UDC 621.391.12

THREE APPROACHES TO THE QUANTITATIVE DEFINITION OF INFORMATION

A. N. Kolmogorov

Problemy Peredachi Informatsii, Vol. 1, No. 1, pp. 3-11, 1965

There are two common approaches to the quantitative definition of "information": combinatorial and probabilistic. The author briefly describes the major features of these approaches and introduces a new algorithmic approach that uses the theory of recursive functions.

1. The Combinatorial Approach

Assume that a variable x is capable of taking values in a finite set X containing N elements. We say that the "entropy" of the variable x is

$$H(x) = \log_2 N.$$

By giving x a definite value

$$x = a$$

we "remove" this entropy and communicate "information"

$$I = \log_2 N.$$

If the variables $x_1, x_2, \ldots, x_k$ are capable of independently taking values in sets respectively containing $N_1, N_2, \ldots, N_k$ members, then

$$H(x_1, x_2, \ldots, x_k) = H(x_1) + H(x_2) + \ldots + H(x_k). \tag{1}$$

Transmission of a quantity of information I requires

$$I' = \begin{cases} I & \text{for integral I} \\ [I] + 1 & \text{for fractional I} \end{cases}$$

binary digits. For example, the number of different "words" consisting of k zeros and ones and one two is

$$2^k (k + 1).$$

Hence, the information content of such a message is

$$I = k + \log_2 (k + 1),$$

i. e., the "coding" of such words in a purely binary system requires*

$$I' \approx k + \log_2 k$$

zeros and ones.

Discussions of information theory do not usually go into this combinatorial approach at any length, but I consider it important to emphasize its logical independence of probabilistic assumptions. Suppose, for example, that we are faced with the problem of coding a message written in an alphabet consisting of s letters, it being known that the frequencies

$$p_r = \frac{s_r}{s} \tag{2}$$

of occurrence of individual letters in a message of length n satisfy the inequality

$$\chi = -\sum_{r=1}^{s} p_r \log_2 p_r \leqslant h. \tag{3}$$

*Here and in what follows $f \approx g$ indicates that the difference $f - g$ is bounded, while $f \sim g$ indicates that the ratio $f : g$ approaches one.

It is easy to see that for large n, the binary logarithm of the number of messages satisfying requirement (2) has the asymptotic estimate

$$H = \log_2 N \sim nh.$$

In transmitting such messages, therefore, it is sufficient to use approximately nh binary digits.

A universal coding method that permits the transmission of any sufficiently long message in an alphabet of s letters with no more than nh binary digits is not necessarily excessively complex; in particular, it is not essential to begin by determining the frequencies p_r for the entire message. In order to make this clear, it is sufficient to note that by splitting the message S into m segments $S_1, S_2, \ldots, S_m$, we obtain the inequality

$$\chi \geqslant \frac{1}{n}[n_1\chi_1 + n_2\chi_2 + \ldots + n_m\chi_m]. \tag{4}$$

However, I will not go into the details of this special problem here. It is only important for me to show that the mathematical problems associated with a purely combinatorial approach to the measure of information are not limited to trivialities.

It is perfectly natural to take a purely combinatorial approach to the notion of the "entropy of language" if we have in mind an estimate of its "flexibility," an index of the diversity of the possibilities for developing a language with a given dictionary and given rules for the construction of sentences. M. Ratner and N. Svetlova obtained the following estimate for the binary logarithm of the number N of Russian texts of length n, expressed as the "number of symbols including spaces," composed of words in S. I. Ozhegov's Russian dictionary and subject only to the requirement of "grammatical correctness"

$$h = \frac{\log_2 N}{n} = 1.9 \pm 0.1.$$

This is considerably larger than the upper estimate for the "entropy of literary texts" that can be obtained by various methods of "guessing continuations." This discrepancy is quite natural, since literary texts must meet many requirements beyond simple "grammatical correctness."

It is more difficult to estimate the combinatorial entropy of texts subject to definite, more elaborate constraints. It would, for example, be of interest to estimate the entropy of Russian texts that could be regarded as sufficiently accurate (in terms of content) translations of a given foreign-language text. It is only "residual entropy" that makes it possible to translate poetry, where the "entropy cost" of adhering to a given meter and rhyme scheme can be calculated rather accurately. It can be shown that the classical rhyming iambic tetrameter, with certain natural restraints on the frequency of syllables, etc., requires a freedom in handling verbal material characterized by a "residual entropy" of the order of 0.4 (this estimate is based on the above method of measuring the length of a text in terms of the "number of symbols, including spaces"). On the other hand, if we take into account the fact that the stylistic limitations of a particular genre probably reduce the above estimate of the "total" entropy from 1.9 to no more than 1.1-1.2, the situation becomes remarkable both in the case of translation and in the case of original poetry.

I trust the reader of a utilitarian bent will forgive me this example, but it should be noted that the broader problem of measuring the information connected with creative human endeavor is of the utmost significance.

At this point, let us turn to a discussion of the extent to which a purely combinatorial approach permits one to estimate the information conveyed by a variable x with respect to a related variable y. The relation between the variables x and y, which respectively take values in the sets X and Y, consists in that not all pairs (x, y) belonging to the Cartesian product X × Y are "possible." The set U of possible pairs determines the set Y_a of y such that for a given $a \in X$

$$(a, y) \in U.$$

It is natural to define the conditional entropy by the equation

$$H(y/a) = \log_2 N\ (Y_a) \tag{5}$$

(where $N(Y_X)$ is the number of members of Y_X) and the information conveyed by x with respect to y by the formula

$$I\ (x : y) = H(y) - H(y/x). \tag{6}$$

For the case shown in the table, for example, we have

x \ y	1	2	3	4
1	+	+	+	+
2	+	−	+	−
3	−	+	−	−

$$I(x=1:y)=0,$$
$$I(x=2:y)=1,$$
$$I(x=3:y)=2.$$

Clearly, H(y/x) and I(x: y) are functions of x (whereas y takes the form of a "bound variable").

It is not difficult to introduce in a purely combinatorial conception the notion of the "quantity of information necessary to designate an object x with given requirements imposed on the accuracy of the designation." (Apropos of this see the extensive literature on the "ε-entropy" of sets in metric spaces.)

It is obvious that

$$H(x/x)=0, \qquad I(x:x)=H(x). \tag{7}$$

2. The Probabilistic Approach

The possible advantages of further developing information theory on the basis of definitions (5) and (6) have been overshadowed by the fact that if we make the variables x and y "random variables" with given joint probability distributions, we can obtain a considerably richer system of concepts and relationships. Paralleling the quantities introduced in §1, here we have

$$H_W(x)=-\sum_x p(x)\log_2 p(x), \tag{8}$$

$$H_W(y/x)=-\sum_y p(y/x)\log_2 p(y/x), \tag{9}$$

$$I_W(x:y)=H_W(y)-H_W(y/x). \tag{10}$$

As before, $H_W(y/x)$ and $I_W(x:y)$ are functions of x, and we have the inequalities

$$H_W(x)\leqslant H(x),\ H_W(y/x)\leqslant H(y/x), \tag{11}$$

where the equality holds when the corresponding distributions (on both X and Y_x) are uniform. The quantities $I_W(x:y)$ and I(x: y) are not related by an inequality of a particular direction. As in §1,

$$H_W(x/x)=0,\ I_W(x:x)=H_W(x). \tag{12}$$

The difference lies in the fact that we can form the mathematical expectations

$$MH_W(y/x),\ MI_W(x:y),$$

while the quantity

$$I_W(x,y)=MI_W(x:y)=MI_W(y:x) \tag{13}$$

symmetrically characterizes the "closeness of the relation" between x and y.

However, it should be noted that the probabilistic approach gives rise to a paradox: In the combinatorial approach, I(x: y) is always non-negative, which is natural in a naive conception of information content, but $I_W(x:y)$ may be negative. Now only the averaged quantity $I_W(x, y)$ is a true measure of the information content.

The probabilistic approach is natural in the theory of information transmission over communications channels carrying "bulk" information consisting of a large number of unrelated or weakly related messages obeying definite probabilistic laws. In this type of problem there is a harmless and (in applied work) deep-rooted tendency to mix up probabilities and frequencies within a sufficiently long time sequence (which is rigorously justified if it is assumed that "mixing" is sufficiently rapid). In practice, for example, it can be assumed that the problem of finding the "entropy" of a flow of congratulatory telegrams and the channel "capacity" required for timely and undistorted transmission is validly represented by a probabilistic treatment even with the usual substitution of empirical frequencies for probabilities. If something goes wrong here, the problem lies in the vagueness of our ideas of the relationship between mathematical probability theory and real random events in general.

But what real meaning is there, for example, in asking how much information is contained in "War and Peace"? Is it reasonable to include this novel in the set of "possible novels," or even to postulate some probability distribution for

this set? Or, on the other hand, must we assume that the individual scenes in this book form a random sequence with "stochastic relations" that damp out quite rapidly over a distance of several pages?

Actually, we are just as much in the dark over the fashionable question of the "quantity of hereditary information" necessary, say, for the reproduction of particular form of roach. Still, within the limits of the probabilistic approach, two variants are possible. In the first variant, we must consider the set of "possible forms" with a probability distribution of uncertain origin on this set*. In the second variant, the characteristics of the form are assumed to be a set of weakly dependent random variables. The real nature of the mechanism of mutation provides arguments favoring the second variant, but these arguments are undermined if we assume that natural selection causes a system of consistent characteristics to appear.

3. An Algorithmic Approach

Actually, it is most fruitful to discuss the quantity of information "conveyed by an object" (x) "about an object" (y). It is not an accident that in the probabilistic approach this has led to a generalization to the case of continuous variables, for which the entropy is infinite but, in a large number of cases,

$$I_W(x, y) = \iint P_{xy}(dx\,dy) \log_2 \frac{P_{xy}(dx\,dy)}{P_x(dx)\,P_y(dy)}$$

is finite. The real objects that we study are very (infinitely) complex, but the relationships between two separate objects diminish as the schemes used to describe them become simpler. While a map yields a considerable amount of information about a region of the earth's surface, the microstructure of the paper and the ink on the paper have no relation to the microstructure of the area shown on the map.

In practice, we are most frequently interested in the quantity of information "conveyed by an individual object x about an individual object y." It is true, as we have already noted, that such an individual quantitative estimate of information is meaningful only when the quantity of information is sufficiently large. It is, for example, meaningless to ask about the quantity of information conveyed by the sequence

0 1 1 0

about the sequence

1 1 0 0.

But if we take a perfectly specific table of random numbers of the sort commonly used in statistical practice, and for each of its digits we write the unit's digit of the units of its square according to the scheme

0 1 2 3 4 5 6 7 8 9
0 1 4 9 6 5 6 9 4 1,

the new table will contain approximately

$$\left(\log_2 10 - \frac{8}{10}\right) n$$

bits of information about the initial sequence (where n is the number of digits in the tables).

Accordingly, below we propose to define

$$I_A(x : y)$$

so that some indeterminacy remains. Different equivalent variants of this definition will lead to values equivalent only in the sense that $I_{A_1} \approx I_{A_2}$, i. e.,

$$|I_{A_1} - I_{A_2}| \leqslant C_{A_1A_2},$$

where the constant $C_{A_1A_2}$ depends on the two basic ways of defining the universal methods of programming A_1 and A_2.

Consider an "indexed domain of objects," i. e., a countable set

$$X = \{x\},$$

*binatorial calculation of the number of possible forms extant (or once extant) on the earth would ...er limit (something like < 100 bits).

with a finite sequence n(x) of zeros and ones, beginning with a one, associated with each element as its index. Denote the length of the sequence n(x) by $l(x)$, and assume that:

1) the correspondence between X and the set D of binary sequences of the form described above is one-to-one;

2) $D \subset X$, the function n(x) on D is generally recursive [1], and for $x \in D$

$$l(n(x)) \leqslant l(x) + C,$$

where C is a constant;

3) together with x and y, the set X contains the ordered pair (x, y), whose index is a generally recursive function of the indices of x and y and

$$l(x, y) \leqslant C_x + l(y),$$

where C_X depends only on x.

Not all of these requirements are essential, but they do simplify the discussion. The end result of the construction is invariant under transition to a new indexing n'(x) that has the same properties as the old system, and can be generally recursively expressed in terms of it; moreover, X retains its properties when embedded in a larger system X' (provided that, for the members of the initial system, the index n' in the expanded system can be generally recursively expressed in terms of the initial index n). The new "complexity" K and quantity of information remain equivalent under these transformations in the sense of $\approx$.

As the "relative complexity" of an object y with a given x, we will take the minimal length l(p) of the "program" p for obtaining y from x. The definition thus formulated depends on the "programming method," which is nothing other than the function

$$\varphi(p, x) = y,$$

that associates on object y with a program p and an object x.

In accordance with the views now universally accepted in modern mathematical logic, we must assume that the function φ is partially recursive. For any such function we have

$$K_\varphi(y/x) = \begin{cases} \min\limits_{\varphi(p, x)=y} l(p) \\ \infty, \text{ if there is no p such that } \varphi(p, x) = y. \end{cases}$$

In this case a function

$$v = \varphi(u)$$

of $u \in X$ with range $v \in X$ is said to be partially recursive if it generates a partially recursive function of the index transformation

$$n(v) = \Psi[n(u)].$$

In order to understand the definition, it is important to note that, in general, partially recursive functions are not defined everywhere, and there is no fixed method for determining whether application of the program p to an object k will lead to a result or not. As a result, the function K_φ(y/x) cannot be effectively calculated (generally recursive) even if it is known to be finite for all x and y.

Fundamental theorem. There exists a partially recursive function A(p, x) such that for any other partially recursive function φ(p, x) we have the inequality

$$K_A(y/x) \leqslant K_\varphi(y/x) + C_\varphi,$$

where the constant C_φ does not depend on x or y.

The proof is based on the existence of a universal partially recursive function

$$\Phi(n, u),$$

which has the property that by fixing an appropriate index n, we can use the formula

$$\varphi(u) = \Phi(n, u)$$

to obtain any other partially recursive function. The function A(p, x) we require is given by the formula*

$$A\ ((n,\ q),\ x) = \Phi\ (n,\ (q,\ x)).$$

Indeed, if

$$y = \varphi\ (p,\ x) = \Phi\ (n\ (p,\ x)),$$

then

$$A\ ((n,\ p),\ x) = y,$$
$$l\ (n,\ p) \leqslant l\ (p) + C_n.$$

We will call functions A(p, x) that satisfy the requirements of the fundamental theorem (and the programming methods defined by them) asymptotically optimal. It is clear that the corresponding "complexity" $K_A(y/x)$ is finite for all x and y. For two such functions A_1 and A_2

$$|K_{A_1}(y/x) - K_{A_2}(y/x)| \leqslant C_{A_1A_2},$$

where $C_{A_1A_2}$ does not depend on x and y, i.e., $K_{A_1}(y/x) \approx K_{A_2}(y/x)$.

