J. Rissanen
G. G. Langdon, Jr.

Arithmetic Coding

The earlier introduced arithmetic coding idea has been generalized to a very broad and flexible coding technique which includes virtually all known variable rate noiseless coding techniques as special cases. An outstanding feature of this technique is that alphabet extensions are not required. A complete decodability analysis is given. The relationship of arithmetic coding to other known nonblock codes is illuminated.

Introduction

In the excellent textbook on coding [1], Abramson neatly splits all codes into two classes: block codes and nonblock codes. Of these he selects only the block codes as being of much use and for which the important decodability results hold. His term, block code, is characterized by the familiar process in which the code words of the symbols are concatenated to form coded messages. A further important subclass of block codes is the class of instantaneous codes for which a both necessary and sufficient condition is the prefix property: No code word is to be a prefix of another. Finally, as there is hardly any conceivable reason to use other than compact codes, which can be formed by Huffman's algorithm, it appears that all "useful" noiseless coding should be confined to Huffman codes. But that has not turned out to be the case at all, mainly because of the necessity to resort to the alphabet extension to achieve a desired compression, particularly for small alphabets. We must therefore conclude that block codes, despite their simplicity and well-known decodability theory, are unsatisfactory to cover all coding needs, especially when alphabet extension is required.

In the neglected and more diffuse class of nonblock codes, two types of codes have appeared: Elias' code [1], and the so-called enumerative codes [2, 3], which, although clearly beset by the practical difficulty of requiring unlimited arithmetic precision, have the attractive feature that no alphabet extension is needed to achieve near-optimum compression. More recently, further classes of nonblock codes were introduced by Rissanen [4] and (not independently) by Pasco [5], which have the same attractive feature of near-optimum compression and whose practicability is comparable to the best block codes. Although these codes have a number of common features, to some extent clarified by Pasco [5], they still remain as so many distinct codes, and they are totally different from the traditional block codes. We should add that in recent textbooks the notion of block code has been used whenever "blocking" is used, be it for alphabet extension or for truncation of the numbers represented by a symbol string to a manageable size. Although virtually all codes then become block codes, such a unification is meaningless because the same decodability results do not hold for, say, the recently discovered arithmetic codes.

In this paper we study coding in which the basic notions in block codes have been generalized as follows: concatenation to addition, code words to rational numbers with finitely many fractional bits, integer lengths to rational lengths, and the prefix property to magnitude order. All these were introduced in a restricted form in [4], as further discussed in the next section. The symbol and the code word selection is done either by table look-up or by a combination of table look-up and arithmetic operations. Finally the code string is generated recursively by growing it either to the left or to the right, much as in the usual block codes and Pasco's codes as well. Both these codes, called arithmetic codes, are formulated with the aid of a finite state machine and constitute the first main contribution of this paper. They include the old block codes as a proper subset, and they also specialize to the early arithmetic codes, as discussed in more detail in the next section.

The second main contribution of this paper is the construction of an independent and self-contained decodability theory for the arithmetic codes, which then covers all codes based upon the described generalized "shift and add" operations. Because of the lack of integer-length code words and the associated trees, only a fraction of the needed results could have been derived by traditional means. For this reason, we do all of the work in a novel manner.

We feel that the main impact of this study is to isolate and lay bare the common fundamental principles in all known variable rate, noiseless coding, which permits a unification of both block and known nonblock codes under arithmetic coding. In the new coding there is no longer need for such artificial *ad hoc* contrivances as alphabet extensions nor the forming of blocks to avoid the requirement of unlimited precision in Elias' code or the enumerative codes. Similarly, the recognition that the length parameters, traditionally given by the number of symbols in the code words, have an independent function contributes to a simpler and cleaner implementation of compact codes.

In what follows a considerable amount of notation is introduced. We have therefore ordered these symbols alphabetically in the Appendix, which should serve as a quick reference for the reader.

Background; coding equations

Because of the ambiguous usage of the term "block code" in the literature, we use the clumsier but quite accurate name "concatenation code" for a code which constructs the code string by concatenation of the code words, which themselves are concatenations of the code symbols. In what follows, the code strings are over the binary alphabet, and logarithms are to the base 2.

We describe first the familiar concatenation coding operations in an arithmetic manner. Let a code word $A(k)$ of length $\ell(k)$ correspond to each symbol k of the source alphabet $\langle 0, 1, \cdots, N-1\rangle$. Let $C(s)$ denote the code of the finite string s. To obtain recursively the code of the string sk, where k is the next source symbol, $A(k)$ is appended either to the left (most significant) or the right (least significant) end of $C(s)$. If the encoded string is grown to the left, $C(sk)$ can be decoded from left to right into $A(k)$ and $C(s)$ "instantaneously," provided that the code words have the prefix property. We call this case last-in, first-out (LIFO) decoding. Dually, when $A(k)$ is appended to the right, the prefix property enables a first-in, first-out (FIFO) type of decoding. The code string must always be decoded from the most significant end.

