Universal Modeling and Coding

JORMA RISSANEN AND GLEN G. LANGDON, JR, SENIOR MEMBER, IEEE

Abstract—**The problems arising in the modeling and coding of strings for compression purposes are discussed. The notion of an information source that simplifies and sharpens the traditional one is axiomatized, and adaptive and nonadaptive models are defined. With a measure of complexity assigned to the models, a fundamental theorem is proved which states that models that use any kind of alphabet extension are inferior to the best models using no alphabet extensions at all. A general class of so-called first-in first-out (FIFO) arithmetic codes is described which require no alphabet extension devices and which therefore can be used in conjunction with the best models. Because the coding parameters are the probabilities that define the model, their design is easy, and the application of the code is straightforward even with adaptively changing source models.**

I. INTRODUCTION

DATA compression problems arising in digital processing differ in one important respect from the traditionally studied ones in communication theory: there is no well defined statistical information source to which the code can be tuned. Moreover, often a set of strings to be compressed cannot even be adequately modeled by a single source, of say, the Markov type. Instead, one is given a long string of symbols in some alphabet, often binary, after which another different string is received and so on. An example of this is the finite but indefinite set of scanned black and white documents consisting of text, drawings, tables, and so forth.

What is needed in such problems is a universal modeler encoder. In broad terms, modeling involves a determination of certain source-string events and their contexts, which uniquely describe the source string. We regard the *model* as consisting of two parts: 1) the *structure* which is the set of events and their context, and 2) the *parameters* which are the probabilities assigned to the events.

The structure is intended to capture the redundancies in the entire set of source strings under consideration, such as the set of black and white documents, while the parameters are tailored to each individual string separately. This

Manuscript received July 26, 1979; revised October 18, 1979.

The authors are with the IBM Corporation, 5600 Cottle Rd., San Jose, CA 95193.

0018-9448/81/0100-0012$00.75

way a degree of "universality" can be achieved. The encoder, in turn, encodes the string using the statistics provided by the model. It should clearly be capable of doing its job without imposing restriction on the modeler, and it should produce a code string with a length close to the ideal that the modeled source can provide.

Model building starts with the decision whether or not to use an alphabet extension of some kind. The selection of the alphabet is particularly important because it affects the nature and complexity of the source model. Usually alphabet extension is done by grouping the initially given symbols into fixed or variable-length blocks. Particularly with binary alphabets, the groups formed of runs of zeros or ones, or of both, are popular. In the case of scanned black and white images, some of the early model structures used straight run-lengths while other more sophisticated ones condition the end of a white run to that on the line above, the deviations forming a new alphabet. Yet other models read two lines at a time which are converted into run-like segments as further-derived symbols. All of these structures appear to be based on and intertwined with a preselected coding technique that tends to obscure the important role played by the source itself.

In the light of such a multitude of models, it may seem appropriate to study the problems of modeling in a systematic manner with the hope of demonstrating that some models are inherently better than others. We begin with a notion of an information source which differs from the customary one in a subtle but significant manner. The models for such sources are partitioned into two classes: those that use the original symbols in which the strings are described, and those that use some form of alphabet extension, i.e., extended sources. We further distinguish in each class between stationary and adaptive models, where the latter in particular are not just nonstationary but nonstationary in a special way which permits their parameters to change only in accordance with certain sound estimation principles. In order to be able to compare the performance of different models we introduce a *model cost*, which essentially is the number of independent parameters needed to describe that part of the model which is not shared by all of the strings to be encoded. This enables us to prove the main result in this paper: there is nothing to gain and something to lose with alphabet extension, and the best models with a given cost use no alphabet extension of any kind.

An implication of this theorem is that the coding must be done without the use of tables larger than the number of parameters in the model. This means in particular that with small alphabets, the traditional "concatenation" codes that require alphabet extension for good code efficiency, are very difficult to apply. How then is the coding to be done? One answer is by arithmetic coding. (This should not be confused with "arithmetic error coding" which is an entirely different subject.) Arithmetic coding was introduced by Rissanen in 1975 in the form of a last-in-first-out (LIFO) code [1], which may be regarded as a practicable derivative of the earlier enumerative codes due to Lynch [13], Davisson [14], Schalkwijk [4], and Cover [5], in that the inherent problem of a growing precision was solved. In the following year, Pasco constructed a first-in-first out (FIFO) code [2], starting from Elias' code [6]. A dual pair of LIFO and FIFO codes were further described by Rissanen and Langdon [3], and recently other versions of FIFO arithmetic codes under different names have been proposed [7], [8]. In Section VI we study a very large class of arithmetic codes in an abstract manner, making them thereby independent of specific program and hardware implementations. This class includes all of the above codes as special cases.

II. INFORMATION SOURCE

In this paper we select a statistical framework within which we wish to describe, or equivalently, encode the principal objects of interest—the strings. The most obvious choice for the statistical structure in question is a random variable. In some way we characterize a family of strings with a probability distribution such that our given string is one of the outcomes. The string can then be encoded with a codelength determined by its probability. The strings we are interested in are very long, however, and to calculate their extremely small probabilities in a practicable manner, we must be able to assign a probabilistic meaning to their prefixes. In other words, in addition to the entire string, we also want their parts and above all their prefixes, to be valid outcomes. This means that we need another statistical structure—the information source.

Traditionally, for communication purposes an information source has been taken to be a random process $\{x(i)\}$ that emits outcomes of random variables $x(i)$ in a never-ending stream. The time index i then runs through all the integers. In addition to the random variables $x(i)$, the process also defines all the finite joint variables $(x(i), x(j), \cdots)$. The most frequently studied information sources are either independent, stationary, or Markovian.

In the traditional notion of an information source, we particularly object to the idea of a source emitting symbols in a never-ending manner. We feel that such a view is not only unrealistic but that it also puts an emphasis upon the wrong things concerning an information source and its coding problems. We therefore construct a different and more realistic notion of an information source. The set of events of interest in this is the set S^* of all *finite* sequences of the symbols from a d-element alphabet S, including the empty string "null". The absolute-time instance of the occurrence of the ith symbol in the string, and in particular that of the first symbol, is totally irrelevant. The relevant thing instead, is the symbol's position counted, for example, from the left end of the string. Similarly, we are not interested in outcomes which are not sequences.

The set S^* of valid outcomes, also called messages, can be conveniently viewed as a d-ary tree with the null string as the root at the top. From the node representing string s there is an arc corresponding to each symbol x and

connecting s to the successor node sx, associated with the string sx. Formally, we define an *information source*, or just a source, to be the pair $<S^*, P>$, where P is a function from S^* into [0,1] satisfying the conditions:

$$P(\text{null}) = 1,$$
$$P(s) = \sum_{x \in S} P(sx), \tag{2.1}$$

for all s in S^*. Accordingly, we may view an information source as the infinite d-ary *tree* S^* where each node s has the number $P(s)$ attached to it. Condition (2.1) states that the sum of the numbers attached to the immediate successors of node s equals its own number $P(s)$.

