THE COGNITIVE DEVELOPMENT OF MATHEMATICS

SPATIAL AND TIMING MEASURES ASSOCIATED WITH ALGORITHMS

BY D. PAGER

ABSTRACT

The thesis considers in turn measures of algorithms, measures of programs, and measures of computations. We define an algorithm's measure as the average of the space-time requirements of its associated computations, and here, as with measures for programs, a question of optimisation arises: that of finding the algorithm for a function which has the least such measure.

The problem of optimisation for algorithmic measure, in its general form, proves to be unsolvable, but we show that an effective optimisation procedure does exist with regard to algorithms for finite functions, and give in detail the solution of the special case for functions with a domain of two elements. Further, we reduce the determination of the optimum algorithm for infinite functions to that of calculating the value of a primitive recursive function $\lambda(z)\ \chi(z,t)$, for any sufficiently large t. Taking a different viewpoint, we investigate the existence of lower bounds to the measures of algorithms for certain functions.

A similar analysis is applied to the spatial measure defined for programs, program length, and we discuss some of the philosophical ramifications of program brevity. In addition, a pseudo-spatial measure, the number of instructions a program contains, is considered. By using reduction theorems which map onto one another corresponding instructions in equivalent programs, we are able to adapt to this pseudo-spatial measure our results on program length. We then examine a problem of secondary optimisation, which involves a minimisation of both algorithmic and program measure.

Measures of computations have been analysed by Myhill, Trakhtenbrot, Smullyan, Ritchie, Cleave, Rabin, Arbib and Blum. An essential element in Blum's definition of computational measure are the 'measuring predicates' and we investigate certain relations between their spatial and timing requirements (as where an upper bound on one such quantity places a lower bound on another). The relevant literature is discussed in brief, and we take up a number of points which arise.

Most of the arguments and results in the thesis are formulated in terms of Turing machines, but they are applicable to other means of representing algorithms. In the final chapter we investigate how the definition of Turing machines may be extended so as to provide a more authentic model of actual computers both in regard to space-time measure, and to the domain of functions they encompass.

CONTENTS

NOTATION

Theorems are numbered in a single sequence, whereas lemmas, corollaries and formulae are numbered anew in each section.

The notation in Davis' "Computability and Unsolvability" is taken as our standard. Minor deviations are that we employ lower case 's-symbols' in our Turing machines and write the predicate $T_n(z,x_1,\ldots x_n,y)$ as $T(z,x_1,\ldots x_n,y)$. We indicate the initial appearance of symbols defined in Davis by the mark $^{\supset}$, and a list of such symbols, together with an informal description of their use, is given in Appendix II.

Turing machines and algorithms are designated by upper case letters, whilst their Gödel numbers are represented by the corresponding lower case letters. Further, gn(Z) denotes the Gödel number of Z. As in Davis, by 'number' we mean 'non-negative integer', unless otherwise specified, e.g. "for any (number) $k \neq 1$" stands for "for any non-negative integer $k \neq 1$". Variables for such numbers are denoted by lower case letters, as are those functions which are defined and employed only within the proof of a single theorem. In general, functions referred to more widely than this are represented in upper case or by Greek letters.

The symbols a, b, ℓ, p, S are employed throughout the thesis to designate certain parameters we define. Symbols such as $\mu_{a,b}$, $\gamma_{a,b,p}$, $\phi_{a,b,p}$ are used to denote functions defined in terms of these parameters; however where no ambiguity arises, the subscripts are dropped.

CHAPTER 1

MEASURES OF ALGORITHMS

We give in this chapter our basic definitions of space-time measure, and with this quantity as criterion, discuss algorithms from the point of view of efficiency. Our interest in providing a formal treatment of this subject is prompted by the amount of attention it receives in practice. The question of algorithmic efficiency occurs in (a) computer programming, where it forms the first consideration in the construction of programs; economy in spatial and timing requirements is particularly vital in the design of library programs such as sine, cosine, exponentiation and matrix routines; (b) Numerical Analysis, in devising and evaluating various methods for tackling linear programming problems, calculating determinants, and solving systems of linear equations, etc.; (c) Operations Research, in cases such as the Travelling Salesman[†] and the m x n scheduling problems,[†] where algorithms for deciding certain questions (typically by considering all possible cases) are known, and the end being pursued is that of finding substantially faster ones; (d) Games

[†] See Appendix I for description.

Theory, where a similar goal arises with respect to finite games with perfect information, as occur in economics and chess.

By giving a precise definition to an algorithm's space-time measure, we enable the evaluation and comparison of proposed solutions to problems such as these to be precisely made. On the other hand we provide a language in which one can formulate the possibility that certain design objectives, such as that of the Travelling Salesman problem, may be unattainable.

However, we have not limited ourselves to questions of direct practical applicability. It has seemed natural, in considering the problem of finding better algorithms, to ask whether an optimum algorithm exists, and if so, whether it be effectively determinable. To this last we would take the answer as being 'yes', no matter how laborious the method of determination might be.

SECTION 1 BASIC DEFINITIONS

The following are the main concepts which are defined in the two succeeding subsections:

computation system

calculation

the space-time measure of a computation; a pure-time measure

the space-time measure of an algorithm

probability function; significant argument; significant set

the efficiency of computations and algorithms

the optimisation function and the restricted optimisation function

the spatial and timing measures of computations of Turing machines

SECTION 1.1 ABSTRACT DEFINITIONS

COMPUTATION SYSTEM

By a computation system we mean a way of representing algorithms on a finite alphabet, plus a specification of the associated calculation procedure. The concept thus embodies the algorithmic systems associated with Turing machines, actual computers, Kleene systems of equations [17], and the U.R.M.'s of Shepherdson and Sturgis [31].

The references to algorithms in the definitions which follow assume some such system.

CALCULATION

In writing a program to calculate a function over its domain, we are not concerned with the behaviour of the program for values of the dependent variables outside the domain. We place no requirement as that imposed by Davis' definition of a Turing machine '<u>computing</u>' a function, that the program be non-terminating in this case[9]. This prompts the following definition of '<u>calculation</u>':

An algorithm <u>calculates</u> a function if it supplies the value of the function for every member of the function's domain.†

THE SPACE-TIME MEASURE OF A COMPUTATION

Rabin [24] and Blum [5] have defined axiomatically closely related concepts of a 'measure of proofs' and a 'measure of computations' respectively. The definition of a measure for the space and time required by a computation which we give below is a particular measure

† Whereas an algorithm <u>computes</u> a function if it calculates it, <u>and is undefined</u> for all arguments that are not in the function's domain.

of the last type. Rabin's and Blum's axioms are too wide for our purposes. They do not isolate the specific properties of a computation's spatial and timing attributes. Blum's definition, for instance, encompasses equally well as a valid measure the number of steps a computation involves or a reciprocal thereof (such as $\left[\frac{k}{\text{number of steps}}\right]$, for some constant k).

We assume that some measure is defined on the space which the computation involves. In general this space consists of two components, that occupied by the program, and that used in the computation procedure itself. We assume further that a measure is defined on the amount of time required. This is usually identified with the number of steps, as in Wang's 'speed function'[38].

The space-time measure of a computation is taken to be the following function of the space and time involved:

a × space involved + b × time involved,[†]
where a,b are non-negative preselected parameters, not both zero.

We denote this, with respect to a computation of an algorithm Z for an argument $(x_1,x_2,\ldots,x_n)$ as
$\mu_{a,b}(z,x_1,x_2,\ldots,x_n)$.

[†] A linear function is of course not the only one possible. We take up the question of other possible measures later.

If $a = o$, the measure is referred to as a pure-time measure, whereas if $a \neq o$, it is said to have a spatial component.

THE SPACE-TIME MEASURE OF AN ALGORITHM

Let the algorithm be one for calculating the values of a function $f(x_1,x_2,\ldots,x_n)$ over a domain S of n-tuples. We assume, as is the case in practice, that we will not be called upon to calculate $f(x_1,x_2,\ldots,x_n)$ for each member of S with equal probability. It is possible for instance that no x_i can occur larger than some limit k; in this case the probability of having to calculate $(k+1,k+1,\ldots,k+1)$ is zero.

In general we postulate the existence of a probability function $p(x_1,x_2,\ldots,x_n)$, where $p(x_1,x_2,\ldots,x_n)$ is the probability of having to calculate $f(x_1,x_2,\ldots,x_n)$, such that

(1) $p(x_1,x_2,\ldots,x_n)$ is recursive[†],

(2) $o \leqslant p(x_1,x_2,\ldots,x_n) \leqslant 1$,

[†] in the original sense of Turing [36]; i.e. we can calculate, for any i, the ith binary digit.

(3) $o < \sum_{x_1,x_2,\ldots,x_n} p(x_1,x_2,\ldots,x_n) \leqslant 1$, [†]

(4) $p(x_1,x_2,\ldots,x_n) > o \implies (x_1,x_2,\ldots,x_n) \in S$.

So if Z is an algorithm which calculates f, then $p(x_1,x_2,\ldots,x_n) > o \implies (x_1,x_2,\ldots,x_n) \in$ domain of Z, and with respect to such sets of probabilities we define the <u>space-time measure of an algorithm Z</u> as

$$\sum_{x_1,\ldots,x_n} p(x_1,\ldots,x_n) \times \text{space-time measure of the computation of Z for } (x_1,\ldots,x_n)$$

$$= \sum_{x_1,\ldots,x_n} p(x_1,\ldots,x_n)\ \mu_{a,b}(z,x_1,\ldots,x_n) \quad .$$

We denote this by $\underline{\gamma_{a,b,p}(z)}$;

<u>PROBABILITY FUNCTION, SIGNIFICANT SET</u>

When speaking of a <u>probability function</u> $p(x_1,x_2,\ldots,x_n)$, we will in all cases assume properties

---

† By $\sum_{x_1,\ldots,x_n} p(x_1,\ldots,x_n)$ we mean $\lim_{m\to\infty} \sum_{\substack{\text{all } x_1,\ldots,x_n \\ \text{such that } x_i<m \\ \text{each } i \text{ in } 1(1)n}} p(x_1,\ldots,x_n)$. We allow the less important case of $\sum_{x_1,\ldots,x_n} p(x_1,\ldots,x_n) < 1$, so that one can evaluate and compare the expected space-time involved in separate parts of an algorithm which consists of a set of distinct procedures for different categories of arguments.

(1) to (3) above. By the significant set S associated with a probability function we refer to: $\{(x_1, x_2, \ldots, x_n) \mid p(x_1, x_2, \ldots, x_n) > 0\}$, and the members of this set are called the significant arguments. By a proper probability function for an algorithm, or function, Z, we mean a probability function whose associated significant set is contained in the domain of Z.

Efficiency of a Computation or Algorithm

By the efficiency of a computation or algorithm we mean how small the associated measures are. It is sufficient thus to equate efficiency with the corresponding inverses of these quantities.

Space-Time Measure of a Function

By ~~the~~ an optimum algorithm $\eta_{a,b,p}(f)$ which calculates a function $f(x_1, x_2, \ldots, x_n)$ over a domain S, we mean the most efficient such algorithm. The concept is only defined for probability functions which are proper with respect to S.

Thus $\gamma(gn(\eta_{a,b,p}(f)))$

$$= \min_{\substack{\text{all algorithms } Z \\ \text{of the computation} \\ \text{system which calculate} \\ f(x_1, \ldots, x_n) \text{ for all} \\ (x_1, \ldots, x_n) \text{ such that} \\ p(x_1, \ldots, x_n) > 0}} \gamma_{a,b,p}(z) \quad .$$

$\gamma(gn(\eta_{a,b,p}(f)))$ is also referred to as the <u>space-time measure of the function</u> <u>$\lambda(x_1,x_2,\ldots,x_n)\ f(x_1,x_2,\ldots,x_n)$</u>.[†]

<u>Optimisation Function, Restricted Optimisation Function</u>

By <u>optimising an algorithm</u> Z (with respect to a proper probability function $p(x_1,x_2,\ldots,x_n)$) we mean finding (if it exists) the optimum algorithm which calculates the same function Z does for all $(x_1,x_2,\ldots,x_n)$ such that $p(x_1,x_2,\ldots,x_n) > 0$.

We denote the optimum such algorithm as <u>$\phi_{a,b,p}(z)$</u> or, for given parameters, simply as $\underline{Z}^*$; ϕ is referred to as the <u>optimisation function</u>.

For the sake of completeness we also define <u>$\phi'_{a,b,p}(z)$</u> as the most efficient algorithm which computes the same function as Z does,[‡] and we refer to ϕ' as the restricted optimisation function. (Thus $\gamma(gn(\phi(z))) \leq \gamma(gn(\phi'(z)))$.)[‡‡]

[†] Using Church's lambda notation [8].

[‡] Note the distinction between computation and calculation, see page 6.

[‡‡] If there is more than one algorithm with least space-time measure, <u>each</u> is an optimum algorithm, and the function ϕ refers to in this case is an arbitrary one of these. Expressions of the form "$Z_1 = \phi(Z)$" or "$Z_1 = Z^*$" are to be interpreted as asserting that Z_1 is one of the optimum Turing machines concerned. ϕ' may also be ambiguous similarly.

SECTION 1.2 DEFINITIONS FOR TURING MACHINES

The Computation System of Turing Machines

We identify 'algorithm' here with 'Turing machine' plus a specification of the number of arguments, i.e. the degree of the function evaluated.

Thus if Z is a Turing machine, we denote as $[z]_n$ (Davis' notation) the algorithm which evaluates $[z]_n(x_1,x_2,\ldots,x_n)$.[†]

The time involved in a computation of a Turing machine Z for argument $(x_1,x_2,\ldots,x_n)$ is identified with the number of steps involved i.e. the number of instantaneous descriptions the computation consists of -1. We denote this as $E(z,x_1,x_2,\ldots,x_n)$.

The space involved in such a computation is made up of

(a) that directly employed in the computation procedure. This is taken as measured by the size of the maximum tape expression that occurs. We use Rabin and Wang's definition of the magnitude of a tape expression as being the smallest length of tape which includes the square being scanned and the marked (non-blank) squares[26]. We

---

[†] See our convention of representing the Gödel numbers of Turing machines by corresponding lower case letters, p.1.

denote this as $\underline{M(z,x_1,x_2,\ldots,x_n)}$.

(b) that occupied by the quadruples which define the Turing machine program. We adopt the convention of representing programs for Turing machines, such as

$$Z = \left\{ q_{i_t} s_{j_t} x_t q_{u_t} \;\middle|\; t = 1(1)\,m \right\}$$

where for each t, x_t is either L, R or an s-symbol s_{k_t},

on the alphabet $\{O,1,R,L,\text{'},,;\}$ in the following way:

$$\bar{i}_1,\bar{j}_1,\bar{k}_1,\bar{u}_1;\ \bar{i}_2,\bar{j}_2,\bar{k}_2,\bar{u}_2;\ \ldots\ ;\bar{i}_m,\bar{j}_m,\bar{k}_m,\bar{u}_m;$$

where $\bar{i}_t,\bar{j}_t,\bar{u}_t$ are the binary encodings of the coefficients i_t,j_t,u_t respectively, and $\bar{k}_t = x_t$ in the case that x_t is L or R, whereas $\bar{k}_t$ = the binary encoding of the coefficient k_t in the case that x_t is the s-symbol s_{k_t}.

We denote by $\underline{Q(z)}$ the total number of symbols in such a representation of a Turing machine Z.

We now have to decide on what weighting to give to the two components of spatial measure considered above, the number of symbols in the program and the number of squares of tape. We are influenced by the situation which exists in actual computers, where each instruction,[†] or part of an instruction,[‡] occupies a fixed number of

[†] As in e.g. the IBM 7090.

[‡] As in e.g. the IBM 1401.

registers. This prompts the following definition of our measure of the <u>space involved in a computation of a Turing machine Z for an argument $(x_1,x_2,\dots,x_n)$</u>:

$$M(z,x_1,x_2,\dots,x_n) + \ell\, Q(z)$$
for some preselected ℓ,

it follows that, for such ℓ,

$$\mu_{a,b}(z,x_1,\dots,x_n) = a\Big(M(z,x_1,\dots,x_n) + \ell\, Q(z)\Big) + b\, E(z,x_1,\dots,x_n) .$$

We write this as $\underline{\mu_{a,b,\ell}}$. Similarly our space-time measure over Turing machines is written $\underline{\gamma_{a,b,\ell,p}}$, and this is given by

$$\sum_{x_1,\dots,x_n} p(x_1,\dots,x_n) \left\{a\Big(M(z,x_1,\dots,x_n) + \ell\, Q(z)\Big) + b\, E(z,x_1,\dots,x_n)\right\} .$$

<u>EXAMPLE</u>. To calculate $\gamma_{a,b,\ell,p}(z_o)$ for a Turing machine Z_o computing 2x, where $\ell = 1$, $a = b = 10$, and p is a singulary function such that

$$p(x) = \begin{cases} o & \text{if } x=o\ ; \\ \frac{90}{\pi^4}\frac{1}{x^4} & \text{if } x>o\ . \end{cases}$$

(Thus $\sum_{x=o}^{\infty} p(x) = 1$, since $\sum_{x=1}^{\infty} \frac{1}{x^4} = \frac{\pi^4}{90}$).

A suitable Z_o is the following:

q_1 1 b q_1
q_1 b R q_2
q_2 1 $s_3 q_3$
q_3 s_3L q_3
q_3 1 L q_3
q_3 b 1 q_4
q_4 1 R q_4
q_4 s_3R q_4
q_4 1 1 q_2
q_4 b L q_5
q_5 $s_3$1 q_6
q_6 1 L q_5
(ends at $q_5$1)

$$\begin{aligned} \text{Here}\quad Q(z_o) &= 133, \\ M(z_o,x) &= 2x + 1, \\ \text{and } E(z_o,x) &= 2x^2 + 5x + 3 ; \end{aligned}$$

$$\therefore \gamma(z_o) = \sum_{x=1}^{\infty} \frac{90}{\pi^4} \left(\frac{20x^2 + 70x + 137}{x^4}\right)$$

$$= 217.78\ldots$$

<u>$\eta_{a,b,\ell,p}(f)$</u>, <u>$\phi_{a,b,\ell,p}(z)$</u> and <u>$\phi'_{a,b,\ell,p}(z)$</u> are defined similarly.

In the last two expressions Z is viewed as an algorithm for a function of the same degree as that of the probability function $p(x_1,x_2,\ldots,x_n)$. We indicate this degree in the abbreviations <u>$\phi_n(z)$</u>, <u>$\phi'_n(z)$</u> which we employ.

SECTION 2. ON EFFICIENCY: THE OPTIMISATION PROBLEM

SECTION 2.1 THE GENERAL PROBLEM

Our first three theorems examine in increasing depth (and length) the question of proving that one Turing machine computes the same function as another. These lead us to our main limitation theorem, which is concerned with the problem of finding the optimum of such equivalent Turing machines.

LEMMA

The set of Gödel numbers for total n-ary functions is not recursively enumerable.[†]

PROOF

Assume that the set is recursively enumerable, and that the recursive function f(z) enumerates the Gödel numbers concerned. Then

$$U\big(\min_y T(f(x_1),\ x_1,x_2,\ldots,x_n,y)\big) + 1$$

is total and thus, by the hypothesis, for some z_o,

$$U(\min_y T(f(x_1),x_1,x_2,\ldots,x_n,y)) + 1 \;=\; U(\min_y T(f(z_o),x_1,\ldots,x_n,y)) .$$

Setting $x_1 = z_o$ now gives a contradiction.

[†] This is a simple generalisation of Davis' theorem 6.1 Chapter 5, which gives the result for singulary functions.

NOTATION

By $\underline{k_n}$ we denote the Gödel number of a Turing machine $N(U_1^n(x_1,x_2,\ldots,x_n))$, where $N(x) = 0$, and by $\underline{B_n}$, the recursive binary function with the property that †

$$[B_n(u,v)]_n(x_1,\ldots,x_n) = [u]_1([v]_n(x_1,\ldots,x_n)) \quad \text{all } u,v .$$

THEOREM 1. FIRST EQUIVALENCE THEOREM

The predicate $P_n(x,y) \Leftrightarrow [x]_n = [y]_n$ which asserts that the Turing machines X and Y compute the same n-ary function, is not recursive.

PROOF

$$[z]_n \text{ is total} \quad \Leftrightarrow \quad [B_n(k_1,z)]_n = [k_n]_n \quad .$$

Thus the recursiveness of $P_n(x,y)$ would give the recursiveness of " $[z]_n$ is total" in contradiction to the lemma.

DEFINITION

By an <u>index</u> of a partial recursive function, we mean a Gödel number of a Turing machine which computes it.

---

†That such a recursive function exists follows from Kleene's Iteration theorem (theorem 23, p.342 [17]).

DEFINITION

A function $g^{(n)}$ [†]is said to be a subfunction of a function $f^{(n)}$ if the domain D of $g^{(n)}$ is a subset of that of $f^{(n)}$, and if for all $(x_1,\dots,x_n) \in D$, $g^{(n)}(x_1,\dots,x_n) = f^{(n)}(x_1,\dots,x_n)$. (We use the term 'subfunction' in preference to that of 'restriction'.)

A predicate $P(x_1,\dots,x_n)$ is said to be a subpredicate of a predicate $Q(x_1,\dots,x_n)$ if its characteristic function is a subfunction of that of $Q(x_1,\dots,x_n)$.

Since many of the practical problems described in the introduction involve (or by taking completions can be made to involve) total functions, it is of particular interest to examine the predicate $[x]_n = [y]_n$ with regard to the indices of these functions. To this end we can strengthen the first equivalence theorem as follows:

THEOREM 2. SECOND EQUIVALENCE THEOREM

No subpredicate of $[x]_n = [y]_n$ defined over a domain which includes the indices of the total n-ary functions is partially recursive.

PROOF

Let the characteristic function of $[x]_n = [y]_n$ be

[†] Davis' notation for an n-ary function.

$r(x,y)$, and let $\alpha(x)$ be the function (using Davis' notation) such that

$$\alpha(x) = \begin{cases} 0 \text{ if } x > 0, \\ 1 \text{ if } x = 0. \end{cases}$$

Consider then a function $f_n(z)$ defined as follows:

$$f_n(z) = \alpha\big(r(B_n(k_1,z),\ k_n)\ \big) - 1.$$

It is clear that if $r(x,y)$ is partially recursive, so is $f_n(z)$. But the domain of $f_n(z)$ consists exactly of the indices of the total n-ary functions, and this would then give their recursive enumerability, which is in contradiction to the lemma. Thus $r(x,y)$, and hence $[x]_n = [y]_n$, is not partially recursive.

At this stage we note that a form of the limitation theorem which follows later, is immediately obtainable with regard to the restricted optimisation function.

<u>THEOREM 3</u>. <u>'LIMITATION THEOREM FOR THE RESTRICTED OPTIMISATION FUNCTION</u>

There are values of the parameters a,b,ℓ,p for which neither the restricted optimisation function $\phi'_{a,b,\ell,p}(z)$, nor any subfunction of it which is defined over a domain that includes the indices of the total recursive functions, is partially recursive.

PROOF

Let a and ℓ be non-zero and $p(x_1, \ldots, x_n)$ any probability function, e.g. one with a finite significant set, with respect to which $\phi'_n(k_n)$ is defined. Since a and ℓ are non-zero, there is an upper bound on the Q-measure that any of the optimum Turing machines for the function $[k_n]_n$ may have, and hence the set of such Turing machines is a finite one. Denote this set as H.

Then

$$[z]_n \text{ is total} \iff \text{the predicate } \phi'_n(B_n(k_1, z)) \in H \text{ is true .} \qquad (A$$

If ϕ'_n (or a subfunction of it defined over a domain which includes the indices of the total n-ary functions) were partially recursive, (A) would provide a means of recursively enumerating the Gödel numbers of the total n-ary functions in contradiction to the lemma.

NOTE. The above result holds even if we restrict the significant sets considered to finite ones. This is in direct contrast to the result we later obtain for the optimisation function $\phi_n(z)$. See theorem 7, P.36.

DEFINITION

If, for $i = 1(1)m$, $Q_i(x_1,\ldots,x_n)$ is a predicate and $f_i(x_1,\ldots,x_n)$ is a function, then the function $g(x_1,\ldots,x_n)$ denoted by the conditional form

$$\left[Q_1(x_1,\ldots,x_n) \rightarrow f_1(x_1,\ldots,x_n);\ Q_2(x_1,\ldots,x_n) \rightarrow f_2(x_1,\ldots,x_n);\ldots \right.$$
$$\left. \ldots;Q_m(x_1,\ldots,x_n) \rightarrow f_m(x_1,\ldots,x_n) \right]$$

is defined as $f_t(x_1,\ldots,x_n)$ for the minimum coefficient t in $1(1)m$ such that $Q_t(x_1,\ldots,x_n)$ is true but $Q_i(x_1,\ldots,x_n)$ is false for all $i < t$. (g is undefined if this function $f_t(x_1,\ldots,x_n)$ is undefined, or if $Q_s(x_1,\ldots,x_n)$ is undefined where S is the least value of i for which $Q_i(x_1,\ldots,x_n)$ is not false, or if all the predicates are false.)

DEFINITION

The symbols F. and T. are used to denote arbitrary tautologously false and true predicates respectively.

The latter, used in place of $Q_m(x_1,\ldots,x_n)$ in the conditional form above, would have the sense of "or otherwise".

Kleene [17] has proved the partial recursiveness of the function defined by the conditional form (which he calls "definition by cases") in the special circumstances where (a) the predicates and functions involved are total recursive (p.229); (b) the predicates are partially recursive and mutually exclusive (p.337). Neither proof is extendable to the general case we require, so we supply this as follows:

<u>THEOREM 4.</u> <u>CONDITIONAL FORM THEOREM</u>

If the predicates and functions involved in McCarthy's conditional form are partially recursive, so is the function which it defines.

<u>PROOF</u>[†]

Let $t(x)$ be the function $\mathcal{L}(x)\,\mathrm{GL}x$.

If $g(x_1,\ldots,x_n) = [Q_1(x_1,\ldots,x_n) \rightarrow f_1(x_1,\ldots,x_n); \ldots$
$\ldots; Q_m(x_1,\ldots,x_n) \rightarrow f_m(x_1,\ldots,x_n)]$,

---

[†] We employ here the predicate T and functions U, min, $\mathcal{L}$, GL defined in Davis. "$T(z,x_1,\ldots,x_n,y)$" (a standard notation) stands for "y is the Gödel number of a computation of Z for the argument $(x_1,x_2,\ldots,x_n)$"; U(x) is the number of tallies in the last of a sequence of instantaneous descriptions with Gödel number x. "$\min_y$" is an abbreviation of "the minimum y such that"; $\mathcal{L}(x)$ is the sequence number (in the sequence of all primes arranged in ascending order) of the highest prime factor of x. xGLy is the exponent of the prime with sequence number x in the prime factorisation of y.