Consider now a LIFO decoding where the code is grown to the left. Let the code words as well as the encoded string be viewed as fractional numbers with the binary point at the left end. To append $A(k)$ to the previously encoded string $\bar{C}(s)$, $\bar{C}(s)$ is shifted to the right by $\ell(k)$ places [length of $A(k)$], and $A(k)$ is added to the left into the vacated $\ell(k)$ bit positions. This can be expressed as an arithmetic process as follows (see [4, 5]):

$$\bar{C}(sk) = A(k) + 2^{-\ell(k)} \cdot \bar{C}(s). \tag{1}$$

As an example, let $k = 0, 1, 2$ give rise to the code words 1, 00, 01 of lengths 1, 2, 2, respectively. With $\bar{C}(s) = .010001$, the result of appending symbol 0 yields $\bar{C}(s0) = .1010001 = .1 + 2^{-1} \times .010001$. The decoding process on $\bar{C}(s0)$ extracts symbols 0, 2, 1, 2 in that order. In decoding, after $A(k)$ is removed, the remaining string is shifted left by $\ell(k)$ positions for the next decoding recursion.

The reader can easily visualize the analogous arithmetization of the other case where the code word is added to the right end of the previously encoded string (see [5]):

$$C(sk) = C(s) + 2^{-L(s)} \cdot A(k). \tag{2}$$

Here $L(s)$ is the number of bits in $C(s)$, formed by summing the individual lengths $\ell(k)$ of all symbols encoded in $C(s)$. In both cases, due to the nonoverlapping of the code words, the addition is in fact a concatenation.

Observe that the recursive "shift-and-add" technique used by arithmetic coding as described herein may be viewed as a generalization of the traditional concatenation mechanization in that the code words may now overlap and that the integer-length shift $\ell(k)$ is replaced, in effect, by a rational-length shift. Also the prefix property for decoding is replaced by the magnitude order relation for the code words, which then are viewed as binary numbers. Because the notion of noninteger-length shift is somewhat intricate and clearly the least obvious of these three generalized items, we illustrate the basic ideas by a fairly detailed discussion of Elias' coding, which when appropriately reinterpreted includes all these notions in a natural, albeit restricted, way. We then continue with a brief exposition of the two early versions of arithmetic codes [4] and [5], and we conclude this section by defining two new dual and wider classes of arithmetic codes.

The following viewpoint on Elias' code appears in Pasco [5]; however, Fig. 1 does not. In Elias' coding the alphabet is ordered, and the order is extended lexically to the source strings with priority to left, say. If $p(k)$ is the probability of symbol k, then define the cumulative probability $P(k) = p(0) + \cdots + p(k-1)$, $P(0) \triangleq 0$. Each source string s has a probability, and corresponding to the lexical ordering of the strings we also obtain a cumulative probability in the space of all strings over the considered alphabet $\langle 0, 1, \cdots, N-1 \rangle$. The cumulative probability of a string s with length t is the sum of the probabilities of all strings of length t smaller than s under the given order, and this probability can be taken as the code $C(s)$ of s.

The recursive construction of $C(s)$ can be written as a sum of terms, one for each symbol in s. The first term is the probability of strings smaller than the first symbol; i.e., it is $P(k)$ if k is the first symbol. The second term is the probability of all strings whose first symbol is k and the second symbol smaller than the second symbol, say, i of s. Similarly, the other terms are defined.

The recursive calculation of these terms, as well as that of $C(s)$, is best illustrated graphically by the example in Fig. 1 for the string $s = 1, 3, 2, 3$, which differs from the customary graphical depictions of Elias' coding.

In Fig. 1 the first term is $P(1)$, which is $p(0)$, the probability of all strings beginning with symbol 0. The second term is $p(1)$ multiplied by $P(3)$. We write the second term as

$$2^{-\ell(1)} \cdot P(3) = 2^{-y(1)}[2^{-x(1)}P(3)],$$

where $y(k)$ and $x(k)$ are the integer and the fractional parts, respectively, of $\ell(k) = -\log p(k)$. This illustrates what is meant by a noninteger-length shift: the composite of ordinary integer-length shift and multiplication by 2 raised to the power indicated by the fractional part. This device was introduced in [4].

To continue the example, we can see that the code of s is

$$\begin{aligned} C(1, 3, 2, 3) &= P(1) + p(1)P(3) + p(1)p(3)P(2) \\ &\quad + p(1)p(3)p(2)P(3) \\ &= C(1, 3, 2) + 2^{-L(1,3,2)} \cdot P(3), \end{aligned} \tag{3}$$

which except for an interpretation is of the type of Eq. (2). The product $T(1, 3, 2) = p(1)p(3)p(2)$ gives the total shift

$$L(1, 3, 2) = -\log T(1, 3, 2) = \ell(1) + \ell(3) + \ell(2).$$

The terms in (3) can also be accumulated from the right:

$$\begin{aligned} C(1, 3, 2, 3) &= P(1) + p(1)C(3, 2, 3) \\ &= P(1) + 2^{-\ell(1)} \cdot C(3, 2, 3). \end{aligned}$$

