_{t(n)}$ to the next sections.

Definition 5.2.12 For any limiting function $t:N_+ \to N_+$, the class of $T_{t(n)}$ total-bounded t(n)-computable functions, is defined as $T_{t(n)} = \{f|$ there exists a machine M which computes $f:\Sigma^* \to \Gamma^*$ such that

$(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) + \ell_O(x) \le n \Rightarrow t_M(x) \le t(n)]\}$.

Theorem 5.2.13 For any limiting function $t:N_+ \to N_+$, $P_{t(n)} \subseteq T_{t(n)}$.

Proof: Let $f:\Sigma^* \to \Gamma^*$. For any input $x \in \Sigma^*$, $\ell_p(x) \le \ell_I(x) + \ell_O(x)$. Suppose $f \in P_{t(n)}$. Hence, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_p(x) \le n \Rightarrow t_M(x) \le t(n)]$. Hence, $(\forall x \in \Sigma^*)[\ell_O(x) + \ell_I(x) \le n \Rightarrow \ell_p(x) \le n \Rightarrow t_M(x) \le t(n)]$ and $f \in T_{t(n)}$. □

We can summarize the theorems in this section in the following diagram, where a line connects two classes if the lower class is contained in the upper.

$$\begin{tikzcd}
 & T_{t(n)} \arrow[d, dash] & \\
 & P_{t(n)} \arrow[d, dash] & \\
 & M_{t(n)} \arrow[d, dash] & \\
 & I_{t(n)} \cup O_{t(n)} \arrow[dl, dash] \arrow[dr, dash] & \\
I_{t(n)} \arrow[dr, dash] & & O_{t(n)} \arrow[dl, dash] \\
 & I_{t(n)} \cap O_{t(n)}. &
\end{tikzcd}$$

5.3 Real-Time Computations

In the real-world sense, a real-time computation is one in which the output must be generated as quickly as the input is entered. For example, consider a process control unit for a nuclear reactor. Time in such a sensitive application is of the essence to ensure the safe usage of the reactor. However, for any given input, more than one command from the control unit may be issued. Thus, the common restriction to functions whose length of output is at most one greater than the length of input does not seem to capture the real-world sense.

In this section, we propose to try to choose the proper class of functions which best fits our intuitive notion of a real-time computation. To do so, we will study the classes introduced in section 5.2 with $t(n) = n$. We first give an intuitive definition of each class.

A function f is input-bounded n-computable if there exists a machine computing f which reads on every step and halts if the endmarker is encountered. Obviously, an input-bounded n-computable function cannot have an output whose length is more than one greater than the length of the input.

A function f is output-bounded n-computable if there exists a machine computing f which writes during every step (except perhaps the first) and halts after producing the last digit. Note that f_λ is not output-bounded real-time computable by theorem 5.2.5.

A function f is maximum-bounded n-computable if there exists a machine computing f which reads on every step and halts on reaching the endmarker in the case that the length of the input is greater than the length of the output or which writes on every occasion (except perhaps the first) and halts after producing the last digit, otherwise.

A function f is profile-bounded n-computable if there exists a machine computing f which reads immediately on producing the last digit (if any) of the output string produced by the previous input string and produces an output digit at every possible occasion.

The concept of total-bounded n-computable functions cannot be expressed in terms of the operation of the IO of the machines computing them.

We now show that the inclusions shown in the previous section are proper for the real-time case.

Theorem 5.3.1 $I_n \cup O_n \neq M_n$

Proof: We define a function $f:\{0,1\}^* \to \{0,1\}^*$ such that half of the inputs are longer than the outputs and half are shorter.

1). $f(\lambda) = \lambda$;

2). for $x \in \Sigma^*$, $f(0x) = \lambda$;

3). for $x \in \Sigma^*$, $x = a_1\, a_2\, a_3 \ldots a_n$, $a_i \in \{0,1\}$,

$f(1x) = 11\, a_1\, a_1\, a_2\, a_2 \ldots a_n\, a_n$.

That is, if an input begins with a 0, no output is made. If an input begins with a 1, each letter of the input is produced twice in the output string. Thus, $t_M(0v) \geq (|0v| + 1)$, $t_M(1v) \geq (2.\,|1v| + 1)$, for any $v \in \{0,1\}^*$.

Suppose $f \in I_{t(n)}$, then $(\forall n_0 \in N_+)$ $(\forall x \in \{0,1\}^*)[\ell_I(x) = n_0 \Rightarrow t_M(x) = t(n_0)]$. Thus, $(2.|1v| + 1) \leq t_M(1v) \leq t(|1v| + 1)$. Hence, $t(n_0) \geq 2n_0 + 1$ and $f \notin I_n$. Suppose $f \in O_{t(n)}$, Then $(\forall n_1 \in N_+)(\forall x \in \{0,1\}^*)[\ell_0(x) \leq n_1 \Rightarrow t_M(x) \leq t(n_1)]$. Furthermore, $\ell_0(0v) = 1$. Consequently, $(|0v| + 1) \leq t_M(0v) \leq t(1)$ and t must be unbounded. Hence, $f \notin O_n$.

Now, we construct a machine M which computes f. If the first input is a 0, M reads the input string and halts when ϵ_0 is encountered. Now, $\ell_I(0v) = |0v| + 1$ and $\ell_0(0v) = 1$. Hence, max $(\ell_I(0v), \ell_0(0v)) = \ell_I(0v)$ and by construction $t_M(0v) = \ell_I(0v)$. If the first letter of the input string is a 1, M reads on every other step and produces two copies of the input in between. M halts if ϵ_0 is encountered. Now, $\ell_I(1v) = |1v| + 1$ and $\ell_0(1v) = 2.|1v| +1$ Hence, max $(\ell_I(1v), \ell_0(1v)) = \ell_0(1v)$ and by construction $t_M(1v) = \ell_0(1v)$. Furthermore, $t_M(\lambda) = 1 = \ell_I(\lambda) = \ell_0(\lambda)$. Hence, $(\forall n \in N_+)$

$(\forall x \in \{0,1\}^*)[\max(\ell_I(x), \ell_0(x)) \leq n \Rightarrow t_M(x) = \max(\ell_I(x), \ell_0(x)) \leq n]$ and $f \in M_n$. □

<u>Theorem 5.3.3</u> $M_n \neq P_n$.

<u>Proof</u>: We define a function $f:\{0,1\}^* \to \{0,1\}^*$ such that for some inputs x, $\ell_0(x) < \ell_I(x)$ and $t_M(x) > \ell_I(x)$.

1). $f(\lambda) = \lambda$;

2). for $x \in \{0,1\}^*$, $f(x0) = f(x)$;

3). for $x \in \{0,1\}^*$, $f(x1) = f(x)01$

f is the gsm mapping which outputs a 01 for every 1 in the input string and ignores the 0's.