The interpretation here is that P assigns a probability to each outcome or sequence. However, the sum of these probabilities over all sequences is generally greater than unity; in fact, the probabilities of the d sequences of length one already add up to one, which means that an information source is not a random variable. Nevertheless, the important condition (2.1) insures that the probability $P(s)$ of string s is unambiguous and independent of the countably many random variables that include s as an outcome that we might wish to pick. To make this more precise, consider a finite subtree T of S^* sharing the root. Call T *complete* if all of its interior nodes have all their d immediate successors in T. The noninterior nodes are called *leaves*. It follows from (2.1) that the set of leaves of each complete subtree with null as the root defines a random variable; i.e., the sum of the numbers $P(s)$ over the leaves is one. Clearly, each node s of S^* belongs to the countably many such random variables, and in *all* of them the probability of s is the same, namely $P(s)$.

To further clarify the difference between a random variable and an information source, we add that the outcomes of a random variable are independent objects competing for the total probability mass. In contrast, the outcomes of an information source are not independent by their very construction, since each string is a nested collection of its prefixes. Condition (2.1) reflects this generic constraint in the construction of the strings. Finally, (2.1) is related to the compatibility condition for random processes. An information source however, is a weaker structure than a random process because the valid events are only sequences. This is what reduces the more complex compatibility conditions for random processes to the single condition (2.1).

Further, we can express stationarity in this setting as follows. An information source is called *stationary* if

$$P(s) = \sum_{x \in S} P(xs) \tag{2.2}$$

for all s in S^*. Observe the perfect symmetry between (2.1) and (2.2). This notion of stationarity implies the usual shift-invariance property of the joint probabilities, as the reader can easily verify. Finally, a source is called *independent* if

$$P(ss') = P(s)P(s') \tag{2.3}$$

for all s and s' in S^*.

In the context where s is an outcome of a random variable with probability $P(s)$, the quantity

$$I(s) = -\log P(s) \tag{2.4}$$

is known as the self-information. Because of the unambiguity of $P(s)$ in the notion of an information source, we can now extend the same interpretation to information sources, and we rename $I(s)$ as the *information content* of the string s. There are various ways to justify the information content as the *ideal codelength* for string s. For instance, its mean, i.e., the entropy,

$$H(n) = (1/n) \sum_{s \in S^n} P(s)I(s), \tag{2.5}$$

over the set S^n of all strings with length n is for large n an increasingly good approximation of the minimum mean per symbol codelength, where a code is defined to be any one-to-one function from S^* into B^*, B denoting the binary alphabet. The reason we do not require the code to be a concatenation code, i.e., one which preserves the concatenation, let alone a prefix code, is that the powerful arithmetic codes are of neither kind. Because of this, the entropy above is not a lower bound on the minimum codelength, and the "ideal codelength" interpretation loses some of its justification. Other justifications exist, however, which we hope to be able to discuss in another context; for some such results we refer to [12]. Somewhat curiously, as seen in Subsection III-B, the ideal codelength in an optimized model also has the interpretation of an entropy.

To conclude this section we write the probability of a string $s = s(1) \cdots s(n)$ as follows:

$$P(s) = P(s(1))P(s(2)|s(1)) \cdots P(s(n)|s(1) \cdots s(n-1)), \tag{2.6}$$

where the conditional probabilities are uniquely defined by P:

$$P(x|s) = P(sx)/P(s). \tag{2.7}$$

Conversely, such conditional probabilities determine P; hence they serve as generators for P. When we construct models for sources, the information source function P has to be generated. The conditional probabilities serve as a convenient way for doing this as discussed further in the next sections.

III. MODELING

The discussion of an abstract information source in Section II does not say how to obtain or calculate the function P which assigns the probabilities to the strings. The purpose of this section is to describe how such functions P can be constructed when one or more strings are given. In other words, rather than first having an information source as a generator of strings, we realistically start with a string and construct or model a source to fit it. We first consider the case where the atomic events are the original symbols and defer the more general alphabet extension case to Section IV.

A. Model Structure

The problem of obtaining the function P is basically the problem of estimating P from the data provided by the given string or strings. As in any estimation, we somehow must select a structure within which the numerical values determining P can be estimated. A quite natural structure is suggested by (2.6) and (2.7) that define P in terms of the conditional probabilities. Conditional probabilities concern events with context formalized as follows. Let f: $S^* \rightarrow Z$, $Z = \{0, \cdots, K-1\}$, be a general recursive function which partitions the set of all strings into K equivalence classes or "contexts". We call f a *structure* function, and the equivalence classes (*conditioning*) *classes.*

The conditioning classes will be used as a means to simplify the estimation of the conditional probabilities $P(x|s)$ of (2.7). Depending on how the estimation is to be done, we distinguish between two basic cases: the adaptive and the nonadaptive models discussed in detail in the subsequent two subsections. Here we merely describe the essential features of these two cases which are pertinent to the structure of models. In the nonadaptive models, the dependency of the conditional distribution on s is restricted to the conditioning class $f(s)$ determined by s, which means that we only need to estimate the numbers $P(x|f(s))$. If d is the size of the alphabet, this involves the estimation of $K(d-1)$ numbers, because for each class the d probabilities add up to one. In the adaptive models we permit $P(x|s)$ to depend both on $z=f(s)$ and on the sequence of the "next" symbols at the previous *occurrences* of the conditioning class z in the string. To make this more precise, consider a string $s=s(1)s(2)\cdots s(t)$. Let $f(s(1))\cdots s(i)) = z$ be the first occurrence of class z, $f(s(1)\cdots s(j)) = z$ the second, and so on. Denote by $s[z]$ the string $s(i+1)s(j+1)\cdots$ of the "next" symbols at the successive occurrences of class z in the string s. Then in adaptive models we put

$$P(x|s) = P(x|z, s[z]), \tag{3.1}$$

where $z=f(s)$. The nature of the function P will be specified in Subsection III-C. Clearly the nonadaptive models form the special case where $s[z]$ is taken as the null string, and we write $P(x|z, \text{null}) = P(x|z)$.

The structure function and the probability assignment define the source $<S^*, P>$, where P is obtained by

$$P(sx) = P(s)P(x|f(s),s[f(s)]), \quad P(\text{null})=1. \tag{3.2}$$

The term (*recursive*) *model* is now refined to mean the set of strings S^* together with the structure function f and the algorithm specifying the conditional distributions (3.2). Often we do not distinguish between a model and the source induced by it.