These recursive functions and predicates are part of a numbered list pp.58-62 [9], which we shall refer to in future simply as <u>Davis' definitions (1)-(26A)</u>.

and if, for $i = 1(1)m$, r_i is an index of the characteristic function of Q_i and z_i an index of f_i,

then $g(x_1,\ldots,x_n) =$ [†]

$$\begin{aligned} U\, t\, \min_y \Big(& \{ T(r_1,x_1,\ldots,x_n,1GLy) \wedge U(1GLy) = o \wedge T(z_1,x_1 \ldots,x_n,2GLy) \} \\ \vee & \{ T(r_1,x_1,\ldots,x_n,1GLy) \wedge T(r_2,x_1,\ldots,x_n,2GLy) \wedge \\ & U(1GLy) = 1 \wedge U(2GLy) = o \wedge T(z_2,x_1,\ldots,x_n,3GLy) \} \\ \vee & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \vee & \{ T(r_1,x_1,\ldots,x_n,1GLy) \wedge \ldots \wedge T(r_m,x_1,\ldots,x_n,mGLy) \wedge \\ & U(1GLy) = 1 \wedge \ldots \wedge U(m-1GLy) = 1 \wedge U(mGLy) = o \wedge T(z_m,x_1,\ldots,x_n,m+1GLy) \} \Big) \end{aligned}$$

The partial recursiveness of $g(x_1,\ldots,x_n)$ follows.

NOTATION

If f and g are n-ary functions, we denote the fact that $f(x_1,\ldots,x_n) = g(x_1,\ldots,x_n)$ for all arguments $(x_1,\ldots,x_n)$ in some set S by $f \underset{S}{=} g$.

[†] Where this enhances clarity, we omit parentheses enclosing singulary arguments. In the above formula the argument of t and that of the left-most occurrence of U are written in this way.

THEOREM 5. THIRD EQUIVALENCE THEOREM

Corresponding to any recursive infinite set S of n-ary arguments, there exist a primitive recursive function $G_S(r)$ and a number z_S such that $\left[G_S(r)\right]_n$, for each r, and $\left[z_S\right]_n$ are primitive recursive functions, and such that

$$\left[G_S(r)\right]_n \underset{S}{=} \left[z_S\right]_n \iff \neg \bigvee_y T(r,r,y)$$

Taking S as the set of all n-tuples we obtain immediately the corollary:

COROLLARY

No subpredicate of $[x]_n = [y]_n$, defined over a domain which includes the indices of the primitive recursive functions, is potentially partially recursive.

PROOF OF THEOREM 5

Method of Proof

We define primitive recursive functions $f_S(x_1,\dots,x_n)$ and, for each r, $f_{S,r}(x_1,\dots,x_n)$ in such a way that for $(x_1,\dots,x_n) \in S$, $f_S(x_1,\dots,x_n)$ enumerates the domain of $\min_y T(x,x,y)$, whereas for each r, $f_{S,r}(x_1,\dots,x_n)$ is specified as follows: for all $(x_1,\dots,x_n) \in S$ such that $f_S(x_1,\dots,x_n) \neq r$, $f_{S,r}(x_1,\dots,x_n) = f_S(x_1,\dots,x_n)$, whereas for all $(x_1,\dots,x_n) \in S$ such that $f_S(x_1,\dots,x_n) = r$,

$f_{s,r}(x_1,\ldots,x_n) \not\simeq f_s(x_1,\ldots,x_n)$. It will then follow that

$$f_{s,r} \underset{S}{\equiv} f_s \leftrightarrow \neg \bigvee_y T(r,r,y) \quad , \qquad (B)$$

and the theorem is then obtained by a consideration of the Gödel numbers involved.

<u>Definition of $f_s(x_1,\ldots,x_n)$ and $f_{s,r}(x_1,\ldots,x_n)$</u>

Observe first that

$$\text{if } x_o = 2^{2^9 3^{11} 5^{11} 7^9} ,$$

then $x_o \not\in$ domain of $\min_y T(x,x,y)$, since x_o in this case is the Gödel number† of the non-terminating Turing machine $\{q_1 1 1 q_1\}$.

Secondly, note that if $J(x,y) = ((x+y)^2 + 3x + y)/2$, then, as is shown in Davis [9]:

(1) $J(x,y)$ effects a 1-1 mapping of 2-tuples onto the integers;

(2) there exist recursive functions $K(w)$, $L(w)$ such that $K(w) = x \leftrightarrow \bigvee_y (J(x,y) = w)$ and $L(w) = y \leftrightarrow \bigvee_x (J(x,y) = w)$.

Now define:

$$h(o) = x_o ,$$

$$h(w+1) = \left[T(L(w),L(w),K(w)) \rightarrow L(w);\ T. \rightarrow x_o \right] ;$$

and $$h_r(w) = \left[h(w) \neq r \rightarrow h(w);\ T. \rightarrow x_o \right] .$$

† We employ Davis' system of Gödel numbering p.56 [9].

At least for singulary functions and total S, h and h_r satisfy (B) (if identified with f_s and $f_{s,r}$ respectively). However for non-total S, the forward implication is no longer valid, since even if $\bigvee_y T(r,r,y)$, h and h_r may still compute the same function over S in the case that S contains no w such that

$$L(w-1) = r$$
$$\text{and } T(r,r,K(w-1)).$$

We must thus modify h and h_r in the way we do below (at the same time generalising to n-ary functions).

<u>Auxiliary Functions J_n, L_n^t, and U_s</u>

Define <u>$J_n(x_1,\ldots,x_n)$</u> for $n \geqslant 2$

as $$\underbrace{J(x_1,J(x_2,\ldots\ldots J(x_{n-1},x_n)\ldots))}_{\substack{\text{n-1 applications of}\\ \text{the J function}}},$$

$L_n^t(y)$ for $1 \leqslant t < n$

as $$K(\underbrace{L(L\ldots L(L(y))\ldots)}_{\substack{\text{t-2 applications}\\ \text{of the L function}}}),$$

and $L_n^n(y)$ as $$\underbrace{L(L(L\ldots L(L(y))\ldots))}_{\substack{\text{n-1 applications}\\ \text{of the L function}}}.$$

It follows that for each n ($\geqslant 2$) $J_n(x_1,\ldots,x_n)$ is a primitive recursive function which effects a 1-1 mapping of the set of all n-tuples onto the non-negative integers, and for each t in $[1,n]$, L_n^t is a primitive recursive function such that

$$L_n^t(y) = v \iff \bigvee_{x_1,\ldots,x_n} (x_t = v \wedge J_n(x_1,\ldots,x_n) = y).$$

Denote by $\underline{U_s(x_1,\ldots,x_n)}$ the number of n-tuples $(y_1,\ldots,y_n)$ such that $J_n(y_1,\ldots,y_n) \leqslant J_n(x_1,\ldots,x_n)$ and such that $(y_1,\ldots,y_n) \not\in S$. $U_s(x_1,\ldots,x_n)$ is then primitive recursive since $U_s(x_1,\ldots,x_n) =$

$$\sum_{t=0}^{J_n(x_1,\ldots,x_n)} \left[(L_n^1(t), L_n^2(t), \ldots, L_n^n(t)) \not\in S \rightarrow 1;\ T. \rightarrow 0 \right].$$

These auxiliary functions enable us to give the required definitions for $f_s(x_1,\ldots,x_n)$ and $f_{s,r}(x_1,\ldots,x_n)$ viz:

$$f_s(x_1,\ldots,x_n) = h(J_n(x_1,\ldots,x_n) - U_s(x_1,\ldots,x_n)),$$
$$f_{s,r}(x_1,\ldots,x_n) = h_r(J_n(x_1,\ldots,x_n) - U_s(x_1,\ldots,x_n)).$$

S is infinite, and thus $f_s(x_1,\ldots,x_n)$ enumerates the <u>whole</u> domain of $\min_y T(x,x,y)$, and $f_{s,r}(x_1,\ldots,x_n)$ the whole domain except r.

Thus

$$\bigvee_y T(r,r,y) \Rightarrow f_{s,r} \neq f_s \text{ .} \qquad \text{(C)}$$

Denote the n-tuples in the ordering

$$(L_n^1(x), L_n^2(x), \ldots, L_n^n(x)) \text{ for } x = 1,2,3,\ldots$$

by $a_1, a_2, a_3, \ldots$ respectively. Then if for all $j < k$, $f_s(a_j) = f_{r,s}(a_j)$, and if $a_k \notin S$, then, since

$$f_s(a_k) = f_s(a_{k-1})$$

$$\text{and } f_{r,s}(a_k) = f_{r,s}(a_{k-1}),$$

therefore $f_s(a_k) = f_{r,s}(a_k)$.

Hence the least i such that $f_s(a_i) \neq f_{s,r}(a_i)$ must involve an n-tuple a_i such that $a_i \in s$. So

$$f_{s,r} \neq f_s \Rightarrow f_{s,r} \underset{s}{\neq} f_s \text{ .} \qquad \text{(D)}$$

(C) and (D) supply

$$\bigvee_y T(r,r,y) \Rightarrow f_{s,r} \underset{s}{\neq} f_s \text{ .}$$

On the other hand obviously

$$\neg \bigvee_y T(r,r,y) \Rightarrow f_{s,r} = f_s$$

and thus in particular $f_{s,r} \underset{s}{=} f_s$. These relations give (B).

The Gödel Numbers of f_s and $f_{s,r}$

By our result on McCarthy's conditional form, $h(w)$ is recursive, and since both J_n and U_s are recursive, so is f_s. We may thus denote by z_s, the Gödel number of a Turing machine which computes it. We show further that there exists a primitive recursive function $G_s(r)$ such that for each r

$$\left[G_s(r)\right]_n = f_{s,r} .$$

The proof employed in the conditional form theorem ensures that $h_r(w)$ is recursively expressible in terms of w and the Gödel numbers of $h(w)$, x_o (a constant function), and the characteristic function $N(w) = o$ of the predicate T., as well as the Gödel number, for each r, of the recursive singulary predicate $h(w) \neq r$. Denote this predicate by $H_r(w)$. Let v be the Gödel number of the characteristic function of $h(w) \neq r$, considered as a binary predicate in w,r. Then, by Kleene's iteration theorem, there exists a recursive binary function S^1 such that $S^1(v,r)$ is, for each r, the Gödel number of the characteristic function of $H_r(w)$. Thus, for some recursive function d,

$$h_r(w) = d(S^1(v,r),w) \text{ for each } r,$$

and hence from the recursiveness of J_n and U_s we obtain that for some recursive function q_s

$$f_{r,s}(x_1,\ldots,x_n) = q_s(r,x_1,\ldots,x_n).$$

If C_s computes the right-hand side, considered as a $n+1$ -ary function in $r,x_1,\ldots,x_n$, then $[S^1(c_s,r)]_n$ computes $f_{r,s}(x_1,\ldots,x_n)$ for each r. Denote $S^1(c_s,r)$ by $G_s(r)$. The theorem follows by substituting for Gödel numbers in (B) which gives

$$[G_s(r)]_n \underset{s}{=} [z_s]_n \quad \longleftrightarrow \quad \neg \bigvee_y T(r,r,y).$$

We can now treat the fundamental question involved in the optimisation of algorithms: "Can optimisation be effectively achieved?"

<u>THEOREM 6</u>. <u>THE MAIN LIMITATION THEOREM</u>

There are values of the parameters a,b,ℓ,p for which neither the optimisation function $\phi_{a,b,\ell,p}(z)$, nor any subfunction of it which is defined over a domain which includes the indices of the primitive recursive functions, is potentially partially recursive.

PROOF

Let a and ℓ be non-zero and $p(x_1,\ldots,x_n)$ any probability function with an infinite significant set S with respect to which $\phi_n(z_s)$ is defined. One example of such a probability function is

$$p(x_1,\ldots,x_n) = \frac{1}{2\mu(z_s,x_1,\ldots,x_n)(J_n(x_1,\ldots,x_n)+1)^2}$$

for all $(x_1,\ldots,x_n)\in S$,

in which case $\gamma(z_s) = \frac{1}{2}\Sigma\frac{1}{(J_n(x_1,\ldots,x_n)+1)^2} \leqslant \frac{\pi^2}{12}$.

It can easily be shown that a Turing machine $G_s(r)$ may be constructed from Z_s such that

$$\mu(G_s(r),x_1,\ldots,x_n)\Big/\mu(z_s,x_1,\ldots,x_n) \quad \text{is bounded}$$

by a linear function of r independent of $x_1,\ldots,x_n$. It follows that whenever $\phi_n(z_s)$ is defined, so is $\phi_n(G_s(r))$ for each r.

Since a and ℓ are non-zero, there exists a finite set H of optimum Turing machines such that $[z]_n \mathrel{\overline{\overline{\varsigma}}} [\phi(z_s)]_n$. We then have

$$\phi_n(G_s(r))\in H \iff [G_s(r)]_n \mathrel{\overline{\overline{\varsigma}}} [z_s]_n \quad .$$

Hence, by the equivalence theorem,

$$\phi_n(G_s(r))\in H \iff \neg\bigvee_y T(r,r,y) \quad ,$$

and the result follows.

PROOF

Let z_s and $G_s(r)$, for each r, be the Gödel numbers of primitive recursive functions as defined in the equivalence theorem above.

Then

$$\phi_n(z_s) = \phi_n(G_s(r)) \Leftrightarrow [z_s]_n \underset{s}{\equiv} [G_s(r)]_n \, .$$

Thus by the equivalence theorem

$$\phi_n(z_s) = \phi_n(G_s(r)) \Leftrightarrow \neg \bigvee_y T(r, r, y) \, ,$$

giving the present result.

We have given our most negative result on optimisation first. Within the limitation this theorem imposes, we consider now to what extent optimisation <u>can</u> in fact be accomplished. We examine separately in the first and second optimisation theorems the case in which the significant set is finite and that (such as was treated above) in which it is infinite.

SECTION 2.2 THE PROBLEM FOR FINITE FUNCTIONS

A finite function is one defined over a finite domain. In considering the optimisation of algorithms over finite significant argument sets, it is with such functions that we are concerned.

It might be thought at first sight that these functions may be adequately dealt with by programs of a 'table of values' kind, in which a Turing machine goes into a unique state for each argument (or for each difference between the number of tallies in the argument and that required by the value of the function)[†], and then, using corresponding quadruples, adds or deletes the appropriate number of "1"s. However this is not the case if efficiency criteria are taken into account. Consider, for example, that we wished to calculate $x + 2$ over the domain $[0, 1{,}000{,}000]$. Clearly an algorithm for addition such as

$$q_1 1 L q_1$$
$$q_1 b 1 q_2$$

would prove to be more efficient with respect to any

[†] To do this it must first move to the argument's left-hand end.

space-time measure than one which 'stored' (by virtue of its quadruples) the value of the function for each member of the domain.

In most practical cases we are in fact only interested in calculating functions over some such finite domain. Finite functions are of particular relevance to Games Theory. This is borne out by the following observation.

OBSERVATION

The goal of finding the best algorithm for evaluating the next move in any given position in a finite game with perfect information, such as chess or draughts, may be considered as one of optimising an algorithm for a finite singulary function.

Consider some Gödel numbering of all the positions a player may be faced with, as well as of all the moves he may make. Using a method based on the min-max theorem for Games (Von Neumann and Morgenstern [37]), one can effectively determine the best move in each position. As the number of possible positions is finite, this provides an evaluation of the finite function f(x), where f(x) is the Gödel number of a player's best move in the position with number x. (It involves examining the tree of every possible sequence of moves by the players and selecting the best subtree.) The Games Theory problem then becomes that of finding the optimum algorithm for evaluating f(x).

Here again the table of values method, listing the best mōve for every possible position, is not necessarily the most efficient. While for the particular games of chess and draughts the situation is not known, much superior algorithms have been found for some games. A striking example of this is the game of Nim (described e.g. in Hardy and Wright [11]). The algorithm for playing this bears no comparison to that of consulting a table that embraces all possible rows.

The following definitions and lemmas lead up to our optimisation theorem for finite functions, but are also employed elsewhere in the thesis.

DEFINITION

By an act of a Turing machine we mean one of the following operations on the tape:

(a) a move by the scanning head one place to the left;

(b) a move one place to the right;

(c) writing a specified symbol on the scanned square.

<u>DEFINITION</u>

By an <u>action</u> of a Turing machine, we mean a sequence of acts.

<u>LEMMA 1</u>

Corresponding to any action ζ consisting (say) of m individual acts, and any initial instantaneous description, a Turing machine of m quadruples can be defined for which the action ζ is that which would occur in a computation starting with the given instantaneous description.

<u>LEMMA 2</u>

Let ζ_i, $i = 1(1)t$, be a set of actions consisting respectively of m_i acts, $i = 1(1)t$, and let α_i, $i = 1(1)t$, be a set of instantaneous descriptions with the same internal configuration. Then a necessary and sufficient condition that a Turing machine Z can be defined such that, for $i = 1(1)t$, ζ_i is the action which would occur in the computation of Z starting with α_i, is that for all u,v in $1(1)t$,

$\zeta_u \neq \zeta_v \Rightarrow$ for some $w \leqslant m_u, m_v$, the symbol scanned after w acts of ζ_u starting α_u is different from that after w acts of ζ_v starting with α_v.

Further the question of whether such Turing machines as Z exist or not is effectively decidable, and if they do, it is effectively possible to construct one (consisting of $\leqslant \sum_{i=1}^{t} m_i$ quadruples).

THEOREM 7. OPTIMISATION THEOREM FOR FINITE FUNCTIONS

If its significant set is finite, $\phi_n(z)$ is partially recursive.[†]

This may be interpreted as stating that it is effectively possible to find an optimum algorithm for a finite function.

PROOF

If Z_o is any given Turing machine, we begin our evaluation of $\phi_n(z_o)$ by calculating $\gamma(z_o)$. If S is the significant set being considered, this is given by

$$\gamma(z_o) =$$

$$\sum_{(x_1,\ldots,x_n)\in S} p(x_1,\ldots,x_n)\left\{a\Big(M(z_o,x_1,\ldots,x_n) + \ell\, Q(z_o)\Big) + b\, E(z_o,x_1,\ldots,x_n)\right\}.$$

Note that $\gamma(z_o)$ is not defined, nor is $\phi_n(z_o)$, unless $[z_o]_n(x_1,\ldots,x_n)$ is defined for all $(x_1,\ldots,x_n) \in S$,

---

[†] No matter what values are preselected for parameters a, b, ℓ, p.

whereas if $[z_o]_n(x_1,\ldots,x_n)$ is defined for all such arguments, $\gamma(z_o)$ is effectively calculable, as S is finite.[†]

Our space-time measure is defined in terms of the non-negative parameters a,b,p,ℓ, where a and b are not both zero. In devising ways of reducing the set of Turing machines from which Z_o need be selected, to a finite one, we consider separately the following cases:

(1) $b = \ell = o$,

(2) $a \neq o \wedge \ell \neq o$,

(3) $b \neq o \wedge (a = o \vee \ell = o)$.

<u>CASE 1</u> $b = \ell = o$

The measure of computation involved here is simply $aM(z,x_1,\ldots,x_n)$. Using lemma 2, we can directly construct an optimum Turing machine which evaluates $[z_o]_n$ over S by actions that involve setting the left-hand tally blank, moving to the square on the right of those representing the argument, and then

---

[†] We leave it to intuition that the spatial and timing measures $M(z,x_1,\ldots,x_n)$ and $E(z,x_1,\ldots,x_n)$ are computable over the same domain as $[z]_n(x_1,\ldots,x_n)$. A formal proof of this is given in Chapter 3.

adding or deleting the required number of tallies. (By setting the left-hand tally blank before moving right, Z_o^* avoids employing a tape length greater than the initial one in cases where this is possible because $f(x_1,\dots,x_n) \leq$ the number of squares on the initial tape.)

CASE 2. $a \neq o \wedge \ell \neq o$

If $p = \min\limits_{(x_1,\dots,x_n)\in S} p(x_1,\dots,x_n)$, the optimum Turing machine Z_o^* must be such that

$$Q(z_o^*) < \frac{\gamma(z_o)}{p\ a\ \ell} .$$

This places an upper bound on the maximum coefficient of a s- or q- symbol which can occur in Z_o^* , and hence on the number of Turing machines which Z_o^* could be. Consider the subset of these which calculate $[z_o]_n$ over S and for which

$$M(z,x_1,\dots,x_n) < \frac{\gamma(z_o)}{p\ a} \quad \text{for all } (x_1,\dots,x_n) \in S . \qquad \text{(A)}$$

Let Z be any such Turing machine with (say) u s-symbols and v q-symbols. For each argument of S, Z must halt. Since it cannot repeat an instantaneous description, the maximum possible number of steps any of its computations can involve must be less than the number of possible instantaneous descriptions which can be formed using its alphabet on a tape bounded by (A),

$$\text{i.e.} \leq \frac{\gamma(z_o)}{p\,a} \vee u^{\frac{\gamma(z_o)}{pa}}.$$

We can thus effectively select the subset in question, and evaluate $\gamma(z)$ in each case. Z_o^* is a Turing machine for which this is a minimum.

<u>CASE 3</u>. $b \neq o \wedge (a = o \vee \ell = o)$

The limitation we have here on Z_o^*, provided by the bound

$$E(z_o^*,x_1,\ldots,x_n) < \frac{\gamma(z_o)}{p\,b},$$

(or that for $M(z_o^*,x_1,\ldots,x_n)$ if $a \neq o$), does not reduce the set of Turing machines to a finite one. Our method is instead to consider the set of possible actions such Turing machines may have.

If Z is any Turing machine defined over S, some of whose actions involve writing symbols other than a tally, or a blank, a corresponding Turing machine Z' may be defined, using lemma 2, which writes a blank instead for each such act, but has all other acts in common. It follows here that

$$[z']_n \underset{s}{=} [z]_n ,$$
$$\text{and} \quad \gamma(z') = \gamma(z) .$$

Thus we can restrict the Turing machines considered in the determination of $\phi_n(z_o)$, to those involving only the acts:

(1) move left,

(2) move right,

(3) write a tally,

(4) write a blank.

Further, as each action of $\phi_n(z_o)$ for arguments in S must be of length $\leqslant \frac{\gamma(z_o)}{pb}$, and there are at most $4^{\frac{\gamma(z_o)}{pb}}$ such actions, it follows that if S consists of t significant arguments, there are $\leqslant 4^{\frac{\gamma(z_o)t}{pb}}$ possible sets of actions of this kind. Using lemma 2, each set may be examined as to whether it corresponds to any Turing machine Z, and if so, whether or not Z calculates $[z_o]_n$ over S may be determined. Of those Turing machines obtained which do, $\gamma(z)$ can be calculated in each case, and a Turing machine for which $\gamma(z)$ is least selected as $\phi_n(z_o)$. Since

$$\neg (a = o \wedge b = o) \implies$$
$$\big((b = o \wedge \ell = o) \vee (a \neq o \wedge \ell \neq o) \vee (b \neq o \wedge (a = o \vee \ell = o))\big)$$

is a tautology, all possible cases have now been considered.

As the particular measures of computation $E(z,x_1,\ldots,x_n)$ and $M(z,x_1,\ldots,x_n)$ are of particular interest, we state explicitly the following corollaries:

<u>COROLLARY 1</u>.

Given a Turing machine Z_o and a set of probabilities as to the likelihood of any of a finite set S of arguments occurring, it is effectively possible to determine a Turing machine calculating $[z_o]_n$ over S, for which the expected average value of $E(z,x_1,\ldots,x_n)$ is a minimum.

<u>COROLLARY 2</u>.

Similarly for $M(z,x_1,\ldots,x_n)$.

<u>NOTE</u>. The theorem remains provable in a wide range of cases even when the space-time measure employed is other than the linear one considered in this thesis. For instance, where the critical factor is the maximum space required, the following definitions of $\gamma(z)$, applicable only to finite functions, are of interest:

$$\gamma(z) \;=\; a \max_{(x_1,\ldots,x_n)\in S} \bigl(M(z,x_1,\ldots,x_n) + \ell\, Q(z)\bigr) + b \sum_{(x_1,\ldots,x_n)\in S} p(x_1,\ldots,x_n)\, E(z,x_1,\ldots,x_n)$$

or, more strictly,

$$\gamma(z) \;=\; \nu\Bigl(k, \max_{(x_1,\ldots,x_n)\in S} \bigl(M(z,x_1,\ldots,x_n) + \ell\, Q(z)\bigr)\Bigr) + b \sum_{(x_1,\ldots,x_n)\in S} p(x_1,\ldots,x_n)\, E(z,x_1,\ldots,x_n) \quad ,$$

where $\nu(k,x) = o$ if $k \geqslant x$,

and is arbitrarily large if $k < x$.

It may readily be verified that our proofs remain applicable to such choices for the measure function.

SECTION 2.3 A PRACTICABLE SPECIAL CASE

We have emphasised the practical importance of optimising algorithms for finite functions. In the theoretical study above we proved such optimisation to be effectively possible, but the algorithm we supplied to show this involved testing a prohibitively large (though finite) class of Turing machines. It seems that it should be possible to design the most efficient

Turing machine directly, given the significant arguments, their probabilities, and the associated function values. The question we are now considering is that of finding an algorithm, efficient enough to be of practical value, for calculating the optimisation function. This unfortunately turns out to be exceptionally difficult. We confine ourselves to pure-time measures, and give a complete solution in the case where the significant set consists of exactly two arguments. (The problem is trivial if the significant set consists of only one argument.)

NOTATION

We denote the optimum Turing machine required as Z^* and the significant arguments in ascending order as a_1-1, a_2-1,† the function value $[z^*]_1(a_1-1)$ as b_1, and $[z^*]_1(a_2-1)$ as b_2, and the probability that arguments a_1-1, a_2-1 will occur as p, 1-p‡ respectively.

---

†In Turing machines an argument x is stored as x+1 tallies; thus a_1-1, a_2-1 are represented respectively by a_1, a_2 "1" s.

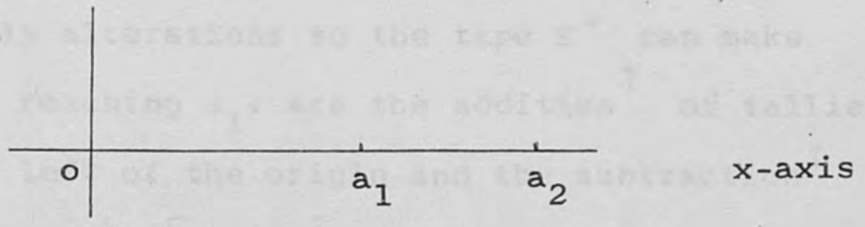
‡The case, which we have allowed, in which the sum of the probabilities is less than unity, is obtainable by writing p as p_1 and replacing 1-p by p_2 in all its occurrences in expressions for $\Upsilon(z^*)$ that we give below.