Finally,

$$K_A(y) = K_A(y/1)$$

can be taken for the "complexity of y" and we can define the "quantity of information conveyed by x about y" by the formula

$$I_A(x : y) = K_A(y) - K_A(y/x).$$

It is easy to show** that this quantity is always essentially positive,

$$I_A(x : y) \gtrsim 0,$$

which means that $I_A(x:y)$ is no less than some negative constant C that depends only on the characteristics of the selected programming method. As we have already noted, the theorem was designed for application to a quantity of information so large that, in comparison, |C| is negligibly small.

Note, finally, that $K_A(x/x) \approx 0$, $I_A(x:x) \approx K_A(x)$.

Of course, one can avoid the indeterminacies associated with the constant C_φ, etc., by considering particular domains of the objects X, indexing, and the function A, but it is doubtful that this can be done without explicit arbitrariness. One must, however, suppose that the different "reasonable" variants presented here will lead to "complexity estimates" that will converge on hundreds of bits instead of tens of thousands. Hence, such quantities as the "complexity" of the text of "War and Peace" can be assumed to be defined with what amounts to uniqueness. Experiments on guessing continuations of literary texts make it possible to obtain an upper estimate for the conditional complexity in the presence of a given consumption of "a priori information" (about language, style, textual content) available to the guesser. In tests conducted at the Moscow State University Department of Probability Theory, such upper estimates fluctuated between 0.9 and 1.4. The estimates of the order of 0.9-1.1 obtained by N. G. Rychkov have led less successful guessers to suggest that he telepathically communicated with the authors of the texts.

I believe that the approach proposed here yields, in principle, a correct definition of the "quantity of hereditary information," although it would be difficult to obtain a reliable estimate of this quantity.

4. Conclusion

The concept discussed in §3 have one important disadvantage: They do not allow for the "difficulty" of preparing a program p for passing from an object x to an object y. By introducing appropriate definitions, it is possible to prove rigorously formulated mathematical propositions that can be legitimately interpreted as an indication of the existence of cases in which an object permitting a very simple program, i.e., with a very small complexity K(x), can be restored by short programs only as the result of computations of a thoroughly unreal duration. Sometime in the future, I intend to study the relationship between the necessary complexity of a program

*$\Phi(n,\ u)$ is defined only when $n \in D$, and A(p, x) is defined only when p is of the form (n, q), $n \in D$.

**By choosing a "comparison function" of the form $\varphi(p,\ x) = A(p,\ 1)$, we obtain $K_A(y/x) \leq K_\varphi(y/x) + C_\varphi = K_A(y) + C_\varphi$.

and its permissible difficulty t. The complexity K(x) that was obtained in §3, is, in this case, the minimum of $K^t(x)$ on the removal of constraints on t.

It is beyond the scope of this article to consider the use of the constructions of §3 in providing a new basis for probability theory. Roughly speaking, the situation is as follows: If a finite set M containing a very large number of members N admits determination by means of a program of length negligibly small in comparison with $\log_2 N$, then almost all members of M have complexity K(x) close to $\log_2 N$. The elements $x \in M$ of this complexity are also treated as "random" members of the set M. An incomplete discussion of this idea may be found in [2].

REFERENCES

1. V. A. Uspenskii, Lectures on Computable Functions [in Russian], Fizmatgiz, Moscow, 1960.

2. A. N. Kolmogorov, "On tables of random numbers," Sankhya. The Indian Journal of Statistics, Series A, 25, 4, 369-376, 1963.

9 January 1965

A PRELIMINARY REPORT ON A GENERAL THEORY OF INDUCTIVE INFERENCE

R. J. Solomonoff

Abstract

Some preliminary work is presented on a very general new theory of inductive inference. The extrapolation of an ordered sequence of symbols is implemented by computing the a priori probabilities of various sequences of symbols. The a priori probability of a sequence is obtained by considering a universal Turing machine whose output is the sequence in question. An approximation to the a priori probability is given by the shortest input to the machine that will give the desired output. A more exact formulation is given, and it is made somewhat plausible that extrapolation probabilities obtained will be largely independent of just which universal Turing machine was used, providing that the sequence to be extrapolated has an adequate amount of information in it.

Some examples are worked out to show the application of the method to specific problems. Applications of the method to curve fitting and other continuous problems are discussed to some extent. Some alternative theories of inductive inference are presented whose validities appear to be corollaries of the validity of the first method described.

Contents

PREFACE TO REVISED VERSION

This paper was prepared as a statement of work in progress prior to my trip to the conference "Cerebral Systems and Computers" held at the California Institute of Technology, February 8-11, 1960 and copies were distributed at the conference. Since then, further development has made several parts of the paper obsolete, and continuing work makes it impossible at present to prepare a definitive statement of this line of development. For this reason, and because of a number of requests for the paper, it appears useful to present the original paper together with some of the more important revisions which will be pointed out in this preface.

A. The main point of the report is Equation (5), Section 11. It is clear that this equation, if taken literally, causes difficulties, since, as was shown by Turing, it is not always possible to tell whether a given input to a Turing machine will ever allow the machine to stop, and thereby produce a meaningful output. To overcome this objection, we can limit our discussion to machines that are not "universal machines" in the most general sense, but machines with limited, but large, memory capacities. These correspond to limiting a Turing machine to a tape of finite length. We could then perhaps *approach* Equation (5) of Section 11, by allowing the total memory capacity (or tape length) to *approach* infinity.

B. In Section 7, an expression is given for the *a priori* probability of a Bernoulli sequence of two symbol types. A somewhat better expression can be obtained using more rigorous arguments. The expression obtained corresponds to some extent to one obtained by Carnap and gives probability ratios that are the same as those given by Laplace's rule of succession. For d different symbol types the a priori probability is

$$\frac{(d-1)!\prod_{i=1}^{d} n_i!}{\left(d-1+\sum_{i=1}^{d} n_i\right)!}$$

n_i is the number of times that the ith symbol type appears in the sequence. The probability ratio of the next symbol being the kth symbol type rather than the jth type is

$$\frac{n_k+1}{n_j+1}$$

C. Equation (6), Section 15 is incorrect. A more satisfactory solution to the problem is to calculate

$$
\begin{aligned}
A &\equiv s_1(p_1 \cdot lnp_1 + (1-p_1)\cdot ln(1-p_1)) \\
&\quad +(1-s_1)(r_1 \cdot lnr_1 + (1-r_1)\cdot ln(1-p_2)) \\
B &\equiv s_2(p_2 \cdot lnp_2 + (1-p_2)\cdot ln(1-p_2)) \\
&\quad +(1-s_2)(r_2 \cdot lnr_2 + (l-r_2)\cdot ln(1-r_2))
\end{aligned}
$$

s_1 is the probability that a randomly chosen 50-year-old man will have had pneumonia.

r_1 is the probability that if a 50-year-old man has not had pneumonia, he will live to 60.

s_2 is the probability that both parents of a randomly chosen 50-year-old man will have lived more than 90 years.

r_2 is the probability that if one or both parents of a 50-year-old man died before they were 90, then the 50-year-old man will live to 60. If $A > B$ then the probability that the man in question will live to 60 is p_1. If $B > A$ the probability is p_2.

This discussion is for cases in which the probability values used have been obtained from large samples. If the samples are small, and/or A is very close to B, then the final probability obtained involves the sample sizes of the various kinds of data used, as well as the costs of describing the various statistical categories involved.

It will be noted that we required r_1, r_2, s_1, and s_2 - four pieces of data - rather than the single probability that a 50-year-old man, both of whose parents lived to over 90, will live to 60. However, these four pieces of data are usually more readily obtained, or approximated, than the one, otherwise critical, piece of data.

D. An informal proof has been obtained, that for any finite state Markov process, Equation (5) of Section 11 approaches the correct probability values, arbitrarily closely, if a long sample sequence is used.

R.J.Solomonoff

November 30, 1960

PREFACE TO ORIGINAL VERSION

(Report V-131, February 1960)

This memo is an outline of some preliminary work on a completely general theory of inductive inference, for universes containing continuous, discontinuous, numerical and non-numerical objects.

The most important previous attempts to obtain a unified theory have been those of R. A. Fisher and of R. Carnap. It is felt that there is a good possibility that the method outlined here overcomes some of the serious shortcomings of the methods of Fisher and of Carnap.

The final statement of the present method is Equation (5) of Section 11. The rest of the memo deals with successive approximations leading to Equation (5), and some outlines of applications of Equation (5) to specific problems.

Although the gross approximations used to obtain some of the results of the application of Equation (5) lead the author to have incomplete confidence in them, it is felt that Equation (5) itself is fairly likely to be correct.

The specific inductive inference problem dealt with is the extrapolation of an ordered sequence of discrete symbols. The methods may, however, be used to extrapolate unordered sets of objects. In order to deal with continuous data, any consistent method of converting from continuous to digital symbolism may be used, and then the regular method can be used with the digital symbols.

The method described is used only for the extrapolation of sequences of symbols. If predictions about objects in the real world are desired, one must devise some method of making a correspondence between the symbol sequences and events in the world. It is believed that using the present extrapolation method on the symbol sequences will result in probability values that correspond to those in the real world, and that the probability values obtained for real-world events in this way will be largely independent of the nature of the correspondence that is devised between the symbols and the events they represent - just as long as the correspondence is not "unreasonably complicated," is used consistently, and does not lose too much information.

February 4, 1960

A PRELIMINARY REPORT ON A GENERAL THEORY OF INDUCTIVE INFERENCE

R. J. Solomonoff

1 INTRODUCTION

We shall be concerned primarily with the problem of extrapolation of a very general time series, whose members may be numbers or non- numerical objects, or mixtures of these. At first, a fairly simple extrapolation formula will be given. Its shortcomings will be discussed, and it will be progressively improved upon, until a final formula that seems to overcome all of these difficulties will be presented.

Consider a very long sequence of symbols — e.g,, a passage of English text, or a long mathematical derivation. We shall consider such a sequence of symbols to be "simple" and have high a priori probability, if there exists a very brief description of this sequence — using, of course, some sort of stipulated description method. More exactly, if we use only the symbols 0 and 1 to express our description, we will assign the probability 2^{-N} to a sequence of symbols, if its shortest possible binary description contains N digits.

2 THE CONCEPT OF "BINARY DESCRIPTION"

Suppose that we have a general purpose digital computer M_1 with a very large memory. (Later we shall consider Turing machines — essentially computers having infinitely large memories.)

Any finite string of 0's and 1's is an acceptable input to M_1. The output of M_1 (when it has an output) will be a (usually different) string of symbols, usually in an alphabet other than the binary. If the input string S to machine M_1 gives output string T, we shall write

$$M_1(S) = T$$

Under these conditions, we will say that "S is a description of T with respect to machine M_1." If S is the shortest such description of T, and S contains N digits, then we will assign to the string, T, the a priori probability, 2^{-N}.

3 THE FIRST APPROXIMATE EQUATION

Let us apply this a priori probability to time series extrapolation. Suppose that T is a string of symbols that constitutes a time series. We want to know the relative probability that the next symbol in the series will be the symbol "a" rather than the symbol "b".

Let $T \frown a$ represent the string of symbols that is T concatenated with the symbol a.

Let $T \frown b$ be similarly defined.

Let S_a be the shortest description of $T \frown a$, with respect to machine M_1.

Let S_b be the correspondingly minimal description for $T \frown b$.

Let N_{S_a} be the number of digits in S_a.

Let N_{S_b} be the number of digits in S_b.

Then the relative probability of a, rather than b, as continuation of the sequence T, will be, with respect to machine M_1,

$$2^{-N_{S_a}} + N_{S_b} \tag{1}$$

which is the ratio of the a priori probabilities of $T \frown a$ and $T \frown b$.

4 FIRST OBJECTION: THAT EQUATION (1) IS MACHINE DEPENDENT

There are several very serious objections that immediately come to mind. First, it is quite clear that N_{S_a} and N_{S_b} will depend very much upon just what machine is selected — in fact, by properly selecting machines, we can give $N_{S_b} - N_{S_a}$ any value we like.

We will later (in Section 9) try to make it plausible that if T is a very long sequence of symbols that contains all of the kinds of data that a man is likely to observe in his lifetime, then $N_{S_b} - N_{S_a}$ will be machine independent over a rather large, "natural" set of machines.

5 SECOND OBJECTION: THAT THE PROBABILITIES OF EQUATION (1) DO NOT CONVERGE

Another objection is that if we assign a priori probability 2^{-N} to a binary string of length N, then the total a priori probability of all binary strings does not converge — i.e., there are 2 strings for $N = 1$; their individual probabilities are $1/2$ each, their total probability is 1. There are 4 strings for $N = 2$, their total probability is 1 also. Similarly, the total probability of all strings of length N

will be 1, for *any* value of N. Clearly the sum of all these probabilities does not converge.

We can, however, think of the binary descriptions as being formed by a simple Markov process. The digit 0 is produced with probability 1/2. The digit 1 is also produced with probability 1/2. Clearly such a Markov chain has no means to terminate. It must be of infinite length.

We can remedy this difficulty in a very natural way by giving the digits 0 and 1, each probability $1/2 - 1/2\epsilon$ and have the probability of termination of the string be ϵ. Since we will deal only with very long descriptions, ϵ will be very small. Using the ϵ formalism, we find that though the total a priori probability of all sequences, does indeed converge, our prediction probabilities have not changed much. Instead of

$$2^{-N_{S_a}} + N_{S_b}$$

we now write

$$[1/2(1-\epsilon)]^{N_{S_a}} - N_{S_b}$$

Since ϵ is much less than 1,

$$(1-\epsilon)^{N_{S_a}} - N_{S_b} \approx 1$$

and

$$[1/2(1-\epsilon)]^{N_{S_a}} - N_{S_b}$$

is very close to

$$2^{N_{S_a}} + N_{S_b}$$

It is clear that the expected length of a description is about $1/\epsilon$

6 THIRD OBJECTION: THAT ALL THE PROBABILITY RATIOS OF EQUATION (1) ARE INTEGRAL POWERS OF TWO

Another objection that comes immediately to mind is that $N_{S_b} - N_{S_a}$ must always be an integer, and so the relative probabilities of the two possible continuations of the sequence would have to be integral powers of 2 — certainly this is not a realistic restriction, since, in general, probabilities may have *any* values between zero and one.

We will overcome this difficulty by three different devices. The first is somewhat ad hoc, and will be discussed immediately. The second will overcome

another difficulty in addition to the present one, and will be discussed in Section 11. These two methods do not interfere with one another. A third method is discussed in Section 13.

It will be noted that the present difficulty seems associated with the use of just two symbol types in our description strings. If we used more than two types of symbols (i.e., N types) there would be even more trouble, since the probability ratios would then be restricted to integral powers of N — an even coarser gradation than integral powers of 2.

An apparently direct source of trouble is that if an integral number of symbol types is used, there is usually some "wastage of bits" in expressing integers. For example, to express the integer 7 in binary notation, we use 3 bits in the sequence 111. However, to express the integer 8, 4 digits are needed, i.e., 1000. It seems unlikely that 8, which is only 14% larger than 7, should require a whole extra bit. Also the numbers 9 through 15 all require only 4 bits.

Much "bit wastage" can be avoided if we allow a "cost" of just log_2 bits for the number n, if n occurs in a context in which the value zero would be meaningless. If zero is meaningful, a cost of $log_2(n+1)$ should be assigned to the number n.