We define the *complexity* of the model to be the number $K(d-1)$. In the nonadaptive models, the complexity is seen to be the number of free parameters required to specify the conditional probabilities $P(x|z)$. Another related way to view the complexity is to imagine that each probability is to be written with, for example, q binary digits. Then it takes $K(d-1)q$ binary digits to specify P if we exclude the lengths of the algorithms needed to describe f and the product (3.3), which may be regarded as overhead costs shared by all the strings to be encoded for which the same structure function is being used. In contrast, these $K(d-1)q$ bits of information must be communicated to the decoder for each string. In adaptive models, these parameters need not be sent to the decoder, because there is an algorithm calculating values for each symbol along the string. The decoder needs $K(d-1)$ registers, however, each having width q, to store these parameter values for decoding the symbol. In either case $K(d-1)$ is seen to be an appropriate measure of the complexity in implementing the coding system based on these models.

The structure function permits one to model an information source schema for a set of strings that are similar in some respects. Each string will get its own information source defined by the common structure function and the probabilities (3.2) tailored to that string. This is very important because often such natural sets of strings cannot be adequately modeled by a fixed information source. An example which the authors have worked on quite extensively [9], is the set of black and white scanned documents. There obviously is something that such documents have in common; for instance, text documents have a characteristic of their own due to the predominance of printed letters in their makeup which differs from that in, for example, black and white photographs. The purpose of the structure function is to capture such characteristic features. Exactly how this is to be done is a different matter; a consolation however, is that to find an optimum structure function is an undecidable problem, as can be shown by arguments similar to those used by Kolmogorov [10].

A familiar and attractive subclass of recursive sources is obtained by restricting the structure function so that it is definable by a *finite state machine* (FSM) as follows:

$$\begin{aligned} x(t) &= F(x(t-1),s(t)), \qquad x(0) = a, \\ z(t) &= G(x(t)), \end{aligned} \tag{3.3}$$

where $x(t)$ is a *state* from, for example, the set $\{0, \cdots, N-1\}$, $z(t)$ is a conditioning class in Z, and the input $s(t)$, an element of the alphabet S, denotes the successive symbols in $s = s(1)\cdots s(n)$. The output $z(n)=G(x(n))$, resulting from the terminal state $x(n)$ where the machine stops after s has been processed, defines the value of the structure function f at s.

The purpose in separating the conditioning classes from the internal state is to permit a large number of states and still have a relatively small number of classes and hence a small number of free parameters in the source: namely, $K(d-1)$. An example of this is the FSM model used in black and white image processes, where the state $x(t-1)$ is taken as the string $s(t-1)\cdots s(t-M)$ with M as large as 2000. The output $G(x(t-1))$, on the other hand, is the binary number $z(t-1)$ defined by the substring $s(t-1)s(t-M+1)s(t-M)$. In geometric terms, $s(t-1)$ is the symbol preceding the "current" symbol $s(t)$, $s(t-M+1)$ is the symbol above $s(t)$ in a two-dimensional raster scan of the

image, and $s(t—M)$ is the symbol nearest to $s(t)$ in the upper left direction. These three symbols can be expected to have a strong influence on the value of the current symbol $s(t)$ and hence cause the conditional probability $P(s(t)) = 0\backslash z(t\text{-}1))$ to be far away from 1/2.

B. Nonadaptive Models

For a given structure function f and string s, the conditional probabilities $P(x\backslash z)$ can be determined in such a manner that the probability $P(s)$ of (3.2) is maximized, and hence the ideal length $-\log P(s)$ is minimized. It is readily shown that such optimizing probabilities are given by the ratios

$$P(x|z)=c(x|z)/c(z), \tag{3.4}$$

where $c(z)$ denotes the number of times class z "occurs" in the string s, and $c(x|z)$ denotes the number of times the "next" symbol at these occurrences is x. More precisely, $c(x\backslash z)$ denotes the number of times the following pairs of equalities hold:

$$\begin{aligned} &f(\text{null}) = 0, && \text{and} \quad s(1)=x,\\ &f(s(1))=z, && \text{and} \quad s(2)=x,\\ &\quad\vdots\\ &f(s(1)\cdots s(n\text{-}1)) = z, && \text{and} \quad s(n)=x, \end{aligned}$$

and $c(z)$ is the sum of the $c(x|z)$ over the symbols x in S. Here we have arbitrarily set the value of $f(\text{null})$ to be zero.

The minimized ideal codelength itself is given by

$$I(s)=\sum_{z=0}^{K-1} c(z)\log c(z)-\sum_{z=0}^{K-1}\sum_{x\in S} c(x|z)\log c(x|z). \tag{3.5}$$

Observe that this expression is also the same as the entropy of the random variable whose outcomes are the strings of the same length as s in the just-optimized source. So in this sense we may perfectly well speak about the entropy of the string s. We illustrate nonadaptive models with an example.

Example 1: For the string $s = 00011010100101$, put $S=\{0,1\}$ and count the symbols conditioned on the previous symbol; i.e., model this as a first-order Markov source. With the initial state as zero we get the conditional counts

$$c(0|0) = 4, \quad c(1|0) = 5,$$
$$c(0|1) = 4, \quad c(1|1) = 1.$$

Put $P(0|0) = 4/9$, and $P(0|1) = 4/5$, and define $P(s)$ recursively by (3.2). We find that the ideal codelength with this model is

$$I(s) = 9\log 9 + 5\log 5-(5\log 5 + 8\log 4) = 12.53,$$

where the result is given to two decimal places. Two parameters are required to describe P.

C. Adaptive Models

From a practical standpoint the main shortcoming of the nonadaptive recursive models is the fact that the string must be prescanned by the encoder to get the optimum probabilities (3.4). Moreover, these or their equivalents must be transmitted to the decoder as a preamble in the code string. To avoid a prescan and the preamble, as well as to provide an opportunity to adjust the probabilities as the string is being encoded, we consider adaptive models in this section. The main problem here is to define the currently prevalent diffuse notion of adaptation in a meaningful manner.

In Subsection III-A we showed how the structure function permits consideration of the conditional probabilities of the form $P(x|z, s[z])$, where $s[z]$ denotes the string of the "next" symbols at the successive past occurrences of the class $z=f(s)$. Here s refers to a growing prefix of the string to be encoded that enables the decoder to decode the symbols shortly after they have been encoded.

The fact that the estimates of the conditional probabilities are to be made from the processed portion s of the string implies certain natural restrictions on the recursive function P performing the estimation. For instance, suppose that we have estimated the probabilities $P(x|z, s[z])$ at class $z=f(s)$ from the past string $s = s(1)\cdots s(i)$, and we observe $s(i+1)=y$. Let the next occurrence of the class z be $z=f(s(1)\cdots s(t))$. Because the latest observed symbol at class z was y, and we are not supposed to have information about the string other than what can be extracted from the scanned string $s' = s(1)\cdots s(t)$, we must not decrease the old estimate; i.e., we must require that

$$P(y|z,s[z])<P(y|z,s'[z]), \tag{3.6}$$

for all y, where the symbol immediately following s is y, and s' is the shortest extension of s such that $f(s')=z$. This condition serves as a criterion for what we mean by *adaptive* recursive models, provided that the adjustments of the probabilities are done at every occurrence of class z. For practical reasons, however, it may be desirable to perform the adjustments only after certain sets of observations are made, which is why we wish to modify the criterion accordingly.

