INFORMATION AND CONTROL 25, 30–44 (1974)

Representative Samples of Programmable Functions

L. K. SCHUBERT

Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

Any computable function ϕ may be viewed as a "generalization" of a finite function. Specifically, there is a "sample" (finite subset) of ϕ such that every minimal program for the sample is a program for ϕ. Like the representation of a function by a program, its representation by a sample is machine dependent. However, relative to any finite number of machines on which ϕ is programmable, there is a sample of ϕ which represents ϕ for each of the machines. On the other hand, given a representative sample of ϕ, the values of ϕ for arguments in its domain can only be found in the limit in general. If it is known that some program length b suffices for a function ϕ, then an upper bound can be effectively inferred from b on the cardinality of any representative sample of ϕ which does not contain redundant elements. Conversely, a bound on representative sample "size" (interpreted not as cardinality, but as a finite-one function of finite functions) effectively supplies a bound on requisite program length. Apart from these general considerations, certain detailed relationships between representative samples and minimal programs are also developed. For example, it is shown that any decision function with a representative sample of l elements can be programmed with no more than $(4 + \lceil \log_2 m \rceil)(l - 1) + c$ bits, where the largest argument appearing in the representative sample is m bits long. Such bounds can be interpreted as bounds on the information-theoretic complexity of the representative samples (finite functions) concerned.

1. INTRODUCTION

The existence of "convergent" learning machines such as the Perceptron (Minsky and Papert, 1969) suggests that certain types of functions (such as the linear threshold functions) are adequately characterized by rather small samples (subsets) of themselves. That is, if convergence is complete after some training period, the labelled patterns seen up to that point comprise a "representative sample" of the pattern-response relation.

In fact, Gold (1965) has shown that convergent function identifiers exist for all re classes of total recursive functions. So for any such class a finite subset of each function suffices to distinguish it from all other functions in the

Copyright © 1974 by Academic Press, Inc.
All rights of reproduction in any form reserved.

class. However, Gold also showed that there is no convergent identifier for the class of *all* total recursive functions. It may be asked, however, whether members of function classes for which there is no convergent identifier are nevertheless characterized by samples (finite subsets) of themselves in some sense. In particular, is this the case for the class of all partial recursive functions?

It is shown in Section 2 that the members of any re class of partial recursive functions are characterized by "representative" samples of themselves, in the sense that all minimal programs for a representative sample are programs for the function represented. In other words, each partial recursive function may be regarded as a *generalization* of a finite function. The use of minimal programs as a basis for function generalization was suggested by the seemingly important "simplicity principle" in science (e.g., Born, 1971); a similar idea lies at the root of the work on inductive inference by Solomonoff (1964), Willis (1970), and others. Representative samples are shown to provide a relatively machine-independent (though in general noneffective) characterization of programmable functions.

In Section 3 relations between program length and representative sample size for arbitrary partial recursive functions are explored. It is found that a given bound on program length adequate for programming a particular function entails a bound on the cardinality of representative samples of that function. This bound on cardinality can be found effectively if all programs of a given length or less can be found effectively. On the other hand, if representative sample size is measured not by cardinality but by a measure of size analogous to measures of length used for programs, then no bound on representative sample size can be inferred from a bound on program length. The situation is reversed for the converse problem of going from bounds on sample size to bounds on program length. Clearly, sample cardinality provides no bound on program length, since for any n there are infinitely many n-element functions requiring distinct programs. On the other hand, if finite function size is defined so that only finitely many finite functions are of a particular size or less and these functions can be found effectively, then representative sample size can be used to infer a bound on minimal program length.

In Section 4 an attempt is made to obtain good upper bounds on minimal program length for a function in terms of detailed properties (not merely "size") of a representative sample of the function. Attention is restricted to program length measures based on the number of "bits" in a program. It is shown, for example, that any decision function with a representative sample of l elements can be programmed with no more than $(4 + \lceil \log_2 m \rceil)(l - 1) + c$

bits, where the largest argument appearing in the representative sample is m bits long.

Finally, the degree of unsolvability of some problems concerning representative samples is considered briefly in Section 5. It is possible to go from representative samples to programs for functions limiting recursively but not recursively in general. The converse problem is not recursively solvable either, but it is not known whether it is limiting recursively solvable. It is then shown that the problem of finding the values of a function, given a representative sample of it, is limiting recursively solvable but not recursively solvable in general. However, if the machine under consideration computes total functions only (e.g., the primitive recursive functions), then the problem is recursively solvable.