In the previous paragraph, and in the following example, it will seem as though the means used for representing numbers in descriptions are rather arbitrary. It can, however, be made plausible that the probability ratios obtained using these rather "arbitrary methods" are identical with the ratios obtained using the more intuitively reasonable Equation (5) of Section 11.

7 A SIMPLE EXAMPLE OF INDUCTION

A very simple example is afforded by a sequence of a A's and b B's. The letters A and B occur in arbitrary order. We are then asked "What is the relative likelihood of an A rather than a B following this sequence?"

To describe the sequence of a A's and b B's, we first note that there are just $(a+b)!/a!b!$ different sequences containing just a A's and b B's. A complete description of the sequence would then be given by the string $RABabk$. k tells which of the $(a+b)!/a!b!$ different orderings of the symbols A and B actually occurred and

$$1 \leq k \leq \frac{(a+b)!}{a!\,b!}$$

R tells the computer just what sort of notation is being used. In general, there will be several different symbols of this type.

To compute the bit cost of this description, we would have to know how A, B and R are to be represented in our system. Suppose A costs C_A bits, and B costs C_B bits and R costs C_R bits (C_A, C_B and C_R are all irrelevant to the final probability ratio to be computed).

The numbers a and b cost $log_2 a$ and $log_2 b$ bits, respectively.

k will cost $log_2[(a+b)!/a!b!]$ bits.

The cost of k seems a bit arbitrary - should it not be $log_2 k$?

First of all, k differs from a and b, in that k has both upper and lower limits. k is a choice between $(a+b)!/a!b!$ alternatives. On the average, k will have about $log_2[(a+b)!/a!b!]$ bits in its binary representation - but this does not justify using a cost of $log_2[(a+b)!/a!b!]$ bits for k, when k does *not* have that many digits in its binary representation.

Again the *true* justification to using this bit cost for k is that it results in the same probability ratios as the more intuitively reasonable Equation (5) of Section 10.

The total bit cost obtained for the description $RABabk$ is $C_A + C_B + C_R + log_2 a + log_2 b + log_2[(a+b)!/a!b!]$. The resultant a priori probability is

$$2^{(-C_A - C_B - C_R)} \frac{(a-1)!(b-1)!}{(a+b)!}$$

Let us now consider the same sequence of A's and B's, to which an additional A has been appended. The resultant sequence will have $a+1$ A's and b B's. Its a priori probability is therefore

$$2^{(-C_A - C_B - C_R)} \frac{a!(b-1)!}{(a+b+1)!} \tag{2}$$

Appending an A has multiplied the a priori probability of the resultant sequence by a factor of

$$\frac{a}{a+b+1}$$

We may view $-log_2(a/(a+b+1))$ as the bit cost of the symbol A, in that particular situation, and we shall call $(a+b+1)/a$ the "raw cost" of the symbol A in that situation.

Similarly, the a priori probability of the sequence after B has been appended is

$$2^{(-C_A - C_B - C_R)} \frac{(a-1)!b!}{(a+b+1)!} \tag{3}$$

The bit cost of the appended B is $-log_2(b/(a+b+1))$ and the raw cost of B was $(a+b+1)/b$.

The relative probability of A rather than B following the original sequence of a A's and b B's is the ratio of the a priori probabilities in expression (2) and expression (3) - This is

$$\frac{2^{(-C_A - C_B - C_R)}}{2^{(-C_A - C_B - C_R)}} \frac{a!(b-1)!/(a+b+l)!}{(a-1)!b!/(a+b+l)!} = \frac{a}{b}$$

which is approximately what is expected. Note also that

$$\frac{a}{b} = \frac{\text{raw cost of} B}{\text{raw cost of} A}$$

a relationship which continues to be true when suitably generalized.

This simple result, which gives the frequency ratio of 2 kinds of events as an estimate of their probability ratio, is called, in inductive inference circles, "The Straight Rule." An important objection to it is that if $a = 2$ and $b = 0$, then it tells us that we have a probability of 1 for the next symbol being A. This seems intuitively unreasonable, since we would certainly not be absolutely certain of the next symbol after so short a sequence.

"Laplace's rule" gives the value $(a + 1)/(b + 1)$.

Carnap (Ref. 1, page 568) gives $(a+k_1)/(b+k_2)$, with the values of k_1 and k_2 dependent upon the exact nature of the properties whose relative frequency one is measuring. If we consider a universe in which very many properties exist, k_1 and k_2 become quite large, and the probability ratio obtained becomes almost independent of empirical data, unless the amount of empirical data is very large.

A more detailed analysis reveals that Equation (1) (as modified by the considerations of Section 6) does *not* give the objectionable ratio a/b, for small values of a or b. This is true because, under these circumstances, the code $RABabk$ is *not* a minimal code. It is more economical to write the sequence itself than to use the "R" method.

let us use the symbol V to denote the identity code, so that if we use the sequence $VABBA$ as input to machine M_1, its output would be $ABBA$. Symbolically,

$$M_1(VABBA) = ABBA$$

or, more generally,

$$M_1(V \frown X) \equiv X$$

for any sequence X.

The cost of coding AB using the "V" method is $C_A + C_B + C_V$

The cost of coding AB using the "R" method is

$$C_A + C_B + C_R + log_2(\frac{2!}{(1-l)!(1-1)!}) = C_A + C_B + C_R + 1$$

The "V" coding method will be more economical than the "R" coding method in this case if

$$C_V < C_R + 1.$$

In general, the raw cost of a symbol type (e.g., R or V) will be about equal to the reciprocal of its relative frequency of use in the previous part of the code.

As a result, the V notation will be used here if, in the past, the V notation has been used more than 1/2 as often as the R notation. If short strings of random symbols have occurred quite often in the sequence to be described, then the V notation will be used very often, and will have a low bit cost.

If V has a very low bit cost, then if we want to extrapolate the sequence AAB, the cost of $AABB$ is $C_V+2C_A+2C_B$ the cost of $AABA$ is $C_V+3C_A+C_B$. The relative probabilities of A and B following will then be

$$2^{(-C_A+C_B)} = 2^{C_B}/2^{C_A}$$

This will be about equal to the ratio of the frequency of occurrence of the symbol A and the symbol B in the sequence preceding the subsequence AAB. If A and B have never occurred before, we might obtain the ratio 1, or if the symbols A and B have other structural features, we might obtain some other ratio corresponding to Carnap's k_1/k_2.

However, if the present sequence is quite long, e.g., $ABBABABAAABAA$, then the R notation is likely to cost less than the V notation, and the computed relative probabilities of A and B following will be independent of their frequencies in the part of the sequence preceding the part under present consideration.

In Section 10, an improved inductive inference method will be described, in which *all* possible methods of describing a sequence contribute to its probability rather than just the "minimal method" of description. Using this method, the probability ratio $(a + k_1)/(b + k_2)$ appears to be approximately correct. The values of k_1 and k_2 are, however, not the same as those of Carnap.

8 CODING AND RECODING

The method used in coding a sequence is to first write a code description of it, using any convenient symbols.

This description will sometimes contain the R and V symbols of Section 7, a space symbol, and various letters and numbers. The numbers are recoded by special methods that take advantage of either the fact that the range of possible values of the number is known, or else that the first digit of the number must be 1, the second digit is more probably a zero than a 1, and so on.

The R V, and space symbols are recoded using the R notation,

If any regularities are found in the resultant code sequence, it is recoded again in a manner that takes advantage of these regularities. The final "minimal" code for a sequence will contain about an equal number of zeros and ones, and will display no "significant" statistical regularities *at all*.

9 REPLY TO THE FIRST OBJECTION

At this point the reader may note that the original premises have apparently been discarded entirely - that while the original idea was to devise a minimal description for a sequence using an *arbitrarily* chosen machine, we have instead made a description for a special machine that must be *very narrowly* specialized to interpret that description!

To answer this criticism it will be necessary to modify the premises a bit. Let us designate by S, the sequence consisting of a A's and b B's. Unless the sequence S is *very* tong, the present methods are not very useful for extrapolating S alone, However, let us define S' to be a very long sequence of symbols containing all the kinds of data that a man is likely to observe in his lifetime. It would be well if this man had a broad background in the kinds of material that we will be extrapolating, but this is not absolutely necessary.

The present methods will be useful for extrapolating the sequence $S' \frown S$, Note that S' need not have any material bearing directly on the sequence to be extrapolated. The relationship of S' to S will be seen presently.

We shall try to make it plausible that the last few symbols in the minimal description of the sequence $S' \frown S$ will be largely independent of just what computer is to be used, as long as that computer is a "universal machine" — which is a kind of general purpose computer - also, that these last few symbols will probably be $RABabk$, or equivalent symbols having the same bit costs.

First the concept of "universal machine" will be defined, A "universal machine" is a sub–class of universal Turing machines that can simulate any other Turing machine in a certain way.

More exactly, suppose M_2 is an arbitrary Turing machine, and $M_2(x)$ is the output of M_2, for input string x. Then if M_1 is a "universal machine," there exists some string, α (which is a function of M_1 and M_2, but not of x), such that for any string, x,

$$M_1(\alpha \frown x) = M_2(x)$$

α may be viewed as the "translation instructions" from M_2 to M_1.

Let us suppose that M_2 is a machine that is able to perform the decoding from the code string $RABabk$, to the sequence S, so that

$$M_2(RABabk) = S$$

Suppose that M_1, a universal machine, has some other method of coding the sequence S, so that

$$M_1(D) = S$$

and that the sequence D is longer (has more bits) than the sequence $RABabk$. Furthermore, let us suppose that the sequence S' contains many subsequences

similar to S, in the sense that the same kind of coding method would apply. Let us assume that the $RABabk$ method of coding used by M_2 is, on the average, better than that used by M_1, so that on the average, it costs M_1 3 more bits than M_2 to code a sequence like S. if M_2's coding method is in any sense "optimum" (the method described is, indeed, close to optimum), then the assumptions mentioned are reasonable.

If S' contains 1000 sequences of "type S," then M_1 will take 3000 more bits to code this part of S' than will M_2. Let

$$M_2(E \frown RABabk) = S' \frown S$$

and

$$M_1(F) = S' \frown S$$

be the normal methods of coding for M_1 and M_2. Then

$$M_1(\alpha \frown E \frown RABabk) = S' \frown S$$

and if the string α contains less than 3003 bits, the code string $\alpha \frown E \frown RABabk$ will be *shorter* than the code F, so the "minimal" codes for both M_1 and M_2 will terminate in the sequence $RABabk$.

The figure "3003 bits" was arbitrary. In general, α will have a fixed number of bits, but the figure "3003" will be proportional to the length of the sequence $S' \frown S$. As a result, *all* universal machines will tend to code long sequences ending in S by code sequences ending in $RABabk$, because coding methods of this type will be shortest in the long run.

It will be noted that this latter statement on the similarity of minimal codes for universal machines is not much more than a strong conjecture, with suggestions of how a proof might, under certain circumstances, be constructed.

More exactly, if S' is a very long sequence of a kind containing the kinds of information that a man would normally observe in his lifetime and

S is a short sequence.

M_1 and M_2 are both universal machines.

G_1, G_2, H_1 and H_2 are the shortest strings such that

$$\begin{aligned} M_1(G_1) &= S', & \qquad M_2(G_2) &= S', \\ M_1(H_1) &= S' \frown S, & \qquad M_2(H_2) &= S' \frown S. \end{aligned}$$

N_{G_1} is the number of bits in G_1, with similar definitions for N_{G_2} etc.

Then we would like it to be true that

$$N_{H_1} - N_{G_1} = N_{H_2} - N_{G_2}$$

for all fairly short sequences, S, and all pairs of universal machines, M_1 and M_2.

The truth of this conjecture is a sufficient condition for the probability estimate of Equation (1) to be independent of just what machine was used (providing, of course, that it was a "universal machine").

10 A FOURTH OBJECTION: THAT EQUATION (1) CONSIDERS ONLY "MINIMAL" DESCRIPTIONS

Another objection to the method outlined is that Equation (1) uses only the "minimal binary descriptions" of the sequences it analyzes. It would seem that if there are several different methods of describing a sequence, each of these methods should be given *some* weight in determining the probability of that sequence.

In accordance with this idea. we will modify Equation (1) and write the probability that a, rather than b, will be the continuation of sequence T,

$$\lim_{\epsilon \to 0} \frac{\sum_{i=1}^{\infty} \left(\frac{1-\epsilon}{2}\right)^{N_{S_{ai}}}}{\sum_{i=1}^{\infty} \left(\frac{1-\epsilon}{2}\right)^{N_{S_{bi}}}} \tag{4}$$

$$M_1(S_{a1}) = M_1(S_{a2}) = M_1(S_{a3}) = ... = M_1(S_{a\infty}) = T \frown a$$

The S_{ai} are all the descriptions of $T \frown a$. Similarly,

$$M_1(S_{b1}) = M_1(S_{b2}) = M_1(S_{b3}) = ... = M_1(S_{b\infty}) = T \frown b$$

$N_{S_{ai}}$ is the number of digits in S_{ai}.

The limit $\epsilon \to 0$ has been incorporated into the equation to overcome the objection in Section 5, that the sum of all the probabilities diverged. In Equation (4) it may not be necessary for ϵ to approach zero. It may be both expedient and adequate to let it take some small value like 0.001.

11 LAST OBJECTION: THAT THE MORE DISTANT FUTURE OF THE SEQUENCE SHOULD BE CONSIDERED

The final objection that we will discuss at any length is that Equation (4) does not consider in any serious way the more distant future of the sequence being

extrapolated. Consider, for example, the sequence *abcdabcdabcdab*. The next symbol is probably *c*, and this is so because the sequence *abcdabcdabcdabcd* has a particularly simple description, and is therefore very probable.

We take all possible future continuations of the sequence into account in the following further refinement of Equation (4):

$$\lim_{\epsilon \to 0} \lim_{r \to \infty} \frac{\sum_{k=1}^{r^n} \sum_{i=1}^{\infty} \left(\frac{1-\epsilon}{2}\right)^{N_{(S_{TaC_{n,k}})_i}}}{\sum_{k=1}^{r^n} \sum_{i=1}^{\infty} \left(\frac{1-\epsilon}{2}\right)^{N_{(S_{TbC_{n,k}})_i}}} \tag{5}$$

$C_{n,k}$ is a sequence of a symbols in the *output* alphabet of the universal machine. There are r different symbols so there are rn different sequences of this type. $C_{n,k}$ is the kth such sequence. k may have any value from 1 to r^n.

$TaC_{n,k}$ is the same as $T \frown a \frown C_{n,k}$.

$(S_{TaC_{n,k}})i$ is the ith description of $TaC_{n,k}$ with respect to Machine M_1.

$N_{\left(S_{TaC_{n,k}}\right)_i}$ is the number of digits in $(S_{TaC_{n,k}})_i$.

It can be shown that Equation (5) also eliminates the Third Objection in a very satisfactory way - i.e., the "bit wastage" in both numerator and denominator average out to be the same, and so they cancel. This cancellation does not ordinarily occur in Equation (4).