Method of Analysis

As the size of the program is not relevant, we need only specify the type of action which would occur in the optimum Turing machine. It will be apparent that the encoding in each case of quadruples to effect the action required presents no difficulty. The choice of Z^*'s action (and hence of Z^*) is given, in each of 6 cases which we distinguish (according to the relative magnitudes of a_1, a_2, b_1, b_2) by calculating $\gamma(z\ )$ for not more than 4 alternatives, according to expressions supplied in terms of the parameters a_1, a_2, b_1, b_2, p, and chosing that for which $\gamma(z\ )$ is least. We allow the determination of Z^* to be made according to such a simple selection by reducing, in the four theorems which follow, the set of Turing machines that need be considered.

Graphical Representation

The various types of actions are represented graphically. The initial position of the reading head (scanning the leftmost tally) is taken as the origin, and the squares of the tape as the integer coordinates along the x- axis. Thus a_1, a_2 (the number of tallies in the argument) correspond to points on the axis as shown:



<u>DEFINITION</u>

<u>Event 1</u> is the contingency in which argument a_1-1 occurs; <u>event 2</u>, that in which a_2-1 occurs.

<u>THEOREM 8</u>

(a) Any steps in the actions for events 1 and 2 which involve introducing symbols other than the tally or blank may be replaced, without altering the total number of tallies at the computation's termination, or the timing measure, by setting (or leaving) the square concerned blank. We may thus confine ourselves to Turing machines which employ only blanks and tallies.

(b) Unless $b_1 - a_1 = b_2 - a_2$, Z^* must proceed eventually to a_1[†] in order to distinguish between event 1 and event 2.

[†] i.e. to the square with coordinate a_1.

(c) The only alterations to the tape Z^* can make before reaching a_1, are the addition[†] of tallies to the left of the origin and the subtraction[†] of tallies in $[o,a_1-1]$; the most efficient way it can do this (involving one or no changes of direction) is to move left and add tallies if so required, then move right to a_1, subtracting tallies where necessary from within $[o, a_1-1]$ 'on the way'.

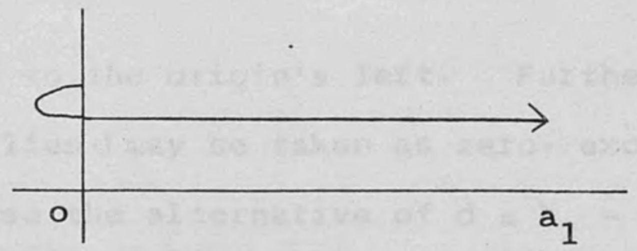
DEFINITION

The action of a Turing machine before reaching a_1 is called its P- action ("P" standing for "preliminary"). The most efficient actions for events 1 and 2 which can occur after a P- action is complete are called the corresponding F- actions ("F" standing for "following").

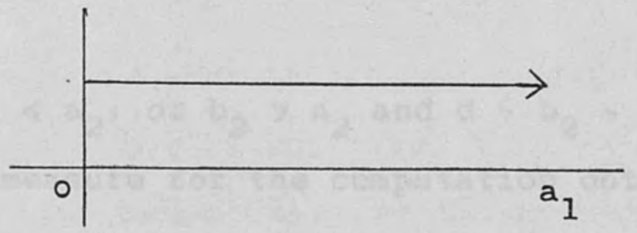
Thus, if $b_1 - a_1 \neq b_2 - a_2$, the P- action for Z^* is either of the form

---

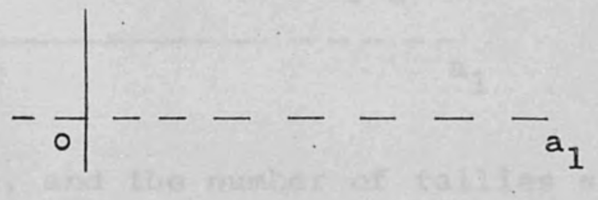
[†] By 'adding a tally', we mean changing the symbol in one of the tape squares from a blank to a "1". Similarly, by 'subtracting a tally', we mean changing a "1" to a blank.



or



The P- action may also be represented as

 (A)

in which the dashes indicate groups of tallies and the gaps between them blanks.

NOTATION d, t

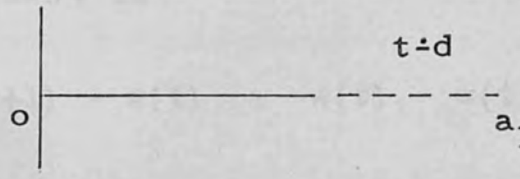
Denote the number of tallies added to the left of the origin as d, and the number of tallies subtracted in $[o,a_1-1]$ as t.

THEOREM 9

It is sufficient to consider for Z^*, Turing machines with the tallies, if any, created by the P- action to the left of the origin, grouped in an unbroken sequence immediately to the origin's left. Further the number of such tallies d may be taken as zero, except if $b_2 > a_2$, in which case the alternative of $d = b_2 - a_2$ may be required.

PROOF

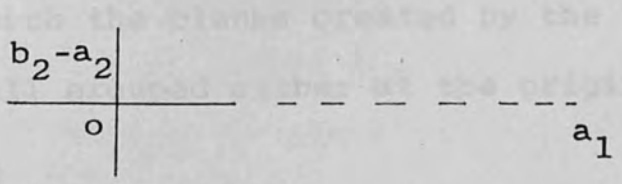
If $b_2 \leqslant a_2$, or $b_2 > a_2$ and $d < b_2 - a_2$, compare the timing measure for the computation obtained in events 1 and 2 with P- action as in (A), to that with P- action as in (B):



(B)

where $d' = o$, and the number of tallies subtracted in $[o, a_1 - 1]$ is reduced to $t \dot{-} d$ by eliminating subtraction from the left. In each case a gain in efficiency is obtained by using (B)'s P- action.

If $b_2 > a_2$ and $d \geqslant b_2 - a_2$, a similar gain occurs if the P- action in (C) is employed:



(C)

Here $d' = b_2 - a_2$ and the number of tallies subtracted in $[o, a_1 - 1]$ is reduced to $t \dot{-} ((b_2 - a_2) - d)$ by eliminating subtraction from the left.

THEOREM 10. A 'STABILITY' CRITERION FOR t

Let $w(t)$ be the total average timing measure for a P- action which creates t blanks in $[o,a_1-1]$ followed by its corresponding F- action. Similarly let $w(t-1)$, $w(t+1)$ be that for the cases respectively in which t is reduced by 1 by eliminating the leftmost (rightmost) blank, or in which t is increased by 1 by creating an additional blank immediately to the left of the present leftmost one (to the right of the present rightmost one). Then, if

$$w(t+1) - w(t) \neq w(t) - w(t-1) ,$$

one can improve efficiency by altering t by 1.

PROOF

Obvious.

THEOREM 11

It is sufficient to consider for Z^* Turing machines in which the blanks created by the P-action in $[o,a_1-1]$ are all grouped either at the origin or at a_1-1.

PROOF

Consider how alterations to the distribution of blanks in $[o,a_1-1]$ (but not to their total number) affect efficiency. Note that

(a) in event 1, if the F- action involves addition, changes to the distribution do not alter the total average timing measure for this event;

(b) in either event, if the F- action involves subtraction, the blanks are best grouped at the origin;

(c) In event 2, if the F- action involves either addition of tallies to the left of the origin, or to the right of a_2, the distribution of blanks is irrelevant;

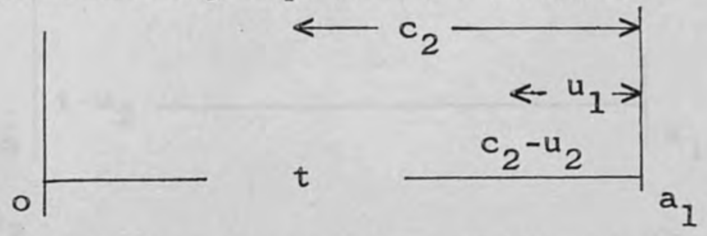
(d) in event 2, if the F-action involves addition of tallies in $[o,a_1-1]$ (i.e. in positions made blank in the P- action), the blanks are best grouped at a_1-1.

It follows that the blanks may be grouped at the origin or at a_1-1 in all combinations of cases for events 1 and 2, except the as yet unresolved one, where the F- actions involve subtraction in event 1 and addition in event 2 within $[o,a_1-1]$.

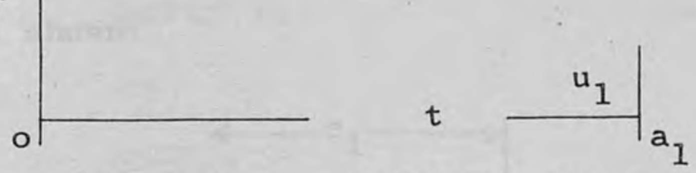
Let u_1,u_2 denote the number subtracted or added respectively in events 1 and 2, and let u_1' , u_2' denote the new values for u_1,u_2 in the various modified actions we present for comparison. Let c_1 be the distance the Turing machine considered moves to the left during the F- action (which starts at a_1) in event 1, and let c_2 be that for event 2.

If $c_2 \geqslant c_1$

Modifying the P- action by altering the distribution of blanks in $[a_1-c_2, a_1-1]$, will not increase the timing measure for event 2; for event 1, the best arrangement is obtained if the blanks in the interval are all grouped at the left. Further, altering the distribution in $[o, a_1-c_2-1]$ does not affect the timing measure in either case. By grouping the blanks at the left in the former interval, and at the right in the latter, we obtain the following improved P- action:

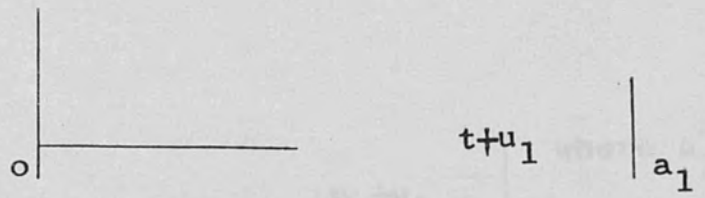


This can be further improved upon (for both events) by displacing the group of blanks $(c_2-u_2)-u_1$ squares to the right:



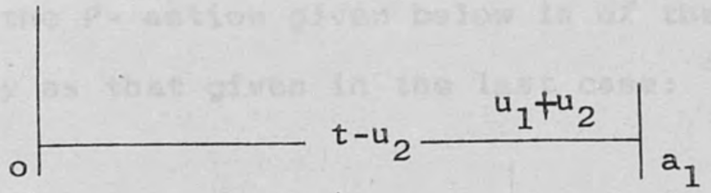
(D)

Increasing t, the number of blanks created, by one, causes $u_2' = u_2+1$, $u_1' = u_1-1$ (provided $u_1 \geqslant 1$). If δw is the increment this produces on the total average timing measure, then $-\delta w$ will be that involved if t is reduced by one (provided $u_2 \geqslant 1$). Thus by the stability criterion, theorem 10, one or other of the following P- actions is superior to (D):

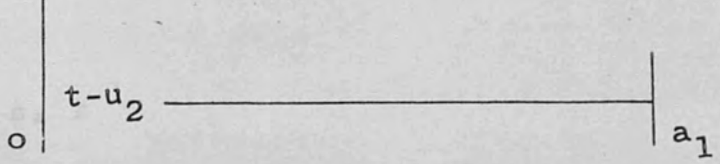


in which $u_1' = o$, $u_2' = u_2 + u_1$,

or

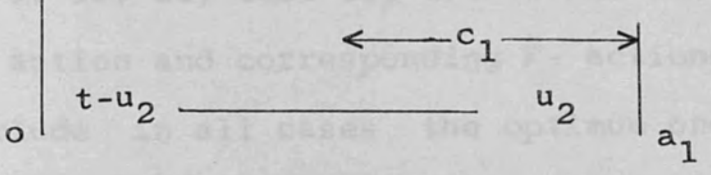


in which $u_1' = u_1 + u_2$, $u_2' = o$; and in the last case, the following P- action is as efficient:

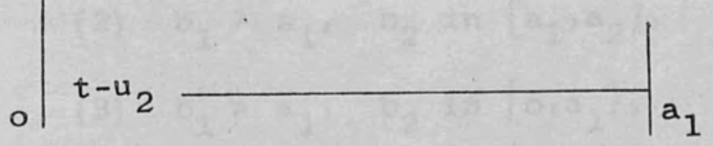


<u>If $c_1 > c_2$</u>

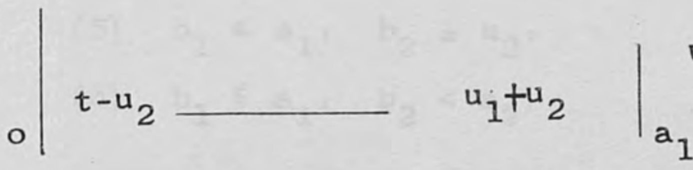
An argument similar to the above shows that we can obtain a more efficient P- action by distributing the blanks as shown:



The stability criterion then provides that one of the following P- actions is a further improvement

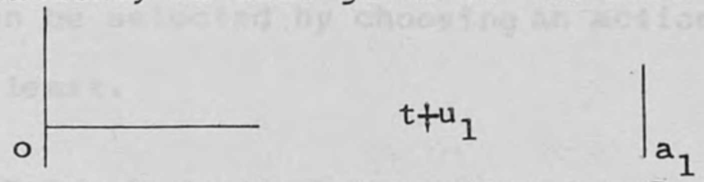


where $u_1' = u_1+u_2$,
$u_2' = o$;

or 

where $u_1' = o$,
$u_2' = u_1 + u_2$.

Further, the P- action given below is of the same efficiency as that given in the last case:



We have thus shown that any P- action can be improved by grouping the blanks at o or a_1-1.

<u>NOTATION</u> s, r

We denote the number of blanks grouped at the origin which the P- action creates as s, and the number grouped at a_1-1 as r.

It may readily be verified, using theorems 8, 9, 10, 11, that the set of alternatives for the P- action and corresponding F- actions which we now provide include in all cases the optimum ones. The six cases we consider are for

(1) $b_1 \geqslant a_1$, $b_2 \geqslant a_2$,

(2) $b_1 \geqslant a_1$, b_2 in $[a_1, a_2)$,

(3) $b_1 \geqslant a_1$, b_2 in $[o, a_1)$,

(4) $b_1 \leq a_1$, $b_2 > a_2$,

(5) $b_1 \leq a_1$, $b_2 = a_2$,

(6) $b_1 \leq a_1$, $b_2 < a_2$,

and in each we give expressions for the timing measure $\gamma(z)$ in terms of the parameters a_1, a_2, b_1, b_2, p. Z^* may then be selected by choosing an action for which $\gamma(z)$ is least.

<u>CASE 1</u> $b_1 \geq a_1$, $b_2 \geq a_2$

(a) If $b_1 - a_1 = b_2 - a_2$ add this number of tallies to the origin's left, otherwise:

The alternative parameters for the P- action that need be considered	The corresponding F- actions[†]
either (b) $d = o$ and $r = o$	In event 1, add to the right. In event 2, add to the left of the origin or to the right of a_2 according to whether $a_1 \leq \frac{1}{2} a_2$ or not.

[†] We omit obvious details, such as how many tallies need be added or subtracted, or that the addition, subtraction in the direction indicated is to be made respectively at the first blank or marked squares encountered.

or (c) $d = b_2 - a_2$ and $r = o$	In event 1, subtract to the left or add to the right (according to whether $(b_1-a_1) > (b_2-a_2)$ or not). No F- action is necessary for event 2.
or (d) $d = b_2 - a_2$ and $r = (b_2-a_2) \dot{-} (b_1-a_1)$ †	In event 1, no F- action is necessary. In event 2, add to the left.

$\gamma(z)$ in these cases is‡

(a) $2(b_1-a_1)$

(b) $a_1 + 2p(b_1-a_1) + (1-p)\big(\min.(a_1, a_2-a_1) + 2(b_2-a_2)\big)$

(c) $a_1 + 3(b_2-a_2) + 2p\,|(b_2-a_2) - (b_1-a_1)|$

(d) $a_1 + 3(b_2-a_2) + (b_2-a_2) \dot{-} (b_1-a_1) + 2(1-p)((b_2-a_2) \dot{-} (b_1-a_1))$

---

† Theorem 10 precludes r being in $(o, (b_2-a_2) \dot{-} (b_1-a_1))$.

‡ Note that a Turing machine takes one time unit per movement of its head, plus an extra time unit for each change of symbol that it makes.

(Only (b) and (c) need be considered if $b_2-a_2 < b_1-a_1$; and only (c) and (d) if $b_2-a_2 > b_1-a_1$. (c) is better than (d) if $p < \frac{3}{4}$)

<u>CASE 2</u> $b_1 \geqslant a_1$, b_2 in $[a_1,a_2)$

<u>Alternative parameters for the P- action</u>	<u>Corresponding F- action</u>
(a) $s = o$	In event 1, add to the right
(b) $s = \min.(a_2-b_2,a_1)$	In event 2, subtract to the right,

$\gamma(z)$ in these cases is

$$a_1 + s + 2p(b_1-a_1+ s) + 2(1-p)((a_2-b_2) \dot{-} s).$$

((a) is in fact better than (b) if $p > \frac{1}{4}$.)

<u>CASE 3</u> $b_1 \geqslant a_1$, b_2 in $[o,a_1)$

<u>Alternative parameters for the P- action</u>	<u>Corresponding F- actions</u>
(a) $s = o$	Event 1: add right. Event 2: if $a_1 > \frac{1}{2}a_2$, subtract $(a_2-b_2) \dot{-} a_1$ to the right, then move left subtracting a further $\min.(a_1,a_2-b_2)$ tallies; if $a_1 < \frac{1}{2}a_2$ follow a similar algorithm with directions reversed.

(b) $s = \min.(a_2-b_2, a_1)$ Event 1: add right.

Event 2: subtract $(a_2-b_2) \dot{-} s$ tallies to the right.

((c) $s = \min.(a_2-b_2, 2a_1-a_2, a_1)$)

Use of theorems 8-11 reduces the possible cases to (a), (b), (c) as shown; however we can also eliminate (c), since it only forms a distinct case if

$$2a_1-a_2 < \min.(a_2-b_2, a_1),$$

and here the following implication holds:

case (c) is better than case (a)

$\Longrightarrow$ case (b) is better than case (c).

$\gamma(z)$ for (a) is

$$a_1 + 2p(b_1-a_1) + (1-p) \min. \{ 3((a_2-b_2) \dot{-} a_1) + 2\min.(a_1, a_2-b_2), a_1 + 2a_2 - 3b_2 \} .$$

$\gamma(z)$ for (b) is as in case 2b.

<u>CASE 4</u>. $b_1 \leq a_1$, $b_2 > a_2$

<u>Alternative parameters for P- action</u>	<u>Corresponding F- actions</u>
(a) $d = o$ $r = o$	Event 1: subtract left. Event 2: add to the left or right according to whether $a_1 < \frac{1}{2}a_2$ or not.

(b)	$d = b_2-a_2$ $\quad r = o$	Event 1:	subtract left.
		Event 2:	no action necessary.
(c)	$d = o$ $\quad r = a_1-b_1$	Event 1:	no action necessary.
		Event 2:	add to the left or right according to whether $a_1 < a_2-b_1$ or not.
(d)	$d = b_2-a_2$	Event 1:	no action necessary,
	$r = (a_1-b_1 + b_2-a_2)$	Event 2:	add to the left.

(if $a_1-b_1 + b_2-a_2 > a_1$)

$\gamma(z)$ is

(a) $a_1 + 2p(a_1-b_1) + (1-p)\left\{2(b_2-a_2) + \min.(a_1,a_2-a_1)\right\}$

(b) $a_1 + 3(b_2-a_2) + 2p(a_1-b_1 + b_2-a_2)$

(c) $2a_1-b_1 + (1-p)\left\{2(b_2-a_2) + (a_1-b_1) + \min.(a_1,a_2-b_1)\right\}$

(d) $a_1 + 3(b_2-a_2) + (a_1-b_1 + b_2-a_2) + 2(1-p)(a_1-b_1 + b_2-a_2)$

CASE 5. $b_1 \leqslant a_1$, $b_2 = a_2$

<u>Alternative parameters for P-action</u>		<u>Corresponding F-actions</u>
(a) $r = o$	Event 1:	subtract left.
	Event 2:	no action necessary.
(b) $r = a_1-b_1$	Event 1:	no action necessary,
	Event 2:	add left.

$\gamma(z)$ is

(a) $a_1 + 2p(a_1-b_1)$

(b) $2a_1-b_1 + 2(1-p)(a_1-b_1)$

((a) is better than (b) if $p < \frac{3}{4}$.)

CASE 6. $b_1 \leqslant a_1 \quad b_2 < a_2$

Starting from the origin, subtract to the right $\min.\left\{(a_1-b_1), (a_2-b_2)\right\} = k$ (say) tallies. (This will involve 2k time units.)

If

$$a_1' = a_1-k$$

and $a_2' = a_2-k$,

consider the new problem now formed with parameters a_1', a_2', b_1, b_2, p :

(a) if $a_1-b_1 = a_2-b_2$, the calculation is complete;

(b) if $a_1-b_1 < a_2-b_2$, $b_1 = a_1'$ and $b_2 < a_2'$ and thus either case 2 or 3 applies;

(c) if $a_1-b_1 > a_2-b_2$, $b_1 < a_1'$ and $b_2 = a_2'$, and case 5 applies.

SECTION 2.4 THE PROBLEM FOR INFINITE FUNCTIONS

In spite of our main limitation theorem, we showed that optimisation is effective in the case where the significant set considered is finite; we now consider to what extent it can be achieved with infinite significant sets. Our first two theorems are concerned with delimiting the domain over which $\phi(z)$ is at least defined; the third reduces the determination of $\phi(z)$, for cases when it is defined, to the calculation of a particular recursive function $\chi(t)$ for any sufficiently large value t.[†]

THEOREM 12

For infinite significant sets and pure-time measures:

No optimum Turing machine exists for any function $f(x)$ such that $f(x)-x$[‡] is monotonically increasing.

[†] How large t must be is not, however, in general effectively determinable.

[‡] For formulations of Turing machines which use a coding of numbers based on a radix $k > 1$ instead of the unary coding employed in Davis, the corresponding result applies to functions for which $\frac{f(x)}{x}$ is monotonically increasing.

PROOF

Let the arguments in the significant set S, taken in ascending order, be denoted by $x_o, x_1, x_2, \ldots$. Assume that an optimum Turing machine Z^* calculating $f(x)$ over S does exist, and denote the number of q-symbols it contains by m. As S is infinite, for each argument x_i, Z^* must eventually move to the first blank on the right (at square $x_i + 1$), otherwise $[Z^*]_1(x_t) - x_t$ will be constant for all $t \geqslant i$. The shortest possible computation in each case is one in which Z^* moves to $x_i + 1$, and then, without reversing, adds $f(x_i) - x_i$ tallies to the right. But Z^* cannot do this for more than m arguments, as it requires a different internal configuration on reaching $x_i + 1$ for each i if it is to then add a different amount of tallies without changing direction.

In contrast, consider a Turing machine Z_h, $h > m$, which behaves in the optimum way for arguments $x_o, x_1, \ldots, x_h$, while for arguments x_i, $i > h$, it reverses direction after reaching $x_h + 1$ and alters the tape in $[o, x_h]$ to the condition Z^* would have produced by the time it reached $x_h + 1$; after this Z_h returns to $x_h + 1$ and behaves from then on as Z^* would. Z_h's computations must involve less steps than those of Z^* for at least one of the arguments (say x_t) in $\{x_o, x_1, \ldots\ldots, x_m\}$;

on the other hand they are at least as short as those of Z^* for all the arguments in $\{x_o, x_1, \ldots, x_m, \ldots, x_h\}$.

The following inequality thus holds:

$$\gamma(z^*) - \gamma(z_h) \geqslant p(x_t)(E(z^*, x_t) - E(z_h, x_t)) - 3\left(\sum_{i=h+1}^{\infty} p(x_i)\right)(x_h + 1)$$

where $p(x_t) > o$ since $x_t \in S$, and
$E(z^*, x_t) - E(z_h, x_t) \geqslant 1$. This gives:

$$\gamma(z^*) - \gamma(z_h) \geqslant p(x_t) - 3\left(\sum_{i=h+1}^{\infty} p(x_i)\right)(x_h + 1) \qquad \text{(A)}$$

We show that the right-hand side is $> o$ for sufficiently large h.

Since each computation involves moving to square x_i+1, $x_i \leqslant E(z^*, x_i)$ for each i ,

and thus $\sum_{i=o}^{\infty} p(x_i) x_i \leqslant \sum_{i=o}^{\infty} p(x_i)\, E(z^*, x_i) = \frac{\gamma(z^*)}{b}$, which, by the hypothesis for Z^*, is defined.

Hence $\sum_{i=o}^{\infty} p(x_i) x_i$ is also defined. This implies

$$\lim_{h \to \infty} \sum_{i=h+1}^{\infty} p(x_i) x_i = o \; ;$$

which in turn gives

$$\lim_{h \to \infty} \left(\sum_{i=h+1}^{\infty} p(x_i)\right) x_h = o \; .$$

It follows that, for sufficiently large h,

$$p(x_t) - 3(\sum_{i=h+1}^{\infty} p(x_i))(x_h+1) > 0 .$$

This inequality in conjunction with (A) provides a contradiction to our definition of Z^*; the theorem follows.

THEOREM 13. EXISTENCE THEOREM

Let the significant set considered be infinite and $\ell \neq 0$. Then, if and only if the space-time measure employed has a spatial component, the following implication holds:

wherever $\gamma(z)$ is defined, $\phi(z)$ exists

(whatever values are preselected for parameters a, b, ℓ, p).

We regard $\phi(z)$ as existing even when it is not effectively determinable.[†]

PROOF

Necessity. $a = 0$ (i.e. the space-time measure is a pure-time one). If a is zero, we can provide a counterexample to the implication concerned. Consider the example on page 14 where the space-time measure of a Turing machine Z_0 computing 2x is evaluated. Clearly

[†]This view would not be accepted by the Intuitionists.

$\gamma(z_o)$ is defined there,[†] but on the other hand, according to theorem 12, no $\phi(z_o)$ exists.