Let $x = 001$. Hence, $f(x) = 01$, and $P_s(\hat{f}, x\,\epsilon_0) = 00101\epsilon_0$. Furthermore, $\ell_I(x) = 4$, $\ell_0(x) = 3$ and $\ell_p(x) = 5$. Let M be any machine computing f. M must perform the three reads before any output could be generated (or else M would not be computing f). Since two outputs are produced, $t_M(x) \geq 5$. Thus, choosing $x = 001$ and $n_1 = 4$, we obtain $[\max(\ell_I(x), \ell_0(x)) = 4 \leq n_1 \;\&\; t_M(x) > n_1]$ and $f \notin M_n$.

We now construct a machine which computes f in $\ell_p(x)$ steps for an input x.

<u>Stage i</u>, $i > 0$ M reads the i^{th} input letter a_i. If $a_i = \epsilon_0$, M halts. If $a_i = 0$, M goes to stage $i + 1$. If $a_i = 1$, M produces a 0 followed by a 1 and goes to stage $i + 1$.

Now, for any $x \in \{0,1\}^*$, let $k \geq 0$ be the number of 1's in x and $j \geq 0$ be the number of 0's in x. Then the number of output digits is $2.k$. The number of unoverlapped input digits is $P_s(\hat{f}, x\,\epsilon_0)$ is $j + 1$, since there are $k + j + 1$ input digits (including ϵ_0) and only k outputs are followed by an input letter in $P_S(\hat{f}, x\,\epsilon_0)$, $(\hat{f}(x) = f(x)$, as f is sequential). Thus, $\ell_p(x) = 2.k + j + 1$.

Now, by construction M takes $2.k$ steps to produce the output. Of the inputs (including ϵ_0) $j + 1$ are not overlapped in a step with an output produced by the previous input. Hence, $t_M(x) = 2k + j + 1 = \ell_p(x)$. Thus, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_p(x) \leq n \Rightarrow t_M(x) = \ell_p(x) \leq n]$ and $f \in P_n$. □

Theorem 5.3.4 $P_n \neq T_n$.

Proof: We will define a function f: $\{0,1,s\}^* \to \{0,1\}^*$ from a function which Hartmanis and Stearns[9] showed not to be in I_n.

We define $f:\{0,1,s\}^* \to \{0,1\}^*$ as:

1). for $x \in \{0,1\}^*$, $f(x) = 0^{|x|+1}$

2). for $x = u\ v\ s\ w\ y$, where $u \in \lambda \cup \{0,1,s\}^* s$
$v \in \{0,1\}^*$, $w \in \{\{0,1\}^* \cdot s\}^*$, $y \in \{0,1\}^*$,

$$f(x) = \begin{cases} 0^{|x|}1, & \text{if } y = Im(v) \\ 0^{|x|+1}, & \text{otherwise} \end{cases}$$

We will call the strings over $\{0,1\}^*$ which are surrounded by s's or surrounded by an endmarker and an s as (0,1)-words. Thus, f is the function which produces a 0 for every input letter and a 0 or 1 if the last (0,1)-word is the mirror image (reversal) of some previous (0,1)-word.

Now, $(\forall x \in \{0,1, s\}^*)[\hat{f}(x) = 0^{|x|}]$ and $(\forall x \in \{0, 1, s\}^*)$ $[\hat{f}(x\ \epsilon_0) = f(x)]$. Also, if $x = a_1\ a_2 \ldots a_n$, $P_S(\hat{f}, x\ \epsilon_0) = a_1\ 0\ a_2\ 0\ a_3 \ldots 0a_n\ 0\ \epsilon_0 i$, where $i \in \{0,1\}$. Hence, $\ell_p(x) = |x| + 2$ ($|f(x)| = |x| +1$ and the first input is not overlapped). Also, $\ell_I(x) + \ell_0(x) = 2.|x| + 3$.

Now suppose $f \in P_n$. Thus, there exists a machine T with d states, M tapes and at most a choice of k symbols per square on the work tapes and T produces $f(x)$ for an input x in at most $\ell_p(x)$ steps. Now, as in [9], we can bound the number of past histories T can look up in $i + 2$ steps by $d.k^{(2i + 5)m}$. Suppose T has been operating for some number of steps and has an s under its read-head. There are 2^i possible (0,1)-words of length i and 2^{2^i} subsets of these words that T may have to look up to see if the last (0,1)-word is the mirror image of one of these words. Now, if x is of length i, then T must produce the final digit of f(x) in at most $i + 2$ steps. Now, for large enough i, $2^{2^i} > d.k^{(2i+5)}$. But then T could not compute f(x), a contradiction. Hence, $f \notin P_n$.

We now define a machine M which computes f in $2.|x| + 3$ steps for any input x.

Stage $i \geqslant 1$ M reads an input. If the input is ϵ_0 the machine goes to the end-routine. If not, M copies the input onto a buffer[2] and a work tape and produces a 0 before going to stage $i + 1$.

End-routine M begins comparing the reverse of last (0,1)-word from one of the work tapes to the (0,1)-words in the buffer. When it reaches the end of each (0,1)-word from the buffer, M outputs a 1 if the last (0,1)-word is the reverse of the (0,1)-word from the buffer. Otherwise, it returns to the end of the last (0,1)-word and tries the next (0,1)-word from the buffer. If it depletes the buffer, it produces a 0 whether the word is the reverse of itself or not.

For the end-routine, $2.|x| + 1$ steps are taken. We can however use the techniques of the constant speedup of Hartmanis and Stearns [9] to get $|x| + 1$ steps. Hence, $(\forall x \in \{0, 1, s\}^*)[t_M(x) = 2.|x| + 2 < \ell_I(x) + \ell_0(x)]$.

[2]Fischer, Meyer and Rosenberg have shown that a buffer can be simulated by a Turing machine with enough tapes without loss of time in [6].

Therefore, $(\forall n \in N_+)(\forall x \in \{0,1,s\}^*)[(\ell_I(x) + \ell_0(x)) \le n \Rightarrow t_M(x) < \ell_I(x) + \ell_0(x) \le n]$ and $f \in T_n$. □

We make two notes concerning theorems 5.3.3 and 5.3.4. First, note that T_n contains some nonsequential functions. The real-world sense of 'real-time' implies that all of the result for a given input is produced before the next input letter is read. Thus, a realistic representative of the class of 'real-time' computable functions, should be limited to sequential functions. We thus reject the class T_n as too broad a class.

In theorem 5.3.3, we showed that there is a gsm mapping f such that $f \notin M_n$. As was the case for an on-going process, in the author's opinion the class of gsm mappings should be included in any class purporting to represent the concept of 'real-time'. Thus, we are led to reject the class M_n as too narrow a class.