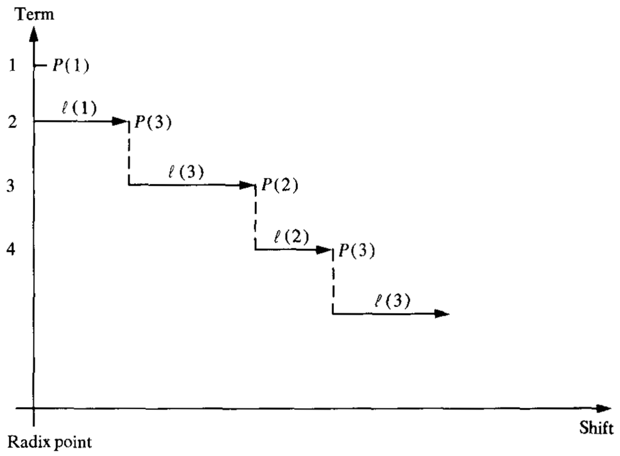

Figure 1 Graphical interpretation of Elias' code for string $s = 1, 3, 2, 3$.

Observe that this recursion starts from the right end of s; i.e., it acts from left to right on the reverse string $\bar{s} = 3, 2, 3, 1$. Hence, if we put $\bar{C}(1, 3, 2, 3) = C(3, 2, 3, 1)$, we transform Elias' code into its dual:

$$\bar{C}(1, 3, 2, 3) = P(3) + 2^{-\ell(3)} \cdot \bar{C}(1, 3, 2), \tag{4}$$

which is of the type of Eq. (1).

When the symbol probabilities are rational numbers, it can be seen that the number of bits in the fractional numbers $\bar{C}(s)$ and $C(s)$ is essentially given by the integer part of $L(s)$, and the per-symbol length approaches the entropy. However, the precision required in the coding calculations grows with the string. We see that in (4) all the past string is to be multiplied by the number 2^{-x}, where x is the fractional part of $\ell(3)$. This is not so serious, though, because viewing the code string as merely an intermediary between the encoder and decoder, it is the relative displacement of the two terms in (4) which is relevant. Accordingly, a key step toward realizability of the dual is to modify the code as follows:

$$\bar{C}(1, 3, 2, 3) = 2^{x} \cdot P(3) + 2^{-y} \cdot \bar{C}(1, 3, 2), \tag{5}$$

where x and y are the fractional and the integer parts of $\ell(3)$, respectively. Now the fractional shift only applies to a fixed number of bits. We also retain the same letter $\bar{C}$ for the code, although the two code strings in (4) and (5) may differ by a multiplication factor. Finally, as a matter of terminology, $P(3)$ in (5) is called the code element [5], and the product $2^{x} \cdot P(3)$ is called the augend $A(x, 3)$. [Note that Eq. (3) could also be trivially reformulated to give $C(1, 3, 2)$ an integral shift left, leaving a fractional shift for the code element $P(3)$.]

In [4] Rissanen studied codes of the general type of Eq. (5), called "arithmetic codes," in which the ideal lengths $\ell(k) = -\log p(k)$ were based on the probabilities, and the numbers 2^x were approximated by rational numbers with a fixed number of fractional bits. This then removed the basic defect (a requirement for unlimited precision) of such codes. While the decodability of the codes of the type in Eqs. (3) and (5) with the exact, nonapproximated parameters is obvious from the fact that the cumulative probabilities form a monotonic nondecreasing sequence, the same becomes a matter of a delicate analysis with the approximated parameters. However, it was shown in [4] that when the approximations are done properly, the code will have a unique inverse, and a practicable non-concatenation code results.

In his excellent thesis [5], Pasco, while being aware of [4], studied the approximation and other issues of Elias' code of the type of Eq. (3), and his codes, which he also called arithmetic codes, removed the requirement for unlimited precision inherent in Elias' code. His main contribution was to obtain a FIFO type of arithmetic code. The key observation was to represent the product T of the probabilities in the string as a floating point number, a fractional part truncated to K bits and the integer part consisting of the number of leading 0's plus one. The fractional part multiplied by the code element $P(k)$ just as in (5) forms the augend, and the integer part of T indicates where in the code string the augend is to be added. Had the fractional part been rounded up instead of truncated, it would have corresponded to a length displacement less than the ideal, and the decodability would have been lost, as Pasco demonstrated. Finally, Pasco proposed a unification of the arithmetic codes by listing eight distinct types depending on whether the code string is grown to the left (LIFO) or to the right (FIFO), whether shift factor T is calculated by a product of probabilities or by its logarithm L treated as a length, and whether the code string or the code element is shifted. However, in the present work, we feel only the LIFO versus FIFO property is significant enough for generalization purposes, as the decodability conditions we derive are valid independently of the implementation details of the shift requirement.

Although the two basic types [Eq. (3) and Eq. (5)] of arithmetic codes indeed provide nonconcatenation codes, they still do not adequately unify the hitherto unexplored class of nonblock or nonconcatenation codes. For instance, as we shall see, there is no LIFO type of dual code of Pasco's main code, which suggests that a meaningful unification involves more than just superficial permutation of the cases. Moreover, from a coding theoretic standpoint, it is a defect if, despite having generalized the basic coding operations, we arrive at a class which still does not include the traditional concatenation codes as special cases.