Sufficiency. $a \neq o$ (i.e. the space-time measure has a spatial component). Let Z_o be a Turing machine for which $\gamma(z_o)$ is defined. As the alphabet of any Turing machine Z for which $\gamma(z) \leq \gamma(z_o)$ cannot involve s-symbols or q-symbols with coefficients

$$\frac{\gamma(z_o)}{a \, \ell \sum\limits_{(x_1,\dots,x_n)} p(x_1,\dots,x_n)} > 2,$$

the number of such Turing machines is finite. The subset of these such that $[z]_n = [z_o]_n$ must in turn include a subset T of Turing machines for which $\gamma(z)$ is defined. But T is finite and non-empty (since it at least contains Z_o itself). Thus there exists $\min\limits_{Z \in T} \gamma(z)$, and $\phi(z_o)$ is some Z' in T such that

$$Z' = \min_{Z \in T} \gamma(z) .$$

[†] viz: $\gamma(z_o) = \frac{90b}{\pi^4} \sum\limits_{x=1}^{\infty} \frac{(20x^2 + 70x + 137)}{x^4}$.

Let us employ, as suggested by the last theorem, measures which include a spatial component. Consider then the following theoretical approach one might take towards finding the optimum Turing machine for an infinite function, such as a total singulary function f(x), where f(x)-x is monotonically increasing. The optimisation theorem for finite functions shows that is possible to determine the optimum Turing machine for f(x) over any finite domain $x = 1(1)t$. One could thus, by successively increasing t, obtain the optimum Turing machines involved over larger and larger domains, and examine how (if at all) they varied. For small t, the optimum Turing machine would be of a table of values kind, in which a distinct action was programmed for each argument. But as we pointed out in the proof of theorem 12, this requires in general at least one quadruple per argument. If Z^* is the optimum Turing machine for f(x), all x, then Z^* is also a candidate in the selection of the optimum Turing machine for f(x) over any finite domain. Further Z^*'s space-time measure over 1(1)t is less than the corresponding measure over all x. Thus an upper bound on the number of quadruples which the optimum Turing machine for f(x) over any finite domain can have, is given by $\frac{\gamma(z^*)}{5\,a\,\ell}$. If

$t \gg \frac{\gamma(z^*)}{5\,a\,\ell}$, this number is too small a quantity to allow the Turing machine to cater for each argument individually. It thus seems reasonable to expect that for sufficiently large t, any efficient algorithm for calculating f(x), for x = 1(1)t, must be cast in an abstract form that in fact calculates f(x) for all x, and that the optimum algorithm for such t will also be the optimum algorithm for computing f(x) for all x. That this <u>is</u> the case for <u>all</u> infinite functions, is shown in the following theorem.

<u>THEOREM 14.</u> <u>OPTIMISATION THEOREM FOR INFINITE FUNCTIONS</u>

For infinite significant sets and measures with a spatial component where $\ell \neq o$:

There exists a primitive recursive function χ_n [†] such that, wherever $\phi_n(z)$ exists ,

$$\chi_n(z,t) = \phi_n(z) \text{ for almost all } t .$$

<u>PROOF</u>

Let Z_o be any Turing machine for which $\phi_n(z_o)$ (Z_o^*) exists. By methods such as that employed in the third equivalence theorem, we may order the n-ary arguments of the significant set S as $a_1, a_2, a_3, \ldots$.

[†] Corresponding to the specific parameters a,b,ℓ,p employed.

Let $\gamma_t(z)$ be the corresponding quantity $\sum_{i=1}^{t} p(a_i)\,\mu(z,a_i)$, where p is the probability function, and Z_t^* that Turing machine which calculates $[z_o]_n$ over a_i, for $i = 1(1)t$, for which $\gamma_t(z)$ is least (i.e. the optimum Turing machine for $[z_o]_n$ over the domain $\{a_1,a_2,\ldots,a_t\}$). The theorem will be shown to hold for a recursive function χ_n such that

$$\chi_n(z_o,t) = \dot{Z}_t^* \quad \text{for each } t,$$

so justifying the heuristic approach we described in the preliminary remarks.

As we mentioned there, Z_o^* also calculates $[z_o]_n$ over $\{a_1,a_2,\ldots,a_t\}$, and hence

$$Q(z_t^*) \text{ must be} \leqslant \frac{\gamma(z_o^*)}{a \,\iota\, p(a_1)} \quad \text{for each } t .$$

If the finite set of Turing machines for which this inequality holds is P_1, Z_t^* is in P_1 for each t, and so is Z_o^*.

Corresponding to any Turing machine Z in P_1 which calculates a different function from that of Z_o over S, there must be some argument $a_t \in S$, such that $[z]_n(a_t) \neq [z_o]_n(a_t)$. Let t_Z be the least coefficient of 'a' for which this is the case; P_2, the subset of P_1 of such Turing machines; and t_{max}, the maximum value of

t_Z for $Z \in P_2$. Clearly then, for $t > t_{max}$, Z_t^* and Z_o^* compute the same function over S, and so are both members of the smaller set $P_1 - P_2$.

It is conceivable, however, that Z_t^* might in certain cases 'oscillate' indefinitely with increasing t among the members of $P_1 - P_2$ other than Z_o^*. We show that this is not the case.

If $[z]_n \underset{S}{\equiv} [z_o]_n$, since $\gamma_t(z)$ is monotonically increasing with respect to t, either there exists $\lim\limits_{t \to \infty} \gamma_t(z)$ or $\lim\limits_{t \to \infty} \gamma_t(z) = \infty$. But for $Z = Z_o^*$, $\lim\limits_{t \to \infty} \gamma_t(z)$ does exist, viz: $\gamma(z_o^*)$. It follows that

$\lim\limits_{t \to \infty} \gamma_t(z) = \infty \implies \gamma_t(z) > \gamma_t(z_o^*)$ for almost all t,

and hence $Z \neq Z_t^*$ for all $t > t_Z'$, some t_Z'. On the other hand if $\lim\limits_{t \to \infty} \gamma_t(z)$ is defined, so is $\gamma(z)$, which is the limit concerned, and in this case, $Z \in P_1 - P_2$ implies that $\gamma(z) \geqslant \gamma(z_o^*)$ (by definition of Z_o^*). Therefore $d_Z = \gamma(z) - \gamma(z_o^*) \Rightarrow d_Z \geqslant o$; and if in fact $d_Z > o$,

$$\gamma_t(z) > \gamma(z_o^*) + \tfrac{1}{2} d_Z > \gamma_t(z_o^*)$$
$$\text{for all } t > t_Z'' \text{ some } t_Z''.$$

Let P_3 be the subset of Turing machines Z of $P_1 - P_2$ such that $\lim\limits_{t \to \infty} \gamma_t(z) = \infty$; and P_4 the subset such that

$\lim\limits_{t \to \infty} \gamma_t(z)$ exists and is $\neq \gamma(z_o^*)$. Then, for $t > \max.(t_{max}, \max\limits_{Z \in P_3}.t'_Z, \max\limits_{Z \in P_4}.t''_Z)$, Z_t^* and Z_o^* are members of $(P_1 - P_2) - (P_3 \cup P_4)$; moreover this set consists solely of Turing machines that calculate $[z_o]_n$ over S and are of optimum efficiency. Hence $Z_t^* = Z_o^*$ for all such t.

The theorem now follows if we write Z for Z_o, and take $\chi_n(z,t)$ as the function defining Z_t.[†]

SECTION 3. ON INEFFICIENCY

Our inquiry so far has been concerned with the question of efficiency. We now raise the reverse consideration, that of inefficiency. Rabin [24], and also Blum [5] have proved the existence of 'arbitrarily difficult' functions. Their notion of difficulty (which we discuss in chapter 3) is quite different from our concept of space-time measure, but one may ask whether

[†] χ_n's primitive recursiveness is left for the present to the intuition; we take up the question of assertions of this kind later (page 145).

a parallel result holds. The question in this form, "Are there functions of arbitrarily large space-time measure (so that there is an arbitrarily small upper bound on the efficiency of their algorithms)?", proves trivial, since we can specifically define functions with values so large that their space-time measures of necessity exceed that of any proposed upper bound.

THEOREM 15

There are functions with arbitrarily large space-time measures.

PROOF

Corresponding to any k, we can define a function f(x) such that if Z is any Turing machine which computes it, $\gamma(z) > k$. Let x_o be an argument-value for which $p(x_o) > o$; then f(x) may be taken as any function such that

$$f(x_o) = \left[b \neq o \rightarrow x_o + 1 + \left[\frac{k}{b\ p(x_o)}\right];\ T. \rightarrow 1 + \left[\frac{k}{a\ p(x_o)}\right] \right].^{\dagger}$$

Then, if Z computes this function, and $b \neq o$,

$p(x_o)\ b\ E(z,x_o) > k$, whereas if $a \neq o$,

$p(x_o)\ a\ M(z,x_o) > k$. Thus in all cases,

$p(x_o)\,\mu(z,x_o) > k$; the result follows.

† Using '$\left[\frac{x}{y}\right]$' for 'the largest integer $\leq \frac{x}{y}$'.

The question becomes a material one if we exclude functions of the type employed in the above theorem, for which a value of x is postulated such that f(x)-x is arbitrarily large. Let us instead consider functions where f(x)-x is bounded. It turns out then that the answer hinges upon whether the significant set considered is a finite or infinite one, as the following two theorems show.

NOTATION

$l(x_1,\ldots,x_n)$ is the length of the argument $(x_1,\ldots,x_n)$. In Davis type Turing machines $l(x_1,\ldots,x_n) = \sum_{i=1}^{n} x_i + 2n-1$.

THEOREM 16

If the significant set S involved is finite, and if h is any fixed number, there is an upper bound to the space-time measure of functions for which

$$f(x_1,\ldots,x_n) - l(x_1,\ldots,x_n) < h \quad \text{for all } (x_1,\ldots,x_n) \in S$$

(corresponding to the parameters a,b,ℓ,S employed).

PROOF

Let $f(x_1,\ldots,x_n)$ be any such function, and let Z be any Turing machine computing it according to a table of values method. We show that $Q(z)$, $M(z,x_1,\ldots,x_n)$, and $E(z,x_1,\ldots,x_n)$ are bounded. The number of quadruples which Z requires, is $< m(3m+2h)$ where $m = \max_{(x_1,\ldots,x_n) \in S} l(x_1,\ldots,x_n)$. This implies that

$$Q(z) < m(3m+2h) \times (2 \log_2 m(3m+2h) + 8) \quad ;$$

moreover $M(z,x_1,\ldots,x_n) < m + h$ for all $(x_1,\ldots,x_n) \in S$ and $E(z,x_1,\ldots,x_n) < 3m + 2h$ for all $(x_1,\ldots,x_n) \in S$.

Hence $$\gamma(z) = \sum_{(x_1,\ldots,x_n) \in S} p(x_1,\ldots,x_n) \Big\{ a\big(M(z,x_1,\ldots,x_n) + \ell\, Q(z)\big) + b\, E(z,x_1,\ldots,x_n) \Big\}$$

is bounded, and so therefore is $\gamma(z^*)$.

<u>THEOREM 17</u>

If the significant set S involved is infinite, corresponding to any k no matter how great, there exists a recursive function $f(x)$ with space-time measure $> k$, yet which is such that

$$f(x) = x \text{ or } x + 1 \quad \text{for all } x \in S .$$

<u>PROOF</u>

Define recursively $(f(x), S_x)$, where S_x is a set of Turing machine indices, as follows:

$$f(o) = o ,$$
$$S_o = \wedge .$$

To evaluate $f(y+1)$ if $p(y+1) \neq o$, consider the least number (say z_o) of

$$\{1, 2, \ldots\ldots, y+1\} - S_y$$

which is the Gödel number of a Turing machine and which is such that

$$\mu(z_o, y+1) \leqslant \frac{k}{p(y+1)}, \qquad \text{(A)}$$

and let $f(y+1) = \left[[z_o]_1 (y+1) = y+1 \longrightarrow y+2;\ T. \longrightarrow y+1 \right]$ (B)

and $S_{y+1} = S_y \cup \{z_o\}$;

if none of the numbers satisfy (A), or if $p(y+1) = o$, let $f(y+1) = y+1$, and $S_{y+1} = S_y$.

Now consider the inequality

$$\mu(z,x) \leqslant \frac{k}{p(x)}$$

with respect to any Turing machine Z computing $f(x)$. This can hold for at most 2z values of x for which $p(x) \neq o$, for otherwise a contradiction arises in that z occurs as z_o in (B) for the evaluation of some value of $f(x)$. Accordingly for at least <u>some</u> y, $p(y)\,\mu(z,y) > k$, and thus

$$\gamma(z) = \sum_{x \in S} p(x)\,\mu(z,x) > k .$$

As this applies to any Turing machine computing $f(x)$, $\gamma(z^*)$ is also $> k$.

We now turn to discuss, in the next two chapters, the individual spatial and timing measures in terms of which our space-time measure for algorithms was defined. These are the measures $M(z,x_1,\ldots,x_n)$, $Q(z)$ and $E(z,x_1,\ldots,x_n)$ over computations, and that over programs $Q(z)$ (considered from a different point of view); we also discuss a related measure over programs $W(z)$ (the number of instructions involved).

CHAPTER 2

MEASURES OF PROGRAMS

SECTION 1. ON PROGRAM LENGTH

As we wish to draw attention to the range of application of our concept of the spatial measure of a program, we make an exception in the first part of this chapter to our practice of formulating results in terms of Turing machines, and instead give them in an abstract form applicable to any computation system.

We first discuss the definition in question, and then go on to give associated results on translation between programs and on the problem of finding programs of minimum length.

DEFINITION

The set of programs of a computation system is a recursive set of words on a finite alphabet by which the algorithms of the system are represented. Each such word is called a program of the system.

DEFINITION

Clearly a system of Gödel numbering may be applied to any set of programs. We employ our convention of representing algorithms by upper case letters and their Gödel numbers by the corresponding lower case letters. Further $\left[z \right]_n$ denotes the n-ary function the algorithm Z evaluates.

DEFINITION

We take as the spatial measure Q(z) of a program Z, its length (i.e. the number of letters of which it is composed).

This concept may be interpreted for different computation systems as follows:

Applications of the Definition

(1) Turing Machines

Q(z) has already been defined for Turing machines.

(2) Recursive Expressions

One may represent any function expressed by minimalisation and compostion in terms of Davis' functions (2)-(6), p.41, on the alphabet:

$\{o, 1, \ldots 9, x, y, z, (,), ', M, S, U, +, \dot{-}, x\}$

if one denotes

(a) subscripted variables such as "x_{12}" as "x12";

(b) "U_i^n" as "Ui" (the value of n in U_i^n is obtainable by counting the number of arguments) ;

(c) minimalisation quantifiers such as "$\min_y$" as "My".

<u>EXAMPLE</u>. If f(x) is defined by $S(S(S(x))) \dot{-} S(S(x \dot{-} x))$, then the spatial measure of this particular expression or program for the function is 20, whereas the equivalent program S(x) has a measure 4.

(3) <u>Other Classes of Mathematical Expressions</u>

One may augment the alphabet in (2) by constants such as π and e, and additional function letters representing exponentiation, proper division, partial products and sums, etc..

(4) <u>Kleene Systems of Equations</u>

One can quite easily represent such a system on a single string, separating the individual equations by semicolons. Our definition of spatial measure may then be applied.

(5) <u>Computer Programs</u>

Computer programs whether in machine language, assembly language, or in a compiler language such as Fortran, can be represented without difficulty as in (4).

<u>EXAMPLE</u> The Fortran program which evaluates $z = \sqrt{x^2-y^2}$

viz: z - (x**2 - y** 2)** 1./2.

has length 20.

Q(z)'s interpretation in other computation systems may be made in ways similar to those given above.

<u>DEFINITION</u>

We refer to Davis' functions (2)-(6)[†], p.41[9], (which correspond to Kleene's "basis A") as <u>basis D</u>. <u>A recursive expression</u> is a w.f.f. constructed from basis D by composition and minimalisation.

Davis has produced an analysis whereby programs given in the form of recursive expressions may (by means of the programming techniques, examples and theorems quoted in the proof of his Theorem 1.2 (Chapter 3, p.42)) be translated into ones for Turing machines, whereas on the other hand (using defintions (1) to (26A) and theorem 1.4 (Chapter 4, p.62)), the functions represented by Turing machines may be defined by recursive expressions. Let Q_T,Q_R, respectively, denote the program length functions in the computation systems of Turing machines

[†] viz: $S(x) = x+1$; $U_i^n(x_1,\ldots,x_n) = x_i$, $1 \leqslant i \leqslant n$; $x+y$; $x \dot{-} y$; xy

and recursive expressions. Then, by using the results quoted, one can make the further observation:

THEOREM 18.

(a) There exists a constant k such that, if V is any given recursive expression, a Turing machine Z can be effectively found which computes the same function, and which is such that

$$Q_T(z) < k^{Q_R(v)}$$

(b) There exists a constant k' such that, if Z is any given Turing machine, a recursive expression V can be effectively found which computes the same function, and which is such that

$$Q_R(v) < k' \log z \, .$$

Generalising the problem, we consider translation between arbitrary computation systems.

THEOREM 19

For finite significant sets:

It is effectively possible, given any program of one computation system, to find a corresponding program for it in another.

PROOF

Let C_1 be the computation system in which the program is given and C_2 that in which it is required, and let the programs of C_2 be, in order of increasing length, $Z_1, Z_2, Z_3 \ldots$. Calculate the values of the function defined by C_1 over the significant set S. Then consider successively, for increasing n, up to n steps of the computations of $Z_1, Z_2, Z_3, \ldots, Z_n$ for each of the arguments in S, until an algorithm is obtained which calculates this function within the number of steps examined.

Before going on to consider the problem of finding programs of minimum length, we make some remarks about the philosophical implications this question holds.

INDUCTION

In science the classical task is to induce an underlying law from experimental evidence. What is sought is not an algorithm which merely fits the data - a simple list would suffice for this - but one which has predictive capacity. It is felt that in general the 'simplest' explanation (algorithm) is most likely to be the one associated with the law involved. The concept of simplicity is an intuitive one, not capable of exact definition. Simplicity and conciseness are however closely related and it is of heuristic value to use the latter as a measure of the former.

For example, assume we were investigating an unknown function $y = f(x_1,\ldots,x_n)$ (such as Kepler may have considered with regard to the variations in time of the apparent positions of the planets). It would certainly be of the greatest heuristic value for determining f, or for fitting a graph, to find the shortest program (in a computation system such as that of example (3)) which was consonant to the accuracy known with the readings for $x_1,\ldots,x_n,y$. Though exceptions can be constructed, our theorem 21 below provides that for a sufficiently large number of readings, such a program will in general be one which represents the underlying function concerned.

Conservation of Space

Shannon [29] has shown that in order to store an arbitrary set of numbers of maximum magnitudes $a_1,a_2,\ldots,a_n$,

$$\sum_{i=1}^{n} \log_2 a_i$$

bits of storage are necessary.

In particular cases, fitting the shortest program and storing this instead of the numbers themselves, allows us to conserve space, e.g. 32;64;128;256;9;512;1024 ,

which requires 24 letters, is more concisely stored as

2^x, x = 5 to 10; 9 ,

which requires only 11.

If the list of numbers is large, no matter how arbitrarily they are selected, it is to be expected that subgroups of the set will be connected by some such laws as that above. The <u>shortest</u> program evaluating the list will probably be much shorter than the list itself, and it is possible in this case to store the set in a space less than that of Shannon's measure.

We defined $\phi_n(z_o)$ as that Turing machine Z_o^* which calculates the same function as Z_o over S for which

$$\sum_{\text{all}(x_1,\ldots,x_n)} p(x_1,\ldots,x_n)\ \mu(z,x_1,\ldots,x_n)$$

is a minimum. If instead of the computational measure $\mu(z,x_1,\ldots,x_n)$, we employ that of programs, $Q(z)$, we obtain the following definition .

<u>DEFINITION</u>

For any given computation system and significant set S, $\phi_n^*(z_o)$ denotes that algorithm Z, of those which calculate $[z_o]_n$ over S, for which Q(z) is a minimum.[†]

[†] Or equivalently that for which $\sum_{(x_1,\ldots,x_n)} p(x_1,\ldots,x_n)Q(z)$ is a minimum.

Note that from the definition of significant set, $\phi_n^*(z_o)$ is only defined if $S \subset$ domain of Z_o (as was the case for $\phi_n(z_o)$).

<u>DEFINITION</u>

By a <u>complete</u> computation system C, we mean one which

(a) contains algorithms for all the recursive functions;

(b) has the property that corresponding to any given n-ary algorithm Z in C and any number k, it is effectively possible to select a (n-1)-ary algorithm in C which computes

$$[z]_n(k,x_1,\ldots,x_n) \quad .^{\dagger}$$

---

† A complete computation system and its associated space-time measure is a particular type of 'M-computer' as defined by Arbib [2] and Blum [5]. We have only postulated (in property (b) of the definition) a form of Kleene's iteration theorem, whereas M-computers require also that the recursion theorem be applicable under the new interpretation of Gödel numbers. However this requirement is in fact superfluous, as the proof used in Kleene, theorem 28, p.352 [17] (or Davis, theorem 7.4, p.176) remains valid under such a reinterpretation of Gödel numbers and shows in each case that the recursion theorem follows from the iteration theorem.

The first result we give follows from the 3rd equivalence theorem and the property specified in the definition above.

THEOREM 20

In any complete computation system, if the significant set considered is infinite, $\phi^*_n(z)$ is not potentially partially recursive, nor is any subfunction of it which is defined over a domain that includes the indices of the primitive recursive functions.

PROOF

Let the computation system be C. Consider the functions $f_1(r,x_1,\ldots,x_n) = [G_s(r)]_n(x_1,\ldots,x_n)$ and $f_2(x_1,\ldots,x_n) = [z_s]_n(x_1,\ldots,x_n)$ defined in theorem 5 for the computation system of Turing machines. By the first property of completeness, C must contain a n+1 -ary algorithm V_1 and a n-ary V_2 such that

$$[v_1]_{n+1}(r,x_1,\ldots,x_n) = f_1(r,x_1,\ldots,x_n)$$
$$\text{and } [v_2]_n(x_1,\ldots,x_n) = f_2(x_1,\ldots,x_n) .$$

(We do not assume in the proof that V_1 and V_2 are effectively determinable.) By the second property, there exists a recursive binary function t such that $t(v_1,r)$ is the Gödel number of an algorithm in C for which

$$[t(v_1,r)]_n(x_1,\dots,x_n) = [v_1]_{n+1}(r,x_1,\dots,x_n)$$
for all $r,x_1,\dots,x_n$.

If H is the finite set of algorithms which calculate $[v_2]_n$ over S, we obtain by theorem 5,

$\phi_n^*(t(v_1,r)) \in H \iff \neg \bigvee_y T(r,r,y)$, and the result follows.

However, in analogy with theorem 14, we have the following:

THEOREM 21

In any computation system, there exists a primitive recursive function χ_n^* such that

$$\chi_n^*(z,t) = \phi_n^*(z) \text{ for almost all } t.$$

PROOF

Let the algorithms of the computation system, considered in order of program length, be $Z_1,Z_2,Z_3,\dots$. and let the n-tuple arguments of the significant set S arranged in some sequence be $a_1,a_2,a_3,\dots$. Let

$$\chi_n^*(z,o) = 1 \text{ and } h_o = 1.$$

To evaluate $\chi_n^*(z,t+1)$ and h_{t+1}, examine up to t+1 steps of the computations of $Z_1,Z_2,\dots,Z_{t+1}$ for each argument

a_i, $i=1(1)\,h_{t+1}$. If none of these algorithms is thereby found to evaluate $[z]_n$ over the arguments considered, let

$$\chi_n^*(z,t+1) = \chi_n^*(z,t) \text{ and } h_{t+1} = h_t \;;$$

otherwise select ~~the index of~~ the algorithm with the shortest program as $\chi_n^*(z,t+1)$, and let $h_{t+1} = h_t+1$ if S is infinite, or if S is finite, let $h_{t+1} = \min.\ (h_t+1$, the number of elements in S).

Of the finite set of algorithms which have shorter programs than $\phi_n^*(z)$, all those that calculate values other than Z's over S or are only defined on a proper subset of S must eventually be permanently eliminated from consideration in the selection of $\chi_n^*(z,t)$. The result follows.

Changing our viewpoint, we now consider whether there are functions which require programs of arbitrarily great length.

THEOREM 22.

In any computation system and for any ~~infinite~~ significant set S and number k, there exists a recursive function which is such that the length of any program calculating it over S is greater than k.

PROOF

Note we assert only that the function is effectively calculable, not that the Turing machine computing it is effectively determinable.

Let $Z_1, \ldots, Z_t$ be those programs for singulary functions of length $\leq k$ which are defined for at least some member of S, and let $x_1, \ldots, x_t$ respectively be the least such members and $y_1, \ldots, y_t$ the corresponding function-values. If x_{i_j}, for $j = 1(1)u_i$, are the arguments in $\{x_1, \ldots, x_t\}$ equal to x_i, define $f(x)$ by

$$f(x_i) = \prod_{j=1}^{u_i} Pr(i)^{y_{i_j}}, \quad \text{for } i = 1(1)t;$$

and $f(x) = o$ otherwise. Since $f(x)$ is zero for all but a finite number of arguments, it is effectively calculable; but $f(x)$ is different from any of the functions calculated over S by Z_i for $i = 1(1)t$, and thus if Z is any program which calculates it over S, $Q(z) > k$.

SECTION 2. ON THE NUMBER OF INSTRUCTIONS A PROGRAM CONTAINS

Closely associated with the length of a program is the number of instructions it contains. Strictly speaking this is not a true spatial measure (unless the number of different instructions available is finite and all are of equal length). Lee /18/ has raised the general problem of 's-minimisation', finding equivalent programs with the least number of instructions. Wanabe /32/, Minsky /21/ and many others have attempted in particular to find the 'smallest' programs for the universal Turing machine (computing $\mathrm{Umin}_y T(z,x,y)$) The criterion of size used in the latter cases is a related measure due to Shannon /30/· the number of s-symbols the Turing machine contains × the number of q-symbols. We identify the number of instructions with the number of quadruples, and consider in this section the latter quantity, but all our results remain applicable if Shannon's measure is substituted instead.

It is natural to attempt to modify the proofs we have made in terms of program length so as to obtain corresponding results for number of instructions. The complication immediately arises, however, that there are an infinite set of Turing machines with any given number of quadruples. To meet this difficulty, we supply the following two theorems:

<u>NOTATION</u>

The number of quadruples a Turing machine Z contains is denoted by W(z).

THEOREM 23. FIRST REDUCTION THEOREM

There are a finite number of distinct Turing machines of any given number of quadruples.

PROOF

We begin by making a definition, which is of use both in the present and subsequent theorems.