Theorem 5.3.5 Let $f:\Sigma^* \to \Gamma^*$ be a gsm mapping. Then, $f \in P_n$.

Proof: In operation a gsm computes a gsm mapping in $\ell_p(x)$ steps, if we allow an input to be overlapped with the previous output letter. Obviously, we can construct a Turing machine which matches the speed of a gsm while computing f. Hence, there exists a machine M such that $(\forall x \in \Sigma^*)[t_M(x) = \ell_p(x)]$ and hence, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_p(x) \le n \Rightarrow t_M(x) \le n]$ and $f \in P_n$. □

We now show that P_n, like T_n, is too broad a class.

Theorem 5.3.6 There exists a nonsequential function $f:\{0,1\}^* \to \{0,1\}^*$ such that $f \in P_n$.

Proof: Consider $f:\{0,1\}^* \to \{0,1\}^*$ defined as:

1). for $x = \lambda$, $f(\lambda) = \lambda$;

2). for $x = u.0$, where $u \in \{0,1\}^*$, $f(x) = 0.f(u)$;

3). for $x = u.1$, where $u \in \{0,1\}^*$, $f(x) = 1.f(u)$.

That is, f is the mirror image function as defined in theorem 2.2.3. Now, we know that f is not sequential and $\hat{f}:\{0,1,\epsilon_0\}^* \to \{0,1\}^*$ is defined as: $(\forall x \in \{0,1\}^*)[\hat{f}(x) = \lambda$ & $\hat{f}(x\,\epsilon_0) = f(x)]$. Hence, for $x \in \{0,1\}$, $P_s(\hat{f}, x\,\epsilon_0) = x\,\epsilon_0\ f(x)$ and $\ell_p(x) = 2.|x| + 1$. We define a machine M which reads the input tape and copies it input by input onto a work tape. When ϵ_0 is encountered, M copies in reverse from the work tape to the output tape, then halts. Since the read-head of the work tape is over the last input letter, M produces $f(x)$ for an input x. Furthermore, $t_M(x)$ = the number of reads executed plus the number of writes = $(|x| + 1) + |x| = 2.|x| + 1$, for any $x \in \{0,1\}^*$. Hence, $(\forall n \in N_+)(\forall x \in \{0,1\}^*)[\ell_p(x) \leq n \Rightarrow t_M(x) = \ell_p(x) \leq n]$ and $f \in P_n$. □

Since P_n is the smallest class we have defined that contains the gsm mappings, we will limit P_n to the sequential functions.

Notation We denote as S_n the class of sequential functions f such that $f \in P_n$.

Remark If f is a gsm mapping, $f \in S_n$.

Proof: By theorem 5.3.5, $f \in P_n$ if f is a gsm mapping. By theorem 2.3.3, every gsm mapping is sequential. □.

We are interested in showing that the class S_n contains those functions which were considered to be 'real-time' computable by previous authors. We note that this implies that the functions are 'length-preserving' in the following sense.

Definition 5.3.7 A function $f:\Sigma^* \to \Gamma^*$ is called essentially length-preserving if $(\forall x \in \Sigma^*)[\ell_I(x) = \ell_0(x)]$.

We use 'essentially length-preserving' instead of 'length-preserving' because we allow one output to be produced before any input is read. That is, we allow both Moore and Mealy type computations to be essentially length-preserving.

Notation We denote the class of essentially length-preserving functions f such that $f \in I_n$ (O_n, M_n, S_n, P_n) as I_n^{ℓ} $(O_n^{\ell}, M_n^{\ell}, S_n^{\ell}, P_n^{\ell})$

Essentially, I_n^{ℓ} is the class of functions previous authors normally called 'real-time' computable. We first show that for essentially length-preserving functions, the classes M_n^{ℓ}, I_n^{ℓ}, O_n^{ℓ} are the same class.

Theorem 5.3.8 $M_n^{\ell} = I_n^{\ell} = O_n^{\ell}$.

Proof: Since $I_n \subseteq M_n$ and $O_n \subseteq M_n$, it follows that $I_n^{\ell} \subseteq M_n^{\ell}$ and $O_n^{\ell} \subseteq M_n^{\ell}$. Now, suppose $f \in M_n^{\ell}$. That is, $(\forall x \in \Sigma^*)[\ell_I(x) = \ell_0(x) = \max(\ell_I(x), \ell_0(x))]$. Since $f \in M_n$, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\max(\ell_I(x), \ell_0(x)) = \ell_I(x) \leq n \Rightarrow t_M(x) \leq n]$ and $f \in I_n^{\ell}$. Similarly, $f \in O_n^{\ell}$ if $f \in M_n^{\ell}$. □

We now characterize I_n^{ℓ}.

Theorem 5.3.9 $I_n^{\ell} = I_n \cap O_n$.

Proof: By theorem 5.3.8, $I_n^{\ell} = O_n^{\ell} = I_n^{\ell} \cap O_n^{\ell}$. Furthermore, $I_n^{\ell} \subseteq I_n$ and $O_n^{\ell} \subseteq O_n$. Hence, $I_n^{\ell} = I_n^{\ell} \cap O_n^{\ell} \subseteq I_n \cap O_n$.

Now suppose $f \in I_n \cap O_n$, $f:\Sigma^* \to \Gamma^*$. We need only show that f is essentially length-preserving. Since $f \in I_n, (\forall n_0 \in N_+)$ $(\forall x \in \Sigma^*)[\ell_I(x) \leq n_0 \Rightarrow t_M(x) \leq n_0]$. Now, $t_M(x) \geq \ell_I(x)$ for each x, since M must read every input letter and ϵ_0 before halting. Thus, letting $n_0 = \ell_I(x)$ we obtain $(\forall n_0 \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) = n_0 \Rightarrow t_M(x) \leq n_0 = \ell_I(x)]$. Hence, $(\forall x \in \Sigma^*)[\ell_I(x) \leq t_M(x) \leq \ell_I(x)]$, and we can conclude that $(\forall x \in \Sigma^*)[t_M(x) = \ell_I(x)]$.

Now, since $f \in O_n$, $(\forall n_1 \in N_+)(\forall x \in \Sigma^*)[\ell_0(x) \leq n_1 \Rightarrow t_M(x) \leq n_1]$. But M must produce every output letter. Therefore $t_M(x) \geq |f(x)|$.

If $\hat{f}(\lambda) = \lambda$, then M cannot produce the first output until during step 2. Thus, $t_M(x) \geq \ell_0(x)$. Replacing n_1 by $\ell_0(x)$, we obtain $(\forall n_1 \in N_+)(\forall x \in \Sigma^*)[\ell_0(x) = n_1 = t_M(x) \leq n_1 = \ell_0(x)]$. Hence, $(\forall x \in \Sigma^*)[\ell_0(x) \leq t_M(x) \leq \ell_0(x)]$ and we can conclude that $(\forall x \in \Sigma^*)[t_M(x) = \ell_0(x)]$.