Let an adjustment of the conditional probabilities at a class z be made after the string s has been received; i.e., $f(s)=z$, and let the next adjustment at the same class be made after s' has been received, hence $f(s')=z$. Of course, a rule is required to specify what these adjustment points $s, s', \cdots$, are. Suppose that in the examination interval, i.e., in the segment between s and s', the symbol y was observed in class z, a total of $c(y|z)$ times. Let

$$c(z)=\sum_{y\in S} c(y|z).$$

The general adaptation criterion can now be expressed as follows. If

$$c(x|z)/c(z) \geqslant P(x|z,s[z]),$$

then

$$P(x|z,s[z]) \leqslant P(x|z,s'[s]). \tag{3.7}$$

As a matter of notation we write $P(y|s)$ if the adaptive model has only the trivial conditioning class S^*. A simple

example of adaptive recursive sources is obtained by putting

$$P(y|s) = c(y|s)/|s|,$$

where $c(y|s)$ denotes the number of times the symbol y occurs in s and $|s|$ denotes the length of s. Clearly (3.6) holds. Fairly sophisticated adaptive sources for black and white documents are constructed in [9] with performance quite superior to that of the nonadaptive ones of the same complexity.

IV. ALPHABET EXTENSION

Frequently the symbols in which the string to be encoded has originally been given are extended to certain segments of the primary symbols. This is done in particular for small alphabets in order to achieve a better per-symbol codelength. Another reason is the fact that the per-symbol entropy of a source decreases as the size of the groups is increased; a frequently quoted example of this is the family of sources for an English text resulting when first the individual letters, then every pair, and finally all words are regarded as independent. In fact, there is a strong feeling that a suitable grouping of the symbols captures certain natrual features of the strings, and thereby a low entropy source should result.

For the purpose of the main theorem in the next section we describe a quite general alphabet-extension process. For a d-element alphabet S, let $T(0)$,• • •, $T(m - 1)$ denote the subsets of S^* such that: 1) in each $T(i)$ no string is a substring of another (the prefix property); and 2) the subtree of S^* defined by $T(i)$ with its elements as the leaves is complete. Further, there is defined an m-state machine with state space $\{0, • • •, m - 1\}$ whose input space T is the union of the $T(i)$. The state transitions are defined by a function $R(w, t)$, where w is a state, and t is an (extended) symbol in $T(w)$.

A string s in S^* is *parsed* into segments as follows. Starting at the initial state zero, a string $t(0)$ in $T(0)$ is recognized by the prefix property as the first segment in s. The machine moves to the next state $R(0, t(0))$, and the process is repeated with $s=t(0) • • • t(k)s'$ as the result. The last segment s' is a proper prefix of some of the symbols in the final tree determined by the final state $R(k, t(k))$. To simplify matters we "pad" such leftovers to form a complete extended symbol in the last tree, so that s' is null, and we regard s to be a sequence in T^*, i.e., a sequence over the extended alphabet.

Example 2: For a binary alphabet, let $T = T(0) = \{1,01,00\}$. Observe how these symbols appear as leaves in a binary tree as illustrated in Fig. 1. The leaves represent "runs" of symbol zero of lengths zero and one, while the last symbol is used to describe runs of any length. These symbols occur in the string s of Example 1, a total of two, four, and two, times, respectively, which gives the probabilities $P(1) = 1/4$, $P(01) = 1/2$, and $P(00)=1/4$ for the symbols. Extend these to strings by independence, which defines a nonadaptive run-length model. We get the ideal

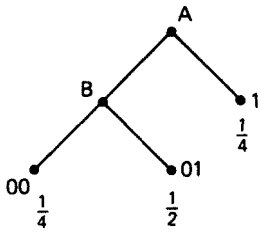

Fig. 1. Binary tree for run-length source.

codelength for the string s of Example 1 to be

$$I(s) = 81og8 - (41og4 + 41og2) = 12.$$

Two parameters are again enough to describe P, so this model is better for the given string than the one in Example 1.

Example 3: This is an example of a double run-length extension. There are four extended symbols for runs of zero and three for runs of one. Let $T(0)= \{1,01,001,000\}$, and $T(1) = \{0,10,11\}$. Put $R(0,1)=R(0,01) = R(0,001)=1$, $R(0,000)=0$, and $R(1,0)=R(1,10)=0$, $R(1,11)=1$. When starting at state zero, the string s=000110101001011 parses into the segments 000, 1, 10, 1, 0, 1, 001, 0, 11.

The recursive models of Section III can also be constructed for extended alphabets. The states of the machine affecting the alphabet extension are then included as a component of the equivalence classes, so that the structure function maps T^*, i.e., the set of all strings in the extended alphabet, *into* $\{0, • • •, K-1\}$ x $\{0, • • •, m - 1\}$. This means that we may write the structure as (f, g), where f: T*→ $\{0,• • •, K\text{-}1\}$ is any recursive function as discussed in Section III, and g: T^*→$\{0,• • •, m - 1\}$ takes each parsed sequence $s = t(0) • • • t(k)$ to the terminal state $w=g(s) = R^*(0, s)$, where the machine stops after receiving the successive symbols in s when started in the initial state zero.

Because the symbol following the last read-in symbol $t(k)$ in s must be taken from $T(w)$, which is determined by the last state $w=R^*(0, s)$, the parameters in the extended model are the conditional probabilities

$$P(t|(z,w), s[z,w]), \quad t \text{ in } T(w), \qquad (4.1)$$

in which $z=f(s)$, $w=g(s)$, and $s[z,w]$ denotes the sequence of the past occurrences of the "next" symbols in T at the conditioning class (z, w) as explained in Subsection III-A. These, of course, must satisfy the condition (3.6).

If $T(w)$ has $D(w)$ symbols, then $D(w)$ — 1 probability parameters are required for every pair (z, w). Hence the total number of parameters in an extended model is given by

$$\sum_{w=0}^{m-1} (D(w)-1)K(w), \qquad (4.2)$$

where $K(w)$ denotes the number of classes in the set

$$\{f(s)|g(s) = w, s \text{ in } T^*\}.$$

We illustrate these notions with an example.

Example 4: We construct a first-order Markov model of the extended symbols in Example 3, i.e., of the runs of

zero, $T(0)$, and the runs of one, $T(1)$. We have to construct the overall structure function (f, g). The two-state machine in Example 3 defines the second component g by $g(s) = R^*(0, s)$, where $s = t(0) \cdots t(k)$. The first component f is defined by $f(t(0) \cdots t(k)) = t(k)$. We next define the conditional probabilities (4.1). For the symbols in $T(0)$, define the conditional probabilities at each conditioning class $(f(s), g(s))$ = (preceding symbol, state) by the following table.