DEFINITION OF $\psi(z)$

Let Z be any Turing machine with q-symbols, other than q_1, in ascending order:

$$q_{i_2}, q_{i_3}, \ldots\ldots, q_{i_v} ,$$

and s-symbols, other than s_o, s_1, in ascending order:

$$s_{j_2}, s_{j_3}, \ldots\ldots, s_{j_u} .$$

Now define

$$R' = R ,$$

$$L' = L ,$$

$$q'_{i_t} = q_t \quad \text{for } t = 2(1)v ,$$

$$s'_{j_t} = s_t \quad \text{for } t = 2(1)u ;$$

further, if any of q_1, s_o, s_1 are members of the alphabet of Z, define

$$s'_o = s_o ,$$

$$s'_1 = s_1 ,$$

$$q'_1 = q_1 ,$$

accordingly. Then, if Z consists of $\{a_i b_i c_i d_i\}$, where, for each i, a_i is a q-symbol, b_i is a s-symbol, and c_i is either R or L or a s-symbol, and d_i is a q-symbol, let $\psi(z)$ be the Turing machine $\{a_i' \; b_i' \; c_i' \; d_i'\}$.

It now follows immediately that for all Z

$$[\mathrm{gn}\ \psi(z)]_n = [z]_n ,$$

$$\text{and} \quad W(\mathrm{gn}\ \psi(z)) = W(z).$$

Thus for any t, the finite set of Turing machines which consist of t quadruples on the alphabet

$$R, L, q_1, q_2, \ldots\ldots, q_{2t},$$

$$s_o, s_1, \ldots\ldots, s_{2t},$$

contains all the distinct Turing machines for which $W(z) = t$.

We can in fact state the stronger result:

<u>THEOREM 24.</u> <u>SECOND REDUCTION THEOREM</u>

For any t, the recursive set of Turing machines Z such that

$$W(z) = t$$

$$\text{and } \psi(z) = Z$$

is finite, and contains all the distinct Turing machines with t quadruples.

<u>DEFINITION</u>

For any given significant set S, $\phi_n^{\dagger}(z_o)$ is that Turing machine Z of those which calculate $[z_o]_n$ over S, for which W(z) is a minimum.

The reduction theorems, and the 3rd equivalence theorem, enable us to prove the following three theorems in analogy with theorems 20, 21 and 22.

<u>THEOREM 25</u>

If the significant set considered is infinite, $\phi_n^{\dagger}(z)$ is not potentially partially recursive, nor is any subfunction of it which is defined over a domain that includes the indices of the primitive recursive functions.

<u>THEOREM 26</u>

For each n there exists a primitive recursive function $\chi_n^{\dagger}$ † such that

$$\chi_n^{\dagger}(z,t) = \phi_n^{\dagger}(z) \text{ for almost all } t.$$

<u>PROOF</u>

Let the set of Turing machines such that

---

† Corresponding to the significant set employed.

$$W(z) = t$$

and $\psi(z) = Z$,

defined in the second reduction theorem, be denoted by R_t. Then in place of the sequence $Z_1, Z_2, Z_3, \ldots$, defined in order of program length in theorem 21, consider instead the sequence $Z'_1, Z'_2, Z'_3, \ldots$ formed by writing all the members of R_1 in alphabetical order, then those of R_2, R_3 etc. The present theorem may then be proved in an analogous manner.

A similar adaptation of theorem 22, on arbitrarily long programs, yields:

THEOREM 27

For any ~~infinite~~ significant set S and any number k, there exists a recursive function such that if Z is any Turing machine that calculates it over S, then

$$W(z) > k .$$

SECTION 3. SECONDARY OPTIMISATION

In our proof of the optimisation theorem for finite functions we took advantage of the case where $\ell = o$, by defining Turing machines in terms of sets of actions for which the average value of $E(z,x_1,\ldots,x_n)$ was as small as possible, but the size of the Turing machine obtained exceptionally large. We consider now how one can in fact determine the shortest such Turing machine, or alternately, that with the least number of quadruples.

THEOREM 28

For any finite set of significant arguments S and any Turing machine Z_o defined over it, it is effectively possible to pick out from among those Turing machines Z which compute $[z_o]_n$ over S and for which

$$\gamma(z) = \sum_{(x_1,\ldots,x_n)} p(x_1,\ldots,x_n)\, E(z,x_1,\ldots,x_n)$$

is a minimum, a Turing machine for which Q(z) is least, or alternately, one for which W(z) is least.[†]

LEMMA 1

Given two instantaneous descriptions α_1, α_2, it is effectively possible to decide whether or not there exists a quadruple Q such that

$$\alpha_1 \to \alpha_2 \ (Q);$$

[†] For the purposes of the theorem, $\gamma(z)$ could be taken more generally as

$$\sum_{(x_1,\ldots,x_n)} p(x_1,\ldots,x_n) \left\{a\, M(z,x_1,\ldots,x_n) + b\, E(z,x_1,\ldots,x_n)\right\}$$

where $b \neq o$.

moreover if such a quadruple does exist, it is unique, and effectively determinable.

PROOF

Obvious.

LEMMA 2

Given a finite set of finite sequences of instantaneous descriptions

$$\alpha_{k_1}, \alpha_{k_2}, \ldots\ldots, \alpha_{k_{t_k}}$$

for $k = 1(1)m$, it is effectively possible to determine whether or not there exists some Turing machine Z such that each of the sequences is a computation of Z; moreover if such Turing machines do exist, it is effectively possible to find one. (We show how to find that one with the least length and least number of quadruples.)

PROOF

Using lemma 1, consider whether or not there exist quadruples Q_{k_i}, for $k = 1(1)m$, $i = 1(1)t_k-1$ such that

$\alpha_{k_i} \rightarrow \alpha_{k_{i+1}} \; (Q_{k_i})$ for each $k = 1(1)m$, and $i = 1(1)t_k-1$.

If this is not the case, no Turing machine of the kind sought does exist. If, on the other hand ,

(a) in each case such quadruples do exist,

(b) $\{Q_{k_i}\}$ represents a Turing machine (which implies that no two quadruples start with the same pair of symbols),

(c) $\alpha_{h_{t_h}}$ is final with respect to $\{Q_{k_i}\}$ for each h in $1(1)m$,

then $Z_1 = \{Q_{k_i}\}$ is the required Turing machine. Moreover if there exists any Turing machine Z_2, such that each of the sequences $\alpha_{k_1}, \ldots, \alpha_{k_{t_k}}$ is a computation of Z_2 for $k = 1(1)m$, then $Z_1 \subset Z_2$, since by lemma 1, each Q_{k_i} must be contained in Z_2; further in this case Z_1 is also a Turing machine for which each of the sequences is a computation, since conditions (a), (b), (c) clearly all then hold.

<u>PROOF OF THEOREM 28</u>

Let Z_o be any Turing machine and let p be $\min_{(x_1,\ldots,x_n)\in S} p(x_1,\ldots,x_n)$. Denote $\frac{\gamma(z_o)}{p}$ (which forms an upper bound for $E(z_o,x_1,\ldots,x_n)$ over S) by u, the number of significant arguments in S by d, and the maximum length of $\overline{(x_1,\ldots,x_n)}$ for $(x_1,\ldots,x_n) \in S$ by m. Then the size of the maximum instantaneous description involved in the d computations over S of the optimum Turing machine Z_o^* is $\leqslant m + u$, since the size of an instantaneous description can increase by at most one symbol at a time; the number of different q-symbols which could occur $\leq du$; and the number of different s-symbols $\leq d(2+u)$ (because only one new s-symbol can be introduced at each step).

Restricting ourselves for the moment to the letters

$$q_1, q_2, \ldots\ldots, q_{du} ,$$
$$s_o, s_1, \ldots\ldots, s_{d(2+u)} ,$$

consider the finite class Γ of all sets of d sequences of instantaneous descriptions of size $\leq m+u$ formed using letters given above, such that each sequence starts with a different member of $\left\{ q_1(\overline{x_1, \ldots, x_n}) \right\}$ for $(x_1, \ldots, x_n) \in S$, and is of length $\leq u$. Using lemma 2, we can determine whether or not any such set of sequences represents a set of computations for some Turing machine, and if it does, effectively obtain a Turing machine of this kind. On the other hand, any set H of d sequences of instantaneous descriptions defined on an alphabet

$$q_{i_2}, q_{i_3}, \ldots\ldots, q_{i_{du}} ,$$
$$s_{j_2}, s_{j_3}, \ldots\ldots, s_{j_{d(2+u)}} ,$$

plus possibly q_1, s_o, s_1, does or does not represent a set of proofs on some Turing machine Z, according to whether this is or is not the case for a set $H' \in \Gamma$, which is obtainable from H by leaving q_1, s_o, s_1 unaltered, and replacing

$$q_{i_t} \text{ by } q_t, \text{ for each } t \text{ in } [2, du],$$
$$\text{and } s_{i_t} \text{ by } s_t, \text{ for each } t \text{ in } [2, d(2+u)] .$$

Further if H' does represent a Turing machine (say) Z', it must follow that

$$[z']_n(x_1,\dots,x_n) = [z]_n(x_1,\dots,x_n) \text{ for all } (x_1,\dots,x_n) \in S$$

and that $\gamma(z') = \gamma(z)$.

Further $Z' = \psi(z)$,

and so $W(z') = W(z)$

and $Q(z') \leqslant Q(z)$.

Thus it is sufficient to examine each of the finite set of Turing machines obtained using lemma 2 from the sets of d sequences of instantaneous descriptions in Γ, to select from these all those Turing machines Z such that $[z]_n \underset{s}{=} [z_o]_n$, and of these Turing machines to choose those such that $\gamma(z)$ is a minimum; we may then in turn select from this last set an element for which $Q(z)$ or $W(z)$ is least. This will be the required Turing machine.

CHAPTER 3

MEASURES OF COMPUTATIONS

SECTION 1. BLUM'S DEFINITION

If $P(x_1,\ldots,x_n)$ is a partially defined predicate with domain D, let $\Omega P(x_1,\ldots,x_n)$ be a predicate defined by

$$\Omega P(x_1,\ldots,x_n) \Longleftrightarrow \begin{cases} P(x_1,\ldots,x_n), \text{ for all } (x_1,\ldots,x_n) \in D; \\ F., \text{ for all } (x_1,\ldots,x_n) \notin D. \end{cases}$$

We may then express Blum's definition of a measure of computation [5] (adapted to apply to Turing machines and generalised to n-ary arguments) as any function $\Phi(z,x_1,\ldots,x_n)$ such that

PROPERTY (1): $\Phi(z,x_1,\ldots,x_n)$ is partially recursive, and is defined if and only if $[z]_n(x_1,\ldots,x_n)$ is defined;

PROPERTY (2′): $\Omega\,(\Phi(z,x_1,\ldots,x_n) = k)$ is recursive.

Since

$$\Omega\,(\Phi(z,x_1,\ldots,x_n) = k) \text{ is recursive}$$
$$\leftrightarrow \Omega\,(\Phi(z,x_1,\ldots,x_n) < k) \text{ is recursive,}$$

we may place property ($2'$) by the right-hand side of the above bi-implication which we call <u>property (2)</u>.

Intuitively our space-time measure of computation $\mu(z,x_1,\ldots,x_n)$ is such a measure (and hence so are each of the specific measures $E(z,x_1,\ldots,x_n)$ and $M(z,x_1,\ldots,x_n) + \ell\ \Omega(z)$ (or simply $M(z,x_1,\ldots,x_n)$)). The first property holds because we can envisage a process for evaluating $\mu(z,x_1,\ldots,x_n)$ which attempts first to generate all the instantaneous descriptions involved, whereas to decide the predicate specified in the second, we need only carry out the computation $\min.(\frac{k}{b}, \frac{k}{a} V(z)\ Y(z)^{\frac{k}{a}})$ steps, where $Y(z)$ is the number of s-symbols in Z and $V(z)$ the number of q-symbols. These ideas are easily expressed in a formal proof:

<u>THEOREM 29</u>

$\mu(z,x_1,\ldots,x_n)$ is a measure of the type defined by Blum.

<u>PROOF</u> †

<u>Property (1)</u>

This follows, as may readily be verified, from the

---

†We employ here the following notation defined in Davis: $IC(x)$, $AL(x)$, $Pr(i)$, $\overset{t}{\underset{y=o}{\mathfrak{M}}} P(y,x_1,\ldots,x_n)$, $INIT_n(x_1,\ldots,x_n)$, $YIELD(x,y,z)$, $R(x,y)$, $\left[\frac{x}{y}\right]$, $x * y$, $T(z,x_1,\ldots,x_n,y)$, α, U, $\min_y$, $\mathcal{L}$, GL. See Appendix II.

following recursive expressions for $Q(z)$, $M(z,x_1,\ldots,x_n)$, $E(z,x_1,\ldots,x_n)$.

If the number of binary digits representing a symbol is denoted by BIN(x),

$$\mathrm{BIN}(x) = \left[\mathrm{IC}(x) \rightarrow \min_t \left(2^t > \left[\frac{x-5}{4}\right]\right); \quad \mathrm{AL}(x) \rightarrow \min_t \left(2^t > \left[\frac{x-7}{4}\right]\right.; \quad T. \rightarrow 1 \right],$$

and $Q(z)$ is thus $= \sum_{i=1}^{\mathcal{L}(z)} \left(4 + \sum_{j=1}^{4} \mathrm{BIN}(j \text{ GL } (i \text{ GL } z)) \right)$.

Let $t(x)$ be as defined on page 22.

If $\underset{y=1}{\overset{t}{\mathrm{MAX}}} \; f(y,x_1,\ldots,x_n)$ denotes

$$\min_h \left\{ \bigwedge_{y=1}^{t} (h \geqslant f(y,x_1,\ldots,x_n)) \wedge \bigvee_{y=1}^{t} (h = f(y,x_1,\ldots,x_n)) \right\},$$

and if $d(x)$ denotes the result of dropping left-hand and right-hand blanks from an instantaneous description x, with recursive expression

$$\underset{y=o}{\overset{x}{\mathfrak{m}}} \; \bigvee_{u=o}^{x} \; \bigvee_{v=o}^{x} \left(x = \prod_{i=1}^{u} \mathrm{Pr}(i)^7 * y * \prod_{i=1}^{v} \mathrm{Pr}(i)^7 \right),$$

then

$$M(z,x_1,\ldots,x_n) = L \min_w \left\{ T(z,x_1,\ldots,x_n,K(w)) \wedge \right.$$

$$\left. L(w) = \underset{i=1}{\overset{\mathcal{L}(K(w))}{\mathrm{MAX}}} \left(\mathcal{L}(d(i \text{ GL } K(w))) - \left[\mathrm{IC}(t(d(i \text{ GL } K(w)))) \rightarrow o; \; T. \rightarrow 1 \right] \right) \right\}.$$

Finally $E(z,x_1,\ldots,x_n) = \mathcal{L} \min_y T(z,x_1,\ldots,x_n,y)$.

Property (2)

NOTATION

If $P(x_1,\ldots,x_n)$ is a predicate, we denote its characteristic function by $\Delta P(x_1,\ldots,x_n)$.

Preliminary Recursive Expressions

(a) Let $RES(z,x_1,\ldots,x_n,t)$ be the Gödel number of the t^{th} instantaneous description of the computation of $[z]_n(x_1,\ldots,x_n)$ if this exists, and O otherwise. It may be defined recursively as follows:

$$RES(z,x_1,\ldots,x_n,o) = INIT_n(x_1,\ldots,x_n) \quad ,$$

$$RES(z,x_1,\ldots,x_n,t+1) =$$

$$\left[z*RES(z,x_1,\ldots,x_n,t)*z \bigvee_{y=1} YIELD(RES(z,x_1,\ldots,x_n,t),y,z) \rightarrow y;\ T. \rightarrow o \right].$$

(b) $\langle \frac{x}{y} \rangle$, the least integer $\geqslant \frac{x}{y}$, $= \left[R(x,y) \neq o \rightarrow \left[\frac{x}{y}\right] + 1;\ T. \rightarrow \left[\frac{x}{y}\right] \right]$.

(c) $MIN(x,y) = \left[x<y \rightarrow x;\ T. \rightarrow y \right]$.

(d) The number of s-symbols $Y(z)$ in Z

$$= \sum_{i=1}^{\mathcal{L}(z)} \sum_{j=1}^{4} a\ \Delta(AL(j\ GL\ (i\ GL\ z)))\ .$$

(e) The number of q-symbols $V(z)$ is recursively expressible in a similar manner.

Using the above, and our proof of property (1), we obtain the <u>partial</u> recursiveness of

$$\Omega(\ \mu(z,x_1,\dots,x_n) < k)$$

from the recursive expression of its characteristic function:

$$\Bigg[\Big\{\mathrm{RES}(z,x_1,\dots,x_n,\ \Big[\, b=o \rightarrow \langle\tfrac{k}{a}\rangle V(z)\, Y(z)^{\langle\frac{k}{a}\rangle};\ a=o \rightarrow \langle\tfrac{k}{b}\rangle;\ T. \rightarrow \mathrm{MIN}(\langle\tfrac{k}{b}\rangle, \langle\tfrac{k}{a}\rangle V(z)\ Y(z)^{\langle\frac{k}{a}\rangle})\Big]) = o\Big\} \longrightarrow \Delta\big(\mu(z,x_1,\dots,x_n) < k\big);\ T. \longrightarrow o\ \Bigg].$$

Further, as RES is primitive recursive and $\mu(z,x_1,\dots,x_n)$ is defined if there exists t such that $\mathrm{RES}(z,x_1,\dots,x_n,t) = o$, $\Omega(\ \mu(z,x_1,\dots,x_n) < k)$ is total, and hence <u>recursive</u>.

This last property prompts us to consider predicates, central to the concept of computational measure, of the form (or its negation)

$$\Omega\ (\ \Phi(z,x_1,\dots,x_n) < g(x_1,\dots,x_n))$$
$$\text{where } \Phi \text{ is M or E,}$$

and we refer to these as the <u>measuring predicates</u>. It is well known for instance that $\neg \bigvee_y T(x,x,y)$ is not computable, but on the other hand, if Φ denotes M or E,

$$\neg \bigvee_y \Big\{T(x,x,y)\ \wedge\ \Omega(\Phi(x,x) < k)\Big\}$$

is computable, since

$\neg \bigvee_{y} \left\{ T(x,x,y) \ \wedge \ \Omega(\Phi(x,x) < k) \right\}$ is true

$\Longleftrightarrow$ the measuring predicate $\neg \Omega(\Phi(x,x) < k)$ is true,

giving the result by property (2).

We shall see in fact that transformations such as that of $\neg \bigvee_{y} T(x,x,y)$ into

$$\neg \bigvee_{y} \left\{ T(x,x,y) \ \wedge \ \Omega(\Phi(x,x) < k) \right\}$$

alter the limitations of the predicates concerned from ones which bear upon their decidability, to ones applying to relationships between the spatial and timing measures $M(z,x_1,\ldots,x_n)$, $E(z,x_1,\ldots,x_n)$ and $Q(z)$, that such decisions entail. This is the subject matter of the next seven theorems.

<u>SECTION 2.</u> <u>THE MEASURING PREDICATES</u>

<u>SECTION 2.1</u> <u>MEASURES E AND M CONSIDERED IN RELATION TO Q</u>

We give here the first of the modifications we offer to Davis' model of Turing machines so as to produce a closer correspondence to actual computers with respect to efficiency criteria. A Turing machine will be allowed to signify whether a predicate $P(x_1,\dots,x_n)$ is true or not as soon as this is 'determined', without the requirement that it first erase all, or all but one, of the "1"s on the tape.

<u>DEFINITION</u>

We introduce the new special s-symbols <u>Y</u> and <u>N</u>, and permit Turing machines to have quadruples of the form

$$\left.\begin{array}{ll}\text{type (a)} : & q_i\ s_j\ Y\ q_1\ ; \\ \text{type (b)} : & q_i\ s_j\ N\ q_1\end{array}\right\}\ \text{for any } i,j\ ;$$
$$\text{type (c)} :\ q_1\ N\ N\ q_1\ .$$

<u>DEFINITION</u>

A Turing machine Z <u>R-calculates</u> a predicate $P(x_1,\dots,x_n)$ if it terminates its computations with respect to $(x_1,\dots,x_n)$ by writing a Y or N respectively, according to whether $P(x_1,\dots,x_n)$ is true or false.

R-calculation is our main concept. It also proves technically useful, however, to define an auxilary concept, R'-calculation.

<u>DEFINITION</u>

A Turing machine <u>R'-calculates</u> a predicate $P(x_1,\dots,x_n)$ if it terminates its computations with respect to $(x_1,\ldots,x_n)$ by writing a Y if $P(x_1,\ldots,x_n)$ is true, whereas if $P(x_1,\ldots,x_n)$ is false, it writes a N and then loops.

DEFINITION

A Turing machine Z is said to reach its terminal state when it either halts, or executes the last instruction before beginning a one instruction loop such as represented by type (c).

The equivalence (from the point of view of effectiveness) of these concepts is easily shown:

THEOREM 30.

$$\begin{aligned} &P(x_1,\ldots,x_n) \text{ is computable} \\ &\iff P(x_1,\ldots,x_n) \text{ is R-computable} \\ &\iff P(x_1,\ldots,x_n) \text{ is } R'\text{-computable.} \end{aligned}$$

PROOF

If Z computes $P(x_1,\ldots,x_n)$, using the method described in Davis' proof of lemma 1, p.26, we can construct a Turing machine Z' which also computes $P(x_1,\ldots,x_n)$, but at the same time maintains special markers at either end of the tape. It is then easy to devise a Turing machine Z'' which behaves as Z', but instead of halting when Z' does, searches the tape between the markers, determines whether the number of tallies contained is 0 or 1, and, accordingly, writes either a Y or a N.

A similar method enables the construction of a Turing machine Z''', which computes $P(x_1,\dots,x_n)$, by modifying a Turing machine Z that R-computes it.

Finally if Z R-computes $P(x_1,\dots,x_n)$, then $Z \cup \{q_1 NN q_1\}$ R'-computes it; whereas if Z R'-computes $P(x_1,\dots,x_n)$, it must contain a quadruple of type (c), and the predicate concerned is then R-computed by $Z - \{q_1 NN q_1\}$.

THEOREM 31

PART A. For any $k \geqslant 1$, and any finite set of numbers S, there exists a Turing machine Z_o which for all $x \in S$, R'-calculates the measuring predicate

$$\neg\,\Omega\bigl(E(x,x) \leqslant k + l(x)\bigr),$$

yet, for all such x, reaches its terminal state in

$$\leqslant k + l(x) \text{ steps.}$$

PART B. For any such Turing machine Z_o, $z_o \notin S$.

PROOF

PART A. We produce a Turing machine Z_o which in fact reaches its terminal state in $\leqslant 1 + l(x)$ steps by a straightforward table of values program, viz: if $S = \{x_i \mid i = 1(1)t\}$, and m is its largest element, let Z_o be

$q_i \; 1 \; L \; q_{i+1}$	for $i = 1(1)m+1$;
$q_{x_i+2} \, b \; \eta_i \; q_1$	for $i = 1(1)t$, where η_i in each case is Y or N according to whether or not $\neg\, \Omega\big(E(x,x) \leqslant k + l(x)\big)$ holds;
$q_1 \; N \; N \; q_1$	

Note that Z_o is effectively determinable since $\neg\, \Omega\big(E(x,x) \leqslant k + l(x)\big)$ is recursive.[†]

<u>PART B</u>. If Z_o R'-computes $\neg\, \Omega\big(E(x,x) \leqslant k + l(x)\big)$ for all $x \in S$, then for all such x

$$\neg\, \Omega\big(E(x,x) \leqslant k + l(x)\big) \Leftrightarrow \bigvee_y T(z_o,x,y) \ .$$

But if Z_o reaches its terminal state for all $x \in S$ in $\leqslant k + l(x)$ steps, then for all $x \in S$

[†] The objection might be raised that this method of determination requires all of Z_o's values in S to be calculated in advance; but the purpose of part A is only to show that a Turing machine of the required efficiency actually exists.

$$\bigvee_{y} T(z_o,x,y) \Longleftrightarrow \Omega\big(E(z_o,x) \leq k + l(x)\big) ,$$

and thus

$$\neg\,\Omega\big(E(x,x) \leq k + l(x)\big)$$
$$\Longleftrightarrow \Omega\big(E(z_o,x) \leq k + l(x)\big) \text{ for all } x \in S.$$

Hence $z_o \notin S$, for otherwise a contradiction arises if we substitute z_o for x.

In general Turing machines with large $Q(z)$ have large Gödel numbers, and vice versa; z and $Q(z)$ are in a sense both measures of the size of the program involved. This suggests that by redefining the Gödel numbering system so as to make the relationship between the two quantities a more direct one, we may be able to adapt theorem 31 to provide a limitation on $Q(z_o)$ instead of the Gödel number z_o; then since programs for R'-computation and their Q measures are simply related to corresponding programs for R-computation, we should be able to modify the result further to make it apply to R-computation.

DEFINITION

A R-Gödel number of a Turing machine with program

$$\bar{I}^1_1,\bar{I}^1_2,\bar{I}^1_3,\bar{I}^1_4;\ldots\ldots;\bar{I}^m_1,\bar{I}^m_2,\bar{I}^m_3,\bar{I}^m_4;$$

is the number whose decimal digits are obtained by replacing all commas by "2"s, semicolons by "3"s, "R"s by "4"s, "L"s by "5"s, "Yq_1"s in quadruples of type (a) by "6"s and "Nq_1"s in quadruples of type (b) by "7"s or "8"s according to whether or not a quadruple of type (c) occurs; the presence of a quadruple of type (c) (which has constant coefficients) is thereby indicated, and the quadruple itself may be deleted.

The representation of lists in this way, by means of concatenation and spacing digits, has been investigated by Quine [23] and Smullyan [33], and using the latter's results, Ritchie [27] has shown that the commonly used recursive functions and predicates such as U and T may be redefined so as to apply to Gödel numbers of the above kind. For the remaining theorems of the present subsection, we reinterpret, similarly, references to Gödel numbers in $E(z,x_1,\ldots,x_n)$, $M(z,x_1,\ldots,x_n)$, and $\Omega(z)$. It follows then that

$$\Omega(z) \;=\; \text{the number of decimal digits in the R-Gödel number of } Z.$$

<u>THEOREM 32</u>

<u>PART A</u>. For any $k \geqslant 1$ and any h, there exists a Turing machine Z_o which, for all $x \leqslant h$, R-calculates

$$\neg\, \Omega\big(E(x,x) \leqslant k + l(x)\big),$$

yet satisfies the condition

$$\Omega E(z_o,x) \leqslant k + l(x).$$

PART B. For any such Turing machine Z_o, $\Omega(z_o) \geqslant$ the number of decimal digits in h.

PROOF

PART A. Omit the quadruple q_1NNq_1 from the Turing machine described in theorem 31.