Therefore, $(\forall x \in \Sigma^*)[\ell_I(x) = t_M(x) = \ell_0(x)]$ and f is essentially length-preserving. □

We now will prove a theorem which will be useful in proving that S_n contains I_n^ℓ.

<u>Theorem 5.3.10</u> Let $f:\Sigma^* \to \Gamma^*$. If $f \in I_n$, f is sequential.

<u>Proof</u>: Suppose f were not sequential. Therefore, by theorem 3.4.2, $(\exists x_1 \in \Sigma^*)[\hat{f}(x_1) \neq f(x_1)]$. Consider $t_M(x_1)$. At best, M must read $\ell_I(x_1)$ times. But there must exist some string $w \in \Gamma^+$ such that $f(x_1) = vw$, where v is the string produced by M before reading ϵ_0, or $\hat{f}(x_1) = f(x_1)$. Thus, we can conclude that $t_M(x_1) > \ell_I(x_1)$. Letting $x = x_1$ and $n = \ell_I(x_1)$, we obtain $(\exists n \in N_+)(\exists x \in \Sigma^*)[\ell_I(x) \leq n \ \&\ t_M(x) > n]$ and $f \notin I_n$, a contradiction. □

<u>Corollary 5.3.11</u> $P_n^\ell \neq I_n^\ell$

<u>Proof</u>: Since $I_n^\ell \subseteq I_n$, I_n^ℓ contains only sequential functions. However, theorem 5.3.6 exhibits an essentially length-preserving function f such that $f \in P_n$. □

For some functions f which are almost sequential but not sequential, we will find that $\ell_I(x) < t_M(x) \leq \ell_I(x) + c$, for all x and some constant c. These classes are studied later.

We are now in a position to show that S_n contains I_n^ℓ.

<u>Theorem 5.3.12</u> $I_n \cap O_n = S_n^\ell$.

<u>Proof</u>: Let $f:\Sigma^* \to \Gamma^*$. Suppose $f \in I_n \cap O_n$. Hence, $f \in I_n$ and f is sequential by theorem 5.3.10. Since $I_n \cap O_n \subseteq P_n$, then $I_n \cap O_n \subseteq S_n$.

By theorem 5.3.9, $I_n \cap O_n = I_n^{\ell} \cap O_n^{\ell}$ and therefore, $I_n \cap O_n \subseteq S_n^{\ell}$.

Now suppose $f \in S_n^{\ell}$. Thus, f is sequential, essentially length-preserving and is contained in P_n. Consider $P_s(\hat{f}, x\epsilon_0) = y_0\, a_1\, y_1\, a_2\, y_2 \ldots a_n\, y_n\, \epsilon_0\, y_{n+1}$, where $x = a_1\, a_2 \ldots a_n$, $a_i \in \Sigma$, $f(x) = y_0\, y_1 \ldots y_{n+1}$ and $y_i \in \Gamma^*$. Since f is sequential, $\hat{f}(x) = f(x)$ and $\hat{f}(x) = y_0\, y_1 \ldots y_n$, $y_{n+1} = \lambda$. Suppose $y_0 = \lambda$. That is, $\hat{f}(\lambda) = \lambda$. Hence, $\ell_0(x) = |f(x)| +1 = |x| + 1 = \ell_I(x)$ for any x, since f is essentially length-preserving. Thus, $|f(x)| = |x|$ and $|y_i| = 1$, $1 \leq i \leq n$. Thus, all but the first input (including ϵ_0) is overlapped by an output, and $\ell_p(x) = |f(x)| + 1 = \ell_0(x) = \ell_I(x)$. Suppose $y_0 \neq \lambda$. Since $f(\lambda) = \hat{f}(\lambda)$, $\ell_I(\lambda) = 1 = \ell_0(\lambda) = |y_0|$. Hence, $|f(x)| = |y_0\, y_1 \ldots y_n| = 1 + |y_1\, y_2 \ldots y_n|$. Since f is sequential $|y_i| = 1$, $1 \leq i \leq n$. In this case every input (including ϵ_0) is overlapped and $\ell_p(x) = |f(x)| = \ell_0(x) = \ell_I(x)$.

Since $f \in P_n$, $(\forall n_0 \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) = \ell_p(x) \leq n_0$ $t_M(x) \leq n_0]$ and $f \in I_n$. Furthermore, $(\forall n_1 \in N_+)(\forall x \in \Sigma^*)[\ell_0(x) = \ell_p(x) \leq n_1 \Rightarrow t_M(x) \leq n_1]$ and $f \in O_n$. $\square$.

Thus we can conclude that S_n^{ℓ} is the class of 'real-time' computable functions to which other authors have referred. Since the author is of the opinion that 'real-time' does not necessitate limiting attention to length-preserving functions and by virtue of the above theorem, S_n is the class which should be referred to as the class of real-time computable functions.

5.4 Linear-Time Considerations

In the previous section, we showed that the containments proved in section 5.2 are proper containments for the case when $t(n) = n$. However, by limiting attention to essentially length-preserving the smaller classes were shown to be equal. In this section, we show that the larger classes are equal for the case when $t(n) = c.n$, for some constant $c > 1$.

Theorem 5.4.1 $T_{c.n} = P_{c.n} = M_{c.n}$, for $c > 1$.

Proof: We know that $M_{c.n} \subseteq P_{c.n} \subseteq T_{c.n}$, by theorems 5.2.11 and 5.2.13. Thus, if we show that $T_{c.n} \subseteq M_{c.n}$, we will be done.

Suppose $f \in T_{c.n}$. Hence, there exists a machine M such that $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) + \ell_0(x) \leq n \Rightarrow t_M(x) \leq c.n]$. Now, for any $x \in \Sigma^*$, $\ell_I(x) + \ell_0(x) \leq 2.\max(\ell_I(x), \ell_0(x))$. Therefore $(\forall n \in N_+)$ $(\forall x \in \Sigma^*)[\max(\ell_I(x), \ell_0(x)) \leq n \Rightarrow \ell_I(x) + \ell_0(x) \leq 2.n \Rightarrow t_M(x) \leq 2.c.n]$. Hence, $f \in M_{2cn}$. Now, by the constant speedup theorem of Hartmanis and Stearns[9], $M_{cn} = M_{2cn}$ and $f \in M_{cn}$. □

Since $I_{c.n}$ and $0_{c.n}$ are incomparable by theorems 5.2.4 and 5.2.6, $I_{cn} \cap 0_{cn} \underset{\neq}{\subseteq} I_{c.n} \underset{\neq}{\subseteq} I_{c.n} \cup 0_{c.n}$ and $I_{c.n} \cap 0_{cn} \underset{\neq}{\subseteq} 0_{c.n} \underset{\neq}{\subseteq} I_{c.n} \cup 0_{cn}$. By a trick similar to the one used in the proof of theorem 5.3.2, we show that $I_{c.n} \cup 0_{c.n} \underset{\neq}{\subseteq} M_{c.n}$

Theorem 5.4.2 $I_{c.n} \cup 0_{c.n} \neq M_{c.n}$, for $c > 1$.