	1	01	001	000
(0,1)	1/2	1/4	1/8	1/8
(10,1)	1/4	1/4	1/4	1/4
(000,0)	5/8	1/8	1/8	1/8

Observe that only the symbols 0, 10 and 000 can possibly precede those in $T(0)$. Hence $K(0)$ in (4.2) is three. Similarly, only the symbols 1, 01, 001, and 11 can precede those in $T(1)$, and we define the probabilities for $T(1)$ by

	0	10	11
(1,0)	1/2	1/4	1/4
(01,0)	3/4	1/8	1/8
(001,0)	2/3	1/6	1/6
(11,1)	1/8	5/8	1/4

The total number of free parameters in this model is 9+8=17.

V. MAIN THEOREM

Markov models of some order k, such as illustrated in Example 1 for $k=1$, and block models with block length k illustrate the familiar tug of war between the two techniques in modeling, Markov conditioning, and blocking. Without imposing a cost on the complexity of the model, there is no winner, and even with the cost introduced above, the best type of model depends on the string to be encoded. However, if we widen the Markov models to the set of recursive models, either adaptive, nonadaptive, or both, then we can say something about the best way to do modeling. At the same time we may include in the contest any alphabet extension of the kind described above rather than just simple blocking.

Theorem 1: For every adaptive or nonadaptive recursive model using alphabet extension, there exists another adaptive or nonadaptive recursive model respectively, using no alphabet extension, which has the same number of parameters (and hence requiring the same number of binary digits for its description), and which has the same ideal codelength $-\log P(s)$ for every string s. The converse is not true.

Remarks: The meaning of the converse part of the theorem is that there exists an FSM binary source such that no adaptive or nonadaptive source satisfying the condition (3.7) exists using any kind of alphabet extension, which can produce the same or better ideal codelength with the same number of parameters. Hence, although stationary binary sources can be simulated by extended sources, the required source must have a greater complexity.

Proof: The heart of the proof is to show that the process affecting the alphabet extension can be described or simulated by a finite-state machine model. Before showing this in full generality required in the theorem, we illustrate the simulation process with a simple example. We simulate the run-length model of Example 2 with a two-state FSM model which therefore will also have two parameters.

Assign one state A, which is taken as the initial state, to the set of nodes consisting of the root and the leaves of the tree in Fig. 1, and another B to the node also marked B. Define the state transition function as follows:

$$F(A,0) = B, \quad F(B,0) = A,$$
$$F(A,1) = A, \quad F(B,1) = A.$$

Assign the two conditional probabilities as follows:

$$P(0|A) = P(B) = P(00) + P(01) = 3/4,$$
$$P(0|B) = P(00)/P(B) = (1/4)/(3/4) = 1/3.$$

Extend these conditional probabilities to the strings in S^* by the independence of the transitions:

$$P(sx) = P(s)P(x|z(s)), \quad x=0,1,$$

where $z(s)$ is the state which the machine reaches when started at state A and given the string s. Observe how each extended symbol always takes the machine to the state A. Because of this and the way the conditional probabilities were chosen from the tree, every string has the same probability and hence the same ideal codelength with this model as with the run-length model.

We now describe the general construction. The given extended model assigns the probabilities to the strings of T^* by the recursion

$$P(st) = P(s)P(t|(z,w),s[z,w]), \qquad (5.1)$$

where $s = t(0) \cdots t(k)$, t is an extended symbol in $T(w)$, $z = f(s)$, and $w = g(s)$. This notation is explained further in Section IV.

In order to simulate this model, we first construct for each tree $T(w)$, an FSM model under which every extended symbol t (leaf) gets the correct probability $P(t|(z,w),s[z,w])$. The construction was illustrated above in a special case; the general case is quite similar. We start by assigning a state marked "null" to the set of nodes in $T(w)$ consisting of the root and the leaves. Further, we assign one state to each internal node. If $x(1) \cdots x(j)$ denotes the path from the root to an internal mode, then we denote the corresponding state by this path. Hence the states are in one to one correspondence with the proper prefixes of all strings in $T(w)$. The state transition function of this machine is simply given by

$$(u,x) \dashrightarrow ux, \qquad (5.2)$$

where u is a proper prefix of a string in $T(w)$ and x a

symbol in S; the right side becomes the state "null" if ux is a leaf in $T(w)$.

We next combine these FSM's with the extended model to form a recursive source with alphabet S. The new structure function is

$$su\text{-->}(f(s), g(s), u),$$

where su is any string in S^*, s is the unique prefix of su which is in T^*, and hence either "null" or a string of the form $t(0)\bullet\bullet\bullet t(k)$, and u is a proper prefix of some symbol in $T(w)$.

We need to define the conditional probabilities

$$P(x|(z,w,u), su[z,w,u]).$$

We define them as follows:

$$P(x|(z, w, u), su[z, w, u]) = \frac{\text{probability of node } ux \text{ in } T(w)}{\text{probability of node } u \text{ in } T(w)}, \quad (5.3)$$

where $z=f(s)$, $w=g(s)$, u is a proper prefix of some string in $T(w)$, and x is symbol of S. Further, the probability of a node in the tree $T(w)$ is defined to be the sum of the probabilities of its descendant leaves. These conditional probabilities generate the probability of any string su in S^* by the rule (3.2), which says

$$P(sux) = P(su)P(x|(z,w,u), su[z,w,u]). \quad (5.4)$$

It follows from (5.3) and (5.4) that this probability agrees with that in (5.1) for every string in T^*.

The tree $T(w)$ has $(D(w)\text{-}1)/(d\text{-}1)$ internal nodes. Hence the FSM constructed for $T(w)$ has the same number of states, and for each z and w the number of free probability parameters $P(x|(z, w, u), su[z, w, u])$, one of each state, is $(d - 1)(D(w) - 1)/(d - 1) = D(w) - 1$. This is the same as the number of free parameters in the extended model for each z and w. Accordingly, we have completely simulated the extended model by a nonextended one, which proves the first part of the theorem.

It remains for us to exhibit a nonextended model that cannot be matched by any extended one with the same number of free parameters. It is in this part that condition (3.7) is needed. Consider the two-state binary source where at state A the next state for symbol zero is A, and for symbol one is B. At state B the next state for symbol zero is B and for symbol one is A. The symbol probabilities are $P(0|A) = 3/4$ and $P(0|B) = 1/2$.

Any extended source with two free parameters must have three symbols as leaves in a complete binary tree. Hence they are either 1, 00, and 01, or the binary complements of these, for example, the former. The probabilities $Q(1|s)$ and $Q(00|s)$ assigned to the first two symbols may depend on the past string s, and they determine the third symbol probability.