PART B. Let Z_o be any such Turing machine. Then for $x \leqslant h$, $Z_o \cup \{q_1NNq_1\}$ R'-computes $\neg\,\Omega\big(E(x,x) \leqslant k + l(x)\big)$ and further reaches its terminal state in $\leqslant k + l(x)$ steps. Thus by theorem 31, if z_1 is the R-Gödel number of $Z_o \cup \{ q_1NNq_1 \}$,

$$z_1 \not\in \{ 1,2,\ldots\ldots,h \};$$

$$\text{hence} \quad z_1 > h,$$

and as $\Omega(z_o) =$ the number of decimal digits in z_1, the result follows.

THEOREM 33

If Z_o R-computes

$$\neg\,\Omega\big(E(x,x) \leqslant k + l(x)\big),$$

$$\text{then} \quad E(z_o,x) > k + l(x) \text{ for some } x < 10^{\Omega(z_o)}$$

PROOF

If not, then by theorem 32,

$$\Omega(z_o) \geqslant \text{ the number of decimal digits in } 10^{\Omega(z_o)}$$
i.e. $\geqslant \Omega(z_o) + 1$

which is impossible.

THEOREM 34

PART A. For any k and any h, there exists a Turing machine Z_o which, for all $x \leqslant h$, R-calculates

$$\neg\, \Omega\big(M(x,x) \leqslant k + l(x)\big),$$

yet satisfies the condition

$$M(z_o,x) \leqslant k + l(x).$$

PART B. For any such Turing machine Z_o,

$$\Omega(z_o) \geqslant \text{ the number of decimal digits in } h.$$

THEOREM 35.

If Z_o R-computes

$$\neg\, \Omega\big(M(x,x) \leqslant k + l(x)\big),$$

then $M(z_o,x) > k + l(x)$ for some $x < 10^{\Omega(z_o)}$.

PROOFS

Similar to those for theorems 32 and 33, with minor modifications to obtain the additions to the range of k in theorem 34.

SECTION 2.2 MEASURES E AND M CONSIDERED INDIVIDUALLY

Theorems 36 and 38 which we give below provide a contrast to theorems 32 and 34 respectively.

THEOREM 36

For any $n,k > o$, there exists a primitive recursive function f such that $f(z)$ R-computes

$$\neg\,\Omega\left(E(z,x_1,\ldots,x_n) < k\right),$$

$$\text{yet} \quad E(f(z),x_1,\ldots,x_n) \leqslant k \quad \text{for all } (x_1,\ldots,x_n).^{\dagger}$$

PROOF

If $k = 1$ the predicate concerned has a constant truth-value, and the problem of constructing $f(z)$ is trivial. Consider $k > 1$.

tru Representing quadruples of the form

$$q_i\ s_j\ s_k\ q_u \quad \text{as} \quad (i,j,k,u),$$
$$q_i\ s_j\ R\ q_u \quad \text{as} \quad (i,j,R,u),$$
$$q_i\ s_j\ L\ q_u \quad \text{as} \quad (i,j,L,u),$$

let Z be any Turing machine

$$\left\{ (i_t,j_t,k_t,u_t) \;\middle|\; t = 1(1)m \right\}^{\dagger}.$$

---

† Actually we produce a primitive recursive function $f(n,k,z)$ which for any n,k is that required by the theorem. This comment applies also to the next two theorems.

† Thus for each t, k_t is either 'R' or 'L' or a coefficient.

If $\max_{t=1(1)m}.(i_t, u_t)^{\dagger}$ is w,

$$\left.\begin{aligned} \text{let}\quad i_t^r &= i_t + (r-1)w \\ u_t^r &= u_t + (r-1)w \end{aligned}\right\} \text{ for } t = 1(1)m.$$

Now let Z' be the Turing machine

$$\left\{(i_t^r, j_t, k_t, u_t^{r+1}) \;\middle|\; t = 1(1)m,\ r = 1(1)k-1\right\}$$

$$\cup \left\{ (i_t^k, j_t, Y, 1) \;\middle|\; t = 1(1)m \right\}$$

$\cup \{ (i_t^r, x, N, 1) \mid t = 1(1)m,\ r = 1(1)k$

all x such that $s_x \in$ alphabet of Z, or is the blank

or tally but $q_{i_t} s_x$ is not the first two symbols

of any quadruple of Z $\}$.

That the Turing machine given above does calculate $\neg\,\Omega\left(E(z, x_1, \ldots, x_n) < k\right)$, is shown as follows. Z and Z' each start in state q_1; thus if Z takes on successive states

$$i_{t_1}, i_{t_2}, i_{t_3}, \ldots .$$

and Z does not halt in $< k$ steps, Z' will take on successively the corresponding states

$$i_{t_1}^1 (= i_{t_1}),\ i_{t_2}^2, i_{t_3}^3, \ldots\ldots, i_{t_k}^k, Y \quad ;$$

---

†i.e. Davis' $\theta(Z)$.

whereas if Z does halt in $< k$, (say) in h steps, Z' takes on successively the states

$$i^{1}_{t_1}, i^{2}_{t_2}, \ldots\ldots\ldots, i^{h}_{t_h}, N \ .$$

It can be shown without difficulty that Z' is expressible as a primitive recursive function $f(z)$.

In order to prove a similar result for $M(z,x_1,\ldots,x_n)$ by devising a Turing machine Z' which decides

$$\neg\, \Omega \left(M(z,x_1,\ldots,x_n) < l(x_1,\ldots,x_n) + k\right), \qquad \text{(A)}$$ [†]

it is not sufficient to employ end-markers; $[z]_n(x_1,\ldots,x_n)$ may loop without its tape length exceeding the bound in question, in which case $M(z,x_1,\ldots,x_n)$ is not defined and (A) is true. To determine when this happens and yet prevent itself looping, Z' must count the number of instantaneous descriptions examined in its testing process. The problem is further complicated by the fact that as blanks are written at one end and non-blanks beyond the other, the set of squares considered

---

[†] We cannot, if course, use the same predicate as in theorem 36 (with M substituted for E), as $M(z,x_1,\ldots,x_n) \geqslant l(x_1,\ldots,x_n)$ for all $z,x_1,\ldots,x_n$.

in evaluating $M(z,x_1,\ldots,x_n)$ may be displaced in either direction. To construct such a Turing machine Z' within the limitation that $M(z',x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n) + k$ is thus exceptionally complex; it would be quite unwieldly to give an expression for Z' explicitly in terms of Z. Instead we develop a vocabulary which enables us to define, in terms of its actions, how such a Turing machine may be constructed. The rather 'high-powered' programming techniques involved are of interest in their own right, and should be of general utility in Turing machine design.

Macro-Statement Vocabulary

s-channel

If C is a 1-1 correspondence between the s-symbols of a Turing machine Z and a set of m-tuples, Z is said to have m s-channels with respect to C. If s-symbol i corresponds by C to $(x_1,\ldots,x_m)$, by the content of the t^{th} channel of this symbol we refer to x_t, for $t = 1(1)m$, and if i is written on the tape, we talk similarly of the contents of the t^{th} channel on that square. By the contents of the highest channel we mean x_1, and by that of the lowest, x_m. Such a correspondence C is called a s-symbol coding scheme.

q-channels are defined similarly, as are the contents of the highest and lowest q-channels and the concept of a q-symbol coding scheme.

<u>Conventions</u>

(a) We reserve the highest q-channel as a work channel, corresponding in function to that of ordinary q-symbols, whereas the other channels are used to store information.

(b) The highest s-channel is reserved for symbols which serve as positional markers on the tape.

(c) All the correspondences C between s-symbols and m-tuples used to define channels, have the following two closure properties:

(1) if $(x_1,\ldots,x_{t-1},x_t,x_{t+1},\ldots,x_m)$ is in the domain of C^{-1}, so is $(x_1,\ldots,x_{t-1},0,x_{t+1},\ldots,x_m)$ for each t in $1(1)m$;

(2) o (the blank s-symbol) corresponds by C to $(o,o,\ldots\ldots,o)$.

<u>Setting and Moving a Marker</u>

<u>Setting</u> a marker on a square means changing the symbol to one with a marker symbol in its first channel, but with the same contents in the other channels. <u>Moving</u> a marker is setting the first channel to zero without changing the other channels, and setting up the marker elsewhere.

m-tuple representation of tapes

If Z has m s-channels, we can represent its s-symbols by the corresponding m-tuples:

$$\begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_m \end{pmatrix} .$$

Then

(1) for any t in $[1,m]$, the tape at the t^{th} channel of one of Z's tapes

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ \cdot \\ \cdot \\ \cdot \\ x_t^1 \\ \cdot \\ \cdot \\ \cdot \\ x_m^1 \end{pmatrix} \begin{pmatrix} x_1^2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_t^2 \\ \cdot \\ \cdot \\ \cdot \\ x_m^2 \end{pmatrix} \cdot\ \cdot\ \cdot\ \cdot\ \cdot \begin{pmatrix} x_1^h \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_t^h \\ \cdot \\ \cdot \\ \cdot \\ x_m^h \end{pmatrix}$$

is the sequence of symbols $(x_t^1, x_t^2, \ldots, x_t^h)$.

(2a) moving the t^{th} channel r places to the left, means changing the contents of the t^{th} channel of each square, leaving those of the other channels unchanged to produce:

$$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_t^1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_t^2 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \cdots \begin{pmatrix} 0 \\ \vdots \\ 0 \\ x_t^r \\ 0 \\ \vdots \\ 0 \end{pmatrix} \begin{pmatrix} x_1^1 \\ \vdots \\ x_{t-1}^1 \\ x_t^{r+1} \\ x_{t+1}^1 \\ \vdots \\ x_m^1 \end{pmatrix} \begin{pmatrix} x_1^2 \\ \vdots \\ x_{t-1}^2 \\ x_t^{r+2} \\ x_{t+1}^2 \\ \vdots \\ x_m^2 \end{pmatrix} \cdots \begin{pmatrix} x_1^{h-r} \\ \vdots \\ x_{t-1}^{h-r} \\ x_t^h \\ x_{t+1}^{h-r} \\ \vdots \\ x_m^{h-r} \end{pmatrix} \begin{pmatrix} x_1^{h-r+1} \\ \vdots \\ x_{t-1}^{h-r+1} \\ 0 \\ x_{t+1}^{h-r+1} \\ \vdots \\ x_m^{h-r+1} \end{pmatrix} \begin{pmatrix} x_1^{h-r+2} \\ \vdots \\ x_{t-1}^{h-r+2} \\ 0 \\ x_{t+1}^{h-r+2} \\ \vdots \\ x_m^{h-r+2} \end{pmatrix} \cdots \begin{pmatrix} x_1^h \\ \vdots \\ x_{t-1}^h \\ 0 \\ x_{t+1}^h \\ \vdots \\ x_m^h \end{pmatrix}$$

(2b) Moving the tape to the <u>right</u> is defined similarly.

<u>Imitation.</u>

A Turing machine Z' is said to <u>imitate</u> another Turing machine Z at an instantaneous description using its t^{th} s-channel and r^{th} q-channel (with respect to a s-symbol and a q-symbol coding scheme), if the contents of its t^{th} s-channel is a s-symbol of Z, and that of its r^{th} q-channel a q-symbol of Z, and if, according to whether Z in such a state and scanning such a symbol, moves left, moves right, or writes a symbol i, before going into a state j, Z' respectively moves left, or right, or changes the t^{th} channel of the scanned square to i, before storing the value j in its r^{th} q-channel.

Z' may, in the operation defined above, simultaneously alter the contents of other s- or q-channels (usually, however, this only occurs with regard to the work or marker channels in the cases we consider).

We refer to this type of operation as an <u>imitation.</u>

<u>LEMMA TO THEOREM 38</u>

For any n,k, there exists a primitive recursive function f such that f(z) R-computes

$$\neg\,\Omega\left(M(z,x_1,\dots,x_n) \leqslant l(x_1,\dots,x_n) + k\right),$$

$$\text{yet} \quad M(f(z),x_1,\dots,x_n) \leqslant l(x_1,\dots,x_n) + k.$$

<u>PROOF</u>

Let Z be any Turing machine with (say) v q-symbols and u s-symbols. If Z's successive instantaneous descriptions, with respect to some argument, are α_1, α_2, α_3,......, consider an origin to be fixed on each of these in turn according to the following definition: the origin in α_1 is the leftmost square; the origin in α_{i+1} is the leftmost square if α_{i+1} 's marked tape is $\leqslant l(x_1,\dots,x_n) + k$ long, otherwise it is the same square as the origin in α_i.

Let Z' be a Turing machine with 2 q-channels and 3 s-channels which computes an argument $(x_1,\dots,x_n)$ in the following manner:

Z' initially sets up markers over the leftmost tally and on the k^{th} square past the rightmost tally. To avoid involving a tape of length $l(x_1,\dots,x_n) + 1$ in locating the right-hand end when $k = o$, it first changes the leftmost tally to a blank, restoring the tally afterwards while setting up the right-hand marker. It then begins to imitate, using its lowest s- and q-channels, the successive operations of Z that do not produce a tape of size $> l(x_1,\dots,x_n) + k$.

<u>Before</u> each imitation, Z' moves its two markers if necessary so as to maintain them over the positions in its 3rd s-channel tape which correspond to the origin and $l(x_1,\dots,x_n) + k-1$ in the corresponding computation by Z. Now Z can only increase its tape to a length $> l(x_1,\dots,x_n) + k$ by moving right from square $l(x_1,\dots,x_n) + k-1$, or moving left from the origin when the tape is of length $l(x_1,\dots,x_n) + k$ (i.e. when square no. $l(x_1,\dots,x_n) + k-1$ is a marked one). Clearly its markers enable Z' to distinguish when either condition occurs, in which case it writes a Y and halts.

Let $w = 2vu$.

<u>After</u> each imitation, Z' marks the scanned square with a special marker, then adds 1 to a number coded to a radix w which is stored in the squares of the 2nd s-channel with its low order end at the right-hand marker; this

number is moved correspondingly (one symbol at a time) each time the markers are moved. Z' finds the square scanned after its most recent imitation by virtue of the special marker, and thus is able to proceed with the next cycle.

If Z eventually halts without producing a tape $> l(x_1,\ldots,x_n) + k$, Z' writes a N and halts. If however Z neither exceeds this tape length nor halts, its operations must involve a loop, which moreover must be of length less than the number of possible combinations of Z's q-symbols and contents of squares $[\,o,\ l(x_1,\ldots,x_n) + k-1\,]$,

$$\text{i.e.} \leqslant \left(l(x_1,\ldots,x_n)+k\right) \vee u^{l(x_1,\ldots,x_n)+k} .$$

So if overflow occurs in Z''s 2nd channel (i.e. a carry from the highest order position at the origin), Z' writes a Y and terminates it computation, for in this case it will have imitated

$$w^{l(x_1,\ldots,x_n)+k} \text{ of } Z\text{'s operations,}$$

and $$w^{l(x_1,\ldots,x_n)+k} = (2vu)^{l(x_1,\ldots,x_n)+k}$$

$$> (l(x_1,\ldots,x_n) + k) \vee u^{l(x_1,\ldots,x_n)+k} .$$

Z' is thus the required Turing machine.

In the lemma above, the initial and final tape lengths need be no greater than $l(x_1,\ldots,x_n)$, whereas all the intermediate ones are bounded by $l(x_1,\ldots,x_n) + k$. In such circumstances it seems possible to strengthen the result by increasing the number of q-channels, so as to reduce the maximum tape length required. This proposition, in general form is treated in the following theorem.

THEOREM 37

For any n,k, there exists a primitive recursive function f such that if Z is any Turing machine with the property that, for all $(x_1,\ldots,x_n)$,

$$M(z,x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n) + k$$

and $[z]_n(x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n)$,

then $[f(z)]_n = [z]_n$,

and $M(f(z),x_1,\ldots,x_n) = l(x_1,\ldots,x_n)$.[†]

[†] Trakhtenbrot [35] refers to a machine such as Z for which $M(z,x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n) + k$ for all $(x_1,\ldots,x_n) \in S$ as one which "processes S with bounded extension (с ограниченным удлинением) ", and to one such as f(z) for which $M(z,x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n)$ for all $(x_1,\ldots,x_n) \in S$, as one which "processes S with bounded expansion (с ограниченным растяжением) ".

PROOF

Let Z be any Turing machine with the property specified above, and with (say) v q-symbols and u s-symbols. Consider an origin to be fixed on each of its successive instantaneous descriptions according to the definition employed in the lemma given above. Let Z' be a Turing machine with $2k+2$ q-channels and 2 s-channels which computes an argument $(x_1,\ldots,x_n)$ as follows: Z' initially sets up markers over the left-hand and right-hand tallies using the method described in the lemma. It then begins to imitate, using its lowest s- and q-channels, the successive operations of Z, maintaining its markers over the squares corresponding to Z 's origin and the square no. $l(x_1,\ldots,x_n)-1$. The markers enable Z' to determine if Z's tape ever exceeds $l(x_1,\ldots,x_n)$, since this entails Z moving out of $[o,l(x_1,\ldots,x_n)-1]$. By the hypothesis, Z cannot, before returning to $[o,l(x_1,\ldots,x_n)-1]$ move out of $[-k,-1]$ or $[l(x_1,\ldots,x_n),l(x_1,\ldots,x_n)+k-1]$ as the case may be. It must thus return or else halt in $\leq kvu^k$ steps, and which of these courses it follows, its resultant state, and the contents of squares $[-k,-1]$ or $[l(x_1,\ldots,x_n),\ l(x_1,\ldots,x_n)+k-1]$, are a function of its original state on entering these squares and their contents at the time. As there are a finite number of such combinations, it is possible to program Z' to store in a single step Z 's resultant state in its lowest q-channel, and Z 's new contents for $[-k,-1]$ or $[1(x_1,\ldots,x_n), 1(x_1,\ldots,x_n) + k-1]$ in q-channels 2 to $k + 1$ or $k + 2$ to $2k + 1$ respectively. After Z' has simulated the last of Z 's operations, the number of tallies in its 2nd s-channel, plus the number stored in q-channels 2 to $2k + 1$, are equal to $[z]_n(x_1,\ldots,x_n)$. Z' is thus able to adjust the number of tallies on the tape to this quantity.

It can clearly be shown that Z' is a primitive recursive function of n,k,z ; the theorem follows.

Using this result and the lemma preceding it, we obtain immediately:

<u>THEOREM 38</u>

For any n,k, there exists a primitive recursive function f such that $f(z)$ R-computes

$$\neg\, \Omega\left(M(z,x_1,\ldots,x_n) \leqslant 1(x_1,\ldots,x_n) + k\right),$$
$$\text{yet}\quad M(f(z),x_1,\ldots,x_n) = 1(x_1,\ldots,x_n) .$$

<u>NOTE</u> Theorems 36 and 38 remain valid if the negation signs on the predicates, which $f(z)$ is required to R-compute in each case, are dropped. (The present forms emphasise the contrast with theorems 32 and 34.)

SECTION 3. RELATED WORK

Myhill [22] considers the class of predicates and functions which can be computed under the restriction

$$M(z,x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n),$$

and in particular proves that Smullyan's rudimentary operations [33] are of this category.

A restriction of the form

$$E(z,x_1,\ldots,x_n) \leqslant l(x_1,\ldots,x_n) \log l(x_1,\ldots,x_n)$$

is examined by Trakhtenbrot [35]. He shows that $T(z,x_1,\ldots,x_n,y)$ is such a function.

Ritchie [27] describes a hierarchy of classes of elementary functions $F,F_1,F_2,\ldots$, such that F is the class of functions computable on a finite automaton, whereas the functions of F_{i+1} are those computable by some Turing machine Z, such that $M(z,x_1,\ldots,x_n) \leqslant g(x_1,\ldots,x_n)$ for some function of $g \in F_i$.

Stearns, Hartmanis and Lewis [34] also examine a hierarchy based on memory requirement, on a variety of dual tape Turing machines.

Cleave [6] investigates, on a Shepherdson and Sturgis type "J-limited machine", a related stratification of primitive recursive functions based on a timing criterion, the number of jump instructions executed.

Blum [5] and Rabin [24] produce functions with various categories of lower bound on the least value that the measure of computation of any machine computing them may have, if the arguments considered are sufficiently large.

Arbib and Blum [3] define ways of comparing machines according to the timing measures of their corresponding programs.

Less directly related to the present work is the field of real-time computation on multitape Turing machines. Yamada [39] examines the question of real-time counting, in which a Turing machine detects whenever the total number of its input tallies adds up to the next highest value of a monotonically increasing function; Hartmanis and Stearns [12], and Hennie [13],[14], classify sequences according to the time functions delimiting the rate at which they can be generated. Ruby and Fischer [28] employ such time functions to extend the concept of real-time countability by defining a hierarchy of monotonically increasing functions classified according to their characteristic sequences. These are binary sequences in which the n^{th} digit is 1 if and only if n is a member of the corresponding function's range. Finally, Rabin [25] considers real-time definability, and shows that not all sets definable on a 2-tape Turing machine can be defined on a 1-tape Turing machine.

We wish to discuss three points in connection with the above works.

Firstly there is an error in the paper by Arbib and Blum (to which we refer the reader for the relevant definitions). This is in theorem 2(1) which states:

$$M \underset{s}{\equiv} N \text{ if and only if } M \underset{s}{\geqq} N \text{ and } N \underset{s}{\geqq} M.$$

While the condition is necessary, it is not in all cases sufficient. Consider a class of Turing machines as defined by the authors, except that they require twice the amount of time (number of steps) per instruction executed. In other words, if T is one of Arbib and Blum's Turing machines, and T' one of ours consisting of exactly the same quintuples as T and with the same encoding and decoding scheme, then for all i,x,

$${}^{T'}\Phi(i,x) = 2\ {}^{T}\Phi(i,x).$$

Taking $S = \{e(x,y)\}$, it is clear that $T \underset{s}{\equiv} T'$ according to their definition 2 but, on the other hand, it is not the case that $T \underset{s}{\geqq} T'$.

Secondly, we prove a result arising out of the work of Ritchie and Cleave concerning the function $h(i,x)$, which may be recursively defined by :

$$h(o,x) = x, \quad h(i+1,x) = 2^{h(i,x)}.$$

This function is of particular interest. Bereczki [4] used it to show that Kalmar's elementary functions (K) do not include all the primitive recursive ones. It thus follows that $h(i,x)$ is not a member of Ritchie's hierarchies, since $\bigcup_{i=o}^{\infty} F_i = K.$†

On the other hand, $h(i,x)$ is primitive recursive, and thus must appear in the Cleave hierarchies. This leads us to inquire as to its exact place there. The following theorem is easily proved.

<u>THEOREM 39</u>

The lowest Cleave hierarchy $h(i,x)$ belongs to, is $E_{\omega+1}$.

---

† In fact a simple proof that $h(i,x) \notin K$, follows directly from Ritchie's work. Ritchie has shown that $h(i+1,X) \notin F_i$ for each i. If $h(i,x)$ <u>were</u> a member of F_t some t, it would follow, by the closure of the classes F_i under explicit transformation, that $h(t+1,x) \in F_t$.

PROOF

Since $E_\omega = K$, we are assured that $h(i,x) \notin E_t$ for $t \leqslant \omega$.

Note first that $2^x \in E_1$ as $2^x = [P_1, x+1]$ where P_1 is

$$\begin{array}{r|l} & R_1 = 1 \\ & J_1(1,1) \\ 1 & R_1 = 2R_1 \\ & J_1(1,1) \end{array}$$

Hence $h(i,x)$ belongs to $E_{\omega+1}$, since $h(i,x) = [P_2, i]$ where the single function P_2 employs is a member of E_1: P_2 is

$$\begin{array}{r|l} 1 & R_2 = 2^{R_2} \\ & J_2(1,1) \end{array}$$

The answer appears in register 2.

The third point we discuss concerns Rabin's definition of a measure on proofs, viz: any recursive function $m(L,y)$, where y is the Gödel number of a proof in a logic L, such that, for any k and L, the number of proofs for which $m(L,y) \leqslant k$, is finite and primitive recursive. The most natural interpretation of $m(L,y)$, as Rabin points out, is the length of the proofs concerned, and this brings out an interesting difference between Post systems and Turing systems , whose deductive equivalence is otherwise well known.[†]

NOTATION

(1) If $R(y,x_1,\ldots,x_n)$ is a predicate, by $N_y R(y,x_1,\ldots,x_n)$ we denote the number of different values for y such that $R(y,x_1,\ldots,x_n)$ is true.

(2) By $M_o(L,y)$, we denote the length of a proof with Gödel number y in the logic L.

THEOREM 40

There exists a Turing system L_1 with a single axiom, and with a rule of inference $R(y,x)$ such that $N_y R(y,x) \leq 2$ all x, for which $N_y(M_o(L_1,y) \leq k)$ is not recursive.

PROOF

Let L_1 be a Turing system defined on the alphabet $\{1,H\}$ with the single axiom $\bar{1}\ H$[‡] and rule of inference:

[†] Each consists of a finite set of axioms and rules of inference, but whereas Post systems (as defined in Rabin [24] and Arbib [1]) have productions for rules of inference, Turing systems (defined in Arbib [1] and Davis (who calls them "Logics")) employ recursive predicates on the Gödel numbers of their formulae.

[‡] $\bar{x}$ denotes $\underbrace{111\ldots\ldots1}_{x+1}$ (as in Davis).

$$\bar{u}\,H \longrightarrow Q,$$

if Q is $(\overline{u+1})H$ and, in the case that $T(u,u,v)$, if Q is $\bar{v}$. Clearly $\alpha \longrightarrow \beta$ is definable by a recursive predicate $R(gn(\alpha), gn(\beta))$, for which $N_y R(x,y) \leq 2$. We then have

$$N_y(M_o(L_1,y) \leq k) = k + \sum_{u=1}^{k-1} N_v T(u,u,v).$$

If the left-hand side were recursive, so would be

$$h(k) = \sum_{u=1}^{k-1} N_v T(u,u,v).$$

But this is impossible as

$$N_v T(k,k,v) = h(k+1) - h(k),$$

yet $N_v T(k,k,v)$ is not recursive since

$$\bigvee_v T(u,u,v) \iff N_v T(u,u,v) \neq o.$$

<u>COROLLARY 1</u>

Length of proof, while a valid measure of proof for Post systems, is not so for Turing systems.

<u>COROLLARY 2</u>

There are Turing systems which generate their theorems in a different order from any equivalent Post system.

CHAPTER 4

EXTENSIONS OF TURING MACHINES

Turing machines, considered from the point of view of efficiency, are not a very authentic model of actual computers. In investigating real-time computation, Yamada [39], and subsequent authors, consider an extension of the basic machine by allowing for more than one tape. Stearns, Hartmanis and Lewis [34], and Hennie [13],[14], study variations of the way the input may be received in Turing machines, and of its memory structure, in order to simulate off-line, on-line and push-down computers.