A careful study of the described approximation steps in the early arithmetic codes as well as of the traditional concatenation codes reveals that the two kinds of parameters, the length parameters $\ell(k)$ and the augends $2^x P(k)$, should not both be linked directly to the symbol probabilities $p(i)$, and hence to each other, as was the case in [4]. In fact, in Huffman codes neither the optimum lengths nor the code words are unique; they are needed so to speak only as a vehicle for decodability. For this reason, we study coding where the augends are set free from their direct dependency on the lengths and the symbol probabilities. To be sure, the decodability question becomes then a crucial issue to be analyzed separately.

Observe in Eq. (5) that the previous code string is only being shifted an integer amount y, and that we still "owe" the fractional shift x. The fractional shift amount x is called the *retained fraction*, and we characterize it as the internal state of a finite state machine. The augend $A(x, k)$ for the "shift-and-add" is a function of x. For the next recursion, the length involved is the sum of $\ell(k)$ for the new symbol and x, the retained fraction. This sum has an integer part which determines the code string (or augend) shift and a fractional part which determines the new retained fraction. Thus, the augends $A(x, k)$ added to the old code string are a function of the past history, as represented by the retained fraction.

We next consider a finite state machine formulation of the recursive "shift-and-add" encoding process, where the summands, the augend, and the code string undergo an integral (nonfractional) relative shift y while the fractional part x is incorporated in the augend. We view the apportioning of the relative shift y as an implementation consideration; in Eq. (6) it is applied to the augend, while in Eq. (7) it is applied to the code string.

Each of the dual equations, (1) and (2) or (3) and (5), admits its corresponding generalization. The first or FIFO version is defined by the recursion equations

$$\begin{aligned} C(a, sk) &= C(a, s) + A[x(s), k]2^{-Y(s)}, \\ x(sk) &= z[x(s), k], \\ Y(sk) &= Y(s) + y[x(s), k], \\ x(\lambda) &= a,\ Y(\lambda) = C(a, \lambda) = 0, \qquad \lambda \text{ is the empty string.} \end{aligned} \tag{6}$$

Here, the functions $z(x, k)$ and $y(x, k)$ denote the state transition and the output functions, respectively, of a machine with a state space X, initial state a, input space

$\langle 0, \cdots, N-1 \rangle$, and the output space consisting of nonnegative integers. In all the practicable codes considered in this paper, the state space is finite, the state constituting a retained fractional part from the preceding code string. An example of another case is Elias' code, which is defined by (6) with X the set of all numbers less than 1. We view Elias' code as a limiting case to FIFO arithmetic codes when the state space is infinite. The parameters $A(x, k)$, called augends, are numbers less than 2 with at most, say, r fractional bits in their binary representation.

The dual code is defined by the equations

$$\bar{C}(b, sk) = A[x(sk), k] + \bar{C}(b, s)2^{-\bar{y}[x(s), k]}$$
$$x(sk) = \bar{z}[x(s), k]$$
$$x(\lambda) = b, \; \bar{C}(b, \lambda) = 0, \qquad (7)$$

where $\bar{z}(x, k)$ and $\bar{y}(x, k)$ denote the next state and the output functions of a machine just as in the previous case. This machine has a finite state space except in certain nonpracticable, idealized codes such as the enumerative codes [3].

Letting $\mathrm{F}(\ell)$ and $\mathrm{I}(\ell)$ denote the fractional and integer parts of ℓ, respectively, we are particularly interested in the class of codes where the state space X consists of q-bit fractional numbers, and the machines in (6)-(7) are defined by the functions

$$\bar{z}(x, k) = \mathrm{F}[\ell(k) + x],$$
$$\bar{y}(x, k) = \mathrm{I}[\ell(k) + x];$$
$$z(x, k) = \mathrm{F}\{1 + x - \mathrm{F}[\ell(k)]\},$$
$$y(x, k) = \mathrm{I}[\ell(k) + z(x, k)] = \bar{y}[z(x, k), k]; \qquad (8)$$

where the length parameters $\ell(0), \cdots, \ell(N-1)$ are positive numbers with at most q fractional bits.

Apart from the normalization of the augends, which causes the code to be a number less than 2, the codes (7) with (8) include Rissanen's arithmetic code as a special case. Also, the traditional concatenation codes can be embedded in these in a canonical manner, as will be described below in detail. The code (6) with (8) is new. It permits a FIFO type of decoding, and it will be shown to be closely related to (7). Pasco's code is of type (6) with a special and narrow choice for the augends and in which the next state $z(x, k)$ results from truncation of the product of the old state x and $p(k)$ to, say, K most significant fractional bits, while the number of the leading zeros plus one defines the output $y(x, k)$. In contrast with Pasco's code, the length parameters $\ell(k)$ appear here explicitly in a role which is entirely analogous to the integer lengths in ordinary codes. Hence, as we shall see, the decodability theory of these codes parallels and naturally extends the existing theory for concatenation codes.