12 AN INTERPRETATION OF EQUATION (5)

Equation (5) has at least one rather simple interpretation. Consider all possible sequences of symbols that could be descriptions of all the things a person might observe in his life. These sequences correspond to the sequences being coded in Equation (5), such as $TaC_{n,k}$.

Then a *complete model* that "explains" all regularities observed in these sequences is that they were produced by some arbitrary universal machine with a random binary sequence as its input. Equation (5) then enables us to use this model to obtain a priori probabilities to be used in computation of a posteriori probabilities using Bayes' Theorem. Equation (5) finds the probability of a particular sequence by summing the probabilities of all possible ways in which that sequence might have been created.

This particular model of induction is somewhat similar to that of Carnap (Ref. 1, page 562). Carnap restricts his discussions to only the simplest finite languages, yet be is able to obtain some very reasonable results with this very limited means.

Here, however, we use the full generality of description methods that are available through Turing machines.

A somewhat more general, and equally "complete" model may also be obtained if we allow the input to the Turing machine to be any Markov chain of nonvanishing entropy.

13 USE OF A SKEW INPUT DISTRIBUTION TO OVERCOME THE THIRD OBJECTION

The above model for Equation (5) suggests a very natural way to avoid the "bit wastage" inherent in the representation of numbers using any integral radix.

For Equation (5) we used as input to the universal machine, binary sequences in which 0's and 1's were equally probable. In such a situation, the probability of any particular input sequence was always a power of 2. However, suppose that we use the following type of input sequence for the machine:

probability of 0 is $\delta - 1/2\epsilon$

probability of 1 is $1 - \delta - 1/2\epsilon$

probability of termination of sequence is ϵ

Here again, the "expected length" of a sequence is about $1/\epsilon$. If δ is small, however, we can have very fine gradations of probability available in these sequences — much finer than the integral powers of 2.

It will be noted that the descriptions (i.e., input sequences to the machine) of a given output sequence that are "most probable" are now entirely different from the shortest (and therefore most probable) sequences that were used before for "minimal" descriptions. There exists, however, a translation method, so that it is possible to go from a "shortest" description using equal probabilities for 0 and 1, to a corresponding "most probable" description using the highly skewed distribution.

Using this highly skewed distribution, it is possible to devise sequences that correspond to any integers with arbitrarily little of the "bit wastage" that was evident when an integral radix was used for representation of numbers. In general, the lengths of sequences of highest probability in the skew distribution that are needed to code a given text will be much longer than the corresponding code sequences using a symmetric distribution.

14 APPLICATION OF THE METHOD TO CURVE FITTING

The application of Equation (4) to numerical extrapolation by means of "curve fitting" has been investigated to some extent. The problem is formulated in the following way: We are given a set of pairs of numbers that correspond to empirically observed data points — e. g., a set of pairs of temperature and pressure readings of a gas. We are then required to extrapolate this data —

i. e., given a new temperature reading, to obtain the relative probability of any possible corresponding pressure reading.

An economical method of describing such a set of data points is to give an equation that approximates the data, then give a set of temperatures, then a set of numbers that give the deviations of the empirical pressures from the equation.

We could conceivably try to express the list of temperatures in more compact form, but doing so would not affect the resultant probability ratios.

If the curve fits very well, the cost of the set of deviations will be smaller than for a curve that fits poorly. The cost of describing the equation must also be taken into account, so, in general, a 20–parameter polynomial could give a low cost set of deviations for 20 empirical points, but the cost of the 20 parameters would be high. There will exist some optimum number of parameters that should be used, such that the total cost of the equation description and the deviation descriptions will be minimal. Here we make use of the fact that it costs less to code small numbers than large numbers of the same absolute accuracy.

If using polynomials for curve fitting has been useful in the past, this method of description will have a low bit cost. Using unusual functions that have few parameters in them, yet are complex to describe and have been used infrequently in the past, will be very expensive to use fur extrapolation, so one would tend not to use them unless they gave a very small set of deviations.

These latter notions are certainly what one feels to be true intuitively when one is fitting a curve to empirical data. The present method of analysis seems to put this intuitive idea on a quantitative basis.

An objection might be raised that the curve fitting method described is close to one that assumes a very un-normal distribution of empirical error — certainly a distribution quite different from that which is ordinarily observed.

If, however, in the sequence of data preceding the present problem, there have been many empirical situations in which the deviations had a normal distribution, or if there are enough empirical points in the present problem. then it will be less expensive to describe the deviations *as a normal distribution* than to simply list them. As a result, we would obtain something close to a mean-square-goodness-of-fit criterion — with the added feature of taking the complexity of the curve used into account.

If the empirical data obtained corresponds to a known physical law, then there will be much previous data to corroborate this law. In such a case, the equation will have been used many times in the past, and will be correspondingly less expensive to use in the present case.

If the physical law used has not been personally empirically verified by the curve fitter through previous experimentation, and he simply read about the law in a book, then the cost of the equation is somewhat more difficult to compute. It depends, in part, upon the empirical accuracy in the life of the curve-fitter of physical laws that he has read about in books.

15 THE PROBLEM OF CONFLICTING LINES OF EVIDENCE

An insurance company wants to determine the probability that a man will live over 60 years, and has compiled tables of data to aid in solving this problem.

One day a man drifts into the office of the company and asks to be insured. He is 50 years old, has had pneumonia, and both of his parents died at the age of 95.

The insurance company has tables that tell the probability that a 50-year-old man who has had pneumonia will live to 60. The tables give the probability p_1.

They have tables that tell the probability that a 50-year-old man, both of whose parents lived to be over 90, will live to 60. The tables give the probability p_2.

They have no tables for 50-year-old-men who have had pneumonia and both of whose parents lived more than 90 years.

How shall the company combine the data from the two tables that it has?

It might be argued that it is impossible — that one *must* have a table for the coincidence of the three characteristics before one can make a probability estimate. However, every day we are forced to combine evidence of various kinds to make probability estimates, and in many cases the data is inadequate, as in the above problem — yet we make decisions based on such inadequate data. Indeed, it might be argued that there are few decisions that we do make in which we have "adequate data."

A very approximate analysis of this problem was made, using the coding method of probability evaluation. The probability obtained that the man will live more than 60 years is[1]

$$\frac{p_1 + p_2 + \dfrac{p(V)}{p(R)}}{(1-p_1) + (1-p_2) + \dfrac{p(V)}{p(R)}} \tag{6}$$

Speaking very loosely, $p(V)$ and $p(R)$ are the relative frequencies with which the R coding method and the V coding method of Section 7 have been used in the past. In the present case, $p(V)/p(R)$ is probably much less than 1.

It is characteristic of the present method of induction that most probability values obtained are dependent, to some extent, on sequences of events that are apparently not very closely related to the events whose probabilities are being computed.

It should be noted that the validity of Equation (6) is not very certain, since it was obtained by using some very uncertain assumptions. These un-

[1]This equation is incorrect. See page viii, paragraph C, in the Preface to the Revised Version for a more satisfactory solution to the problem.

certain assumptions need not characterize the method and are symptomatic of the author's present inability to always devise good approximation methods for Equation (5).

16 GENERAL REMARKS ON EQUATION (5) AND ITS APPLICATIONS

While Equation (5) is put forth as what is hoped to be an adequate computation of conditional relative probability, the equation itself will not ordinarily be used directly for probability computation - any more than the definition of a Lebesgue integral is used directly in the numerical evaluation of integrals.

Instead, Equation (5) can be and has been used to obtain theorems about probability from which actual probabilities may be calculated. Among the techniques used are the discovery of coding methods that are simple to use, and nonminimal, yet from which it is possible to obtain the same probability ratios as Those given by Equation (5). The apparently ad-hoc number manipulation of Sections 6 and 7 is an example of this, though a proof has not been given here.

Minimal coding techniques do have important direct applications, however. One of these is information retrieval. The minimal coding enables us to discard information that is least relevant to prediction, or to whatever application the coded information might have. Coded information that is most valuable for prediction is also most likely to be correlated with other data, and for this reason, in coding new data, we examine relationships between it and parts of previously coded data that are of most value in prediction.

Another direct application of minimal coding is in the generalized hill-climbing problem. Here, there is a set of continuous and/or discrete parameters that must be adjusted to maximize tine value of a certain evaluation function. Organic evolution is an important example of a hill-climbing problem with discrete parameters. These parameters are the coded sequences that constitute the chromosomes. The evaluation function of such a set of coded sequences is the expected reproduction rate of the resultant organism.

The method used for hill-climbing in organic evolution of asexual organisms is to make each new set of trial parameters a random change of a few of the parameters of a fairly good organism. This random change corresponds to a mutation.

While there is some reason to believe that the genetic code description of the organism is not a minimal code, it shares with minimal codes the property that a random change of one of the code symbols will yield a code sequence for an organism that has a not-altogether-too-small probability of living and a somewhat smaller probability of being a bit better than his parent.

None of the computing machine simulations of organic evolution have attempted representations of organisms using minimal codes, and it seems like a reasonably good thing to try.

17 REFERENCE

1. R. Carnap, *Logical Foundations of Probability*, University of Chicago Press, 1950.

ON THE LENGTH OF PROGRAMS FOR COMPUTING FINITE BINARY SEQUENCES

Journal of the ACM 13 (1966), pp. 547–569

Gregory J. Chaitin[1]
The City College of the City University of New York
New York, N.Y.

Abstract

The use of Turing machines for calculating finite binary sequences is studied from the point of view of information theory and the theory of recursive functions. Various results are obtained concerning the number of instructions in programs. A modified form of Turing machine is studied from the same point of view. An application to the problem of defining a patternless sequence is proposed in terms of the concepts here

developed.

Introduction

In this paper the Turing machine is regarded as a general purpose computer and some practical questions are asked about programming it. Given an arbitrary finite binary sequence, what is the length of the shortest program for calculating it? What are the properties of those binary sequences of a given length which require the longest programs? Do most of the binary sequences of a given length require programs of about the same length?

The questions posed above are answered in Part 1. In the course of answering them, the logical design of the Turing machine is examined as to redundancies, and it is found that it is possible to increase the efficiency of the Turing machine as a computing instrument without a major alteration in the philosophy of its logical design. Also, the following question raised by C. E. Shannon [1] is partially answered: What effect does the number of different symbols that a Turing machine can write on its tape have on the length of the program required for a given calculation?

In Part 2 a major alteration in the logical design of the Turing machine is introduced, and then all the questions about the lengths of programs which had previously been asked about the first computer are asked again. The change in the logical design may be described in the following terms: Programs for Turing machines may have transfers from any part of the program to any other part, but in the programs for the Turing machines which are considered in Part 2 there is a fixed upper bound on the length of transfers.

Part 3 deals with the somewhat philosophical problem of defining a random or patternless binary sequence. The following definition is proposed: Patternless finite binary sequences of a given length are sequences which in order to be computed require programs of approximately the same length as the longest programs required to compute

[1]This paper was written in part with the help of NSF Undergraduate Research Participation Grant GY-161.

any binary sequences of that given length. Previous work along these lines and its relationship to the present proposal are discussed briefly.

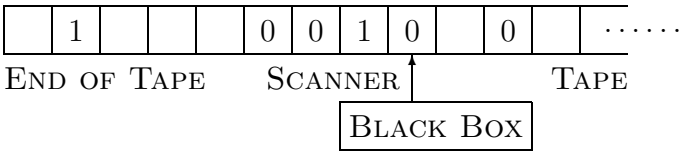
Part 1

1.1. We define an N-state M-tape-symbol Turing machine by an N-row by M-column table. Each of the NM places in this table must have an entry consisting of an ordered pair (i, j) of natural numbers, where i goes from 0 to N and j goes from 1 to $M + 2$. These entries constitute, when specified, the program of the N-state M-tape-symbol Turing machine. They are to be interpreted as follows: An entry (i, j) in the kth row and the pth column of the table means that when the machine is in its kth state and the square of its one-way infinite tape which is being scanned is marked with the pth symbol, then the machine is to go to its ith state if $i \neq 0$ (the machine is to halt if $i = 0$) after performing the operation of

1. moving the tape one square to the right if $j = M + 2$,
2. moving the tape one square to the left if $j = M + 1$, and
3. marking (overprinting) the square of the tape being scanned with the jth symbol if $1 \leq j \leq M$.

Special names are given to the first, second and third symbols. They are, respectively, the blank (for unmarked square), 0 and 1.

A Turing machine may be represented schematically as follows:



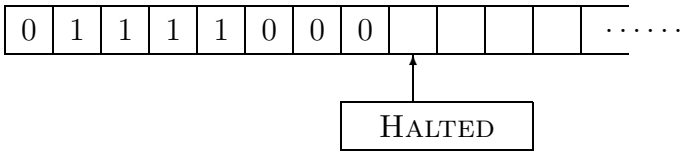
It is stipulated that

(1.1A) Initially the machine is in its first state and scanning the first square of the tape.

(1.1B) No Turing machine may in the course of a calculation scan the end square of the tape and then move the tape one square to the right.

(1.1C) Initially all squares of the tape are blank.

Since throughout this paper we shall be concerned with computing finite binary sequences, when we say that a Turing machine calculates a particular finite binary sequence (say, 01111000), we shall mean that the machine stops with the sequence written at the end of the tape, with all other squares of the tape blank and with its scanner on the first blank square of the tape. For example, the following Turing machine has just calculated the sequence mentioned above:



1.2. There are exactly

$$((N+1)(M+2))^{NM}$$

possible programs for an N-state M-tape-symbol Turing machine. Thus to specify a single one of these programs requires

$$\log_2(((N+1)(M+2))^{NM})$$

bits of information, which is asymptotic to $NM\log_2 N$ bits for M fixed and N large. Therefore a program for an N-state M-tape-symbol Turing machine (considering M to be fixed and N to be large) can be regarded as consisting of about $NM\log_2 N$ bits of information. It may be, however, that in view of the fact that different programs may cause the machine to behave in exactly the same way, a substantial portion of the information necessary to specify a program is redundant in its specification of the behavior of the machine. This in fact turns out to be the case. It will be shown in what follows that for M fixed and

N large at least $1/M$ of the bits of information of a program are redundant. Later we shall be in a position to ask to what extent the remaining portion of $(1 - 1/M)$ of the bits is redundant.

The basic reason for this redundancy is that any renumbering of the rows of the table (this amounts to a renaming of the states of the machine) in no way changes the behavior that a given program will cause the machine to have. Thus the states can be named in a manner determined by the sequencing of the program, and this makes possible the omission of state numbers from the program. This idea is by no means new. It may be seen in most computers with random access memories. In these computers the address of the next instruction to be executed is usually 1 more than the address of the current instruction, and this makes it generally unnecessary to use memory space in order to give the address of the next instruction to be executed. Since we are not concerned with the practical engineering feasibility of a logical design, we can take this idea a step farther.