Proof: We define a function $f:\{0,1\}^* \to \{0,1\}^*$ such that for some inputs, the length of the output is far greater than the length of the input and for the rest of the inputs, the length of the output is bounded.

1). for $x = \lambda$, $f(x) = \lambda$;

2). for $x = 0v$, where $v \in \{0,1\}^*$, $f(x) = 0$;

3). for $x = 1v$, where $v \in \{0,1\}^*$, $f(x) = 1^{2^{|x|}}$

Therefore, for $x = 0v$, where $v \in \{0,1\}^*$, $\ell_0(x) = 2$. Now, $t_M(x) \geq \ell_I(x) = |x| + 1$. Suppose $f \in 0_{c.n}$. Thus, $(\forall n \in N_+)$ $(\forall x \in \{0,1\}^*)[\ell_0(x) = 2 \leq n = t_M(x) \leq c.n]$. Choosing $n = 2$ and x such that $|x| > 2.c$, we obtain $(\exists x \in N_+)(\exists x \in \{0,1\}^*)[\ell_0(x) \leq n \;\&$

$t_M(x) \geq |x| + 1 > 2.c = c.n]$, a contradiction to $f \in 0_{c.n}$. Now suppose $f \in I_{c.n}$. Thus, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) \leq n \Rightarrow t_M(x) \leq c.n]$. Now, for $x = 1v$, where $v \in \{0,1\}^*$, $\ell_0(x) = 2^{|x|} + 1$. Now, $t_M(x) \geq \ell_0(x) = 2^{|x|} + 1$. Choose x such that $c.(|x| + 1) < 2^{|x|} + 1$. Such an x obviously exists. Choose $n = \ell_I(x)$. Thus, $(\exists n \in N_+)(\exists x \in \Sigma^*)$ $[\ell_I(x) \leq n \;\&\; t_M(x) \geq 2^{|x|} + 1 > c\ell_I(x)]$, a contradiction to $f \in I_{c.n}$.

Now, we construct a machine so that $f \in M_{c.n}$.

<u>Step 1</u> M reads the first input. If it is an ϵ_0, it halts. If it is a 0 , it goes to 0-routine. If it is a 1, it goes to 1-routine.

<u>0-routine</u> Output a 0 and read the input tape until the ϵ_0 is reached and then halt.

<u>1-routine</u> For each input, use a real-time counter [17] to write the $2^{|x|}$ 1's. From Yamada's work [17], we know there exists such a counter.

Now we bound $t_M(x)$, for each x. If $x = \lambda$, M executes step 1 and halts. Thus, $t_M(\lambda) = 1$. Now, $\ell_I(\lambda) = \ell_0(\lambda) = 1$. Now, if $x = 0v$, where $v \in \{0,1\}^*$, M executes step 1, overlaps the 0 output with the next read. M then simply reads and $t_M(x) = |x| + 1$ steps. If $x = 0v$, where $v \in \{0,1\}^*$, $\ell_I(x) = |x| + 1 \geq 2$ and $\ell_0(x) = 2$. Hence, $\ell_I(x) = \max(\ell_I(x), \ell_0(x))$. Also, $t_M(x) = \ell_I(x)$. Finally, suppose $x = 1v$, where $v \in \{0,1\}^*$. M executes step 1, and starts outputting 1's using the real-time counter. Thus, $t_M(x) = 2^{|x|} + 1 = \ell_0(x)$. Now, if $x = 1v$; where $v \in \{0,1\}^*$, $\ell_I(x) = |x| + 1$ and $\ell_0(x) = 2^{|x|} + 1$. Hence, $\ell_0(x) = \max(\ell_I(x), \ell_0(x))$.

Therefore, $(\forall n \in N_+)(\forall x \in \{0,1\}^* [\max(\ell_I(x), \ell_0(x)) \leq n \Rightarrow t_m(x) = \max(\ell_I(x), \ell_0(x)) \leq c.n]$ and $f \in M_{c.n}$. □

As was the case for theorem 5.3.8, we would expect to find a subclass of functions for which $I_{cn} = 0_{cn} = M_{cn}$. If this were the case, linear time could be represented by one class.

Definition 5.4.3 A function $f:\Sigma^* \to \Gamma^*$ is called linear if $(\exists c_1 > 0)$ $(\exists c_2 > 0)(\forall x \in \Sigma^*)[\ell_I(x) \le c_1 . \ell_0(x)$ & $\ell_0(x) \le c_2 . \ell_I(x)]$, where c_1 & c_2 are positive real numbers.

Notation We denote the class of linear functions f such that $f \in I_{cn}$ (O_{cn}, M_{cn}) as $I^{lin}_{c.n}$ $(O^{lin}_{c.n}, M^{lin}_{c.n})$.

Theorem 5.4.4 $M^{lin}_{cn} = I^{lin}_{cn} = O^{lin}_{cn}$, for $c > 1$.

Proof: Since $I_{cn} \subseteq M_n$ and $O_{cn} \subseteq M_{cn}$, it follows that $I^{lin}_{cn} \subseteq M^{lin}_{cn}$ and $O^{lin}_{cn} \subseteq M_{cn}$. Now, suppose $f:\Sigma^* \to \Gamma^*$ and $f \in M^{lin}_{cn}$. That is, $(\forall x \in \Sigma^*)$ $[\ell_I(x) \le c_1\ell_0(x)]$. Thus, $\max(\ell_I(x), \ell_0(x)) \le c_1\ell_0(x)$. Since $f \in M_{cn}$, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\max(\ell_I(x),\ell_0(x)) \le c_1\ell_0(x) \le n \Rightarrow t_M(x) \le cn]$. Therefore, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_0(x) \le \frac{n}{c_1} \Rightarrow t_M(x) \le cn]$ and $f \in O_{cc_1n}$. Using the constant speedup theorem [9], $O_{cc_1n} = O_{cn}$. Now, if $c_1 < 1$, then $\max(\ell_I(x), \ell_0(x)) = \ell_0(x)$. Since $f \in M_{cn}$, $(\forall n \in N_+)(\forall x \in \Sigma^*)$ $[\max(\ell_I(x), \ell_0(x)) = \ell_0(x) \le n \Rightarrow t_M(x) \le cn]$ and $f \in O_{cn}$.