We illustrate these coding equations and the notions of "state," "shift," and the two kinds of parameters by two examples.

Example 1
Let the source alphabet be $S = \{0, 1, 2, 3\}$, where the symbols occur with probabilities $p(0) = 0.25$, $p(1) = 0.65$, $p(2) = 0.075$, and $p(3) = 0.025$. Pick the length parameters as those given by the Huffman algorithm:

$$\ell(0) = 2, \qquad \ell(1) = 1, \qquad \ell(2) = \ell(3) = 3.$$

Because there are no fractional bits, q in (8) is 0, and there is but one state, $x = 0$. Consider the set of augends, which later will be shown to be unique with certain conventions (the now needless state variable is dropped):

$$A(0) = 0$$
$$A(1) = 0.01$$
$$A(2) = 0.11$$
$$A(3) = 0.111.$$

The code (6) of the string $s = 2013$ is seen to be the staggered sum

$$\begin{array}{ll}
.11 & = A(2) \\
.000 & = A(0) \cdot 2^{-3} \\
.0000001 & = A(1) \cdot 2^{-5} \\
\underline{.000000111} & = A(3) \cdot 2^{-6} \\
.110001011 & = C(2013).
\end{array}$$

This is an arithmetic implementation of a compact code. As will be seen, the mean per-symbol length of this code is the same as that of the Huffman code:

$$\sum_i p(i)\ell(i) = 1.45.$$

Example 2
With the same symbols and symbol probabilities as in Example 1, pick the length parameters as follows (the numbers in binary):

$$\ell(0) = 0.1, \quad \ell(1) = 10.1, \quad \ell(2) = 100.0, \quad \ell(3) = 100.1.$$

These were chosen so as to minimize the mean per-symbol length

$$\sum_i p(i)\ell(i)$$

subject to the Kraft inequality and the constraint that each $\ell(i)$ has at most $q = 1$ fractional bit. The mean per-symbol length in the limit when the length of the string goes to infinity is now about 1.36; i.e., better than in Example 1. The state space consists of the numbers 0 and .1, the latter of which is identified with 1.

Consider the list of augends, whose determination will be discussed later:

$$\begin{array}{ll} A(0,0) = 0 & A(1,0) = 0 \\ A(0,1) = 0.001011 & A(1,1) = 0.0100001 \\ A(0,2) = 0.110111 & A(1,2) = 1.01000011 \\ A(0,3) = 0.1110111 & A(1,3) = 1.01011. \end{array} \tag{9}$$

The maximum number of fractional bits in these is then $r = 8$.

By putting the initial state $a = 0$ and by writing $C(0, s) = C(s)$, we calculate the code (6) of the string $s = 2013$ as follows:

$$C(2) = 0.110111$$

$$x(2) = z(0, 2) = 0$$
$$Y(2) = y(0, 2) = 4$$

$$C(20) = C(2)$$

$$x(20) = z(0, 0) = 1$$
$$Y(20) = 4 + y(0, 0) = 7$$

$$C(201) = C(20) + A(1, 1)2^{-y(20)} = 0.11011100100001$$

$$x(201) = z(1, 1) = 0$$
$$Y(201) = 7$$

$$C(2013) = C(201) + A(0, 3)2^{-Y(201)} = 0.11100000001111$$

Because of the longer augends in this case than in the previous one, the length of the code is longer, due to the "end effect" of the last symbol. For long strings, the per-symbol length gets shorter, though, because the end effect is bounded above by $[r + 1 - \ell(s_n)]$, as discussed below. This example also illustrates that a carry-over 1 may occasionally propagate towards the left end of the string. Hence, strictly speaking, neither this nor Pasco's code in [5] permits a FIFO decoding unless the carry-over bit is suitably treated. Fortunately, this can be done with a negligible length penalty. We omit a detailed description of such a blocking mechanism. Observe that the LIFO codes of (7) do not have such carry-over problems.

Decoding and code length

We describe the decoding process for the two codes in (6) and (7) with general finite state machines. In (6), let $s = s_1, s_2, \cdots, s_n$; $s_i \in \langle 0, \cdots, N-1\rangle$ be a string, and consider the associated state trajectory

$$a = a_0 \xrightarrow{s_1/y_1} a_1 \rightarrow \cdots \xrightarrow{s_n/y_n} a_n = x(s) = z^*(a, s),$$

where $a_i = z(a_{i-1}, s_i)$, $y_i = y(a_{i-1}, s_i)$, and $z^*(a, s)$ is the n-fold composite of $z(x, k)$ starting at a. By iteration we get from (6)

$$C(a, s) = A(a, s_1) + A(a_1, s_2)2^{-y(a,s_1)} + \cdots + A(a_{n-1}, s_n)2^{-y(a,s_1) - \cdots - y(a_{n-2},s_{n-1})}. \tag{10}$$