2. Representative Samples of Programmable Functions

The following rather arbitrary definition of program machines is made to fix ideas.

Definition. Let $\mathscr{M}$ be the class of two-tape Turing machines (or some equivalent class of algorithmic devices). One tape of each machine is regarded as the input-output (I/O) tape, and the other is regarded as the program tape. For this reason members of $\mathscr{M}$ will be referred to as *program machines*. The finite-state part of each machine is coupled to the tapes through read-write heads. A computation begins with the finite-state automaton in a unique start state and with a program on the program tape and an input expression on the I/O tape. If and when the machine halts, the I/O tape expression represents the output. Suppose there is an effective one-to-one coding from tape expressions onto the integers N (assume the same tape-expression syntax for both tapes). The tape expression corresponding to code number x will be written as $\bar{x}$. If $\bar{x}$ is a program, x is called its index. The terms "input expression" and "input" will serve, respectively, to distinguish an initial I/O tape expression $\bar{y}$ from its code number y; similarly, a distinction is made between an "output expression" $\bar{z}$ and the "output" z. If M eventually halts with output z when supplied with program $\bar{x}$ and input y, this is written as $\phi_x{}^M(y) = z$. If M does not halt, $\phi_x{}^M(y)$ is undefined. Thus, M computes a partial function $\phi_x{}^M$ with program $\bar{x}$. However, it will be convenient to think of $\bar{x}$ not merely as a program for $\phi_x{}^M$ but as a program for any *subset* of $\phi_x{}^M$. In other words, $\bar{x}$ is a program for a function ϕ provided only that

$\phi_x{}^M(y) = \phi(y)$ for all y in the domain ϕ; $\phi_x{}^M(y)$ need not be undefined for y outside that domain.

DEFINITION. $\bar{x}$ is a *program for* ϕ (dependence on a fixed M is understood), where ϕ is a function, if $\phi_x{}^M(y) = z$ for all $\langle y, z\rangle \in \phi$. If such an $\bar{x}$ exists for a given ϕ, ϕ ill be said to be *programmable* (on M). If in addition, $\phi_x{}^M(y)$ is undefined whenever $\phi(y)$ is undefined (i.e., $\phi = \phi_x{}^M$), then $\bar{x}$ is a *strict program* for ϕ and ϕ is *strictly programmable* (on M).

Remarks. (i) It is in the strict sense that the correspondence between programs and functions is usually formalized.

(ii) $\phi_x{}^M$ is the union of functions for which $\bar{x}$ is a program.

(iii) If ϕ is programmable or strictly programmable on M, all subsets of ϕ are programmable on M, including non-re subsets (if any).

(iv) Any program for a total function is a strict program for that function.

(v) Any program is a program for the empty function.

DEFINITION. A *program length measure* assigns a nonnegative integral length to each program such that only a finite number of programs are of any particular length. The length of $\bar{x}$ will be written as $|\ \bar{x}\ |$.

A length measure need not be recursive, though this is a frequent assumption; furthermore, programs are often assumed to be effectively enumerable in order of nondecreasing length. For example, the number of elementary symbols in a program provides such a length measure. Feldman (1969) has given an interesting sufficient condition for enumerability of this type (he uses the term "occams enumerations").

In the following, obviously machine- and length-measure-dependent concepts will often be used without explicit reference to a particular machine or length measure. This should be kept in mind for a correct interpretation of the results. By a *universal machine* will be meant one on which all partial recursive functions are strictly programmable. Note that there are machines on which all partial recursive functions are programmable, but not all are strictly programmable.

DEFINITION. A (*strict*) *minimal program* for a function is a (strict) program for ϕ whose length does not exceed the length of any other (strict) program for ϕ.

DEFINITION. Suppose that each minimal program for a subset ξ of a function ϕ is a program for ϕ. Then ξ is a *representative subset* of ϕ, and ϕ is a *generalization* of ξ. If ϕ is the union of all functions (i.e., the most inclusive function) of which ξ is representative, then ξ is a *strictly representative subset* of ϕ, and ϕ is the *maximal generalization* of ξ. A function which is its own maximal generalization is *maximal*.

Remarks. (i) The relation "is a representative subset of" is reflexive and transitive (and in fact is a partial ordering on the programmable functions).