By replacing $\ell_0(x)$ by $\ell_I(x)$ and $c_1\ell_0(x)$ by $c_2\ell_I(x)$, we can show that $M^{lin}_{cn} = I^{lin}_{cn}$ $\square$

We now show that for linear functions, all of the linear-time classes are the same.

Theorem 5.4.5 $I_{cn} \cap O_{c.n} = I^{lin}_{cn}$, for $c > 1$.

Proof: By theorem 5.4.4, $I^{lin}_{cn} = O^{lin}_{cn} = I^{lin}_{cn} \cap O^{lin}_{cn}$. Furthermore, $I^{lin}_{cn} \cap O^{lin}_{cn} \subseteq I_{cn} \cap O_{cn}$.

Now suppose $f \in I_{cn} \cap O_{cn}$, $f:\Sigma^* \to \Gamma^*$. We need only show that f is linear. Since $f \in I_{cn}$, $(\forall n_0 \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) \le n_0 \Rightarrow t_M(x) \le c.n_0]$. Now, $t_M(x) \ge \ell_0(x)$ for each x, since M must produce all of $f(x)$. Suppose $\ell_0(x) > c\ell_I(x_1)$ for some x_1. Choosing $n_0 = \ell_I(x_1)$, we obtain $(\exists n_0 \in N_+)(\exists x_1 \in \Sigma^*)[\ell_I(x_1) \le n_0$ & $cn_0 = c.\ell_I(x_1) <$

$\ell_0(x_1) \le t_M(x_1)$, a contradiction. Similarly, $\ell_I(x) \le c\ell_0(x)$, for all x and f is linear. □

Corollary 5.4.6 $I^{\ell}_{cn} \cap O^{\ell}_{cn} = T^{\ell}_{cn}$, for $c > 1$.

Proof: If f is essentially length-preserving, f is linear. By theorems 5.4.1, 5.4.4 and 5.4.5, $I^{\ell}_{cn} \cap O^{\ell}_{cn} = M^{\ell}_{cn} = T^{\ell}_{cn}$. □

We can conclude that for the commonly studied functions, the linear-time classes are all the same. We complete this section with the problem of when real-time classes and linear-time classes are equal.

Remark For generation problems, we know that $I_n \cap O_n = I_{cn} \cap O_{cn}$. Since generation problems can be stated as essentially length-preserving functions, we note that for generation problems, all of the classes are the same. However, we also know that there exists recognition problems for which $I_n \cap O_n \ne I_{cn} \cap O_{cn}$. Also, we note that a recognition problem can be stated as a bounded function (a 1 or 0 output for the whole string). Thus, the context free language shown in theorem 5.3.4 could be used to show that $T_n \ne T_{cn}$. Thus, we can conclude that in general, for all of the classes defined, real-time and linear-time are different qualities.

5.5 The Effect of the Off-line Nature of a Function

We observed in the introduction of this chapter that the off-line nature of a function could affect its complexity. In this section, we will study this effect for input-bounded, output-bounded and maximum-bounded classes. We first prove a lemma that will be the basis of our main result in this section.

Lemma 5.5.1 Let $f:\Sigma^* \to \Gamma^*$ and M be any machine computing f. Then, $(\forall x \in \Sigma^*)[t_M(x) > |x| + (|f(x)| - |\hat{f}(x)|)]$.

Proof: We know that for any $x \in \Sigma^*$, $|f(x)| - |\hat{f}(x)|$ is the amount of output that can be made only after M reads the endmarker ϵ_0. Since M must also read all of the input and ϵ_0, the lemma follows immediately. □

We now show that there exists classes between I_n and I_{cn}.

Theorem 5.5.2 For any integer $c \geq 1$, $I_{n+c-1} \subsetneq I_{n+c}$.

Proof: Let $f:\Sigma^* \to \Gamma^*$. If $f \in I_{n+c-1}$, $(\forall n \in N_+)(\forall x \in \Sigma^*)[\ell_I(x) \leq n = t_M(x) \leq n+c-1 < n+c]$ and $f \in I_{n+c}$.

We now define an almost sequential function $f:\{0,1\}^* \to \{0,1\}^*$ which satisfies the conditions of this theorem.

1). For $x \in \{0,1\}^*$, such that $|x| < c$, $f(x) = \text{Im}(x)$ (where Im is as defined in theorem 2.2.3).

2). For $x = a_1 a_2 \dots a_c v$, where $v \in \{0,1\}^*$ and for $1 \leq i \leq c$, $a_i \in \{0,1\}$, $f(x) = va_c a_{c-1} \dots a_2 a_1$.

For example, for $c = 3$, $f(011101) = 101110$, $f(0101101) = 1101010$.

Now, we define a $g:\{0,1,\epsilon_0\}^* \to \{0,1\}^*$.

I). For $x \in \{0,1\}^*$, such that $|x| < c$, $g(x) = \lambda$;

II). For $x = a_1 a_2 \dots a_c v$, where $v \in \{0,1\}^*$ and for $1 \leq i \leq c$, $a_i \in \{0,1\}$, $g(x) = v$;

III). For $x = u \epsilon_0 w$, where $u \in \{0,1\}^*$, $w \in \{0,1,\epsilon_0\}^*$, $g(x) = f(u)$.

Since similar proofs can be found in chapter 4, we state without proof that $g = \hat{f}$, the m.d.s.a. of f.

Now, for any $x \quad \Sigma^*$, $|f(x)| - |\hat{f}(x)| \leq c$ and f is almost sequential. Also, for any x such that $|x| \geq c$, $|f(x)| - |\hat{f}(x)| = c$. Thus, we can choose an x_1 such that $|x_1| \geq c$ and an $n_1 = |x_1|$ and obtain $(\exists n_1 \in N\ )(\exists x_1 \in \Sigma^*)[\ell_I(x) \leq n_1 \ \&\ t_M(x_1) > |x| + c =$

$\ell_I(x) + c-1) = n_1 + (c-1)]$, for any machine M computing f. Therefore, $f \notin I_{n+c-1}$.

We now define a machine M which computes f in no more than $|x| + 1 + c$ steps for any input x.

Stage 1 M reads the input tape. If the input is ϵ_0, M goes to stage 3. If not, M puts the input letter on a work tape and using a counter decides if c inputs have been read.[3] If not, return to the start of stage 1. If c inputs have been read, go to stage 2.

Stage 2 M reads the input and copies it onto the output tape if it is not ϵ_0. When ϵ_0 is reached, go to stage 3.

Stage 3 Copy the c digits on the work tape in reverse order onto the output tape and halt.