No means is provided in the classical formulation of Turing machines for reflecting the savings in time and space actual computers enjoy through special circuitry to compute standard functions, such as addition, subtraction, multiplication, division, exponentiation, integration etc., or even of representing negative or non-integer numbers. We consider here how the classical definition may be modified to serve such ends. This leads to a device sophisticated enough for the investigation of actual computer languages such as Fortran.

SECTION 1. REPRESENTATION OF BUILT-IN FUNCTIONS

The method of representing the effect of built-in functions which immediately suggests itself, is by means of the concept of relative recursiveness. Davis defines computation relative to a set A, by considering Turing machines in which quadruples of the form $q_i s_j q_t q_u$ are allowed under the interpretation that if $Pq_i s_j Q$ is an instantaneous description,

$$\begin{aligned} Pq_i s_j Q &\xrightarrow[A]{} Pq_t s_j Q, \text{ if the number of tallies in } Pq_i s_j Q \text{ is a member of } A; \\ &\xrightarrow[A]{} Pq_u s_j Q \text{ otherwise.} \end{aligned}$$

This definition of relative computation is unsatisfactory for our purposes because (a) it does not define relative computability directly with regard to functions. The device of employing the associated set instead, means that a Turing machine can only make use of the value of (say) $F(x_1,\ldots,x_n)$ in its calculations, by successively generating $\prod_{i=1}^{n} Pr(i)^{x_i} Pr(n+1)^t$ for $t = 0,1,2,\ldots$, and checking in turn whether the number is a member; (b) it defines relative computability with respect to only a single entity, and there is no simple extension of the concept. However it is possible to obtain a more satisfactory formulation in these respects, by re-interpreting Davis' type 4 quadruples $q_i s_j q_t q_u$ in the manner described below.

DEFINITION

If $\underline{A}$ is a finite ordered set of functions $\left\{ f_1^{(n_1)}, f_2^{(n_2)}, \ldots, f_k^{(n_k)} \right\}$ we write $\alpha \xrightarrow[\underline{A}]{} \beta\,(Z)$ if and only if (a) α is of the form

$$P q_i s_j \overline{(x_1, x_2, \ldots, x_{n_t})} Q,$$

where Q is empty or else has a leftmost symbol other than "1";and (b) $q_i s_j q_t q_u \in Z$ for some u, and some t such that $f_t^{(n_t)} \in \underline{A}$; and (c) β is of the form

$$P q_u s_j \overline{f_t^{(n_t)}} (x_1, \ldots, x_{n_t}) Q \;.$$

'A-computation', '$\mathrm{RES}_Z^A(\alpha)$', 'A-partial computability' and 'A-computability' are similarly reinterpreted in the notation '$\underline{A}$-computation', '$\mathrm{RES}_Z^{\underline{A}}(\alpha)$','$\underline{A}$-partial computability' and '$\underline{A}$-computability' respectively.

We can now give the following definitions:

DEFINITION OF RELATIVE SPACE-TIME MEASURE

If Z computes $f(x_1, \ldots, x_n)$ relative to a set of functions $\underline{A}$, the space-time measure relative to $\underline{A}$ of a computation by Z is the quantity $a\big(M(z, x_1, \ldots, x_n) + \ell\, Q(z)\big) + b\, E(z, x_1, \ldots, x_n)$ as employed before, applied to the more general type of computation defined above.

Each use of a built-in function then requires one time unit, but no extra work space.

The space-time measure of a Turing machine relative to a set of functions is defined similarly.

Church's thesis [7] asserts the identification of effective computability (which in our case is defined in terms of Turing machines) and recursiveness. Davis has carried out this identification between relative computability (in his restricted sense) and recursiveness relative to a single quantity (definition 1.1). On the other hand, his definition of relative partial recursiveness with respect to a set of functions, which he gives in terms of completely computable functionals, is a highly artificial one. It serves the purpose, no doubt, of introducing the concept which Kleene defines in terms of sets of equations [17], but there is no satisfactory interpretation of the idea at the level of Turing machines, or in Davis' definition of partial recursiveness. Having extended his concept of relative computability to apply to sets of functions, it seems natural to now extend his concept of recursiveness relative to a single quantity according to similar criteria, and to prove the equivalence of the two extended concepts.

EXTENDED CONCEPT OF RELATIVE RECURSIVENESS

If A is an ordered finite set of functions, a function is A-partial recursive if it can be obtained by a finite number of applications of composition and minimalisation, beginning with the functions of the following list:

(1) the set of functions of $\underline{A}$;

(2) $S(x) = x + 1$;

(3) $U_i^n(x_1,\ldots,x_n) = x_i \quad 1 \leqslant i \leqslant n$;

(4) $x + y$;

(5) $x \dot{-} y$;

(6) xy .

$\underline{A}$-recursiveness is defined correspondingly. We refer to the above functions as basis $\underline{A}$. (We previously defined functions (2) - (6) as basis D.)

THEOREM 41

If $\underline{A}$ consists of a single function (say) $f(x_1,\ldots,x_n)$, and A^* is the associated set of $f(x_1,\ldots,x_n)$, then for any function g

g is A^*-recursive (in Davis' sense)

$\Longleftrightarrow$ g is $\underline{A}$-recursive (in our sense).

PROOF

NECESSITY. If g is A^*-recursive, it is definable, using composition and minimalisation, in terms of functions (2) - (6) and the characteristic function of A^*, $C_{A^*}(x)$.

But $C_{A^*}(x) = \alpha\alpha|f(1GLx, 2GLx, \ldots, nGLx) - (n+1)GLx|$,

and thus $C_{A^*}(x)$ is definable by composition in terms of functions (2) - (6) and $f(x_1, \ldots, x_n)$.

<u>SUFFICIENCY</u>. If g is <u>A</u>-recursive, it is definable in terms of functions (2) - (6) and $f(x_1, \ldots, x_n)$.

But $f(x_1, \ldots, x_n) = \min_y \left(C_{A^*}\left(\prod_{i=1}^{n} Pr(i)^{x_1} \cdot Pr(n+1)^y \right) = o \right)$,

from which the result follows.

<u>THEOREM 42</u>

$g(x_1, \ldots, x_n)$ is <u>A</u>-partially recursive

$\longleftrightarrow$ $g(x_1, \ldots, x_n)$ is <u>A</u>-partially computable .

<u>PROOF</u>

Let <u>A</u> consist of the functions $f_1^{(n_1)}, f_2^{(n_2)}, \ldots, f_k^{(n_k)}$

<u>Necessity</u>. If $f_t^{(n_t)} \in$ <u>A</u>, then $f_t^{(n_t)}$ is <u>A</u>-computable, since in fact the Turing machine

$q_1 1 L q_1$

$q_1 b q_t q_2$

$q_2 b R q_2$

$q_3 1 b q_3$

<u>A</u>-computes it. Thus the functions in basis <u>A</u> are <u>A</u>-computable. It is sufficient then to show that this property of <u>A</u>-computability is closed with respect to

composition and minimalisation. The proof Davis gives with respect to A-computability (lemma 1-4 and theorems 2.1 and 2.4) may be applied with minor modifications.

<u>Sufficiency</u>. Davis' proof that A-partial computability implies A-partial recursiveness hinges on his demonstration that the predicate $T^A(z,x_1,\ldots,x_n,y)$ is A-recursive. Davis does this by means of a series of definitions ((1) - (26A)) in which each new concept is shown to be recursively or A-recursively expressible in terms of its predecessors. In interpreting A-computability, A-recursiveness and the predicate $T^A(z,x_1,\ldots,x_n,y)$ in our sense, we need only modify definition (25A) for his proof to remain valid. By $H_{\underline{A}}(x,y,z)$ we now understand that x and y, in that order, are Gödel numbers of successive instantaneous descriptions of a Turing machine with Gödel number z relative to a set of functions $\underline{A}$. This proposition may be recursively expressed as follows:

$$
\begin{aligned}
&ID(x) \wedge ID(y) \wedge TM(z) \wedge \\
&\bigvee_{p=0}^{x} \bigvee_{q=0}^{x} \bigvee_{r=0}^{x} \bigvee_{s=0}^{x} \bigvee_{h=0}^{x} \bigvee_{u=0}^{z} (x = p*2^r*2^s*q \wedge IC(r) \wedge AL(s) \wedge IC(u) \wedge 1GLh \neq 11 \wedge \\
&\Big(\Big\{TERM(2^r3^s5^17^u,z) \wedge \bigvee_{x_1=0}^{x} \bigvee_{x_2=0}^{x} \cdots \bigvee_{x_{n_1}=0}^{x} (\quad q = MR(x_1)*2^7*MR(x_2)*2^7*\ldots*MR(x_{n_1})*h \\
&\qquad \wedge\ y = p*2^u*2^s*MR(f_1(x_1,\ldots,x_{n_1}))*h\ )\Big\} \\
&\vee \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\
&\vee \Big\{TERM(2^r3^s5^k7^u,z) \wedge \bigvee_{x_1=0}^{x} \bigvee_{x_2=0}^{x} \cdots \bigvee_{x_{n_k}=0}^{x} (\quad q = MR(x_1)*2^7*MR(x_2)*2^7*\ldots*MR(x_{n_k})*h \\
&\qquad \wedge\ y = p*2^u*2^s*MR(f_k(x_1,\ldots,x_{n_k}))*h\ )\Big\} \Big)).
\end{aligned}
$$

$H_{\underline{A}}(x,y,z)$ is clearly $\underline{A}$-recursive; the result follows.

COROLLARY

From the previous two theorems it follows that:

If $\underline{A}$ consists of a single function f and A^* is the associated set of f, then for any function g

g is $\underline{A}$-computable $\longleftrightarrow$ g is A^*-computable.

THEOREM 43

A function is partially recursive in a finite set of functions according to our definition, if and only if it is so according to Davis' (definition 5.1, p.171).

PROOF

Let $\underline{A}$ consist of the functions $f_1^{(n_1)}, f_2^{(n_2)}, \dots, f_k^{(n_k)}$. By Davis' definition we are required to prove that, if $g^{(n)}$ is any function,

$g^{(n)}$ is $\underline{A}$-partially recursive

$\longleftrightarrow$ there exists a completely computable functional F such that

$$g^{(n)}(x_1,\dots,x_n) = F\left(f_1^{(n_1)}, f_2^{(n_2)}, \dots, f_k^{(n_k)}, x_1, \dots, x_n\right).$$

Necessity. If g is expressible in terms of basis $\underline{A}$, then considering its recursive expression as defining a functional F, we obtain at once that F has both properties of compactness (definition 2.2, p.164, Davis), and that

$$\theta(r_1,\ldots,r_k,x_1,\ldots,x_n) = F(r_1^{[n_1]},\ldots,r_k^{[n_k]},x_1,\ldots,x_n)$$

is partially recursive; this gives the complete computability of F (by theorem 2.2, p.165, Davis).

<u>Sufficiency</u>. Assume

$g(x_1,\ldots,x_n) = F\left(f_1^{(n_1)},\ldots,f_k^{(n_k)},x_1,\ldots,x_n\right)$. We show that g is expressible in terms of basis <u>A</u>. Davis' theorem 5.2, p.171, (Kleene's extended normal form), asserts that

$$g^{(n)}(x_1,\ldots,x_n) = F\left(f_1^{(n_1)},\ldots,f_k^{(n_k)},x_1,\ldots,x_n\right)$$

$\longrightarrow$ there exists a number e such that

$$g^{(n)}(x_1,\ldots,x_n) = U\,\min_y T_n^{n_1,\ldots,n_k}\left(e,\langle f_1^{(n_1)}\,|\,y\rangle,\ldots,\langle f_k^{(n_k)}\,|\,y\rangle,x_1,\ldots x_n,y\right).$$

The result follows since $T_n^{n_1,\ldots,n_k}\left(e,t_1,\ldots,t_k,x_1,\ldots,x_n,y\right)$ is primitive recursive and, if

$$\lambda(t,x) = \alpha(t \dot{-} x)x\,,^{\dagger}$$

a recursive expression for $\langle f_t^{(n_t)}\,|\,y\rangle$ is provided by

$$\prod_{x_1=0}^{y}\prod_{x_2=0}^{y}\cdots\prod_{x_{n_t}=0}^{y}\Pr\left(\Pr(1)^{x_1+1}\Pr(2)^{x_2+1}\ldots\Pr(n_t)^{x_{n_t}+1}\right)^{\lambda(y,f_t^{(n_t)}(x_1,\ldots,x_{n_t}))}.$$

† Thus $\lambda(t,x) = \begin{cases} x & \text{if } t \leqslant x\,, \\ 0 & \text{otherwise.}\end{cases}$

COROLLARY

A function is partially computable in a finite set of functions according to our definition if and only if it is according to Davis' (definition 5.1, p.171; Davis uses the adjective 'partially computable' in this sense, as a synonym for 'partially recursive').

SECTION 2. REPRESENTATION OF NEGATIVE AND NON-INTEGER NUMBERS

Turing [36] originally introduced Turing machines so as to define 'computable number'. However the theory of computability subsequently developed has been restricted to functions of non-negative integers, as has the parallel study of recursiveness.

In order to extend these concepts, and consequently the range of application of the results of the thesis to encompass such basic operations as subtraction, or division to a bounded number of steps (which all computers can perform), we suggest a further modification to the definition of Turing machines. Let special s-symbols be set aside to represent the decimal point and minus sign. Arguments allowed may then be any n-tuples of numbers, positive or negative, coded to a bounded number of places in a radix > 1.[†]

This provides a wider class of 'computable' functions. Recursiveness may be similarly redefined by allowing such numbers in Davis' basis D; the extended concepts thus obtained remain equivalent.

Such Turing machines are powerful enough for computer languages such as Algol, or Fortran (in which 80% of programs written, are coded) to be directly compiled in terms of the Turing machine operations. There are several advantages to this property, and to obtaining formal proofs that particular languages can be so represented:

(1) It allows our results on measures of computations and algorithms to be applied to programs in these languages.

(2) More generally it allows results from the theory of recursiveness, such as the unsolvability of the halting problem, to be applied in this way. A further step is obtained in the "rapprochement between the practical and theoretical aspects of computation" supplied by Wang [38], Shepherdson and Sturgis [31], and McCarthy [20].

[†] Arbib and Blum [3] consider, for integer arguments, the effect which the use of different radices has on the associated measures of computation.

(3) Such languages provide means of defining algorithms far superior to that of Turing machines, and variants thereof could be specifically designed for use in the theory of recursiveness. Davis definitions (1) – (26A) are a first step in this direction. A further example is McCarthy's conditional form, which we have found so useful in this thesis. Algorithms in such languages then represent proofs of recursiveness, and the formal recursiveness of the vast library of functions already coded in the computer languages is at once obtained.

We give an outline of the proof required with regard to Fortran. (For a description of the language see e.g. Jamison [15].)

<u>THEOREM 44</u>

Any function coded in Fortran is partially recursive in the functions defined by the subprograms it calls.†

---

† We are here using the words 'function', and 'partially recursive' in the extended sense defined above. The definition of the function which a Fortran program represents is dependent on a specification of the assumed arrangement of the input, and of how the output is to be interpreted.

PROOF

The specification statements such as DIMENSION, COMMON, EQUIVALENCE, TYPE may be regarded as defining, rather than being part of, the program. The DO statement, e.g. DO 3 I = J, K, L ,where the highest statement no. the program uses is (say) 100, is equivalent to

```
        |  |I = J
    101 |  |.  )
        |  |.  |
        |  |.  }   program forming the DO loop's range
        |  |.  )
      3 |  |IF (I.GE.K) GO TO 102
        |  |I = I + L
        |  |GO TO 101
    102 |  |..... instruction in program following
                                  end of DO loop
```

The arithmetical and logical expressions are clearly of the form $y = f(x_1,\ldots,x_n)$, where f is recursive, as are the predicates involved in the control statements such as the computed GO TO and the arithmetic and logical IF's. McCarthy [19] has shown that the function defined by any program consisting of operations which change (the contents of) registers according to functions of other registers and transfer control according to predicates on the registers, can be expressed in his conditional form in terms of the predicates and functions involved. Thus by our theorem 4 the result now follows.

<u>EXAMPLE</u> The claim that the function χ_n, defined in our optimisation theorem for infinite functions, is primitive recursive is based on a question of faith (supported by Church's thesis) that the algorithm we gave for it can in fact be coded on a Turing machine. We could meet this objection by actually presenting the Turing machine concerned or, equivalently, a recursive expression in terms of basis D, but this would be exceptionally tedious. On the other hand, once a stock of standard library functions, such as for $T(z,x,y)$, were developed, a Fortran-type program of quite moderate length could be supplied for χ_n. This would then constitute the formal proof required.

APPENDIX I

Definition of the Travelling Salesman Problem

A travelling salesman is scheduled to visit n cities and return to his starting point. The distance between each pair of cities is known. His problem is to select the order which requires the least travelling.

If $\left\{ (x_i, y_i) \;\middle|\; i = 1(1)n \right\}$ are the coordinates of the cities, the problem may be phrased as that of finding the permutation $\binom{1\ 2\ \ldots n}{a_1 a_2 \ldots a_n}$ for which

$$\sum_{i=1}^{n} \sqrt{(x^2_{a_{i+1}} - x^2_{a_i}) + (y^2_{a_{i+1}} - y^2_{a_i})}$$

is least.

No general method has been found which involves appreciably less than the n! operations required by an exhaustive search. A discussion of the problem is to be found in (e.g.) Flood [10].

Definition of the mxn Scheduling Problem

n objects are to be processed in turn by m machines in a common fixed order of machine. As soon as an object has been processed by the i^{th} machine ($i < m$), it joins the queue, if any, for the $i+1$ th. The order in which the objects are processed is the same for all the machines. The problem is to determine which order involves the least total processing time, from the start of the first object on the first machine, to the completion of the last on the m^{th}.

For a single machine ($m = 1$), the order of the objects is irrelevant. For two machines, the following algorithm has been shown to suffice: If the time the i^{th} object takes on the j^{th} machine is denoted $t_{i,j}$, arrange the set $\left\{t_{i,j} \mid i = 1(1)n, j = 1,2\right\}$ in order of magnitude to produce (say) $\left\{t_{i_1,j_1}, t_{i_2,j_2}, \ldots, t_{i_n,j_n}\right\}$. If $j_1 = 1$, let i_1 be the first object; if $j_1 = 2$, let i_1 be the last. Determine in turn the order of object i_t, for $t = 2,3,\ldots,n$. as follows. If $j_t = 1$, let the object i_t follow all those objects i_x (if any) such that j_x is 1 and x is in $1(1)t-1$; whereas if $j_t = 2$, let it precede all those (if any) for which j_x is 2 and x in $1(1)t-1$.

No such algorithm has been found for $m \geqslant 3$.

A discussion of the problem is to be found in Johnstone [16].

APPENDIX II

DAVIS' NOTATION

We list here notation, defined in Davis, which we employ. In each case we give the page number in Davis where the symbol concerned is introduced, plus an informal description of its use or, alternately, the thesis page where this is supplied.

PAGE NO. IN DAVIS	SYMBOL	DESCRIPTION
145	$[z]_n$	see p.12.
42	$N(x)$	see p.17.
58	$T(z,x_1,\ldots,x_n,y)$	
60	$U(x)$	see p.22 fn.
38	$\min_y$	
16	$U_i^n(x_1,\ldots,x_n)$	see p.78 fn.
162	$f^{(n)}$	see p.18 fn.
42	$\alpha(x)$	see p.19.
58	$\mathcal{L}$	see p.22 fn.
58	GL	
43	J	see p.25.

PAGE NO. IN DAVIS	SYMBOL	DESCRIPTION
44	K	see p.25.
44	L	see p.25.
147	S^1	see p.29.
42	$[x/y]$	see p.70 fn.
12	$S(x)$	see p.78 fn.
15	$x \dot{-} y$	$= x-y$ if $x \geqslant y$; $= o$ otherwise.
6	$\alpha \longrightarrow \beta (Z)$	α and β, in that order, are successive instantaneous descriptions for a Turing machine Z.
9	$(\overline{x_1, x_2, \dots, x_n})$	the tape expression $\underbrace{11\dots1}_{x_1+1}b\underbrace{11\dots1}_{x_2+2}\dots\dots b\underbrace{11\dots1}_{x_n+1}$.
59	$IC(x)$	x is the Gödel number of a q-symbol.
59	$AL(x)$	x is the Gödel number of a s-symbol.
60	$INIT_n(x_1, \dots, x_n)$	the Gödel number of the instantaneous description $q_1(\overline{x_1, \dots, x_n})$ (see description of $(\overline{x_1, \dots, x_n})$ above).

PAGE NO. IN DAVIS	SYMBOL	DESCRIPTION
61	YIELD(x,y,z)	x and y, in that order, are the Gödel numbers of successive instantaneous descriptions for a Turing machine with Gödel number z.
54	Pr(i)	the i th prime in the sequence of primes arranged in ascending order.
52	$\overset{x}{\underset{y=o}{\mathfrak{M}}} P(y,x_1,\ldots,x_n)$	= the minimum number $y \leqslant x$, such that $P(y,x_1,\ldots,x_n)$ is true; = o, if no such number exists.
43	R(x,y)	x mod y.
59	x * y	the Gödel number of the expression formed by concatenation of the expressions represented by the Gödel numbers x,y.
25	θ(Z)	the largest coefficient of a q-symbol occurring in the Turing machine Z.
21	$\alpha \xrightarrow[A]{} \beta \quad (Z)$	α and β, in that order, are successive instantaneous descriptions for a Turing machine Z relative to a set A.

PAGE NO. IN DAVIS	SYMBOL	DESCRIPTION
7	$\mathrm{RES}_Z(\alpha)$	the resultant of a computation on a Turing machine starting with instantaneous description α.
22	$\mathrm{RES}_Z^A(\alpha)$	as for $\mathrm{RES}_Z(\alpha)$, except that the computation concerned is one relative to a set A.
60	$\mathrm{ID}(x)$	x is the Gödel number of an instantaneous description.
60	$\mathrm{TM}(x)$	x is the Gödel number of a Turing machine.
59	$\mathrm{TERM}(x,z)$	z consists of an unbroken ascending sequence of prime factors starting with 2, and x is an exponent of one of these.
60	$\mathrm{MR}(x)$	the Gödel number of the tape expression $\underbrace{11\ldots1}_{x+1}$
62	$H_A(x,y,z)$	x and y, in that order, are Gödel numbers of successive instantaneous descriptions for a Turing machine with Gödel no. z, relative to a set A.

PAGE NO. IN DAVIS	SYMBOL	DESCRIPTION
164	$x^{[n]}(x_1,\ldots,x_n)$	the finite function represented by the integer x.
168 and 171	$T_n^{n_1,n_2,\ldots,n_k}(e,r_1,\ldots,r_k,x_1,\ldots,x_n,y)$	(see, first, description for $x^{[n]}$ above.) y is a Gödel number of a proof for a Turing machine with Gödel number e, with respect to a n+k-tuple argument $(t_1,t_2,\ldots,t_k,x_1,\ldots,x_n)$ such that, for $i = 1(1)k$, $t_i^{[n_i]}$ is a subfunction of $r_i^{[n_i]}$. Further if v is any Gödel number < y which is a proof for the Turing machine with respect to arguments $(s_1,s_2,\ldots,s_k,x_1,\ldots,x_n)$ where, for each i in 1(1)k, $s_i^{[n_i]}$ and $t_i^{[n_i]}$ have a common extension, then the function-values for the computations that y and v represent, are equal.
167	$f^{(n)}\|y$	the finite subfunction of the n-ary function $f^{(n)}$, which is defined over all $(x_1,\ldots,x_n)$ in

PAGE NO. IN DAVIS	SYMBOL	DESCRIPTION
		the domain of $f^{(n)}$ such that $x_i \leqslant y$ for $i = 1(1)n$ and $f^{(n)}(x_1,\ldots,x_n) \leqslant y$.
163	$\langle f^{(n)} \rangle$	the integer which represents the finite n-ary function $f^{(n)}$.

REFERENCES

[1] M.A.ARBIB, Monogenic Normal Systems are Universal, J.Austr.Math.Soc., vol.3(1963), 301-306.

[2] ——— , Speed-up Theorems and Incompleteness Theorems, in Automata Studies (Ed.,E.R.Caianiello), Academic Press Inc.,New York, 1966, 6-24.

[3] M.A.ARBIB and M.BLUM, Machine Dependence of Degrees of Difficulty, Proc.Amer.Math.Soc., vol.16 (1965), 442-447.

[4] I.BERECZKI, On Recursive Functions which are not Elementary, Acta Scientiarum Mathematicarum, vol.13 (1951).

[5] M.BLUM, Measures on the Computational Speed of Partial Recursive Functions, Quart.Progr.Rept.no.72, Res.Lab.Electronics, M.I.T.,1964, 237-253.

[6] J.P.CLEAVE, A Hierarchy of Primitive Recursive Functions, Zeitschr.Math.Logik Grundlagen Math., vol.9 (1963), 331-346.

[7] A.CHURCH, An Unsolvable Problem of Elementary Number Theory, Amer.J.of Maths.,vol 58 (1936), 345-363.

/8/ A.CHURCH, The Calculi of Lambda-Conversion, Annals of Mathematical Studies no.6, Princeton University Press, Princeton 1951.

/9/ M.DAVIS, Computability and Unsolvability, McGraw-Hill, New York, 1958.

/10/ M.M.FLOOD, The Travelling Salesman Problem, Op.Res.,vol.4 (1956), 61-75.

/11/ G.M.HARDY and E.M.WRIGHT, The Theory of Numbers, Oxford University Press, Oxford, 1945.

/12/ J.HARTMANIS and R.E.STEARNS, On the Computational Complexity of Algorithms, Trans.Amer.Math.Soc., vol.117 (1965), 285-306.

/13/ F.C.HENNIE, One Tape, Off-Line Turing Machine Computations, Information and Control, vol.8 (1965), 553-578.

/14/ ——— , On - Line Turing Machine Computations, I.E.E.E., Trans. on Elec.Computers, vol.EC-15 (1966), 35-45.

/15/ R.V.JAMISON, Fortran Programming, McGraw-Hill, New York, 1966.

[16] S.M.JOHNSTONE, Optimal Two- and Three-Stage Production Schedules with Set-Up Time Included, Naval Res.Log.Quart.,vol.1 (1954), 61-68.

[17] S.C.KLEENE, Introduction to Metamathematics, Van Nostrand, New York, 1952.

[18] C.Y.LEE, Categorizing Automata by W-Machine Programs, J. of ACM, vol.8 (1961), 384-389.

[19] J.McCARTHY, Recursive Functions of Symbolic Expression and Their Computation by Machine, Part I, Comm.of ACM, vol.3 (1960), 184-195.