(ii) If ξ is a representative subset of ϕ, then $\bar{x}$ is a minimal program for ξ iff $\bar{x}$ is a minimal program for ϕ.

(iii) Any ξ' obtained by adding further elements of ϕ to a (strictly) representative subset ξ is also (strictly) representative.

(iv) Each ξ programmable on M is a strictly representative subset of some function (its maximal generalization).

(v) Any representative subset of a maximal function ϕ is strictly representative of ϕ.

(vi) Any programmable total function is maximal.

(vii) Any function which is maximal in terms of a particular machine and length measure is partial recursive.

(viii) A programmable function contains a strictly representative subset iff it is maximal.

In the following, a finite subset of a function will be referred to as a "sample" of the function (for brevity).

Theorem 1 states that any programmable function is in a sense characterized by a sample of itself (in much the same sense that it is characterized by a program for it; however, the representative-sample characterization is not always "effective", as is indicated in Section 5).

THEOREM 1. *If ϕ is a function programmable on M, then there is a representative sample of ϕ.*

Proof. The following diagonal argument is similar to one given by Pager (1969) in a theorem on finite decision tables.

Since ϕ is programmable on M, there is at least one minimal program $\bar{x}_0$ for ϕ.

Let P be the set of programs of length $| \bar{x}_0 |$ or less, and let

$$Q = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k\}$$

be the set of programs in P which are *not* programs for ϕ. If Q is empty (i.e., $k = 0$), then the empty function is a representative sample of ϕ since the minimal programs for the empty function are the absolutely shortest programs and these are included in P. If Q is not empty, then for each $\bar{x}_i$ in Q there is an element in the domain of ϕ, say y_i, such that $\phi^M_{x_i}(y_i)$ is undefined or is not equal to $\phi(y_i)$. Let ξ be the sample $\{\langle y_i, \phi(y_i)\rangle \mid i = 1, 2,..., k\} \subseteq \phi$. Then each program in Q is not a program for ξ, since it must fail for at least one element of ξ, but all minimal programs for ξ are in P; hence, all minimal programs for ξ are in $P - Q$—i.e. they are programs for ϕ—and ξ is a representative sample of ϕ.

COROLLARY 1.1. *If M is universal, there is a representative sample of each (subset of each) partial recursive function.*

COROLLARY 1.2. *There is a strictly representative sample of each maximal function programmable on M (and hence of each total function programmable on M).*

COROLLARY 1.3. *Any enumeration (recursive or otherwise) of a function programmable on M yields a representative sample eventually, since all elements of some representative sample are eventually enumerated.*

COROLLARY 1.4. *If ϕ is programmable on some finite number of machines, then there is a sample of ϕ which is representative of ϕ for all of these machines. The union of samples representative for individual machines yields such a sample. Note that different length measures are permitted for different machines.*

Corollaries 1.3 and 1.4 indicate that the characterization of functions by representative samples is less machine specific than characterization by programs.

3. Relations Between Program Length and Representative Sample Size

Can program length be used to obtain bounds on representative sample size, and conversely? The notion of sample size can be interpreted either in terms of sample cardinality or in terms of some measure of length analogous to program length measures. The next two theorems show that an upper bound can be inferred for sample cardinality but not for sample length (at least when the measure of sample length is recursive).

The proof of Theorem 1 provides an upper bound on the cardinality of *some* representative sample of ϕ. It turns out that this upper bound applies to *all* representative samples of ϕ once the samples have been purged of "redundant" elements.

DEFINITION. A representative sample ξ of ϕ is a *critical sample* of ϕ if no proper subset of ξ is a representative sample of ϕ.

In the following, I_ψ denotes the set of indices of minimal programs for a function ψ. If ξ is a representative sample of ϕ, then the following statements can easily be shown to be equivalent:

(a) ξ is a critical sample of ϕ.

(b) For all $u \in \xi$, $\xi - \{u\}$ is not a representative sample of ϕ.

(c) For all $\eta \subset \xi$, η is not a representative sample of ξ.

(d) For all $\eta \subset \xi$, $I_\xi \neq I_\eta$.

(e) For all $\eta \subset \xi$, either $I_\xi \cap I_\eta = \varnothing$ or $I_\xi \subset I_\eta$ (depending on whether or not there is a program for η shorter than any program for ξ).