Under suitable selection of y_i and the augends such that $A(a, k+1) > A(a, k)$ [to be stated precisely later in Eq. (25)], the first term in (10) will be greater than the sum of the others. Then the left-most symbol s_1 can be decoded as the largest index k for which

$$C(a, s) \geq A(a, k). \tag{11}$$

Because the condition $C(a, s) < A(a, 1)$ automatically decodes $s_1 = 0$, the value of $A(x, 0)$ is not needed and will be set to zero. The decoding process can be continued by the equations

$$C[z(a, s_1), s_2 \cdots s_n] = [C(a, s) - A(a, s_1)]2^{y(a,s_1)} \tag{12}$$

until $C(x, s') = 0$ for some state x and postfix s' of s. Because $C(a, s0) = C(a, s)$, we must have an extra indication of the number of the trailing 0's in s, or an end-of-string marker. Observe that we may always set $a = 0$. Also, observe that s_1 can be decoded from the code of any prefix of s; e.g., from $C(a, s_1)$, which permits a FIFO type of decoding. The number of the left-most bits of $C(a, s)$ needed for (2) is at most $r + 1$. Hence, the decoding is not quite "instantaneous" in the traditional sense, but the number of "excess" bits [6] needed is only the difference between the numbers of bits in $A(a, k+1)$ and $A(a, k)$, where k is the current left-most symbol of s.

Consider next the dual code (7) of the same string s. The associated state trajectory is

$$b = b_0 \xrightarrow{s_1/y_1} b_1 \rightarrow \cdots \xrightarrow{s_n/y_n} b_n = x(s) = \bar{z}^*(b, s),$$

where $b_i = \bar{z}(b_{i-1}, s_i)$, $y_i = \bar{y}(b_{i-1}, s_i)$, and $\bar{z}^*(b, s)$ is the n-fold composite of $\bar{z}(b, k)$. From (7) we get

$$\bar{C}(b, s) = A(b_n, s_n) + A(b_{n-1}, s_{n-1})2^{-\bar{y}(b_{n-1},s_n)} + \cdots + A(b_1, s_1)2^{-\bar{y}(b_{n-1},s_n) - \cdots - \bar{y}(b_1,s_2)}. \tag{13}$$

This time it is the right-most symbol s_n of s that can be decoded first as the maximum index i for which

$$\bar{C}(b, s) \geq A(x(s), i). \tag{14}$$

Here, the terminal state $x(s)$ is required. The decoding process can be continued by the equations

$$\bar{C}(b, s_1, \cdots, s_{n-1}) = [\bar{C}(b, s) - A(x(s), s_n)]2^{y(b_{n-1},s_n)},$$
$$b_{n-1} = \bar{z}^{-1}(b_n, s_n), \tag{15}$$

provided that the next state function

$$\bar{z}(-, k) : X \rightarrow X$$

has inverse $\bar{z}^{-1}(-, k)$ for every k. The process terminates when $\bar{C}(b, s') = 0$, which correctly decodes $s_1 = 0$ only when a suitable indication of the number of the leading zeros in s is given or a beginning-of-string marker provided. In contrast with the previous code, the dual code has no carry-over problems.

In Pasco's code the next state function does not in general have an inverse and the dual code (7) with its LIFO type decoding cannot be constructed.

We prove now a theorem which justifies the name "dual" for the two codes.

Theorem 1
If $z(-, k)$ has an inverse for every k, then with $\bar{z}(-, k) = z^{-1}(-, k)$ and $\bar{y}(x, k) = y(\bar{z}(x, k), k)$,

$$C(a, s) = \bar{C}(x(s), \bar{s}),$$

where $\bar{s} = s_n, s_{n-1} \cdots s_1$ and $x(s) = z^*(a, s)$.

Proof
The claim follows from comparison of the corresponding terms in (10) and (13). QED.

With the particular choice of the next state and the output functions as given in (8), we can verify that the conditions in Theorem 1 are satisfied. To see that $\bar{z}(-, k) = z^{-1}(-, k)$, we calculate

$$z(\bar{z}(x, k), k) = \mathrm{F}[1 + \mathrm{F}(x + \ell(k)) - \mathrm{F}(\ell(k))] = x,$$

$$\bar{z}(z(x, k), k) = \mathrm{F}[\ell(k) + \mathrm{F}(1 + x - \mathrm{F}(\ell(k)))] = x.$$

Similarly,

$$\bar{y}(x, k) = \mathrm{I}[\ell(k) + z(\bar{z}(x, k), k)] = y(\bar{z}(x, k), k),$$

and we have the important identity in x and k:

$$z(x, k) + \ell(k) = x + y(x, k). \tag{16}$$

Equations (8) imply further useful results. The state trajectory generates the recursion

$$a_i + \ell(s_{i+1}) = a_{i+1} + y(a_i, s_{i+1}),$$

which by summing up both sides gives

$$L(s) = \sum_{i=0}^{N-1} m(i)\ell(i) = x(s) - a + Y(s),$$

$$x(s) = \mathrm{F}(L(s) + a), \tag{17}$$

where $m(i)$ denotes the number of times the symbol i occurs in s. In particular, the second equation permits one to select the initial state b for the dual code (7) in such a way that the terminal state $x(s) = 0$; namely,

$$b = \mathrm{F}[1 - \mathrm{F}(L(s))],$$

so that the decoding process can be started without storage of $x(s)$. Observe, though, that by setting $b = 0$ and storing $x(s)$, an error check is obtained, because $b \neq 0$ after decoding implies that an error has occurred. This can be an important consideration for a noiseless code.