M can execute each stage in 1 step for each iteration. Hence, $(\forall x \in \{0,1\}^*)[t_M(x) = |x| + 1 + c]$. Therefore, $(\forall n \in N_+)$ $(\forall x \in \{0,1\}^*)[\ell_I(x) \leq n \Rightarrow t_M(x) = |x| + 1 + c = \ell_I(x) + c \leq n + c]$ and $f \in I_{n+c}$. □

Corollary 5.5.3 For any integer $c \geq 1$, $O_{n+c-1} \subsetneq O_{n+c}$, and $M_{n+c-1} \subsetneq M_{n+c}$.

Proof: The function defined for theorem 5.5.2 is length-preserving and for $x \in \{0,1\}^*$, $\ell_I(x) = \ell_O(x) = \max(\ell_I(x), \ell_O(x))$. Hence, the corollary follows immediately. □

We should note that the off-line nature of a function has no equivalent effect for the profile-bounded classes since $\ell_p(x)$ already compensates for it by including $|f(x)| - |\hat{f}(x)|$.

The results of sections 5.4 and 5.5 lead one to using either $M_{t(n)}$ or $I_{t(n)}$ for $t(n) > n$. It is only in the case of n-computable functions that the profile-bounded functions can lend any insight into the complexity of functions.

[3]We could decide if c inputs are read by using c states for stage 1 and advance to the next state for each input read.

CHAPTER 6

RELATED TOPICS

6.1 Introduction

In this chapter, we consider problems which are related to the two central problems discussed in this thesis: i.e., the concepts of 'ongoing processes' and 'real-time computations'. In section 6.2, we show first that if a function f is computable on an off-line machine in t(n) steps, then f is computable on a sequential Turing machine in at most n.t(n) steps, for each input of length n. Next, we show that there is a function f such that $f \in I_{n \log n}$, but f takes at least $O\left(\frac{n^2}{(\log n)^2}\right)$ steps on a sequential Turing machine.

In section 6.3, we discuss some open problems and some future research areas.

6.2 Comparison of Off-line and Sequential Turing Machines

In chapter 2, we introduced the sequential Turing machine and showed that it was not a model of a universal computer: that is, the class of sequential Turing machines does not strongly (i.e., without encodings)com pute all of the recursive functions. In this section, we answer the related question: "Is there any difference between the amount of time (number of steps) a sequential Turing machine takes to compute a function and the time an off-line Turing machine takes for the same function?"

Remark Since we do not put any restriction on our off-line model, every sequential Turing machine is an off-line Turing machine. We can conclude immediately that for any function f, there is an off-line Turing machine which computes f no slower than any sequential Turing machine can compute f.

In this section, we will assume that we are dealing with input-bounded $t(n)$ - computable functions. We will henceforth abbreviate 'input-bounded $t(n)$ - computable' by '$t(n)$ - computable'.

Theorem 6.2.1 Let $f:\Sigma^* \to \Gamma^*$ be a sequential function. If f is $t(n)$ - computable on an off-line Turing machine M, then f is $(\sum_{i=0}^{n} t(i))$ - computable on a sequential Turing machine.

Proof: We describe the operation of T, a sequential Turing machine. T will use one work tape as a dummy input tape to M, which T uses as a submachine.

Stage 0 T uses M to compute $f(\lambda)$ using the blank dummy input tape. The result is produced on the output tape and on a dummy output tape.

Stage $i > 0$ T reads the ith input letter a_i and adds it to the dummy input tape. M then is used to compute $f(a_1 a_2 \ldots a_i)$ from the beginning using the dummy input tape. The output letters produced by M are compared to the letters on the dummy output tape. When output letters not on the dummy output tape (i.e., letters of $\Delta f(a_1 a_2 \ldots a_i)$), are encountered, T produces these on the real output tape and also adds them to the dummy output tape.

Since f is sequential, each state i is possible and T does compute f. Also, T is obviously a sequential Turing machine. Now for each stage i, T takes $t(i)$ steps to compute $f(a_1 a_2 \ldots a_i)$. By the results of Fischer, Meyer and Rosenberg [6], we can assume that the dummy input and output tapes are buffers and no rewinding is necessary. The reading of a_i can be overlapped with the reading of the first input on the dummy work tape. Thus, $t_T(x) \leq \sum_{i=1}^{n} t(i)$. □

Corollary 6.2.2 If f is sequential and linear-time computable on an off-line Turing machine, f is $O(n^2)$ - computable on a sequential Turing machine.

Corollary 6.2.2 represents the worst possible comparison between off-line and sequential Turing machines. For a polynomial $t(n)$, $\sum_{i=0}^{n} (t(i))$ is a polynomial, and for exponential $t(n)$, $\sum_{i=0}^{n} (t(i))$ is exponential in the same base. Thus, for functions which take at least exponential time, the speed of computation is the same up to a constant factor.

Hennie [11] has shown that the above difference is almost achievable for one function.

Theorem 6.2.3 [Hennie] There exists a function $f:\{0,1,2\}^* \to \{0,1\}^*$ such that f is $O(n.\log n)$ - **computable on an off-line Turing machine**, and takes at least $O\left(\frac{n^2}{(\log n)^2}\right)$ **steps on a sequential Turing machine.**

Proof: The set A defined **by Hennie [11] is the basis of f.**

We will say that $x \in A$ if

a). $x = v2$, for $v \in \{0,1,2\}^*$

b). The preceding sequence v consists of a number of identical-length blocks of 0's and 1's, consecutive blocks separated by single 2's.

c). The total number of blocks received so far exceeds 2^k, where k is the number of 0's and 1's in each block.

d). The pattern of 0's and 1's that appears in the very last block to be received matches the pattern of at least one of the first 2^k blocks.

Then, $f:\{0,1,2\}^* \to \{0,1\}^*$ is defined for $x = vi$, where $v \in \{0,1,2\}^*$, $i \in \{0,1,2\}$, as $f(x) = f(v).1$, if $x \in A$.

Hennie showed that for any sequential Turing machine which computes f, must take at least $O\left(\frac{n^2}{(\log n)^2}\right)$ steps and infinitely many n.

Now, to show that f is $O(n \log n)$ - computable on an off-line Turing machine, we need only sort-merge the entire input string if it has more than 2^k blocks and has the right format. Since we can also assume that we have an index to the sorted string without loss of generality, we can tell when one of the first 2^k blocks matches one of the later blocks from the index vector. The sort will cost $O(n \log n)$ steps, and checking the index vector takes another $O(n \log n)$ steps. □

We observe that the above theorem holds because sorting is not a sequential function. Thus, we cannot sort on the sequential Turing machine except for blocks already received. Consequently, although computing a sequential function, an off-line Turing machine can use nonsequential functions as an aid in the computation.

Remark Since the function used in theorem 6.2.3 is a length-preserving function, we could have replaced 'sequential Turing machines' by 'on-line Turing machines' and 'input-bounded' by 'maximum-bounded' and still have a valid theorem.

We can thus conclude that sequential Turing machines are neither as computationally powerful as off-line Turing machines nor as fast for some functions as off-line Turing machines.