It is worth noting that the concept of "criticality" of a function ξ is actually independent of the function ϕ represented by ξ; this is clear from the absence of ϕ in (c), (d), and (e).

In the following theorem $\#S$ stands for the number of elements of S.

THEOREM 2. *If $\bar{x}_0$ is a program for a function ϕ, then for any critical sample ξ of ϕ, $\#\xi < \#\{\bar{x} \mid |\bar{x}| \leqslant |\bar{x}_0|\}$. (The shorter $\bar{x}_0$, the better the upper bound obtained).*

Proof. Let $X = \{\bar{x} \mid |\bar{x}| \leqslant |\bar{x}_0|\}$, $Y = \{\bar{y} \mid \bar{y}$ is a minimal program for $\phi\}$, and $Z = X - Y$. Say that an element u of ξ "eliminates" a program $\bar{z}$ in Z if $\bar{z}$ is not a program for $\{u\}$. It is sufficient to show that each element of ξ eliminates a program in Z which is not eliminated by any other element of ξ. But if this were false for some $u \in \xi$, every program in Z would be eliminated by some element of $\xi - \{u\}$ and contradiction with fact (b) above would result. The strict inequality $\#\xi < \#X$ results from $\#Y > 0$.

It is obvious from fact (b) that every representative sample of a function ϕ contains a critical sample of ϕ. Thus, every representative sample of ϕ whose cardinality exceeds a certain fixed number (dependent on ϕ) contains "redundant" elements.

COROLLARY. *For any recursive program length measure such that programs*

are effectively enumerable in order of nondecreasing length, a bound on critical sample cardinality of a function can be effectively determined from a bound on requisite program length for that function.

Of course, knowing a bound on critical sample cardinality is insufficient in general for constructing a critical sample (see Section 5).

Now let a recursive measure of finite function "length" be given such that only a finite number of such functions are of any particular length. For any such measure of length,

THEOREM 3. *There is no algorithm for computing, from x, an upper bound on the length of a representative sample of ϕ_x^M if M is universal.*

Proof. If all functions computed by M with the absolutely minimal programs are empty, let $y_0 = 0$ and $z_0 = 0$. Otherwise, let y_0 be an input for which some absolutely minimal program produces an output, and let z_0 be some number greater than that output.

Now assume contrary to the theorem that there is a recursive function f such that $f(x)$ is an upper bound on the length of some representative sample of ϕ_x^M for all x. Consider a program $\bar{x}_0$ which carries out the following instructions for any given input y:

Enumerate the singleton functions $\{\langle y_0, z_0\rangle\}$, $\{\langle y_0, z_0 + 1\rangle\}$, $\{\langle y_0, z_0 + 2\rangle\}$,..., computing the length of each until one is found, say $\{\langle y_0, z_1\rangle\}$, whose length exceeds $f(x_0)$. If $y = y_0$, output z_1, otherwise run on forever.

That such an $\bar{x}_0$ exists even though the value of x_0 itself is used in the computation follows from the S-m-n theorem and the recursion theorem. Specifically, it can be shown that there is a recursive function g such that $\phi_{g(x)}^M = \lambda z[\phi_x^M(g(x), z)]$ for all x.

Now clearly $\phi_{\bar{x}_0}^M = \{\langle y_0, z_1\rangle\}$. By the definition of y_0 and z_0, at least one of the absolutely minimal programs fails to give output z_1 for input y_0, so that the empty function cannot be a representative sample of $\phi_{\bar{x}_0}^M$; hence, $\{\langle y_0, z_1\rangle\}$ is the only representative sample of $\phi_{\bar{x}_0}^M$. But its length exceeds $f(x_0)$, so that the purported algorithm for finding an upper bound on the length of a representative sample fails for $x = x_0$.

COROLLARY. *There is no algorithm for computing an upper bound on the length of a representative sample of ϕ_x^M from an upper bound on the requisite program length for ϕ_x^M if M is universal and the length measures concerned are recursive.*

The corollaries of the last two theorems have been concerned with inferences

about representative sample size when a bound on program length is known. Next consider the converse problem of inferring bounds on program length from bounds on representative sample size.

First, observe that sample cardinality cannot provide a bound on program length, since for any n there are infinitely many n-element functions requiring distinct programs. This leaves the question of whether a bound on program length can be inferred from a bound on representative sample "length" in the previously defined sense. The answer is affirmative.