We conclude this section by deriving an inequality for the code length when the special functions (8) are being used. When the augends are normalized to make $0 \leq C(a, s) < 2$ true, we can see from (10) that the smallest term is the last. Therefore, with (15) the total number of bits in $C(a, s)$ is bounded by

$$|C(a, s)| \leq L(s) + r + 1 - \ell(s_n). \tag{18}$$

where the augends have r fractional bits and s_n is the last symbol encoded.

For strings generated by a stationary, ergodic, and independent source with symbol probabilities $p(i)$, the mean per-symbol length is therefore given by (17) and (18) as

$$L = \lim_{n\to\infty} \frac{r+1}{n} + \sum_{i=0}^{N-1} \frac{m(i)}{n} \ell(i)$$
$$= \sum_{i=0}^{N-1} p(i)\ell(i). \tag{19}$$

This clearly justifies the name "length parameters" for $\ell(k)$.

Decodability criteria

In the preceding sections the encoding and the decoding mechanisms have been explained. In this section the invertibility of the encoding process, which we recall was based on a magnitude comparison, is analyzed. The results clearly apply to any code using the same mechanism.

We begin by deriving a decodability criterion for the code in (6). By Theorem 1 the result will be applicable to its dual code as well whenever this latter exists.

Define for each state x in X,

$$B(x) = \sup_s \{C(x, s)\}. \tag{20}$$

These numbers turn out to be well defined whenever the augends are less than 2 and the code has an inverse. The suprema $B(x)$, which have no counterpart in the traditional theory of concatenation codes, play a crucial role in what follows.

Theorem 2 develops necessary and sufficient conditions for decoding arithmetic codes. In addition, it shows that the $B(x)$ values are the limits of the code strings, viewed as numbers, when the strings consist of stretches of the last symbol $(N-1)$ of the alphabet (and the binary point for C remains on the left).

Theorem 2
For every state x in X Eqs. (21) and (22) hold if and only if decoding by Eq. (11) is correct:

$$B(x) = A(x, N-1) + B(z(x, N-1))2^{-y(x,N-1)} \tag{21}$$

$$A(x, k+1) - A(x, k) \geq B(z(x, k))2^{-y(x,k)}, \quad k < N-1. \tag{22}$$

Finally, $B(x) > C(x, s)$ for all (finite) strings s.

Proof
First assume decoding by (11). From (10) we have with $s = (N-1)s'$

$$C(x, s) = A(x, N-1) + 2^{-y(x,N-1)}C(z(x, N-1), s').$$

This leads at once to

$$B(x) \geq A(x, N-1) + 2^{-y(x,N-1)} \cdot B(z(x, N-1)). \tag{23}$$

On the other hand,

$$C(x, s) \leq A(x, N-1) + 2^{-y(x,N-1)} \cdot B(z(x, N-1)). \tag{24}$$

Now, by (11), for every $k < N-1$,

$$A(x, k+1) > C(x, ks') \geq A(x, k), \tag{25}$$

which shows that $0 \triangleq A(x, 0) < A(x, 1) < \cdots < A(x, N-1)$. Because $s = (N-1)s'$,

$$C(x, s) \geq A(x, N-1),$$

which with the previous inequalities implies that $C(x, (N-1)s') > C(x, ks')$. This, in turn, implies that $B(x)$ is the supremum of the codes $C(x, s)$ where the initial symbol of s is $N-1$. By (24), then, the reverse inequality in (23) holds, which proves (21). The first equation in (6) immediately implies that $B(x)$ cannot be attained by any string s, which proves the last claim.

To show (22), we substitute

$$C(x, ks') = A(x, k) + C(z(x, k), s') \cdot 2^{-y(x,k)}$$

from (10) into the first inequality in (25) with the result

$$A(x, k+1) - A(x, k) > C(z(x, k), s') \cdot 2^{-y(x,k)}.$$

This holds even when s' is a string $N-1, N-1, \cdots$. Because $B(x)$ is the supremum of the numbers $C(x, s')$, where s' runs over such strings, (22) follows. Conversely, if (21) and (22) hold for all x and k, then (11) will clearly always result in correct decoding, so that (21) and (22) are both necessary and sufficient conditions for correct decoding.

Equation (21) may be useful in two ways: First, from a set of values $B(x)$, the values of $A(x, N-1)$ can be determined. Conversely, if the values of $A(x, N-1)$ are given, then the values $B(x)$ can be determined. Once the $B(x)$ are determined for given values of $A(x, N-1)$, Eq. (22) can be used to calculate the intermediate values for the $A(x, k)$ tables. When state x is a fraction as in Eq. (8), this is an iterative process for which the values $P(k) \cdot 2^x$ serve as good starting points. A useful final objective would be to find solutions to (21) and (22) which employ a minimum number of fractional bits r. We illustrate this by an example in a later section.