Consider the three strings 1, 00, and 01. The binary source with A as the starting state assigns to these the probabilities 1/4, 9/16, and 3/16, respectively. To match these, the extended source must have

$$Q(1|\text{null}) \geq 1/4,$$
$$Q(00|\text{null}) \geq 9/16,$$
$$Q(01|\text{null}) \geq 3/16,$$

which can be satisfied only when the equalities hold. Next, consider the strings 11, 100, and 101. To these the binary source assigns the probabilities 1/8, 1/16, and 1/16, respectively. Again, to match these the extended source must have $Q(100) = Q(1|\text{null})Q(00|1)=1/16$. Finally, consider the strings 1001, 10001, and 10000, to which the binary source assigns the probabilities 1/32, 1/64, and 1/64, respectively. To match these we must have $Q(00|100) \geq 1/4$, $Q(01|100) \geq 1/4$, and $Q(1|100) \geq 1/2$. If this adjustment is made, then we have a contradiction to (3.6), because $Q(00|\ 100) = 1/4 \leq Q(00|\text{null})$. On the other hand, if we keep the value 9/16 for $Q(00|100)$, as we are allowed to do within the wider notion of adaptation according to (3.7), then, $Q(1|100)$ must be chosen smaller than 1/2. But then the largest probability we can generate for the string 1001 is smaller than 1/32, or less than what the binary source assigns to the same string. Hence, regardless of when the adjustments of the probabilities are made, we cannot match the performance of the binary source. This completes the proof. □

Remarks: The first implication of this theorem is that an alphabet extension has *no* inherent value; absolutely nothing is lost if we confine the search for a good model to the class of recursive d-ary sources without extending the alphabet. This is true if the performance is measured in terms of available compression; if we include other performance measures such as the speed with which the coding operations can be performed, alphabet extension may well have advantages. Secondly, for any set of strings arising in real applications, a given number of parameters is strictly better spent in letting them describe probabilities in the d-ary model than wasting them in the description of the probabilities of the extended symbols, no matter what these symbols are. This result may be somewhat counter intuitive, because one is tempted to believe that by picking the extended symbols in a 'natural' way an economical usage of the available parameters results. Actually, there is a bit of truth behind this belief since if the symbols are perfectly independent, then the resulting model cannot be improved. However, there still is an equally optimum d-ary model. More important, in 'real' strings where the new symbols as random variables are not independent, better d-ary models with the same number of parameters can be found.

Finally, in some respect the most devastating implication of this theorem is that it makes the use of all the traditional codes in conjunction with the best models cumbersome and wasteful, above all when the original alphabet size is small. This is because an alphabet extension is needed for the code to have a near-optimal length. To illustrate the inherent difficulty, suppose that a binary adaptive model assigns the probability $P(0|s)$ for the

symbol following s to be zero, and we wish to design a ran-length code for this source with the symbols $1,01,\cdots,0\cdots01,0\cdots0$, up to the length n. Suppose that the string to be encoded is $s001\cdots$, where the last symbol of the so far encoded string s is a one. The task then, is to encode the run 001 following s. If the encoder wishes to use a compact code, i.e., a Huffman code, he must first work out the probabilities of *all* the runs with help of the binary model

$$P(1)=P(1|s),$$
$$P(01)=P(0|s)P(1|s0),\cdots$$
$$P(0\cdots0)=P(0|s)\cdots P(0|s0\cdots0).$$

Only then can he construct the optimum codeword for the run 001 actually occurring, the calculation of all the other probabilities is wasted. The same is true for the decoder. Because of the adaptive nature of the run probabilities, this entire optimum-code tree can be used only once! It is easy to show that this problem is not peculiar to the Huffman algorithm, which proceeds backwards. There is no forward-progressing algorithm for the construction of the compact codes: *all* probabilities are needed for the construction of even the very first codeword. It is obviously possible to construct forward moving ad hoc code trees, but they are not generally optimal.

VI. CODING UNIT

In our view all coding should be a matter of representing sequences as numbers in an order-preserving manner. The sequences are then thought to have a natural order such as the lexical one, and the numbers of course, are ordered by magnitude. The most familiar example of such coding is the decimal number representation of decimal sequences. The decimal string 3407 can be represented or encoded as the integer $7+4\text{x}10^{2}+3\text{x}10^{3}$, or as the fractional number $3\text{x}10^{-1}+4\text{x}10^{-2}+7\text{x}10^{-4}$, precisely because these codes preserve the lexical ordering of the decimal sequences. Similarly, in enumerative codes ([4] and [5]), in which a sequence is encoded as its index in a certain lexically ordered set, the order is evidently preserved, and the same is true in the closely related Elias code, which in fact can now be viewed as a dual of the enumerative codes [11].

When the numerical coding of strings is done for compression purposes the strings to be compressed are normally long, and the numbers used for their representation are very large indeed, requiring hundreds of thousands of binary digits. It is clear that in the generations of such numbers or codes, special attention must be paid to the question of how to do the calculations in normal-size registers. This problem was addressed and solved in [1] with the first "arithmetic code" as the result. The constructed code was of the LIFO type in which the last encoded symbol was decoded first, i.e., the code string medium was of the nature of a pushdown storage. Subsequently Pasco [2] considered the same problem for Elias' code [6] and constructed a FIFO arithmetic code which decoded the symbols in the same order in which they were encoded. One advantage of FIFO codes is that the decoding process can be started almost immediately after the first symbol has been encoded; that is to say, 'instantaneously' in the practical sense that only a few adjacent symbols need be altered when a symbol is encoded or decoded. Evidently the same criterion for instantaneousness is applicable even to LIFO codes. We should add that because of a carry-over problem, which is inherent in FIFO arithmetic codes, Pasco's code is not instantaneous in our sense. In fact, he specifically wanted a prefix block code. Another class of LIFO and FIFO codes has been discussed in detail in [3], and recently, further versions of Pasco's type of FIFO codes were rediscovered by Jones [7] and Martin [8] both with knowledge of and reference to the original LIFO code in [1].

In view of the several different versions of FIFO codes proposed by the authors referred to previously, it seems appropriate to introduce arithmetic codes from an abstract number representation viewpoint. The number representation viewpoint is natural and free from specific implementation details, and it is sufficiently general to include all the known arithmetic codes as special cases.

In the decimal representation of decimal strings, each term added to the code depends both on the symbols's position in the alphabet and on its position in the string; for instance, the fourth symbol "7" in the above given example string, encodes in the fractional representation as the fourth power of 1/10 multiplied by seven. In arithmetic coding this same principle is generalized. We discuss here the FIFO codes because of their practicality. For each symbol k of the alphabet $S=\{0,\cdots,d-1\}$ as the next symbol to the string s (yielding string sk), we add a term, $B(sk)$, called *augend*, which depends on the symbol's position k in the alphabet as well as on s, rather than merely on the symbol's position in the string. With $C(s)$ denoting the code of string s as a number, we then have the recursion

$$C(\text{null})=0,$$
$$C(sk)=C(s)+B(sk). \tag{6.1}$$

A convenient way to encode k for a given s is to make $B(sk)$ an increasing function of k starting from $B(s0)=0$. With this convention we can write it as

$$B(sk)=A(s0)+\cdots+A(s(k\text{-}1)), \tag{6.2}$$

where the increments $A(si)$ are called *addends*. Because the right-most zeros in the strings cannot be decoded, we append a dummy nonzero symbol to the end of the strings to be encoded.