THEOREM 4. *Let M be a program machine with programs at least for all finite functions, and let a recursive program length measure be given as well as a recursive measure of finite function length such that finite functions are effectively enumerable in order of nondecreasing length. Then there is an effective procedure for obtaining a bound on minimal program length for ϕ (programmable on M) from a bound on representative sample length for ϕ.*

Proof. For the given bound b on representative sample length, enumerate the set of finite functions $\{\xi \mid \text{length of } \xi \leqslant b\}$. For each such ξ generate an effective enumeration of N^2. For each pair $\langle x, t\rangle$ enumerated and for each y in the (finite) domain of ξ, operate M for t steps with program $\bar{x}$ and input y. If M produces output $\xi(y)$ for each such y within t steps, then $\bar{x}$ is a program for ξ and $|\bar{x}|$ provides a bound on the smallest programs for ξ. Such an $\bar{x}$ will be found eventually since ξ is programmable on M. The largest of all bounds found in this way provides an upper bound on minimal program length for ϕ, since at least one of the ξ is a representative sample of ϕ.

4. Bounds on the Minimal Number of Bits in Programs for Finite Functions

More interesting results on minimal program lengths are obtained if more information about representative samples (rather than size only) is used and if attention is restricted to universal machines and "information-theoretic" length measures of the type introduced by Solomonoff (1964), Kolmogorov (1965), and Chaitin (1970); i.e., programs are assumed to be binary strings and the length of a program is the number of bits in the string. Then for any effective one-to-one code from finite functions into binary strings, a fixed amount of additional code concatenated with any such code string $\bar{\xi}$ suffices to form a program for the finite function ξ; the additional code serves to decode $\bar{\xi}$ and to print the output $\xi(y)$ whenever the input y is in the domain

of ξ. Thus, if ξ is a representative sample of ϕ, an upper bound on the length of a minimal program for ϕ is $| \bar{\xi} | + c$, where c is independent of ξ.

Now, while a one-to-one encoding of finite functions is certainly sufficient for the above purpose, it is not in general necessary. In order for the program to regenerate correctly the values $\xi(y)$ for all arguments y in the domain of ξ, it need not "remember" (encode) those arguments but only *distinguish* them. For example, if $\xi = \{\langle y_1 , z_1 \rangle, \langle y_2 , z_2 \rangle\}$, where $\bar{y}_1 = 100110101$ and $\bar{y}_2 = 1110100001$, then only the second bits (say) of $\bar{y}_1$ and $\bar{y}_2$ need to be tested and $\bar{z}_1$ or $\bar{z}_2$ printed out accordingly by a program for ξ. This evidently permits a better upper bound on minimal program length in general than a one-to-one code.

The extent to which the $| \bar{\xi} | + c$ bound can be improved depends, first, on the class of finite functions under consideration and, second, on the class of functions from which the upper bound is chosen. Clearly, the more inclusive the latter class, the closer the upper bound can be to actual minimal program length.[1] On the other hand, only an easily calculated bound expressed in terms of simple parameters of a given finite function can shed any light on the relationship between representative samples and lengths of minimal programs for a function. The following theorem lists upper bounds on minimal program length for several rather "common" classes of finite functions; in view of the particular length measure under consideration, these can be regarded as bounds on the *information-theoretic complexity* K of the function in these classes. In calculating the bounds, it has been assumed that the programs on which they are based need not specify their own end points (thus it would not be clear where each program begins and ends if they were placed end to end on tape); if the programs are to be self-delimiting, an additional term logarithmic in the bound is required with each given bound.

THEOREM 5. *Let the programs of a universal program machine be binary strings whose length is measured in bits. Then the following bounds apply to the minimal length* $K(\xi)$ *of programs for a finite function* ξ. *Evidently, these bounds also apply to programs for any generalization* ϕ *of* ξ.

(a) *If* ξ *is any finite binary sequence, then* $K(\xi) \leqslant l + c$, *where* $l = \#\xi$.

(b) *If* ξ *is any finite integer sequence, then* $K(\xi) \leqslant 2L + c$, *where* $L = \sum_{i=0}^{l-1} | \bar{z}_i |$, $l = \#\xi$, *and* $z_i = \xi(i)$ *for* $i < l$.

[1] However, for any recursive upper bound on minimal program length, there are finite functions for which this upper bound will exceed the actual minimal length by an arbitrarily large amount.