6.3 Further Research

In this section, we discuss problems which are suggested by the results in this thesis. First, we know that by theorem 3.3.4 that some computable functions have a noncomputable m.d.s.a. We also noted that the classes of sequential, truly off-line, almost sequential, almost truly off-line and bounded computable functions had computable m.d.s.a.'s. By the Rice-Myhill-Shapiro Theorem, there is no decision procedure to decide when a computable function has a computable m.d.s.a.

Next, in section 4.5, we exhibited a dense, infinite hierarchy of functions. Does this hierarchy have any close relationship to the previously discovered hierarchies? The author conjectures that no such relationship does exist.

In section 5.3, we introduced the class of profile-bounded n-computable sequential functions, S_n. We characterized the essentially length-preserving functions belonging to S_n as exactly the intersection of the input-bounded and output-bounded n-computable functions: i.e., as the class of 'real-time' computable functions as traditionally studied. Note that we can characterize P_n in terms of the length of the profile and the class of sequential functions by the fact that $P_s(\hat{f},x) = P_s(f,x)$. However, we cannot characterize S_n by restricting the length of the profile to only include the endmarker ϵ_0 and the string preceding it. Denote the restricted length as $\ell_s(x)$ and C_n as the class defined for $\ell_s(x)$. Clearly, $S_n \subset C_n \subset P_n$.

We can show that $C_n \neq S_n$ by considering the following function: **for $x = \lambda$, $f(\lambda) = \lambda$; for x $\{0,1\}$, $f(x) = xx$; for $x = a_1 v a_2$, $f(x) = a_1 a_1 a_2$. Now, f is obviously not sequential and $f \notin S_n$. How**ever, we can construct a machine which reads the first input letter and outputs the first letter once before the second read and once after the second read. When ϵ_0 is encountered, M produces the last output. Thus, M takes $|x| + 2$ steps to compute $f(x)$ and $\ell_s(x) = |x| + 2$. Hence, $f \in C_n$. We conclude by asking: "Is there any characterization of S_n, which will enable us to study it more readily?"

Our last conjecture is that there exists a function f such that f is linear-time computable on an off-line Turing machine, while any sequential Turing machine computing f must take at least O(n log n) steps for an input of length n. The results of Cook and Aanderaa [4] **would possibly be applicable in proving this conjecture.**

This conjecture can be compared to the results of L. Stockmeyer and M.J. Fischer [16]. They showed that for a multiplication-like sequential function f which takes t(n) - steps on an off-line Turing machine there is a sequential Turing machine which computes f in log n.t(n) steps.

Finally, we note that the meaning of 'simulation' has not yet really been decided. Like the concept of 'on-line' and 'real-time', simulation has some relationship to the IO-profile of a machine computation. Exactly how to tie simulation into the present framework is an area to be studied in the future.

BIBLIOGRAPHY

1. Arbib, M.A. (1969), Theories of Automata, Prentice-Hall, Englewood Cliffs, New Jersey.

2. Atrubin, A.J. (1965), A one Dimensional Real-Time Iterative Multiplier, IEEE Transactions on Electronic Computers, EC-14, 394-399.

3. Cole, S.N. (1971), Deterministic Pushdown Store Machines and Real-Time Computation, Journal of the Association for Computing Machinery, 18:2, 306-328.

4. Cook, S.A., and S.O. Aanderaa (1969), On the Minimum Computation Time of Functions, Transactions of the American Mathematical Society, 142, 291-314.

5. Fischer, P.C. (1967), Turing Machines with a Schedule to Keep, Information and Control 9, 364-379.

6. Fischer, P.C., A.R. Meyer, and A.L. Rosenberg (1972), Real-Time Simulation of Multihead Tape Units, Journal of the Association of Computing Machinery 19:4, 590-607.

7. Ginsburg, S., and G.F. Rose (1962), A Comparison of the work Done by Generalized Sequential Machines and Turing Machines, Transactions of the American Mathematical Society 103, 394-402.

8. Hartmanis, J. (1970), Size Arguments in the Study of Computation Speeds, Department of Computer Science, Cornell University, Technical Report Number 70-70.

9. Hartmanis, J., and R.E. Stearns (1964), On the Computational Complexity of Algorithms, Transactions of the American Mathematical Society 117, 285-306.

10. Hennie, F.C. (1965), One Tape, Off-Line Turing Machine Computations, Information and Control 8, 553-598.

11. Hennie, F.C. (1966), On-Line Turing Machine Computations, IEEE Transactions on Electronic Computers EC-15, 35-44.

12. Minsky, M.L. (1967), Computation: Finite and Infinite Machines, Prentice-Hall, Englewoods Cliffs, New Jersey.

13. Rabin, M.O. (1964), Real Time Computation, Israel Journal of Mathematics 1, 203-211.

14. Ritchie, D.M. (1968), Program Structure and Computational Complexity, Doctoral Thesis, Harvard University.

15. Rosenberg, A.L. (1967), Real-Time Definable Languages, Journal of the Association for Computing Machinery 14:4, 645-662.

16. Stockmeyer, L., and M.J. Fischer (1973), private communication.

17. Yamada, H. (1962), Real-Time Computation and Recursive Functions not Real-Time Countable, IRE Transactions on Electronic Computers EC-11, 753-760.

APPENDIX

A.1 Turing Machine Model

A multitape Turing machine M has a one-way read-only input tape, a one-way write-only output tape, and k work tapes, for some k. M is represented by specifying: Q - a finite set of states;

Σ - an input alphabet;

Γ - an output alpabet;

Δ - a work tape alphabet;

and a set of quintuples of the form

$$(q,(a_0,a_1,\ldots,a_k),(b_0,b_1,\ldots,b_k),(m_0,m_1,\ldots,m_k),q'),$$

where $q\in Q$, $a_0\in\Sigma$, $a_i\in\Delta$ for $1\leq i\leq k$, $b_0\in(\Gamma \cup\{\lambda\})$, $b_i\in\Delta$ for $1\leq i\leq k$, $m_0\in\{N,R\}$ $m_i\in\{N,R,L\}$ for $1\leq i\leq k$. (where N means no move, R means a move to the right, and L means a move to the left)

The interpretation of a quintuple as given above is that a machine in state q with symbol a_0 under the read head of the input tape and a_i under the read head of work tape i, $1\leq i\leq k$, performs the following steps:

(i) if $b_0\in\Gamma$, b_0 is written on the output tape; (if $b_0=\lambda$, the output tape does not move; otherwise, it moves to the right)

(ii) replaces a_i by b_i on work tape i, $1\leq i\leq k$;

(iii) moves the input tape in direction m_0, and work tape i in direction m_i, $1\leq i\leq k$;

and (iv) enters state q'.