1.3. In the presentation of the redesigned Turing machine let us begin with an example of the manner in which one can take a program for a Turing machine and reorder its rows (rename its states) until it is in the format of the redesigned machine. In the process, several row numbers in the program are removed and replaced by + or ++ —this is how redundant information in the program is removed. The "operation codes" (which are 1 for "print blank," 2 for "print zero," 3 for "print one," 4 for "shift tape left" and 5 for "shift tape right") are omitted from the program; every time the rows are reordered, the op-codes are just carried along. The program used as an example is as follows:

row 1	1	9	7
row 2	8	8	8
row 3	9	6	1
row 4	3	2	0
row 5	7	7	8
row 6	6	5	4
row 7	8	6	9
row 8	9	8	1
row 9	9	1	8

To prevent confusion later, letters instead of numbers are used in

the program:

row A	A	I	G
row B	H	H	H
row C	I	F	A
row D	C	B	J
row E	G	G	H
row F	F	E	D
row G	H	F	I
row H	I	H	A
row I	I	A	H

Row A is the first row of the table and shall remain so. Replace A by 1 throughout the table:

row 1	1	I	G
row B	H	H	H
row C	I	F	1
row D	C	B	J
row E	G	G	H
row F	F	E	D
row G	H	F	I
row H	I	H	1
row I	I	1	H

To find to which row of the table to assign the number 2, read across the first row of the table until a letter is reached. Having found an I,

1. replace it by a +,
2. move row I so that it becomes the second row of the table, and
3. replace I by 2 throughout the table:

row 1	1	+	G
row 2	2	1	H
row B	H	H	H
row C	2	F	1
row D	C	B	J
row E	G	G	H
row F	F	E	D
row G	H	F	2
row H	2	H	1

To find to which row of the table to assign the number 3, read across the second row of the table until a letter is found. Having found an H,

1. replace it by a +,
2. move row H so that it becomes the third row of the table, and
3. replace H by 3 throughout the table:

row 1	1	+	G
row 2	2	1	+
row 3	2	3	1
row B	3	3	3
row C	2	F	1
row D	C	B	J
row E	G	G	3
row F	F	E	D
row G	3	F	2

To find to which row of the table to assign the number 4, read across the third row of the table until a letter is found. Having failed to find one, read across rows 1, 2 and 3, respectively, until a letter is found. (A letter must be found, for otherwise rows 1, 2 and 3 are the whole program.) Having found a G in row 1,

1. replace it by a ++,
2. move row G so that it becomes the fourth row of the table, and
3. replace G by 4 throughout the table:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	F	2
row B	3	3	3
row C	2	F	1
row D	C	B	J
row E	4	4	3
row F	F	E	D

The next two assignments proceed as in the case of rows 2 and 3:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	E	D
row B	3	3	3
row C	2	5	1
row D	C	B	J
row E	4	4	3

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	+	D
row 6	4	4	3
row B	3	3	3
row C	2	5	1
row D	C	B	J

To find to which row of the table to assign the number 7, read across the sixth row of the table until a letter is found. Having failed to find one, read across rows 1, 2, 3, 4, 5 and 6, respectively, until a letter is found. (A letter must be found, for otherwise rows 1, 2, 3, 4, 5 and 6 are the whole program.) Having found a D in row 5,

1. replace it by a ++,
2. move row D so that it becomes the seventh row of the table, and
3. replace D by 7 throughout the table:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	+	++
row 6	4	4	3
row 7	C	B	J
row B	3	3	3
row C	2	5	1

After three more assignments the following is finally obtained:

row 1	1	+	++
row 2	2	1	+
row 3	2	3	1
row 4	3	+	2
row 5	5	+	++
row 6	4	4	3
row 7	+	++	++
row 8	2	5	1
row 9	3	3	3
row 10			

This example is atypical in several respects: The state order could have needed a more elaborate scrambling (instead of which the row of the table to which a number was assigned always happened to be the last row of the table at the moment), and the fictitious state used for the purposes of halting (state 0 in the formulation of Section 1.1) could have ended up as any one of the rows of the table except the first row (instead of which it ended up as the last row of the table).

The reader will note, however, that 9 row numbers have been eliminated (and replaced by + or ++) in a program of 9 (actual) rows, and that, in general *this process will eliminate a row number from the*

program for each row of the program. Note too that if a program is "linear" (i.e., the machine executes the instruction in storage address 1, then the instruction in storage address 2, then the instruction in storage address 3, etc.), only + will be used; departures from linearity necessitate use of ++.

There follows a description of the redesigned machine. In the formalism of that description the program given above is as follows:

	(1, ,0)	(0, ,1)	(0, ,2)
	(2, ,0)	(1, ,0)	(0, ,1)
	(2, ,0)	(3, ,0)	(1, ,0)
	(3, ,0)	(0, ,1)	(2, ,0)
10	(5, ,0)	(0, ,1)	(0, ,2)
	(4, ,0)	(4, ,0)	(3, ,0)
	(0, ,1)	(0, ,2)	(0, ,2)
	(2, ,0)	(5, ,0)	(1, ,0)
	(3, ,0)	(3, ,0)	(3, ,0)

Here the third member of a triple is the number of +'s, the second member is the op-code, and the first member is the number of the next state of the machine if there are no +'s (if there are +'s, the first member of the triple is 0). The number outside the table is the number of the fictitious row of the program used for the purposes of halting.

We define an N-state M-tape-symbol Turing machine by an $(N+1) \times M$ table and a natural number n $(2 \leq n \leq N+1)$. Each of the $(N+1)M$ places in this table (with the exception of those in the nth row) must have an entry consisting of an ordered triple (i, j, k) of natural numbers, where k is 0, 1 or 2; j goes from 1 to $M+2$; and i goes from 1 to $N+1$ if $k = 0$, $i = 0$ if $k \neq 0$. (Places in the nth row are left blank.) In addition:

(1.3.1) The entries in which $k = 1$ or $k = 2$ are N in number.

Entries are interpreted as follows:

(1.3.2) An entry $(i, j, 0)$ the pth row and the mth column of the table means that when the machine is in the pth state and the square of its one-way infinite tape which is being scanned is marked with the mth symbol, then the machine is to go to its ith state if $i \neq n$

(if $i = n$, the machine is instead to halt) after performing the operation of

1. moving the tape one square to the right if $j = M + 2$,
2. moving the tape one square to the left if $j = M + 1$, and
3. marking (overprinting) the square of the tape being scanned with the jth symbol if $1 \leq j \leq M$.

(1.3.3) An entry $(0, j, 1)$ in the pth row and mth column of the table is to be interpreted in accordance with (1.3.2) as if it were the entry $(p + 1, j, 0)$.

(1.3.4) For an entry $(0, j, 2)$ in the pth row and mth column of the table the machine proceeds as follows:

(1.3.4a) It determines the number p' of entries of the form (0, ,2) in rows of the table preceding the pth row or to the left of the mth column in the pth row.

(1.3.4b) It determines the first $p' + 1$ rows of the table which have no entries of the form (0, ,1) or (0, ,2). Suppose the last of these $p' + 1$ rows is the p''th row of the table.

(1.3.4c) It interprets the entry in accordance with (1.3.2) as if it were the entry $(p'' + 1, j, 0)$.

1.4. In Section 1.2 it was stated that the programs of the N-state M-tape-symbol Turing machines of Section 1.3 require in order to be specified $(1-1/M)$ the number of bits of information required to specify the programs of the N-state M-tape-symbol Turing machines of Section 1.1. (As before, M is regarded to be fixed and N to be large.) This assertion is justified here. In view of (1.3.1), at most

$$N(3(M + 2))^{NM}(N + 1)^{N(M-1)}$$

ways of making entries in the table of an N-state M-tape-symbol Turing machine of Section 1.3 count as programs. Thus only $\log_2$ of this number or asymptotically $N(M - 1) \log_2 N$ bits are required to specify the program of an N-state M-tape-symbol machine of Section 1.3.

Henceforth, in speaking of an N-state M-tape-symbol Turing machine, one of the machines of Section 1.3 will be meant.

1.5. We now define two sets of functions which play a fundamental role in all that follows.

The members $L_M(.)$ of the first set are defined for $M = 3, 4, 5, \ldots$ on the set of all finite binary sequences S as follows: An N-state M-tape-symbol Turing machine can be programmed to calculate S if and only if $N \geq L_M(S)$.

The second set $L_M(C_n)$ $(M = 3, 4, 5, \ldots)$ is defined by

$$L_M(C_n) = \max_S L_M(S),$$

where S is any binary sequence of length n.

Finally, we denote by C_n^M $(M = 3, 4, 5, \ldots)$ the set of all binary sequences S of length n satisfying $L_M(S) = L_M(C_n)$.

1.6. In this section it is shown that for $M = 3, 4, 5, \ldots$,

$$L_M(C_n) \sim \frac{n}{(M-1)\log_2 n}.$$

We first show that $L_M(C_n)$ is greater than a function of n which is asymptotically equal to $(n/((M-1)\log_2 n))$. From Section 1.4 it is clear that there are at most

$$2^{((1+\epsilon_N)N(M-1)\log_2 N)}$$

different programs for an N-state M-tape-symbol Turing machine, where ϵ_x denotes a (not necessarily positive) function of x and possibly other variables which tends to zero as x goes to infinity with any other variables held fixed. Since a different program is required to calculate each of the 2^n different binary sequences of length n, we see that an N-state M-tape-symbol Turing machine can be programmed to calculate any binary sequence of length n only if

$$(1+\epsilon_N)N(M-1)\log_2 N \geq n$$

or

$$N \geq (1+\epsilon_n)\frac{n}{(M-1)\log_2 n}.$$

It follows from the definition of $L_M(C_n)$ that

$$L_M(C_n) \geq (1+\epsilon_n)\frac{n}{(M-1)\log_2 n}.$$

Next we show that $L_M(C_n)$ is less than a function of n which is asymptotically equal to $(n/((M-1)\log_2 n))$. This is done by showing how to construct for any binary sequence S of length not greater than $(1+\epsilon_N)N(M-1)\log_2 N$ a program which causes an N-state M-tape-symbol Turing machine to calculate S. The main idea is illustrated in the case where $M = 3$.

Row Number	Column Number 1	2	3	
1	2, 4	2, 4	2, 4	
2	..., 2	..., 3	3, 4	
3	..., 2	..., 3	4, 4	Section I:
4	..., 2	..., 3	5, 4	approximately
5	..., 2	..., 3	6, 4	$(1-1/\log_2 N)N$
6	..., 2	..., 3	7, 4	rows
7	..., 2	..., 3	8, 4	
8	..., 2	..., 3	9, 4	
⋮	⋮	⋮	⋮	
d	$d+1, 4$	$d+1, 4$	$d+1, 4$	
$d+1$	$d+2, 4$	$d+2, 4$	$d+2, 4$	
$d+2$	$d+3, 4$	$d+3, 4$	$d+3, 4$	Section II:
$d+3$	$d+4, 4$	$d+4, 4$	$d+4, 4$	approximately
$d+4$	$d+5, 4$	$d+5, 4$	$d+5, 4$	$N/\log_2 N$
$d+5$	$d+6, 4$	$d+6, 4$	$d+6, 4$	rows
$d+6$	$d+7, 4$	$d+7, 4$	$d+7, 4$	
$d+7$	$d+8, 4$	$d+8, 4$	$d+8, 4$	
⋮	⋮	⋮	⋮	
f	This section is the same			
$f+1$	(except for the changes			Section III:
$f+2$	in row numbers caused by			a fixed number
$f+3$	relocation) regardless			of rows
$f+4$	of the value of N.			
⋮	⋮	⋮	⋮	

This program is in the format of the machines of Section 1.1. There are N rows in this table. The unspecified row numbers in Section I are all in the range from d to $f-1$, inclusive. The manner in which they are specified determines the finite binary sequence S which the program computes.

The execution of this program is divided into phases. There are twice as many phases as there are rows in Section I. The current phase is determined by a binary sequence P which is written out starting on the second square of the tape. The nth phase starts in row 1 with the scanner on the first square of the tape and with

$$\begin{cases} P = 111\ldots1 & (i \text{ 1's}) \text{ if } n = 2i+1, \\ P = 111\ldots10 & (i \text{ 1's}) \text{ if } n = 2i+2. \end{cases}$$

Control then passes down column three through the $(i+1)$-th row of the table, and then control passes to

$$\begin{cases} \text{row } i+2, \text{ column 1 if } n = 2i+1, \\ \text{row } i+2, \text{ column 2 if } n = 2i+2, \end{cases}$$

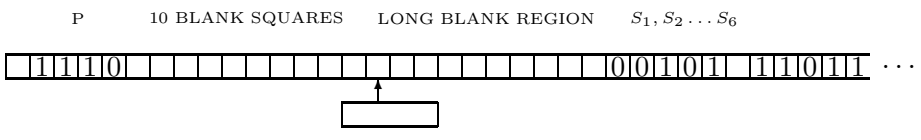
which

1. changes P to what it must be at the start of the $(n+1)$-th phase, and

2. transfers control to a row in Section II.

Suppose this row to be the mth row of Section II from the end of Section II.

Once control has passed to the row in Section II, control then passes down Section II until row f is reached. Each row in Section II causes the tape to be shifted one square to the left, so that when row f finally assumes control, the scanner will be on the mth blank square to the right of P. The following diagram shows the way things may look at this point if n is 7 and m happens to be 11:



Now control has been passed to Section III. First of all, Section III accumulates in base-two on the tape a count of the number of blank squares between the scanner and P when f assumes control. (This number is $m-1$.) This base-two count, which is written on the tape, is simply a binary sequence with a 1 at its left end. Section III then removes this 1 from the left end of the binary sequence. The resulting sequence is called S_n.

Note that if the row numbers entered in

$$\begin{cases} \text{row } i+2, \text{ column 1 if } n = 2i+1, \\ \text{row } i+2, \text{ column 2 if } n = 2i+2, \end{cases}$$

of Section I are suitably specified, this binary sequence S_n can be made any one of the 2^v binary sequences of length $v =$ (the greatest integer not greater than $\log_2(f-d)-1$). Finally, Section III writes S_n in a region of the tape far to the right where all the previous S_j ($j = 1, 2, \ldots, n-1$) have been written during previous phases, cleans up the tape so that only the sequences P and S_j ($j = 1, 2, 3, \ldots, n$) remain on it, positions the scanner back on the square at the end of the tape and, as the last act of phase n, passes control back to row 1 again.

The foregoing description of the workings of the program omits some important details for the sake of clarity. These follow.

It must be indicated how Section III knows when the last phase (phase $2(d-2)$) has occurred. During the nth phase, P is copied just to the right of $S_1, S_2, \ldots, S_n$ (of course a blank square is left between S_n and the copy of P). And during the $(n+1)$-th phase, Section III checks whether or not P is currently different from what it was during the nth phase when the copy of it was made. If it isn't different, then Section III knows that phasing has in fact stopped and that a termination routine must be executed.

The termination routine first forms the finite binary sequence S^* consisting of

$$S_1, S_2, \ldots, S_{2(d-2)},$$

each immediately following the other. As each of the S_j can be any one of the 2^v binary sequences of length v if the row numbers in the entries in Section I are appropriately specified, it follows that S^* can be any

one of the 2^w binary sequences of length $w = 2(d-2)v$. Note that

$$2(d-2)(\log_2(f-d)-1) \geq w > 2(d-2)(\log_2(f-d)-2),$$

so that

$$w \sim 2\left((1-\frac{1}{\log_2 N})N\right)\left(\log_2 \frac{N}{\log_2 N}\right) \sim 2N\log_2 N.$$

As we want the program to be able to compute any sequence S of length not greater than $(2+\epsilon_N)N\log_2 N$, we have S^* consist of S followed to the right by a single 1 and then a string of 0's, and the termination routine removes the rightmost 0's and first 1 from S^*. Q.E.D.