The decoding is done by magnitude comparison with the following rules. Let s' run through all proper prefixes of s starting at null string. The symbol following s' is decoded as that index k for which

$$\text{Rule 1:}\quad C(s)\text{-}C(s')\text{-}B(s'k)\geq 0,$$
$$\text{Rule 2:}\quad C(s)\text{-}C(s')\text{-}B(s'(k+1))<0;$$

the second condition is not needed if Rule 1 holds for $k=d\text{-}1$.

Because we have not yet added any precision requirements to these codes, we may call them "pre-arithmetic" codes. We have deliberately removed the dependency of the addends and the augends of any given source probabilities, as opposed to Elias' scheme, to achieve greater freedom in their selection. It then becomes important to understand when such codes are decodable; which issue we will settle next. The main decodability theorem for the class of FIFO codes (6.1) is as follows.

Theorem 2: All strings with the right-most symbol nonzero can be decoded by Rules 1 and 2 if and only if for all $s=s'k$, k not $d-1$,

i) $A(s) > \lim[B(s(d\text{-}1)) + B(s(d\text{-}1)(d\text{-}1)') + \cdots]$.

If $A(\text{null}) = 1$, then i) implies $C(s)<1$. In particular, decodability is implied by

ii) $A(s) \geq B(sd)$

$$= A(s0) + \cdots + A(s(d\text{-}1)) > 0, \text{ for all } s.$$

Proof: We show that ii) implies i), and i) implies the decodability. By (6.2)

$$B(sd) = B(s(d\text{-}1)) + A(s(d\text{-}1)) \geq B(s(d\text{-}1)) + B(s(d\text{-}1)(d\text{-}1)) + A(s(d\text{-}1)(d\text{-}1)),$$

which by iteration and by one more application of ii) gives i). Let $s = s'(d\text{-}1)s''$. By (6.1), (6.2) and the fact that addends are positive, Rule 2 of the decoding process fails for $k<d-1$ while Rule 1 holds for $k=d-1$. Hence d-1 is decoded correctly. Let $s = s'ks''$ for k not d-1. By (6.1)

$$C(s) - C(s') \geq C(s'k) - C(s') = B(s'k),$$

and Rule 1 holds. Consider

$$C(s) - C(s') - B(s'(k+1)) = -A(s'k) + B(s'kj) + \cdots,$$

where j is the first symbol of s'', or s'' is null. By an application of i) to $A(s'k)$ we see that Rule 2 holds, and k gets decoded correctly.

Suppose next that i) fails; i.e.,

$$A(s) < B(s(d\text{-}1)) + \cdots + B(s(d\text{-}1)\cdots(d\text{-}1)),$$

m repeated symbols $d-1$, for some $s=s'k$, $k<d-1$. Pick $s''=s'k(d-1)\cdots(d-1)$, where the number of repeated symbols is m. Then

$$C(s'')-C(s')-B(s'k) = B(s'k(d\text{-}1)) + \cdots + B(s'k(d\text{-}1)\cdots(d\text{-}1)) \geq 0$$

$$C(s'') - C(s') - B(s'(k+1)) = -A(s'k) + B(s'k(d\text{-}1)) + \cdots + B(s'k(d\text{-}1)\cdots(d\text{-}1)) > 0.$$

Hence Rule 1 holds, and Rule 2 fails, which means that k will not be correctly decoded. The proof is complete. □

Remark: Observe that the equality in ii) is just an instance of (2.1), and the code matches a source precisely if $A(s) = P(s)$.

For practical reasons we wish to put further restrictions on the augends. Above all, we require that the addition in (6.1) must be done in a fixed-size register, and it must not affect more than a fixed number of digits in $C(s)$, when the latter is written for example in a binary notation. In FIFO codes this amounts to the requirement that the augends are added to the right end of the code string and they can have no more than, for example, r significant floating-point binary digits; i.e., there are $r-1$ digits following the first one. Such "instantaneous" codes are called (*proper*) *arithmetic codes.*

The purpose of using arithmetic codes is to obtain compression, which is possible only if the augends are selected appropriately. Ideally, the length of $C(s)$ should be $-\log P(s)$, see Section II, where we assume that the string is taken from an information source with a probability function P. But because the length of the code string is nearly the same as that of the last added augend, the leading zeros included, that has no more than r significant floating-point digits, it follows that the $A(sk)$ must be approximately $P(sk)$. Observe that this argument relies on the assumption that the addends and the augends have a fixed maximum number of significant floating-point digits only. We give now two examples of classes of arithmetic codes.

Example 5: Let

$$A(sk) = [A(s)p(k|s)]' \tag{6.3}$$

where $[x]'$ denotes the number obtained when the binary number x has been truncated to r significant floating-point binary digits, and $p(k|s)$ is a number also with no more than r significant floating-point digits satisfying the condition

$$p(0|s) + \cdots p((d\text{-}1)|s) < 1, \tag{6.4}$$

for all s. Ideally, $p(k|s)$ should be taken as the conditional probability of the symbol following s being k given s.

The code (6.1)-(6.3) is a modification of Pasco's code. One difference is that here the precision of all the parameters $p(j|s)$ is the same, and the addition in (6.1) can be made in the same size register, namely r, as that needed for the calculation of the addends, provided though, that we add the individual $A(si)$ to the code one at a time rather than first collecting them to form the augend (6.2).

Example 6 (N. Martin): Let $p(i|s)$ be the numbers having r-1 fractional digits which satisfy (6.4) with equality, and put

$$P(k|s) = p(0|s) + \cdots + p(k\text{-}1|s), \quad P(0|s)=0.$$

Clearly the numbers $P(k|s)$ also have $r-1$ fractional digits. Suppose $A(s)$ is a number with $r-1$ digits following the first one (the last may well be a zero). The product $A(s)P(k|s)$ has no more than $2r-1$ significant floating-point digits, and because the smallest $p(i|s)$ is at least 2^{1-r}, the leading one of this product remains within the range of the significant digits of $A(s)$. Let $[A(s)P(k|s)]''$ denote the truncation of the product to the range of the significant digits of $A(s)$. In other words, the length of the truncated product is the same as the length of $A(s)$. As an example let $r=5$, $A(s) = 0.00010100$, and $P(1|s) = .0110$. Then $[A(s)P(1|s)]'' = 0.00000111$. Now put

$$A(sk) = [A(s)P(k+1|s)]'' - [A(s)P(k|s)]''.$$

Clearly, each $A(sk)$ has no more than r significant digits, and ii) in Theorem 2 holds with equality.