[20] ———, A Basis for a Mathematical Theory of Computation, in Computer Programming and Formal Systems (Ed.,P.Braffort and D.Hischberg), North Holland, 1963, 33-70.

[21] M.MINSKY, Size and Structure of Universal Turing Machines using Tag Systems: a 4-symbol 7-state Machine, Proc.Symp.on Recursive Function Theory, Amer.Math.Soc.,Providence, R.I.,1962, 229-238.

[22] J.MYHILL, Linear Bounded Automata, WADD Tech. Note no.60-165, Univ.of Pennsylvania, Report no. 60-22, 1960.

[23] W.V.QUINE, Concatenation as a Basis for Arithmetic, J.Symb.Logic, vol.11 (1946), 105-114.

[24] M.O.RABIN, Degree of Difficulty of Computing a Function and a Partial Ordering of Recursive Sets, Tech.Report no.2, Hebrew University, Jerusalem, 1960.

[25] ——— , Real Time Computation, Israel J.of Maths., vol.1 (1963), 203-211.

[26] M.O.RABIN and H.WANG, Words in the History of a Turing Machine with a Fixed Input, J.of ACM, vol.10 (1963), 526-527.

[27] R.W.RITCHIE, Classes of Predictably Computable Functions, Trans.Amer.Math.Soc.,vol.106 (1963), 139-173.

[28] S.S.RUBY and P.C.FISCHER, Translational Methods and Computational Complexity, Proc.5th Ann.Symp. on Switching Theory and Logical Design, Princeton, 1964, 173-178.

[29] C.E.SHANNON, A Mathematical Theory of Communication, Univ.of Ill.Press, Ill.,1948.

[30] ——— , A Universal Turing Machine with Two Internal States, in Automata Studies, Princeton, 1956.

[31] J.C.SHEPHERDSON and H.E.STURGIS, Computability of Recursive Functions, J.of ACM, vol.10 (1963),217-225.

[32] S.WATANABE, 5-Symbol 8-State and 5-Symbol 6-State Universal Turing Machines, J.of ACM,vol.8 (1961) 476-483.

[33] R.M.SMULLYAN, Theory of Formal Systems, Annals of Mathematical Studies, no.47, Princeton University Press, Princeton, 1961.

[34] R.E.STEARNS, J.HARTMANIS and R.M.LEWIS III, Hierarchies of Memory Limited Computations, Proc. 6th Annual Symp.on Switching Theory and Logical Design, 1965.

[35] B.A.TRAKHTENBROT, Turing Machines with Logarithmic Delay (Russian), Algebra i Logika Sem.3, 1964, 33-48.

[36] A.M.TURING, On Computable Numbers, with an Application to the Entscheidungsproblem, Proc.Lond. Math.Soc.,vol.42 (1937), 230-265.

[37] J.V.VON NEUMANN and O.MORGENSTERN, Theory of Games and Economic Behaviour, Princeton University Press, Princeton, 1947.

[38] H.WANG, A Variant to Turing's Theory of Computing Machines, J.of ACM, vol.4 (1957), 63-92.

[39] H.YAMADA, Real-Time Computation and Recursive Functions not Real-Time Computable, IRE Transactions on Electr.Computers, vol.EC-11 (1962), 753-760.

Case	Occ. of y in $\mathfrak{B}$	Occ. of y in $\mathfrak{A}$	Occ. of F in $\mathfrak{A}$ in a scope of y	Occ. of y in $\mathfrak{A}$ in an argument place of F	Occ. of any of $x_1, \dots, x_n$ in $\mathfrak{B}$ in a scope of y	Permissibility of cases in substitutions according to					Case
						restrictions				intuition[6]	
						(1)	(2)	(3)	(1) & (2) & (3)		
	In cases 1–16 y is to be taken as other than $x_1, \dots, x_n$										
1	y does not occur both in $\mathfrak{B}$ and $\mathfrak{A}$					√	√	√	√	√	1
2	free	free		none		√	√	√	√	√	2
3	free	free		some		√	√	√	√	√	3
4	free	bound	none	none		*	*	√	*	*[7]	4
5	free	bound	some	none		†	*	√	*	†[8]	5
6	free	bound	some	some		†	*	√	*	†[8]	6
7	bound	free		none	none	*	√	√	*	*[7]	7
8	bound	free		none	some	*	√	√	*	*[7]	8
9	bound	free		some	none	*	√	*	*	*[7]	9
10	bound	free		some	some	†	√	*	*	†[8]	10
11	bound	bound	none	none	none	√	√	√	√	√	11
12	bound	bound	none	none	some	√	√	√	√	√	12
13	bound	bound	some	none	none	*	√	√	*	*[7]	13
14	bound	bound	some	none	some	*	√	√	*	*[7]	14
15	bound	bound	some	some	none	*	√	*	*	*[7]	15
16	bound	bound	some	some	some	*	√	*	*	*[7]	16
17	y is one of $x_1, \dots, x_n$ (irrespective of of whether it occurs or not in $\mathfrak{A}$)					√	√	√	√	√[9]	17

'√' stands for 'all examples are admissible'.
'*' stands for 'no examples are admissible'.
'†' stands for 'inadmissible examples exist'.

Restriction (1) is necessary to rule out cases 7, 13, and 14.

Restriction (2) is necessary because intuitively unacceptable examples of cases 5 and 6 exist, not ruled out by restriction (1).

Inadmissible example of case 5: In $(Eu)(Ev)(G(w, u) \sim \bar{G}(w, v)) \rightarrow (Ey)(G(w, y) \sim \bar{F}(w))$ which is intuitively valid and derivable (obtainable from μ), we cannot substitute $G(\mathfrak{a}, y)$ for $F(\mathfrak{a})$ as this gives $(Eu)(Ev)(G(w, u) \sim \bar{G}(w, v)) \rightarrow (Ey)(G(w, y) \sim \bar{G}(w, y))$ which is not universally valid since taking the natural numbers as individuals and $z \geqq x$ as $G(z, x)$ renders the formula false.

Inadmissible example of case 6: In $(Ey)F(y) \sim (Ey)((Ew)F(w) \sim (H(y) \vee \bar{H}(y)))$, we cannot substitute $G(\mathfrak{a}, y)$ for $F(\mathfrak{a})$, as this gives

[9] The admissibility of case 1 *includes* that of case 17, even when y occurs also in $\mathfrak{A}$, since replacing value-expressions of F by value-expressions of $\mathfrak{B}[x_1, \dots, x_{n+r}]$ as described above gives the same result as replacing them by value-expressions of $\mathfrak{B}[x_1', \dots, x_n', x_{n+1}, \dots, x_{n+r}]$ where $x_1', \dots, x_n'$ are any individual variables new to $\mathfrak{B}[x_1, \dots, x_{n+r}]$, $\mathfrak{A}$.

$(Ey)G(y, y) \sim (Ey)((Ew)G(w, y) \sim (H(y) \vee \bar{H}(y)))$, which is not universally valid since taking natural numbers as individuals, $z > x$ as $G(z, x)$ and $z = z$ as $H(z)$ renders the formula false.

Restriction (3) is necessary because intuitively unacceptable examples of case 10 exist, not ruled out by restriction (1).

Inadmissible example of case 10: In $(Ew)(F(y) \sim F(w))$, we cannot substitute $(y)(G(\mathfrak{a}) \sim G(y))$ for $F(\mathfrak{a})$ as this gives $(Ew)((y)(G(y) \sim G(y)) \sim (y)(G(w) \sim G(y)))$, which is not universally valid since taking the natural numbers as individuals and '*z is even*' as $G(z)$ renders the formula false.

Now much weaker and simpler rules of substitution for sentential and predicate variables may be used to supplant those of the authors within the authors' framework of axiom system and concept of w.f.f. (i.e. leaving all else unchanged). Such a replacement is inherently desirable. The rules in question have the advantage of being immediately intuitively apparent, and requiring no such elaborate justification as that given here. We show that they are as adequate as those of the authors in that the authors' rules can be derived from them. This at the same time provides a justification for the sufficiency of the authors' restrictions, since the derivations are intuitively legitimate, making use as they do of the intuitively acceptable rule of substitution for individual variables (α2)[10] and the amended rule of relabelling (δ), intuitively acceptable after the analysis above.

New rule of substitution for sentential variables: For any sentential variable X, any w.f.f. $\mathfrak{A}$ in which X occurs, and any w.f.f. $\mathfrak{B}$, we can replace X wherever it occurs in $\mathfrak{A}$ by $\mathfrak{B}$, provided that $\mathfrak{B}$ and $\mathfrak{A}$ have no individual variable in common.

Adequacy: The authors provide restrictions, we saw above, for ruling out instances of cases 3–7, i.e. they allow instances of cases 1 and 2 and combinations of these, whereas we have only provided for instances of case 1 being allowed (which is directly intuitively acceptable); however we can deduce by (α2) the simultaneous admissibility of instances of case 2, i.e. that $\mathfrak{B}$ and $\mathfrak{A}$ may have in common any variables that are free in both. (We can further deduce the simultaneous admissibility of instances of case 6 from (amended) (δ), i.e. that $\mathfrak{B}$ and $\mathfrak{A}$ may have in common any variables that are bound in both, provided that X does not occur in any scope thereof in $\mathfrak{A}$. One wonders why restriction (2) did not simply read 'no *free* individual variable of $\mathfrak{B}$ occurs in $\mathfrak{A}$'.)

[10] "A free individual variable may be replaced by any other individual variable, provided that the replacement be simultaneously effected at all the occurrences of the free variable and that the substituted variable has no bound occurrence in the original formula."

New rule of substitution for predicate variables: For any n-adic predicate variable F, any w.f.f. $\mathfrak{A}$ in which F occurs, and any w.f.f. with $n+r$ free variables, $\mathfrak{B}[x_1, \ldots, x_{n+r}]$, $r \geqq 0$, we can replace value-expressions of F wherever they occur in $\mathfrak{A}$ by value-expressions of $\mathfrak{B}$, according to the rule that any expression $F(\mathfrak{a}_1, \ldots, \mathfrak{a}_n)$ is to be replaced by $\mathfrak{B}[\mathfrak{a}_1, \ldots, \mathfrak{a}_n, x_{n+1}, \ldots, x_{n+r}]$, provided that $\mathfrak{B}$ and $\mathfrak{A}$ have no individual variable in common.

Adequacy The authors rule out instances of cases 4–10 and 13–16, i.e. they allow instances of cases 1, 2, 3, 11, 12, 17 and any combinations of these. We have only provided for instances of case 1 being allowed, but this includes the permissibility of instances of case 17 (see footnote 9), and we can deduce from (α2) the simultaneous admissibility of instances of cases 2 and 3 (i.e. that $\mathfrak{B}$ and $\mathfrak{A}$ may have in common any variables that are free in both (irrespective of whether any of these occurrences in $\mathfrak{A}$ are in argument places of F)) and by (amended) (δ) we can deduce the simultaneous admissibility of instances of cases 11 and 12 (i.e. that $\mathfrak{B}$ and $\mathfrak{A}$ may have in common any variables that are bound in both, provided that F does not occur in any scope thereof in $\mathfrak{A}$ (irrespective of whether there are occurrences of any of $x_1, \ldots, x_n$ in such scopes in $\mathfrak{B}$)).

An analysis similar to the foregoing may be applied to systems with a wider concept of w.f.f., in that the same variable may occur both free and bound in a formula, and scopes may contain scopes of the same variable[11] (e.g. Quine, ***Mathematical Logic*** (1955), or Church, ***Introduction to Mathematical Logic*** (1956)). Under such an interpretation of w.f.f., restriction (1) becomes vacuous in both rules (the other restrictions remain *necessary*). Using a wider rule of relabelling, which is now obtainable[12],

[11] For sentential variables, the categories of occurrence for y in $\mathfrak{A}$, $\mathfrak{B}$ must here be denoted by 'only free occurrences' and 'some bound occurrence'. For predicate variables (1) the table must be enlarged (to 27 cases) so as to encompass 3 categories of occurrence for y in $\mathfrak{B}$, namely: (a) y occurs free, does not occur bound, and is not one of $x_1, \ldots, x_n$; (b) y occurs free, also occurs bound, and is not one of $x_1, \ldots, x_n$; (c) either y occurs free, also occurs bound, and is one of $x_1, \ldots, x_n$, or y does not occur free, but occurs bound, and is not one of $x_1, \ldots, x_n$; (2) the categories of occurrence for y in $\mathfrak{A}$ must be taken as 'only free occurrences' and 'some bound occurrence'; (3) case 17 must be replaced by 'y occurs free but does not occur bound in $\mathfrak{B}$, and is one of $x_1, \ldots, x_n$ (irrespective of whether it occurs or not in $\mathfrak{A}$)'. A little reflection will show that the set of cases obtained is exhaustive.

[12] We consider the exhaustive set of cases: The variable (call it y) is changed into a variable (a) not occurring in the scope of σ, (b) occurring in the scope of σ either free, or bound in a scope not contained in the scope of σ, (c) occurring in the scope of σ only bound in scopes, contained in that of σ, but which do not contain an occurrence of y bound to σ, (d) occurring in the scope of σ bound in a scope, contained in that of σ, and which contains an occurrence of y bound to σ. Consideration of these cases shows that (a) and (c), and only these, are intuitively acceptable.

the authors' rules, with restriction (1) omitted, can be derived from our new rules, showing that the surviving restrictions are *sufficient*, and establishing at the same time that our rules can be adopted in place of those of the authors as before.

REFERENCES

[1] A. Church, ***Introduction to Mathematical Logic,*** vol. I, Princeton (Princeton University Press), 1st ed. 1944, revised ed. 1956 (Princeton Mathematical Series, No. 17).

[2] D. Hilbert and W. Ackermann, ***Grundzüge der theoretischen Logik,*** Berlin (Springer), 1st ed. 1928, 2nd ed. 1938, 3rd ed. 1949, 4th ed. 1959 (Die Grundlehren d. Mathematischen Wissenschaften, Bd. 27), New York (Dover Publications), reprint of 2nd ed. 1946, New York (Chelsea Pub. Co.), English translation of 2nd ed. 1950 (***Principles of Mathematical Logic***).

[3] Reviews of [2]: 1st ed.: C. H. Langford, ***Bulletin of the American Mathematical Society,*** vol. 36 (1930), pp. 22–25. 2nd ed.: J. B. Rosser, ***Bulletin of the American Mathematical Society,*** vol. 44 (1938), pp. 474–5 and W. V. Quine, this Journal, vol. 3 (1938), pp. 83–84. Translation of 2nd ed.: H. B. Curry, ***Bulletin of the American Mathematical Society,*** vol. 59 (1953), pp. 263–7 and G. Zubieta, this Journal, vol. 16 (1951), pp. 52–53. 3rd ed.: H. B. Curry as for translation of 2nd ed., and A. Church, this Journal, vol. 15 (1950), p. 59.

[4] D. Hilbert and P. Bernays, ***Grundlagen der Mathematik,*** vol. I, Berlin (Springer), 1934, reprinted Ann Arbor, Mich. (J. W. Edwards), 1944.

[5] S. C. Kleene, ***Introduction to Metamathematics,*** Amsterdam (North Holland), Groningen (Noordhoff), New York and Toronto (Van Nostrand), 1952.

[6] W. V. Quine, ***Mathematical Logic,*** Cambridge, Mass. (Harvard University Press), 1955.

UNIVERSITY OF LONDON

THE JOURNAL OF SYMBOLIC LOGIC
Volume 27, Number 2, June 1962

PAGER (D.)

Ph.D. 1967

AN EMENDATION OF THE AXIOM SYSTEM OF HILBERT AND ACKERMANN FOR THE RESTRICTED CALCULUS OF PREDICATES

DAVID PAGER

The fundamental role of the restricted calculus of predicates in applications of symbolic logic, and particularly in Hilbert's *Beweistheorie* as summed up by Hilbert and Bernays, makes it important that this logical calculus should be accurately defined. The first standard formulation of the calculus was that of Hilbert and Ackermann's ***Grundzüge der theoretischen Logik.*** This employed (in the first three editions[1]) a finite set of axioms and rules of derivation, with rules of substitution included. A reaction by Hilbert and Ackermann's successors[2] to persistent difficulty encountered with the rules of substitution has been to omit these rules, and instead enlarge the set of axioms and the other rules of derivation so as to encompass all possible substitutions. Such an enlargement seems to me to be undesirable. As an alternative, this note is designed to put the original approach of Hilbert and Ackermann for once and for all on a sound basis. The third edition is still erroneous, and there are also intuitively obscure portions of it, which have a history of amendment, and which are now recommended to intuition only on the grounds that no further errors have been detected. The present paper firstly corrects and justifies these parts of the formulation as it stands; then certain simplifying changes are suggested whereby the need of the justifications is avoided, and the formulation rendered more natural.

To make the discussion which follows less cumbersome, we introduce here some definitions:

(1) (a) A *value-expression* of a w.f.f. is the result of a substitution for some or all of its free variables.

(b) A *value-expression* of an n-adic predicate variable F is a value-expression of $F(x_1, \ldots, x_n)$.

(2) By a *scope* of an individual variable $\mathfrak{a}$, we mean a scope of a quantifier $(\mathfrak{a})$ or $(E\mathfrak{a})$.

(3) A w.f.f. $\mathfrak{B}$ with free variables $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$, and these only, is written $\mathfrak{B}[\mathfrak{a}_1, \ldots, \mathfrak{a}_n]$. The result of substituting in $\mathfrak{B}$ any n individual variables $\mathfrak{a}'_1, \ldots, \mathfrak{a}'_n$ for $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$ respectively, is denoted $\mathfrak{B}[\mathfrak{a}'_1, \ldots, \mathfrak{a}'_n]$.

(4) When we talk of a *restriction* on substitution for sentential and predicate variables, we refer to the restrictions on the occurrences of the

Received May 21, 1960.

[1] The present note does not refer to the recent fourth edition, which departs entirely from what Kleene refers to as a "Hilbert type" axiom system and uses instead a Gentzen type one.

[2] Such as Kleene ***Introduction to Metamathematics*** (1952), Quine ***Mathematical Logic*** (1955), Church ***Introduction to Mathematical Logic*** (1956).

same variable both in the formula in which the substitution is made and in the formula substituted therein.

(5) When we talk of the *necessity* of a restriction, taken from a set of restrictions, we refer to the question of whether it can be omitted, the other restrictions being retained unchanged.

We begin by pointing out an omission in the rule of relabelling (δ) which has survived all three editions, as well as the reviews and criticism by Langford, Rosser, Quine, Curry, Church, and Zubieta.

The rule in effect states: we may replace in a w.f.f. $\mathfrak{A}$ an individual variable in any occurrence σ in $\mathfrak{A}$ of a quantifier binding it and in all its occurrences in the scope of σ, by another individual variable, subject to the restriction that the result is a w.f.f.

Consider the following exhaustive set of cases:

Case	The variable is changed into a variable occurring		Permissability of relabelling according to	
			restriction[3]	intuition
1	not at all in $\mathfrak{A}$		√	√
2	free only in	the scope of σ	√	†
3	free, outside	,, ,, ,, ,,	*	*
4	bound, in	,, ,, ,, ,,	*	*
5	bound, only outside	,, ,, ,, ,,	√	√

'√' stands for 'all examples are admissible'.
'*' stands for 'no examples are admissible'.
'†' stands for 'inadmissible examples exist'.

Inadmissible example of case 2: Consider the formula $(Eu)(Ev)(G(u) \sim \bar{G}(v)) \rightarrow (Ey)(G(y) \sim \bar{G}(h))$. It is intuitively universally valid, since the l.h.s. asserts that $G(z)$ assumes both truth values, truth and falsity for appropriate z, while the r.h.s. asserts that for, some z, $G(z)$ assumes a particular truth value (i.e. that of $\bar{G}(h)$); and it is also derivable[4]; but we cannot relabel by replacing y with h, as this gives $(Eu)(Ev)(G(u) \sim \bar{G}(v)) \rightarrow (Eh)(G(h) \sim \bar{G}(h))$ which is not universally valid since taking the natural numbers as individuals and '*z is even*' as $G(z)$ renders the formula false.

Thus the authors' restriction is not sufficient, in that it allows case 2. If we add to the authors' restriction a restriction on case 2, this will be equivalent to allowing only cases 1 and 5. We will then have the

[3] I.e. the authors' restriction that the result be a w.f.f.

[4] Obtainable from $(\mathrm{Eu})(\mathrm{Ev})(\mathrm{G(u)} \sim \bar{\mathrm{G}}(\mathrm{v})) \rightarrow (\mathrm{Ey})(\mathrm{G(y)} \overline{\sim \mathrm{X}}) \ldots\ldots \mu$. Outline derivation of μ: $[Z \rightarrow (W \sim X)] \rightarrow [[Z \rightarrow (Q \sim X)] \rightarrow [Z \rightarrow (W \sim Q)]]$. Hence, by substitution, $[(y)(G(y) \sim X) \rightarrow (G(u) \sim X)] \rightarrow [[(y)(G(y) \sim X) \rightarrow (G(v) \sim X)] \rightarrow [(y)(G(y) \sim X) \rightarrow (G(u) \sim G(v))]]$. Hence, $(y)(G(y) \sim X) \rightarrow (G(u) \sim G(v))$, and therefore $(y)(G(y) \sim X) \rightarrow (u)(v)(G(u) \sim G(v))$, and therefore $(\bar{u})(v)(G(u) \sim G(v)) \rightarrow (\bar{y})(G(y) \sim X)$, from which result follows.

AMENDED RULE OF RELABELLING: We may replace in a w.f.f. $\mathfrak{A}$ an individual variable in any occurrence σ in $\mathfrak{A}$ of a quantifier binding it and in all its occurrences in the scope of σ, by an individual variable either new to $\mathfrak{A}$, or else occurring bound in $\mathfrak{A}$ in a scope lying outside that of σ.

Now, in constructing an axiomatic model of a category of inference (call it K), as is done in the restricted predicate calculus, we are able to systematise the role that intuition plays in determining the validity of inference in K. We reduce the role of intuition to that of accepting a number of initial premisses: these are that the axioms of the model mirror valid inferences, and that the rules of derivation are such that only valid inferences are thereby derivable. These intuitive steps made, the determination of various types of inference in K is made (or essayed at) by the purely formal means provided by the axiom system.

Thus the basic requirement of such a mathematical model is that the initial premisses provided by the axiom system should either be intuitively apparent, or else be shown to depend on other intuitively apparent premisses.

It is on these grounds that we criticize the authors' obscure rules of substitution for the sentential and predicate variables. The complicated restrictions on such substitutions are justified neither as to their sufficiency nor as to their necessity[5].

We supply this justification here. First we deal with the question of necessity.

SUBSTITUTION FOR SENTENTIAL VARIABLES. The authors' rule in effect states: For any sentential variable X, any w.f.f. $\mathfrak{A}$ in which X occurs, and any w.f.f. $\mathfrak{B}$, we can replace X wherever it occurs in $\mathfrak{A}$ by $\mathfrak{B}$ provided that:

restriction (1): the result is a w.f.f.;

restriction (2): no individual variable of $\mathfrak{B}$ occurs bound in $\mathfrak{A}$.

Consider the following exhaustive set of cases for any particular individual variable y:

[5] The need of such justification is illustrated by the history of amendment of the rules: The rule of substitution for sentential variables in the 1st edition of Hilbert and Ackermann (1928) was erroneous. A new version was given in the 2nd edition (1939). The rule of substitution for predicate variables in the 1st edition was also erroneous. The error was noted and a more correct statement given by Hilbert and Bernays, ***Grundlagen der Mathematik,*** volume I (1934), and also by Quine, ***A System of Logistic,*** (1934). Also a revised statement was given in the 2nd edition of Hilbert and Ackermann. According to Church, ***Introduction to Mathematical Logic,*** part I (1944), none of these statements was correct (but it turned out that Church's attribution of error to Hilbert and Bernays in this regard was itself incorrect). The 3rd edition of Hilbert and Ackermann (1949) contains a further revision of the rule. (A discussion of these difficulties is to be found in the revised (1956) edition of hurch's ***Introduction to Mathematical Logic***).

Case	Occurrence of y in $\mathfrak{B}$	Occurrence of y in $\mathfrak{A}$	Occurrence of X in $\mathfrak{A}$, in a scope of y	Permissibility of cases in substitutions according to restrictions			intuition[6]
				(1)	(2)	(1) & (2)	
1	y does not occur both in $\mathfrak{B}$ and $\mathfrak{A}$			√	√	√	√
2	free	free		√	√	√	√
3	free	bound	none	*	*	*	*[7]
4	free	bound	some	†	*	*	†[8]
5	bound	free		*	√	*	*[7]
6	bound	bound	none	√	*	*	√
7	bound	bound	some	*	*	*	*[7]

'√' stands for 'all examples are admissible'.
'*' stands for 'no examples are admissible'.
'†' stands for 'inadmissible examples exist'.

Restriction (1) is necessary to rule out case 5.

Restriction (2) is necessary to rule out case 4, for which there are intuitively inadmissible examples not ruled out by restriction (1).

Inadmissible example of case 4: In $(Eu)(Ev)(G(u) \sim \bar{G}(v)) \rightarrow (Ey)(G(y) \sim \bar{X})$, which is intuitively universally valid (c.f. previous inadmissible example), and in point of fact derivable (see footnote 4), we cannot substitute $G(y)$ for X, as this gives $(Eu)(Ev)(G(u) \sim \bar{G}(v)) \rightarrow (Ey)(G(y) \sim \bar{G}(y))$, which as before is not intuitively valid.

Substitution for predicate variables. The authors' rule in effect states: For any n-adic predicate variable F, any w.f.f. $\mathfrak{A}$ in which F occurs, and any w.f.f. with $n + r$ free variables $\mathfrak{B}[x_1, \ldots, x_{n+r}]$, $r \geqq 0$, we may replace value-expressions of F wherever they occur in $\mathfrak{A}$ by value-expressions of $\mathfrak{B}$ according to the rule that any expression $F(\mathfrak{a}_1, \ldots, \mathfrak{a}_n)$ is to be replaced by $\mathfrak{B}[\mathfrak{a}_1, \ldots, \mathfrak{a}_n, x_{n+1}, \ldots, x_{n+r}]$, provided that:

restriction (1): the result is a w.f.f.

restriction (2): none of $x_{n+1}, \ldots, x_{n+r}$ occur bound in $\mathfrak{A}$;

restriction (3): no individual variable occurring in a value-expression of F in $\mathfrak{A}$ occurs bound in $\mathfrak{B}$.

Consider the following exhaustive set of cases for any particular individual variable y:

[6] While it is not strictly necessary for the present analysis to give *all* the entries in this column, we do so in order to make the analysis applicable also for other sets of restrictions.

[7] These cases give results which are not w.f.f.'s.

[8] Example given below.