The result just obtained shows that it is impossible to make further improvement in the logical design of the Turing machine of the kind described in Section 1.2 and actually effected in Section 1.3; if we let the number of tape symbols be fixed and speak asymptotically as the number of states goes to infinity, in our present Turing machines 100 percent of the bits required to specify a program also serve to specify the behavior of the machine.

Note too that the argument presented in the first paragraph of this section in fact establishes that, say, for any fixed s greater than zero, at most $n^{-s}2^n$ binary sequences S of length n satisfy

$$L_M(S) \leq (1+\epsilon_n)\frac{n}{(M-1)\log_2 n}.$$

Thus we have: For any fixed s greater than zero, at most $n^{-s}2^n$ binary sequences of length n fail to satisfy the double inequality

$$(1+\epsilon_n)\frac{n}{(M-1)\log_2 n} \leq L_M(S) \leq (1+\epsilon'_n)\frac{n}{(M-1)\log_2 n}.$$

1.7. It may be desirable to have some idea of the "local" as well as the "global" behavior of $L_M(C_n)$. The following program of 8 rows causes an 8-state 3-tape-symbol Turing machine to compute the binary sequence 01100101 of length 8 (this program is in the format of the machines of Section 1.1):

1,2	2,4	2,4
2,3	3,4	3,4
3,3	4,4	4,4
4,2	5,4	5,4
5,2	6,4	6,4
6,3	7,4	7,4
7,2	8,4	8,4
8,3	0,4	0,4

And in general:

(1.7.1) $L_M(C_n) \leq n$.

From this it is easy to see that for m greater than n:

(1.7.2) $L_M(C_m) \leq L_M(C_n) + (m - n)$.

Also, for m greater than n:

(1.7.3) $L_M(C_m) + 1 \geq L_M(C_n)$.

For if one can calculate any binary sequence of length m greater than n with an M-tape-symbol Turing machine having $L_M(C_m)$ states, one can certainly program any M-tape-symbol Turing machine having $L_M(C_m) + 1$ states to calculate the binary sequence consisting of (any particular sequence of length n) followed by a 1 followed by [a sequence of $(m - n - 1)$ 0's], and then—instead of immediately halting—to first erase all the 0's and the first 1 on the right end of the sequence. This last part of the program takes up only a single row of the table; in the format of the machines of Section 1.1 this row r is:

row r	r,5	r,1	0,1

Together (1.7.2) and (1.7.3) yield:

(1.7.4) $|L_M(C_{n+1}) - L_M(C_n)| \leq 1$.

From (1.7.1) it is obvious that $L_M(C_1) = 1$, and with (1.7.4) and the fact that $L_M(C_n)$ goes to infinity with n it finally is concluded that:

(1.7.5) For any positive integer p there is at least one solution n of

$$L_M(C_n) = p.$$

1.8. In this section a certain amount of insight is obtained into the properties of finite binary sequences S of length n for which $L_M(S)$ is close to $L_M(C_n)$. M is considered to be fixed throughout this section. There is some connection between the present subject and that of Shannon in [2, Pt. I, especially Th. 9].

The main result is as follows:

(1.8.1) For any $e > 0$ and $d > 1$ one has for all sufficiently large n: If S is any binary sequence of length n satisfying the statement that

(1.8.2) the ratio of the number of 0's in S to n differs from $\frac{1}{2}$ by more than e,

then $L_M(S) < L_M(C_{[ndH(\frac{1}{2}+e,\frac{1}{2}-e)]})$.

Here $H(p,q)$ $(p \geq 0, q \geq 0, p+q=1)$ is a special case of the entropy function of Boltzmann statistical mechanics and information theory and equals 0 if $p = 0$ or 1, and $-p\log_2 p - q\log_2 q$ otherwise. Also, a real number enclosed in brackets denotes the least integer greater than the enclosed real. The H function comes up because the logarithm to the base-two of the number

$$\sum_{|\frac{k}{n}-\frac{1}{2}|>e} \binom{n}{k}$$

of binary sequences of length n satisfying (1.8.2) is asymptotic to $nH(\frac{1}{2}+e, \frac{1}{2}-e)$. This may be shown easily by considering the ratio of successive binomial coefficients and using the fact that $\log(n!) \sim n\log n$.

To prove (1.8.1), first construct a class of effectively computable functions $M_n(.)$ with the natural numbers from 1 to 2^n as range and all binary sequences of length n as domain. $M_n(S)$ is defined to be the ordinal number of the position of S in an ordering of the binary sequences of length n defined as follows:

1. If two binary sequences S and S' have, respectively, m and m' 0's, then S comes before (after) S' according as $|\frac{m}{n}-\frac{1}{2}|$ is greater (less) than $|\frac{m'}{n}-\frac{1}{2}|$.

2. If 1 does not settle which comes first, take S to come before (after) S' according as S represents (ignoring 0's to the left) a larger (smaller) number in base-two notation than S' represents.

The only essential feature of this ordering is that it gives small ordinal numbers to sequences for which $|\frac{m}{n} - \frac{1}{2}|$ has large values. In fact, as there are only

$$2^{(1+\epsilon_n)nH(\frac{1}{2}+e,\frac{1}{2}-e)}$$

binary sequences S of length n satisfying (1.8.2), it follows that at worst $M_n(S)$ is a number which in base-two notation is represented by a binary sequence of length $\sim nH(\frac{1}{2}+e, \frac{1}{2}-e)$. Thus in order to obtain a short program for computing an S of length n satisfying (1.8.2), let us just give a program of fixed length r the values of n and $M_n(S)$ and have it compute S $(= M_n^{-1}(M_n(S)))$ from this data. The manner in which for n sufficiently large we give the values of n and $M_n(S)$ to the program is to pack them into a single binary sequence of length at most

$$\left[n(1 + \frac{d-1}{2})H(\frac{1}{2} + e, \frac{1}{2} - e)\right] + 2(1 + [\log_2 n])$$

as follows: Consider (the binary sequence representing $M_n(S)$ in base-two notation) followed by 01 followed by [the binary sequence representing n with each of its bits doubled (e.g., if $n = 43$, this is 110011001111)]. Clearly both n and $M_n(S)$ can be recovered from this sequence. And this sequence can be computed by a program of

$$L_M(C_{\left[n(1+\frac{d-1}{2})H(\frac{1}{2}+e,\frac{1}{2}-e)\right]+2(1+[\log_2 n])})$$

rows. Thus for n sufficiently large this many rows plus r is all that is needed to compute any binary sequence S of length n satisfying (1.8.2). And by the asymptotic formula for $L_M(C_n)$ of Section 1.6, it is seen that the total number of rows of program required is, for n sufficiently large, less than

$$L_M(C_{[ndH(\frac{1}{2}+e,\frac{1}{2}-e)]}).$$

Q.E.D.

From (1.8.1) and the fact that $H(p, q) \leq 1$ with equality if and only if $p = q = \frac{1}{2}$, it follows from $L_M(C_n) \sim (n/((M-1)\log_2 n))$ that, for example,

(1.8.3) For any $e > 0$, all binary sequences S in C_n^M, n sufficiently large, violate (1.8.2);

and more generally,

(1.8.4) Let

$$S_{n_1}, S_{n_2}, S_{n_3}, \ldots$$

be any infinite sequence of distinct finite binary sequences of lengths, respectively, $n_1, n_2, n_3, \ldots$ which satisfies

$$L_M(S_{n_k}) \sim L_M(C_{n_k}).$$

Then as k goes to infinity, the ratio of the number of 0's in S_{n_k} to n_k tends to the limit $\frac{1}{2}$.

We now wish to apply (1.8.4) to programs for Turing machines. In order to do this we need to be able to represent the table of entries defining any program as a single binary sequence. A method is sketched here for coding any program $T_{N,M}$ occupying the table of an N-state M-tape-symbol Turing machine into a single binary sequence $C(T_{N,M})$ of length $(1 + \epsilon_N)N(M-1)\log_2 N$.

First, write all the members of the ordered triples entered in the table in base-two notation, adding a sufficient number of 0's to the left of the numerals for all numerals to be

1. as long as the base-two numeral for $N + 1$ if they result from the first member of a triple,

2. as long as the base-two numeral for $M + 2$ if they result from the second member, and

3. as long as the base-two numeral for 2 if they result from the third member.

The only exception to this rule is that if the third member of a triple is 1 or 2, then the first member of the triple is not written in base-two notation; no binary sequences are generated from the first members of such triples. Last, all the binary sequences that have just been obtained are joined together, starting with the binary sequence

that was generated from the first member of the triple entered at the intersection of row 1 with column 1 of the table, then with the binary sequence generated from the second member of the triple..., ...from the third member..., ...from the first member of the triple entered at the intersection of row 1 with column 2, ...from the second member..., ...from the third member..., and so on across the first row of the table, then across the second row of the table, then the third, ...and finally across the Nth row.

The result of all this is a single binary sequence of length $(1 + \epsilon_N)N(M-1)\log_2 N$ (in view of (1.3.1)) from which one can effectively determine the whole table of entries which was coded into it, if only one is given the values of N and M. But it is possible to code in these last pieces of information using only the rightmost

$$2(1 + [\log_2 N]) + 2(1 + [\log_2 M])$$

bits of a binary sequence consequently of total length

$$\begin{aligned}(1 + \epsilon_N)N(M-1)\log_2 N + 2(1 + [\log_2 N]) + 2(1 + [\log_2 M]) \\ = (1 + \epsilon'_N)N(M-1)\log_2 N,\end{aligned}$$

by employing the same trick that was used to pack two pieces of information into a single binary sequence earlier in this section.

Thus we have a simple procedure for coding the whole table of entries $T_{N,M}$ defining a program of an N-state M-tape-symbol Turing machine and the parameters N and M of the machine into a binary sequence $C(T_{N,M})$ of $(1 + \epsilon_N)N(M-1)\log_2 N$ bits.

We now obtain the result:

(1.8.5) Let

$$T_{L_M(S_1),M}, T_{L_M(S_2),M}, \ldots$$

be an infinite sequence of tables of entries which define programs for computing, respectively, the distinct finite binary sequences $S_1, S_2, \ldots$ Then

$$L_M(C(T_{L_M(S_k),M})) \sim L_M(C_{n_k}),$$

where n_k is the length of

$$C(T_{L_M(S_k),M}).$$

With (1.8.4) this gives the proposition:

(1.8.6) On the hypothesis of (1.8.5), as k goes to infinity, the ratio of the number of 0's in

$$C(T_{L_M(S_k),M})$$

to its length tends to the limit $\frac{1}{2}$.

The proof of (1.8.5) depends on three facts:

(1.8.7a) There is an effective procedure for coding the table of entries $T_{N,M}$ defining the program of an N-state M-tape-symbol Turing machine together with the two parameters N and M into a single binary sequence $C(T_{N,M})$ of length $(1+\epsilon_N)N(M-1)\log_2 N$.

(1.8.7b) Any binary sequence of length not greater than

$$(1+\epsilon_N)N(M-1)\log_2 N$$

can be calculated by a suitably programmed N-state M-tape-symbol Turing machine.

(1.8.7c) From a universal Turing machine program it is possible to construct a program for a Turing machine (with a fixed number r of rows) to take $C(T_{N,M})$ and decode it and to then imitate the calculations of the machine whose table of entries $T_{N,M}$ it then knows, until it finally calculates the finite binary sequence S which the program being imitated calculates, if S exists.

(1.8.7a) has just been demonstrated. (1.8.7b) was shown in Section 1.6. (The concept of a universal program is due to Turing [3].)

The proof of (1.8.5) follows. From (1.8.7a) and (1.8.7b),

$$L_M(C(T_{L_M(S_k),M})) \leq (1+\epsilon_k)L_M(S_k),$$

and from (1.8.7c) and the hypothesis of (1.8.5),

$$L_M(C(T_{L_M(S_k),M})) + r \geq L_M(S_k).$$

It follows that

$$L_M(C(T_{L_M(S_k),M})) = (1+\epsilon_k)L_M(S_k),$$

which is—since the length of

$$C(T_{L_M(S_k),M})$$

is

$$(1+\epsilon_k)L_M(S_k)(M-1)\log_2 L_M(S_k)$$

and

$$L_M(C_{(1+\epsilon_k)L_M(S_k)(M-1)\log_2 L_M(S_k)}) = (1+\epsilon'_k)L_M(S_k)$$

—simply the conclusion of (1.8.5).

1.9. The topic of this section is an application of everything that precedes with the exception of Section 1.7 and the first half of Section 1.8. C. E. Shannon suggests [1, p. 165] that the state-symbol product NM is a good measure of the calculating abilities of an N-state M-tape-symbol Turing machine. If one is interested in *comparing* the calculating abilities of *large Turing machines whose M values vary over a finite range,* the results that follow suggest that $N(M-1)$ is a good measure of calculating abilities. We have as an application of a slight generalization of the ideas used to prove (1.8.5):

(1.9.1a) Any calculation which an N-state M-tape-symbol Turing machine can be programmed to perform can be imitated by any N'-state M'-tape-symbol Turing machine satisfying

$$(1+\epsilon_N)N(M-1)\log_2 N < (1+\epsilon''_N)N'(M'-1)\log_2 N'$$

if it is suitably programmed.

And directly from the asymptotic formula for $L_M(C_n)$ we have:

(1.9.1b) If

$$(1+\epsilon_N)N(M-1)\log_2 N < (1+\epsilon''_N)N'(M'-1)\log_2 N',$$

then there exist finite binary sequences which an N'-state M'-tape-symbol Turing machine can be programmed to calculate and which it is impossible to program an N-state M-tape-symbol Turing machine to calculate.

As

$$(1+\epsilon_N)N(M-1)\log_2 N = \\ ((1+\epsilon'_N)N(M-1))\log_2((1+\epsilon'_N)N(M-1))$$

and for x and x' greater than one, $x\log_2 x$ is greater (less) than $x'\log_2 x'$ according as x is greater (less) than x', it follows that the inequalities of (1.9.1a) and (1.9.1b) give the same *ordering* of calculating abilities as do inequalities involving functions of the form $(1+\epsilon_N)N(M-1)$.

Part 2

2.1. In this section we return to the Turing machines of Section 1.1 and add to the conventions (1.1A), (1.1B) and (1.1C),

(2.1D) An entry (i,j) in the pth row of the table of a Turing machine must satisfy $|i-p| \leq b$. In addition, while a fictitious state is used (as before) for the purpose of halting, the row of the table for this fictitious state is now considered to come directly after the actual last row of the program.

Here b is a constant whose value is to be regarded as fixed throughout Part 2. In Section 2.2 it will be shown that b can be chosen sufficiently large that the Turing machines thus defined (which we take the liberty of naming "bounded-transfer Turing machines") have all the calculating capabilities that are basically required of Turing machines for theoretical purposes (e.g., such purposes as defining what one means by "effective process for determining..."), and hence have calculating abilities sufficient for the proofs of Part 2 to be carried out.