Remarks: For a binary alphabet these examples simplify, For instance, in Example 6 we may write the param-eters $p(i|s)$ with a maximum of r significant floating-point digits, and still maintain the same for $A(s)$ provided that we keep track of the symbol which has the higher conditional probability. Let x be that symbol and x' the other symbol. Put $A(sx) = [A(s)p(x|S)]'$, truncated to r significant floating-point digits, and $A(sx') = A(s) — A(sx)$. $A(s)$ will then have no more than r significant floating-point digits. Finally, and more importantly, the multiplication involved in these codes can be avoided with only a small loss of compression. One such scheme is described in [9]. In the nonbinary case, again the multiplications can be avoided by constructing the codes from length parameters rather than from the probabilities $p(k|s)$ as discussed in [3].

It is clear from these remarks that there is great flexibility in the design of arithmetic codes, with ample room for engineering trade-offs, which in view of the great variety of coding needs is highly valuable. Several questions of both conceptual and practical nature arise in this number representation view of coding. For instance, we would like the binary strings that result when the fractional numbers $C(s)$ are first written in binary notation and then the binary point deleted, to fill the tree defined by all binary strings terminating at a one. It is easy to show that this certainly is not achievable unless perhaps the equality in ii) holds. Hence the code in Example 6 is a good candidate for such an onto or "almost" onto map.

We illustrate the coding operations by a specific instance of the code in Example 5. Let $d = 3$, and $s = 0212$. Let $p(0|\text{null}) = p(1|\text{null}) = 0.011$, $p(2|\text{null}) = 0.010$, $p(0|0) = 0.011$, $p(1|0) = 0.001$, $p(2|0) = 0.1$, and for the remaining prefixes s' let $p(0|s') = 0.01$, $p(1|s') = p(2|s') = 0.011$. With r =3 we get the following table.

s	$A(s)$	$B(s)$	$C(s)$
0	0.011	0	0
2	0.00110	0.00110	0.00110
1	0.000100	0.0000110	0.001111
2	0.00000110	0.00000100	0.01

This example also illustrates the *carry-over problem,* which causes only the first trivial prefix code, namely, zero to be a prefix of the final code.

We sketch a solution to the carry-over problem. After an agreed number of consecutive ones, say t, has been detected in the code string, the remaining symbols to the right of the t ones in the code string are shifted right one position and a zero is inserted in the vacated position immediately to the right of the t ones. Moreover, all the future addends are halved so as to preserve their correct position relative to the tail (the working end) of the code string. In reality the addends are added in a fixed register, and the generated symbols of the code string are shifted left out of the register, but the equations above are written as if the addends were shifted right along the code string. Because the decoding is done by magnitude comparison, the important aspect of the code generation is the relative position of the code string and the addends.

Then the decoder, when seeing t consecutive ones, removes and examines the $t + 1'$th symbol. If it is a zero, the decoding proceeds as usual, but if it is a one, a carry-over must have occurred and been stopped by the added zero. Accordingly, this one is added to the t'th one so that the carry-over one ripples through the preceding t — 1 ones. In either case one can show that the decoder has the correct code string as the result. The proof, which we omit, rests on the crucial property that once any symbol in the code string is beyond the r augend bit range (or the working end of the code string), it can receive at most one carry-in. We illustrate this by the preceding example, rewritten for $t = 2$.

s	$A(s)$	$B(s)$	$C(s)$	$C'(s)$
0	0.011	0	0	0
2	0.00110	0.00110	0.0011	0.0011
1	0.000100	0.000011	0.001111	0.0011011
1	0.00000011	0.0000001		0.00111

Here we wrote $C'(s)$ for the modified code string which the decoder receives, and after seeing two consecutive ones, converts to the original code string as described above. Finally, if the code string is a random Bernoullian sequence as it ideally should be, then the probability of having t consecutive ones is 2^{-t}, which is also the per symbol increase in the code string due to this carry-over blocking mechanism. For a typical value of t = 16, the length increase is normally quite insignificant.

If we arrange things in such a way that the last added augend is the smallest, then the length of the code string is determined by the last augend, arid the per-symbol length of the code in Example 6 satisfies

$$(1/n)|C(s)| \leq -(1/n)\log P(s) + r/n + 2^{2-r},$$

where $P(s)$ denotes the probability of the string as the product of the conditional probabilities $p(k|s)$. This inequality shows that the mean per-symbol length does not exceed, the ideal codelength defined by the conditional probabilities by more than the two right-most terms.

ACKNOWLEDGMENT

The authors are greatly indebted to Frank King for suggesting the line of research which led to the main modeling theorem in Section V. They are also indebted to Stephen Todd for stimulating discussions.

REFERENCES

[1] J. Rissanen, "Arithmetic coding of strings," IBM Res. Rep. RJ 1591, June 3, 1975. This appeared in a revised form as "Generalized kraft inequality and arithmetic coding," *IBM J. Res. Dev.,* vol. 20, No. *3,* pp. 198-203, May 1976.

[2] R. Pasco, "Source coding algorithms for fast data compression,"

Ph.D dissertation, Dept. of Elec. Eng., Stanford University, Stanford, CA, May 1976.
[3] J. Rissanen and G. G. Landgon, Jr., "Arithmetic' coding," *IBM J. Res. Dev.*, vol. 23, no. 2, pp. 149-162, Mar. 1979.
[4] J. P. M. Schalkwijk, "An algorithm for source coding," *IEEE Tram. Inform. Theory*, vol. IT-18, no. 3, pp. 395-398, May 1972.
[5] T. M. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory,* vol. IT-19, no. 1, pp. 73-77, Jan. 1973.
[6] N. Abramson, *Information Theory and Coding.* New York: McGraw-Hill, 1969.
[7] C B. Jones, "An efficient coding system for long source sequences," (submitted to *IEEE Trans. on Inform. Theory).*
[8] G. N. N. Martin, "Range encoding: An algorithm for removing redundancy from a digitized message," presented at the Video & Data Recording Conf., Southampton, July 1979.
[9] G. G. Landgon, Jr. and J. Rissanen, "Compression of black-white images with binary arithmetic coding," IBM Res. Rep., Dec. 1979 (submitted to *IEEE Trans. Common.).*
[10] A. Kolmogorov, "Three approaches to the quantitative definition of information," *Prob. Peredach. Inform.,* vol. 1, no. 1, pp. 3-11, 1965, (Russian).
[11] J. Rissanen, "Arithmetic codings as number representations," *Acta Polytech. Scandinavica,* Math. 31, pp. 44-51, Dec. 1979.
[12] S. K. Leung-Yan-Cheong and T. Cover, "Some equivalences between shannon entropy and Kolmogorov complexity," *IEEE Trans. Inform. Theory,* vol. IT-24, no. 3, pp. 331-338, May 1978.
[13] T. J. Lynch, "Sequence time coding for data compression," *Proc. IEEE* (Lett.), vol. 54, pp. 1490-1491, Oct. 1966.
[14] L. D. Davisson, "Comments on 'Sequence time coding for data compression," *Proc. IEEE (Lett.), vol. 54, p.* 2010, *Dec.* 1966.

0018-9448/81/0100-0023$00.75 ©1981 IEEE