(c) *If ξ is any finite binary-valued function (decision table) with arguments of equal length, then* $K(\xi) \leqslant (3 + \lceil \log_2 m \rceil)(l - 1) + c$, *where* $l = \#\xi$, $m = |\bar{y}_0| = \cdots = |\bar{y}_{l-1}|$, *and* $\{y_0, \ldots, y_{l-1}\}$ *is the domain of* ξ.

(d) *If ξ is any finite binary-valued function (decision table), then* $K(\xi) \leqslant (4 + \lceil \log_2 m \rceil)(l - 1) + c$, *where* $l = \#\xi$, $m = \max_{i<l} |\bar{y}_i|$, *and* $\{y_0, \ldots, y_{l-1}\}$ *is the domain of* ξ.

(e) *If ξ is any finite function from N into N, then*

$$K(\xi) \leqslant (3 + \lceil \log_2 m \rceil)(l - 1) + L + \log_2 \binom{L-1}{l-1} + \tfrac{3}{2} \log_2(L - l + 1) + c,$$

where $\xi = \{\langle y_i, z_i \rangle \mid 0 \leqslant i < l\}$, $m = \max_{i<l} |\bar{y}_i|$, *and* $L = \sum_{i=0}^{l-1} |\bar{z}_i|$.

Proofs and Discussion. (a) See Solomonoff (1964), Kolmogorov (1965), or Chaitin (1970); the argument used is very similar to the one above for establishing the $|\bar{\xi}| + c$ bound. The $l + c$ bound is optimal in the sense that for all sufficiently large k the overwhelming majority of binary sequences with $l + c = k$ have $K(\xi)$ in the interval $k \pm c'$, with c' independent of k and ξ.

(b) Finite integer sequences frequently occur in mathematical tables, e.g., the squares of the integers, or a list of prime numbers. One way to encode such a list is to place the binary representations of the successive integers end to end. However, boundaries between successive integers must somehow be identified. One way of doing this is as follows: Suppose the total number of bits in the concatenated binary-coded integers is L. The total number of ways of segmenting a sequence of L bits is $\sum_{k=0}^{L-1} \binom{L-1}{k} = 2^{L-1}$, so that an additional $(L - 1)$-bit prefix will suffice to indicate any particular disposition of boundaries. Thus, $2L + c$ is an upper bound on the minimal program length. This bound is again optimal in that for sufficiently large k the overwhelming majority of sequences with $2L + c = k$ have $K(\xi)$ within the interval $k \pm c'$. The reason is that the code strings (consisting of a prefix and a number of concatenated binary-coded integers) of length $2L - 1$ encode 2^{2L-1} integer sequences, all of which require *distinct* programs, as is easily verified.

(c) Finite decision tables with arguments of equal length (equal numbers of bits) are of importance in switching theory. They are also of interest in pattern recognition, where the problem of mapping digital arrays (images) of fixed size onto two or more pattern classes is frequently encountered. The code strings on which the given bound is based contain four successive segments: an encoding of l (length, $\sim \frac{3}{2} \log_2 l$); an encoding of the form of a decision tree, each of whose $l - 1$ nonterminal nodes corresponds to a conditional

branch contingent on the presence of a 1 bit at a particular position in a given argument and whose l terminal nodes correspond to the l given function values (length, $\sim 2l - \frac{3}{2}\log_2 l$); a listing of the positions of the $l-1$ decision bits used at the nonterminal nodes of the decision three (length, $\sim (l-1)\lceil \log_2 m \rceil$); and a listing of the binary values associated with the given arguments, in the order corresponding to the arrangement of terminal decision tree nodes (length, $\sim l$). The initial code for l consists of the binary representation of l (length, $\lfloor \log_2(x+1) \rfloor$ if the binary string $b_0 b_1 \cdots b_k$ is taken as code of the number $2^{k+1} - 1 + \sum_{i=0}^{k} b_i 2^i$) with an additional prefix which indicates in a type of unary code the length of the representation of l that follows it. Specifically, if the binary representation of l contains b bits (say), the prefix consists of an initial 0 or 1 (corresponding to b even or odd, respectively) followed by $\lfloor b/2 \rfloor$ 0's followed by a 1; thus the over-all length of the code for l is within a constant of $\frac{3}{2}\log_2 l$. The length of the decision tree code is based on the fact that there are $(2l-2)!/[l!(l-1)!]$ distinct binary decision trees with l terminal nodes, where neither nodes nor edges are labelled. If the index of a tree in a systematic enumeration of all of these trees is taken as its code number, the number of bits required to designate a particular tree is found with the aid of Stirling's formula to lie within a constant of $2l - \frac{3}{2}\log_2 l$. All codes for positions of decision bits in the arguments are assumed to be of length $\sim \lceil \log_2 m \rceil$; this is certainly sufficient for m-bit arguments. It is easy to show that the boundaries of various code segments in the complete code string can be inferred from the code string itself (assuming it is known where the complete string ends). The resulting bound $(3 + \lceil \log_2 m \rceil)(l-1) + c$ is not optimal in the strong sense of (a) and (b) above, since the first segment (length, $\sim \frac{3}{2}\log_2 l$) could be made of length $\sim (1+\epsilon)\log_2 l$ for any $\epsilon > 0$, and it is possible to shorten the remainder of the code string as well; however, it is conjectured that the deviation from optimality is logarithmic in the given bound at most.