(2.1D) may be regarded as a mere convention, but it is more properly considered as a change in the basic philosophy of the logical design of the Turing machine (i.e., the philosophy expressed by A. M. Turing [3, Sec. 9]).

Here in Part 2 there will be little point in considering the general M-tape-symbol machine. It will be understood that we are always speaking of 3-tape-symbol machines.

There is a simple and convenient notational change which can be made at this point; it makes all programs for bounded-transfer Turing machines instantly relocatable (which is convenient if one puts together

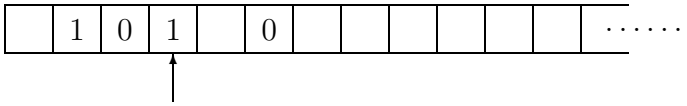
a program from subroutines) and it saves a great deal of superfluous writing. Entries in the tables of machines will from now on consist of ordered pairs (i', j'), where i' goes from $-b$ to b and j' goes from 1 to 5. A "new" entry (i', j') is to be interpreted in terms of the functioning of the machine in a manner depending on the number p of the row of the table it is in; this entry has the same meaning as the "old" entry $(p + i', j')$ used to have.

Thus, halting is now accomplished by entries of the form (k, j) $(1 \leq k \leq b)$ in the kth row (from the end) of the table. Such an entry causes the machine to halt after performing the operation indicated by j.

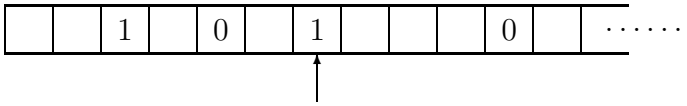
2.2. In this section we attempt to give an idea of the versatility of the bounded-transfer Turing machine. It is here shown in two ways that b can be chosen sufficiently large so that any calculation which one of the Turing machines of Section 1.1 can be programmed to perform can be imitated by a suitably programmed bounded-transfer Turing machine.

As the first proof, b is taken to be the number of rows in a 3-tape-symbol universal Turing machine program for the machines of Section 1.1. This universal program (with its format changed to that of the bounded-transfer Turing machines) occupies the last rows of a program for a bounded-transfer Turing machine, a program which is mainly devoted to writing out on the tape the information which will enable the universal program to imitate any calculation which any one of the Turing machines of Section 1.1 can be programmed to perform. One row of the program is used to write out each symbol of this information (as in the program in Section 1.7), and control passes straight through the program row after row until it reaches the universal program.

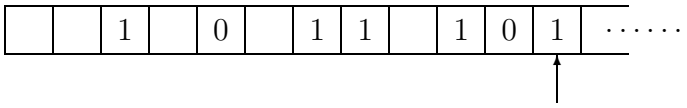
Now for the second proof. To program a bounded-transfer Turing machine in such a manner that it imitates the calculations performed by a Turing machine of Section 1.1, consider alternate squares on the tape of the bounded-transfer Turing machine to be the squares of the tape of the machine being imitated. Thus



is imitated by



After the operation of a state (i.e., write 0, write 1, write blank, shift tape left, shift tape right) has been imitated, as many 1's as the number of the next state to be imitated are written on the squares of the tape of the bounded-transfer Turing machine which are not used to imitate the squares of the other machine's tape, starting on the square immediately to the right of the one on which is the scanner of the bounded-transfer Turing machine. Thus if in the foregoing situation the next state to be imitated is state number three, then the tape of the bounded-transfer Turing machine becomes



The rows of the table which cause the bounded-transfer Turing machine to do the foregoing (type I rows) are interwoven or braided with two other types of rows. The first of these (type II rows) is used for the sole purpose of putting the bounded-transfer Turing machine back in its initial state (row 1 of the table; this row is a type III row). They appear (as do the other two types of rows) periodically throughout the table, and each of them does nothing but transfer control to the preceding one. The second of these (type III rows) serve to pass control back in

the other direction; each time control is about to pass a block of type I rows that imitate a particular state of the other machine while traveling through type III rows, the type III rows erase the rightmost of the 1's used to write out the number of the next state to be imitated. When finally none of these place-marking 1's is left, control is passed to the group of type I rows that was about to be passed, which then proceeds to imitate the appropriate state of the Turing machine of Section 1.1.

Thus the obstacle of the upper bound on the length of transfers in bounded-transfer Turing machines is overcome by passing up and down the table by small jumps, while keeping track of the progress to the desired destination is achieved by subtracting a unit from a count written on the tape just prior to departure.

Although bounded-transfer Turing machines have been shown to be versatile, it is not true that as the number of states goes to infinity, asymptotically 100 percent of the bits required to specify a program also serve to specify the behavior of the bounded-transfer Turing machine.

2.3. In this section the following fundamental result is proved.

(2.3.1) $L(C_n) \sim a^* n$, where a^* is, of course, a positive constant.

First it is shown that there exists an a greater than zero such that:

(2.3.2) $L(C_n) \geq an$.

It is clear that there are exactly

$$((5)(2b+1))^{3N}$$

different ways of making entries in the table of an N-state bounded-transfer Turing machine; that is, there are

$$2^{((3\log_2(10b+5))N)}$$

different programs for an N-state bounded-transfer Turing machine. Since a different program is required to have the machine calculate each of the 2^n different binary sequences of length n, it can be seen that an N-state bounded-transfer Turing machine can be programmed to calculate any binary sequence of length n only if

$$(3\log_2(10b+5))N \geq n \text{ or } N \geq \frac{n}{3\log_2(10b+5)}.$$

Thus one can take $a = (1/(3\log_2(10b+5)))$.

Next it is shown that:

(2.3.3) $L(C_n) + L(C_m) \geq L(C_{n+m})$.

To do this we present a way of making entries in a table with at most $L(C_n) + L(C_m)$ rows which causes the bounded-transfer Turing machine thus programmed to calculate any particular binary sequence S of length $n+m$. S can be expressed as a binary sequence S' of length n followed by a binary sequence S'' of length m. The table is then formed from two sections which are numbered in the order in which they are encountered in reading from row 1 to the last row of the table. Section I consists of at most $L(C_n)$ rows. It is a program which calculates S'. Section II consists of at most $L(C_m)$ rows. It is a program which calculates S''. It follows from this construction and the definitions that (2.3.3) holds.

(2.3.2) and (2.3.3) together imply (2.3.1).[2] This will be shown by a demonstration of the following general proposition:

(2.3.4) Let $A_1, A_2, A_3, \ldots$ be an infinite sequence of natural numbers satisfying

(2.3.5) $A_n + A_m \geq A_{n+m}$.

Then as n goes to infinity, (A_n/n) tends to a limit from above.

[2][As stated in the preface of this book, it is straightforward to apply to LISP the techniques used here to study bounded-transfer Turing machines. Let us define $H_{\rm LISP}(x)$ where x is a bit string to be the size in characters of the smallest LISP S-expression whose value is the list x of 0's and 1's. Consider the LISP S-expression (APPEND P Q), where P is a minimal LISP S-expression for the bit string x and Q is a minimal S-expression for the the bit string y. I.e., the value of P is the list of bits x and P is $H_{\rm LISP}(x)$ characters long, and the value of Q is the list of bits y and Q is $H_{\rm LISP}(y)$ characters long. (APPEND P Q) evaluates to the concatenation of the bit strings x and y and is $H_{\rm LISP}(x) + H_{\rm LISP}(y) + 10$ characters long. Therefore, let us define $H'_{\rm LISP}(x)$ to be $H_{\rm LISP}(x)+10$. Now $H'_{\rm LISP}$ is subadditive like $L(S)$. The discussion of bounded-transfer Turing machines in this paper and the next therefore applies practically word for word to $H'_{\rm LISP} = H_{\rm LISP} + 10$. In particular, let $B(n)$ be the maximum of $H'_{\rm LISP}(x)$ taken over all n-bit strings x. Then $B(n)/n$ is bounded away from zero, $B(n+m) \leq B(n) + B(m)$, and $B(n)$ is asymptotic to a nonzero constant times n.]

For all n, $A_n \geq 0$, so that $(A_n/n) \geq 0$; that is, $\{(A_n/n)\}$ is a set of reals bounded from below. It is concluded that this set has a greatest lowest bound a^*. We now show that

$$\lim_{n\to\infty} \frac{A_n}{n} = a^*.$$

Since a^* is the greatest lower bound of the set $\{(A_n/n)\}$, for any e greater than zero there is a d for which

(2.3.6) $(A_d/d) < a^* + e.$

Every natural number n can be expressed in the form $n = qd+r$, where $0 \leq r < d$. From (2.3.5) it can be seen that for any $n_1, n_2, n_3, \ldots, n_{q+1}$,

$$\sum_{k=1}^{q+1} A_{n_k} \geq A_{(\sum_{k=1}^{q+1} n_k)}.$$

Taking $n_k = d$ $(k = 1, 2, \ldots, q)$ and $n_{q+1} = r$ in this, we obtain

$$qA_d + A_r \geq A_{qd+r} = A_n,$$

which with (2.3.6) gives

$$qd(a^* + e) = (n - r)(a^* + e) \geq A_n - A_r$$

or

$$(1 - \frac{r}{n})(a^* + e) \geq \frac{A_n}{n} - \frac{A_r}{n},$$

which implies

$$a^* + e \geq \frac{A_n}{n} + \epsilon_n$$

or

$$\limsup_{n\to\infty} \frac{A_n}{n} \leq a^* + e.$$

Since $e > 0$ is arbitrary, it can be concluded that

$$\limsup_{n\to\infty} \frac{A_n}{n} \leq a^*,$$

which with the fact that $(A_n/n) \geq a^*$ for all n gives

$$\lim_{n\to\infty} \frac{A_n}{n} = a^*.$$

2.4. In Section 2.3 an asymptotic formula analogous to a part of Section 1.6 was demonstrated; in this section a result is obtained which completes the analogy. This result is most conveniently stated with the aid of the notation $B(m)$ (where m is a natural number) for the binary sequence which is the numeral representing m in base-two notation (e.g., $B(6) = 110$).

(2.4.1) There exists a constant c such that those binary sequences S of length n satisfying

(2.4.2)

$$\begin{aligned} L(S) \quad \leq \quad & L(C_n) - L(B(L(C_n))) - [\log_2 L(B(L(C_n)))] \\ & - L(C_m) - [\log_2 L(C_m)] - c \end{aligned}$$

are less than 2^{n-m} in number.

The proof of (2.4.1) is by contradiction. We suppose that those S of length n satisfying (2.4.2) are 2^{n-m} or more in number and we conclude that for any particular binary sequence $S^{\sim}$ of length n there is a program of at most $L(C_n)-1$ rows that causes a bounded-transfer Turing machine to calculate $S^{\sim}$. This table consists of 11 sections which come one after the other. The first section consists of a single row which moves the tape one square to the left (1,4 1,4 1,4 will certainly do this). The second section consists of exactly $L(B(L(C_n)))$ rows; it is a program for computing $B(L(C_n))$ consisting of the smallest possible number of rows. The third section is merely a repetition of the first section. The fourth section consists of exactly $[\log_2 L(B(L(C_n)))]$ rows. Its function is to write out on the tape the binary sequence which represents the number $L(B(L(C_n)))$ in base-two notation. Since this is a sequence of exactly $[\log_2 L(B(L(C_n)))]$ bits, a simple program exists for calculating it consisting of exactly $[\log_2 L(B(L(C_n)))]$ rows each of which causes the machine to write out a single bit of the sequence and then shift the tape a single square to the left (e.g., 0,2 1,4 1,4 will do

for a 0 in the sequence). The fifth section is merely a repetition of the first section. The sixth section consists of at most $L(C_m)$ rows; it is a program consisting of the smallest possible number of rows for computing the sequence S^R of the m rightmost bits of $S^{\sim}$. The seventh section is merely a repetition of the first section. The eighth section consists of exactly $[\log_2 L(C_m)]$ rows. Its function is to write out on the tape the binary sequence which represents the number $L(C_m)$ in base-two notation. Since this is a sequence of exactly $[\log_2 L(C_m)]$ bits, a simple program exists for calculating it consisting of exactly $[\log_2 L(C_m)]$ rows each of which causes the machine to write out a single bit of the sequence and then shift the tape a single square to the left. The ninth section is merely a repetition of the first section. The tenth section consists of at most as many rows as the expression on the right-hand side of the inequality (2.4.2). It is a program for calculating one (out of not less than 2^{n-m}) of the sequences of length n satisfying (2.4.2) (which one it is depends on $S^{\sim}$ in a manner which will become clear from the discussion of the eleventh section; for now we merely denote it by S^L).

We now come to the last and crucial eleventh section, which consists *by definition* of $(c-6)$ rows, and which therefore brings the total number of rows up to at most $1 + L(B(L(C_n))) + 1 + [\log_2 L(B(L(C_n)))] + 1 + L(C_m) + 1 + [\log_2 L(C_m)] + 1+$ (the expression on the right-hand side of the inequality (2.4.2)) $+ (c - 6) = L(C_n) - 1$. When this section of the program takes over, the numbers and sequences $L(C_n), L(B(L(C_n))), S^R, L(C_m), S^L$ are written—in the above order—on the tape. Note, first of all, that section 11 can:

1. compute the value v of the right-hand side of the inequality (2.4.2) from this data,

2. find the value of n from this data (simply by counting the number of bits in the sequence S^L), and

3. find the value of m from this data (simply by counting the number of bits in S^R).

Using its knowledge of v, m and n, section 11 then computes from the sequence S^L a new sequence $S^{L'}$ which is of length $(n-m)$. The manner

in which it does this is discussed in the next paragraph. Finally, section 11 adjoins the sequence S^R to the right of $S^{L'}$, positions this sequence which is in fact $S^{\sim}$ properly for it to be able to be regarded calculated, cleans up the rest of the tape, and halts scanning the square just to the right of $S^{\sim}$. $S^{\sim}$ has been calculated.

To finish the proof of (2.4.1) we must now only indicate how section 11 arrives at $S^{L'}$ (of length $(n-m)$) from v, m, n, and S^L. (And it must be here that it is made clear how the choice of S^L depends on $S^{\sim}$.) By assumption, S^L satisfies

(2.4.3) $L(S^L) \leq v$ and S^L is of length n.

Also by assumption there are at least 2^{n-m} sequences which satisfy (2.4.3). Now section 11 contains a procedure which when given any one of some particular serially ordered set Q_v^n of 2^{n-m} sequences satisfying (2.4.3), will find the ordinal number of its position in Q_v^n. And the number of the position of S^L in Q_v^n is the number of the position of $S^{L'}$ in the natural ordering of all binary sequences of length $(n-m)$ (i.e., 000...00, 000...01, 000...10, 000...11, ..., 111...00, 111...01, 111...10, 111...11). In the next and final paragraph of this proof, the foregoing italicized sentence is explained.