(d) Here the code is much the same as in (c), except that an additional bit is stored for each nonterminal node of the decision tree. This bit determines which of the two questions "Is the length of argument exceeded ?" or "Is the bit in the specified position a 1 ?" shall be asked at that node.

(e) Here the code is much the same as in (d), except that an encoding of $L - l$ (length $\sim \frac{3}{2}\log_2(L - l + 1)$) is inserted between the code segment for l and the code segment for the form of the decision tree; in addition, the space taken up by the values associated with the given arguments is now $L = |\bar{z}_0| + |\bar{z}_1| + \cdots + |\bar{z}_{l-1}|$, and this sequence of values is prefixed with a piece of code indicating the disposition of the $l-1$ boundaries

between them (requires $\log_2\binom{L-1}{l-1}$ bits). No claim is made about the optimality of this or the preceding bound.

The assumption in Theorem 5 that the given program machine is universal could be weakened; the machine need only be powerful enough to decode the binary strings described above. This is possible with a machine which always halts, in which case the problem of finding a minimal program for ξ is solvable.

5. Some Further Solvability Questions Concerning Representative Samples

In Section 3 some solvability questions concerning bounds on program length and bounds on representative sample size were answered. More generally, the difficulty of going from one "representation" of a function to another is of some interest, and this is briefly discussed below. Then the "effectiveness" of representation of a function by a representative sample is considered.

First consider the problem of going from a representative sample of a function to a program for the function. The predicate [$\bar{x}$ is a minimal program for ξ] is easily shown to be in $\Sigma_2 \cap \Pi_2$ of the Kleene hierarchy. Thus the problem under consideration is limiting recursively solvable, in the terminology of Gold (1965). Löfgren (1967) and Pager (1969) have independently shown that the problem of finding minimal programs for finite functions is not recursively solvable (see also Schubert, 1973).

The converse problem of going from a strict program[2] for a function to a representative sample of the function is not recursively solvable by Theorem 3. It is conjectured that this problem, and the closely related problem of finding a minimal program for ϕ_x, given x, is not limiting recursively solvable, but no proof has been found for this conjecture. (The problem of finding a *strict* minimal program for ϕ_x *is* known to be nonlimiting recursively solvable; see Meyer, 1972).

Finally, how difficult is it to find the values of a function for which a representative sample is given?

[2] It would be meaningless to ask for a representative sample when the given program is not a strict program for the function. *Any* set of input-output pairs generated by a program (including the empty set) is representative of *some* function computed by M with that program—namely, that set itself.

THEOREM 6. *Given a recursive length measure, the values of a programmable function at points in its domain can be found in the limit from a representative sample of the function. If M computes total functions only and programs are re in order of nondecreasing length, then function values can be found effectively from a representative sample. For some program machines, however, no such effective procedure exists.*

Proof. Since a minimal program for a representative sample can be found in the limit, function values can also be found in the limit.

If M computes total functions only and programs are re in order of nondecreasing length, then of course minimal programs for representative samples of functions and, hence, for the functions can be found effectively.

Finally, assume contrary to the theorem that the values of any function programmable on any M in $\mathscr{M}$ can be determined effectively from a representative sample of the function, for some length measure. Then let M be universal, and let f be a partial recursive function of two arguments such that $f(\hat{\xi}, y) = \psi(y)$ for all y in the domain of ψ whenever ξ (with Gödel number $\hat{\xi}$) is a representative sample of ψ.

Enumerate elements $\langle y, n\rangle$ of N^2 and correspondingly run the computation of $f(\hat{\xi}_y, y + 1)$ for n steps, where $\xi_y = \{\langle 0, 0\rangle, \ldots, \langle y, 0\rangle\}$, until a pair $y = y_1$, $n = n_1$ is found such that the corresponding computation halts and yields output 0. Such a pair will be found eventually since, for any sufficiently large y, ξ_y is a representative sample of the function which is zero for all arguments. Correspondingly, define the first $y_1 + 2$ arguments of a function g to be $g(y) = 0$ for $y \leqslant y_1$ and $g(y_1 + 1) = 1$. Now use a similar procedure to find y_2, n_2 such that $f(\hat{\xi}_{y_1+y_2+1}, y_1 + y_2 + 2)$ yields output 1 within n_2 steps, where $\xi_{y_1+y_2+1} = \{\langle 0, 0\rangle, \ldots, \langle y_1, 0\rangle, \langle y_1 + 1, 1\rangle, \ldots, \langle y_1 + y_2 + 1, 1\rangle\}$. Then define $g(y) = 1$ for $y_1 + 1 \leqslant y \leqslant y_1 + y_2 + 1$ and $g(y_1 + y_2 + 2) = 0$. Similarly continue assigning sequences of 0's and 1's as values of g.

Then g is programmable on M but, for infinitely many initial segments $\xi = \{\langle 0, g(0)\rangle, \ldots, \langle y, g(y)\rangle\}$ of g, $f(\hat{\xi}, z) \neq g(z)$ for some z. By Corollary 1.3, f fails for some representative samples of g, and the resultant contradiction establishes the theorem.

6. CONCLUDING REMARKS

It has been shown that generalization of finite functions via minimal programs leads to a class of functions which includes each function ϕ, or an extension of ϕ, which can be computed with the program machine under

consideration. If the machine is universal, each partial recursive function can be regarded as a generalization of a finite subset (representative sample) of itself. The characterization of a function by a representative sample has been shown to be less machine dependent than its characterization by a program. Various relationships between representative samples and programs have been examined. As a byproduct, bounds on minimal program size for various types of finite functions have been obtained.

Acknowledgments

The author is indebted to D. V. Gottlieb of Johns Hopkins University for stimulating discussions on the initial part of this work, and to J. R. Sampson of the University of Alberta for his many useful comments on the manuscript. The work was conducted under a National Research Council Postdoctorate Fellowship.

Received: March 22, 1973; revised: November 8, 1973

References

Born, M. (1971), "The Born-Einstein Letters," p. 163, Macmillan.

Chaitin, G. (1970), On the difficulty of computations, *IEEE Trans. Inf. Theory* **IT-16**, 5–9.

Feldman, (1972), "Some Decidability Results on Grammatical Inference and Complexity," Stanford Artificial Intelligence Project Memo No. AI-93 (1969); also *Information and Control* **20**, 244–262.

Gold, E. M. (1965), Limiting recursion, *J. Symb. Logic* **30**, 28–48.

Kolmogorov, A. N. (1965), Three approaches to the quantitative definition of information, *Inform. Transmission* **1**, 3–11; also *Int. J. Comp. Math.* **2**, 157–168 (1968).

Löfgren, L. (1967), Recognition of order and evolutionary systems, *in* "Computer and Information Sciences," Vol. II (J. Tou, Ed.), 165–175, Academic Press.

Meyer, A. R. (1972), Program size in restricted programming languages, *Information and Control* **21**, 382–394.

Minsky, M. and Papert, S. (1969), "Perceptrons," MIT Press, Cambridge, MA.

Pager, D. (1969), On the problem of finding minimal programs for tables, *Information and Control* **14**, 550–554.

Schubert, L. K. (1973), "Iterated Limiting Recursion and the Program Minimization Problem," Technical Report No. TR 73-2, Dept. of Computing Science, University of Alberta; *J. Ass. Comp. Mach.*

Solomonoff, (1964), A formal theory of inductive inference, *Information and Control* **7**, 1–22, 224–254.

Willis, D. G. (1970), Computational complexity and probability constructions, *J. Ass. Comp. Mach.* **17**, 241–259.