It is sufficient to give here a procedure for serially calculating the members of Q_v^n in order. (That is, we define a serially ordered Q_v^n for which there is a procedure.) By assumption we know that the predicate which is satisfied by all members of Q_v^n, namely,

$$(L(\ldots) \leq v)\,\&\,(\ldots \text{ is of length } n),$$

is satisfied by at least 2^{n-m} sequences. It should also be clear to the reader on the basis of some background in Turing machine and recursive function theory (see especially Davis [4], where recursive function theory is developed from the concept of the Turing machine) that the set Q of

> all natural numbers of the form $2^n 3^v 5^e$, where e is the natural number represented in base-two notation by a binary sequence S satisfying $(L(S) \leq v)$ & (S is of length n)

is recursively enumerable. Let T denote some particular Turing machine which is programmed in such a manner that it recursively enumerates (or, to use E. Post's term, generates) Q. The definition of Q_v^n can now be given:

> Q_v^n is the set of binary sequences of length n which represent in base-two notation the exponents of 5 in the prime factorization of the first 2^{n-m} members of Q generated by T whose prime factorizations have 2 with an exponent of n and 3 with an exponent of v, and their order in Q_v^n is the order in which T generates them.

Q.E.D.

It can be proved by contradiction that the set Q is not recursive. For were Q recursive, there would be a program which given any finite binary sequence S would calculate $L(S)$. Hence there would be a program which given any natural number n would calculate the members of C_n. Giving n to this program can be done by a program of length $[\log_2 n]$. Thus there would be a program of length $[\log_2 n] + c$ which would calculate an element of C_n. But we know that the shortest program for calculating an element of C_n is of length $\sim a^* n$, so that we would have for n sufficiently large an impossibility.

It should be emphasized that if $L(C_n)$ is an effectively computable function of n then the method of this section yields the far stronger result: There exists a constant c such that those binary sequences S of length n satisfying $L(S) \leq L(C_n) - L(C_m) - c$ are less than 2^{n-m} in number.[3]

2.5. The purpose of this section is to investigate the behavior of the right-hand side of (2.4.2). We start by showing a result which is stronger for n sufficiently large than the inequality $L(C_n) \leq n$, namely, that the constant a^* in the asymptotic evaluation $L(C_n) \sim a^* n$ of Section 2.3 is less than 1. This is done by deriving:

[3] [For LISP we also obtain this much neater form of result that most n-bit strings have close to the maximum complexity H_{LISP}. The reason is that by using EVAL a quoted LISP S-expression tells us its size as well as its value. In other words, $H_{\text{LISP}}(x) = H_{\text{LISP}}(x, H_{\text{LISP}}(x)) + O(1)$.]

(2.5.1) For any s there exist n and m such that

$$L(C_s) \leq L(C_n) + L(C_m) + c,$$

and $(n+m)$ is the smallest integral solution x of the inequality

$$s \leq x + [\log_2 x] - 1.$$

From (2.5.1) it will follow immediately that if $e(n)$ denotes the function satisfying $L(C_n) = a^*n + e(n)$ (note that by Section 2.3 $(e(n)/n)$ tends to 0 *from above* as n goes to infinity), then for any s, $L(C_s) \leq L(C_n) + L(C_m) + c$ for some n and m satisfying $(n+m) = s - (1+\epsilon_s)\log_2 s$, which implies

$$a^*s \leq a^*\left(s - (1+\epsilon_s)\log_2 s\right) + e(n) + e(m)$$

or

$$(a^* + \epsilon_s)\log_2 s \leq e(n) + e(m).$$

Hence as n and m are both less than s and at least one of $e(n), e(m)$ is greater than $\frac{1}{2}(a^* + \epsilon_s)\log_2 s$, there are an infinity of n for which $e(n) \geq \frac{1}{2}(a^* + \epsilon_n)\log_2 n$. That is,

(2.5.2) $$\limsup \frac{L(C_n) - a^*n}{a^* \log_2 n} \geq \frac{1}{2}.$$

From (2.5.2) with $L(C_n) \leq n$ follows immediately

(2.5.3) $a^* < 1$.

The proof of (2.5.1) is presented by examples. The notation $T * U$ is used, where T and U are finite binary sequences for the sequence resulting from adjoining U to the right of T. Suppose it is desired to calculate some finite binary sequence S of length s, say $S = 010110010100110$ and $s = 15$. The smallest integral solution x of $s \leq x + [\log_2 x] - 1$ for this value of s is 12. Then S is expressed as $S' * S^T$ where S' is of length $x = 12$ and S^T is of length $s - x = 15 - 12 = 3$, so that $S' = 010110010100$ and $S^T = 110$. Next S' is expressed as $S^L * S^R$ where the length m of S^L satisfies $A * B(m) = S^T$ for some (possibly null) sequence A consisting entirely of 0's, and the length n of S^R is

$x - m$. In this case $A * B(m) = 110$, so that $m = 6$, $S^L = 010110$ and $S^R = 010100$. The final result is that one has obtained the sequences S^L and S^R from the sequence S. And—this is the crucial point—if one is given the S^L and S^R resulting by the foregoing process from some unknown sequence S, one can reverse the procedure and determine S. Thus suppose $S^L = 1110110$ and $S^R = 01110110000$ are given. Then the length m of S^L is 7, the length n of S^R is 11, and the sum x of m and n is $7 + 11 = 18$. Therefore the length s of S must be $s = x + [\log_2 x] - 1 = 18 + 5 - 1 = 22$. Thus $S = S^L * S^R * S^T$, where S^T is of length $s - x = 22 - 18 = 4$, and so from $A * B(m) = S^T$ or $0 * B(7) = S^T$ one finds $S^T = 0111$. It is concluded that

$$S = S^L * S^R * S^T = 1110110011101100000111.$$

(For x of the form 2^h what precedes is not strictly correct. In such cases s may equal the foregoing indicated quantity or the foregoing indicated quantity minus one. It will be indicated later how such cases are to be dealt with.)

Let us now denote by F the function carrying (S^L, S^R) into S, and by F_R^{-1} the function carrying S into S^R, defining F_L^{-1} similarly. Then for any particular binary sequence S of length s the following program consists of at most

$$1 + L(F_L^{-1}(S)) + 1 + L(F_R^{-1}(S)) + 2 + (c - 4) \leq L(C_n) + L(C_m) + c$$

rows with $m + n = x$ being the smallest integral solution of $s \leq x + [\log_2 x] - 1$.

Section I: 1,4 1,4 1,4
Section II consists of $L(F_L^{-1}(S))$ rows. It is a program with the smallest possible number of rows for calculating $F_L^{-1}(S)$.
Section III: 1,4 1,4 1,4
Section IV consists of $L(F_R^{-1}(S))$ rows. It is a program with the smallest possible number of rows for calculating $F_R^{-1}(S)$. (Should x be of the form 2^h, another section is added at this point to tell Section V which of the two possible values s happens to have. This section consists of two rows; it is either 1,4 1,4 1,4 1,2 1,2 1,2 or 1,4 1,4 1,4 1,3 1,3 1,3.)
Section V consists of $c-4$ rows, by definition. It is a program that is able to compute F. It computes $F(F_L^{-1}(S), F_R^{-1}(S)) = S$, positions S properly on the tape, cleans up the rest of the tape, positions the scanner on the square just to the right of S, and halts.

As this program causes S to be calculated, the proof is easily seen to be complete.

The second result is:

(2.5.4) Let $f(n)$ be any effectively computable function that goes to infinity with n and satisfies $f(n+1) - f(n) = 0$ or 1. Then there are an infinity of distinct n_k for which $L(B(L(C_{n_k}))) < f(n_k)$.

This is proved from (2.5.5), the proof being identical with that of (1.7.5).

(2.5.5) For any positive integer p there is at least one solution n of $L(C_n) = p$.

Let the n_k satisfy $L(C_{n_k}) = f^{-1}(k)$, where $f^{-1}(k)$ is defined to be the smallest value of j for which $f(j) = k$. Then since $L(C_n) \leq n$, $f^{-1}(k) \leq n_k$. Noting that f^{-1} is an effectively computable function, it is easily seen that

$$L(B(L(C_{n_k}))) = L(B(f^{-1}(k))) \leq L(B(k)) + c \leq [\log_2 k] + c.$$

Hence, for all sufficiently large k,

$$L(B(L(C_{n_k}))) \leq [\log_2 k] + c < k = f(f^{-1}(k)) \leq f(n_k).$$

Q.E.D.

(2.5.4) and (2.4.1) yield:

(2.5.6) Let $f(n)$ be any effectively computable function that goes to infinity with n and satisfies $f(n+1) - f(n) = 0$ or 1. Then there are an infinity of distinct n_k for which less than $2^{n_k - f(n_k)}$ binary sequences S of length n_k satisfy $L(S) \leq L(C_{n_k}) - (a^* + \epsilon_k) f(n_k)$.

Part 3

3.1. Consider a scientist who has been observing a closed system that once every second either emits a ray of light or does not. He summarizes his observations in a sequence of 0's and 1's in which a zero represents "ray not emitted" and a one represents "ray emitted." The sequence may start

$$0110101110\ldots$$

and continue for a few thousand more bits. The scientist then examines the sequence in the hope of observing some kind of pattern or law. What does he mean by this? It seems plausible that a sequence of 0's and 1's is patternless if there is no better way to calculate it than just

by writing it all out at once from a table giving the whole sequence:

My Scientific Theory
0
1
1
0
1
0
1
1
1
0
⋮

This would not be considered acceptable. On the other hand, if the scientist should hit upon a method by which the whole sequence could be calculated by a computer whose program is short compared with the sequence, he would certainly not consider the sequence to be entirely patternless or random. And the shorter the program, the greater the pattern he might ascribe to the sequence.

There are many genuine parallels between the foregoing and the way scientists actually think. For example, a simple theory that accounts for a set of facts is generally considered better or more likely to be true than one that needs a large number of assumptions. By "simplicity" is *not* meant "ease of use in making predictions." For although General or Extended Relativity is considered to be the simple theory par excellence, very extended calculations are necessary to make predictions from it. Instead, one refers to the number of arbitrary choices which have been made in specifying the theoretical structure. One naturally is suspicious of a theory the number of whose arbitrary elements is of an order of magnitude comparable to the amount of information about reality that it accounts for.

On the basis of these considerations it may perhaps not appear entirely arbitrary to define a patternless or random finite binary sequence as a sequence which in order to be calculated requires, roughly speaking, at least as long a program as any other binary sequence of the same

length. A patternless or random infinite binary sequence is then defined to be one whose initial segments are all random. In making these definitions mathematically approachable it is necessary to specify the kind of computer referred to in them. This would seem to involve a rather arbitrary choice, and thus to make our definitions less plausible, but in fact both of the kinds of Turing machines which have been studied by such different methods in Parts 1 and 2 lead to precise mathematical definitions of patternless sequences (namely, the patternless or random finite binary sequences are those sequences S of length n for which $L(S)$ is approximately equal to $L(C_n)$, or, fixing M, those for which $L_M(S)$ is approximately equal to $L_M(C_n)$) whose provable statistical properties start with forms of the law of large numbers. Some of these properties will be established in a paper of the author to appear.[4]

A final word. In scientific research it is generally considered better for a proposed new theory to account for a phenomenon which had not previously been contained in a theoretical structure, before the discovery of that phenomenon rather than after. It may therefore be of some interest to mention that the intuitive considerations of this section antedated the investigations of Parts 1 and 2.

3.2. The definition which has just been proposed[5] is one of many attempts which have been made to define what one means by a patternless or random sequence of numbers. One of these was begun by R. von Mises [5] with contributions by A. Wald [6], and was brought to its culmination by A. Church [7]. K. R. Popper [8] criticized this definition. The definition given here deals with the concept of a patternless binary sequence, a concept which corresponds roughly in intuitive intent with the random sequences associated with probability half of Church. However, the author does not follow the basic philosophy of the von

[4]The author has subsequently learned of work of P. Martin-Löf ("The Definition of Random Sequences," research report of the Institutionen för Försäkringsmatematik och Matematisk Statistik, Stockholm, Jan. 1966, 21 pp.) establishing statistical properties of sequences defined to be patternless on the basis of a type of machine suggested by A. N. Kolmogorov. Cf. footnote 5.

[5]The author has subsequently learned of the paper of A. N. Kolmogorov, Three approaches to the definition of the concept "amount of information," *Problemy Peredachi Informatsii* [Problems of Information Transmission], *1*, 1 (1965), 3–11 [in Russian], in which essentially the definition offered here is put forth.

Mises–Wald–Church definition; instead, the author is in accord with the opinion of Popper [8, Sec. 57, footnote 1]:

> I come here to the point where I failed to carry out fully my intuitive program—that of analyzing randomness as far as it is possible within the region of *finite* sequences, and of proceeding to *infinite* reference sequences (in which we need *limits* of relative frequencies) only afterwards, with the aim of obtaining a theory in which the existence of frequency limits follows from the random character of the sequence.

Nonetheless the methods given here are similar to those of Church; the concept of effective computability is here made the central one.

A discussion can be given of just how patternless or random the sequences given in this paper appear to be for practical purposes. How do they perform when subjected to statistical tests of randomness? Can they be used in the Monte Carlo method? Here the somewhat tantalizing remark of J. von Neumann [9] should perhaps be mentioned:

> Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number—there are only methods to produce random numbers, and a strict arithmetical procedure of course is not such a method. (It is true that a problem that we suspect of being solvable by random methods may be solvable by some rigorously defined sequence, but this is a deeper mathematical question than we can now go into.)

Acknowledgment

The author is indebted to Professor Donald Loveland of New York University, whose constructive criticism enabled this paper to be much clearer than it would have been otherwise.

References

[1] Shannon, C. E. A universal Turing machine with two internal states. In *Automata Studies,* Shannon and McCarthy, Eds., Princeton U. Press, Princeton, N. J., 1956.

[2] —. A mathematical theory of communication. *Bell Syst. Tech. J. 27* (1948), 379–423.

[3] Turing, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* {2} *42* (1936–37), 230–265; Correction, *ibid., 43* (1937), 544–546.

[4] Davis, M. *Computability and Unsolvability.* McGraw-Hill, New York, 1958.

[5] von Mises, R. *Probability, Statistics and Truth.* MacMillan, New York, 1939.

[6] Wald, A. Die Widerspruchsfreiheit des Kollektivbegriffes der Wahrscheinlichkeitsrechnung. *Ergebnisse eines mathematischen Kolloquiums 8* (1937), 38–72.

[7] Church, A. On the concept of a random sequence. *Bull. Amer. Math. Soc. 46* (1940), 130–135.

[8] Popper, K. R. *The Logic of Scientific Discovery.* U. of Toronto Press, Toronto, 1959.

[9] von Neumann, J. Various techniques used in connection with random digits. In *John von Neumann, Collected Works, Vol. V.* A. H. Taub, Ed., MacMillan, New York, 1963.

[10] Chaitin, G. J. On the length of programs for computing finite binary sequences by bounded-transfer Turing machines. Abstract 66T-26, *Notic. Amer. Math. Soc. 13* (1966), 133.

[11] —. On the length of programs for computing finite binary sequences by bounded-transfer Turing machines II. Abstract 631-6, *Notic. Amer. Math. Soc. 13* (1966), 228–229. (Erratum, p. 229, line 5: replace "P" by "L".)

Received October, 1965; Revised March, 1966