In the rest of this paper we assume the special state transition and output functions (8). With these, the decodability criteria (21) and (22) can be converted into another form which explicitly involves the primary length parameters $\ell(k)$ and shows that these may be selected independently from the augends.

Our first aim is to derive a generalized Kraft inequality as an important part of the decodability criteria. Although the necessity of this inequality could be derived from the length inequality (18) by a modification of the proof by Karush [1], we give an altogether different derivation which is constructive in nature and leads to further results discussed subsequently.

From (22) we get

$$A(x, N-1) \geq \sum_{i=0}^{N-2} B(z(x, i))2^{-y(x,i)},$$

which with (21) leads to

$$B(x) \geq \sum_{i=0}^{N-1} B(z(x, i))2^{-y(x,i)}.$$

With (16), the last inequality becomes

$$B(x)2^{-x} \geq \sum_{j=0}^{N-1} B(z(x, j))2^{-z(x,j)} \cdot 2^{-\ell(j)}, \text{ all } x \text{ in } X. \tag{26}$$

We write this as a vector inequality by writing the elements of X in their natural order $\langle x(1), \cdots, x(M)\rangle$, and $\mathbf{u} = \mathrm{col}(u(1), \cdots, u(M))$,

$$u(i) = B(x(i))2^{-x(i)}.$$

Then (26) turns into

$$\mathbf{u} \geq \mathbf{P}\mathbf{u}, \tag{27}$$

where the $M \times M$ matrix $\mathbf{P} = \{p(i, j)\}$ is given by

$$p(i, j) = \sum_{k \in K(i,j)} 2^{-\ell(k)};$$

$K(i, j)$ is the set of indices k for which $z(x(i), k) = x(j)$, and $p(i, j) = 0$ if $K(i, j)$ is empty. An example will illustrate how the matrix $\mathbf{P}$ gets defined in a straightforward way.

Example 1
Let $N = 3$, $\ell(0) = (10.10)_2$, $\ell(1) = (1.10)_2$, $\ell(2) = (1.11)_2$. The state space X is then given by

$$X = \{.00, .01, .10, .11\};$$

and $\mathbf{P}$ becomes the four by four matrix

$$\mathbf{P} = \begin{bmatrix} 0 & 2^{-\ell(2)} & 2^{-\ell(0)} + 2^{-\ell(1)} & 0 \\ 0 & 0 & 2^{-\ell(2)} & 2^{-\ell(0)} + 2^{-\ell(1)} \\ 2^{-\ell(0)} + 2^{-\ell(1)} & 0 & 0 & 2^{-\ell(2)} \\ 2^{-\ell(2)} & 2^{-\ell(0)} + 2^{-\ell(1)} & 0 & 0 \end{bmatrix}.$$

Lemma 1
For every m and n,

$$\sum_{j=1}^{M} p(m, j) = \sum_{i=1}^{M} p(i, n) = \sum_{k=0}^{N-1} 2^{-\ell(k)}.$$

Proof
We showed earlier that for each k the function $z(-, k)$ has an inverse. Therefore, for each m the nonempty sets in the family $\{K(m, j) \mid j = 1, \cdots, M\}$ partition the alphabet $\{0, \cdots, N - 1\}$, and the same is true for each n about the family $\{K(i, n) \mid i = 1, \cdots, M\}$. By the first property the double sum in

$$\sum_{j=1}^{M} p(m, j) = \sum_{j=1}^{M} \sum_{k \in K(m,j)} 2^{-\ell(k)}$$

has the N terms, $2^{-\ell(i)}$, $i = 0, \cdots, N - 1$. Similarly, by the second property, the double sum in

$$\sum_{i=1}^{M} p(i, n) = \sum_{i=1}^{M} \sum_{k \in K(i,n)} 2^{-\ell(k)}$$

has the same N terms. This proves the claims.

Theorem 3
Inequality (27) has positive solutions u; i.e., $u(i) > 0$, if and only if

$$\sum_{k=0}^{N-1} 2^{-\ell(k)} \leq 1. \tag{28}$$

Proof
Let (28) hold. The vector $\mathbf{u} = \text{col}\ (1, \cdots, 1)$ satisfies (27) as immediately verified with the help of Lemma 1.

Then let

$$\sum_{k=0}^{N-1} 2^{-\ell(k)} = \alpha > 1.$$

If some $\mathbf{u}^0$ satisfies (27), then so does $\mathbf{u}^1 = \mathbf{P}\mathbf{u}^0$, and $\mathbf{u}^k = \mathbf{P}^k\mathbf{u}^0$, so that

$$\mathbf{u}^0 \geq \mathbf{u}^k \geq \mathbf{P}\mathbf{u}^k = \mathbf{u}^{k+1}, \tag{29}$$

because $\mathbf{P}$ has nonnegative elements. Also, for each positive number K, $K\mathbf{u}^0$ satisfies (27). By picking K appropriately we may assume that the smallest component of $\mathbf{u}^0$ equals one. But then the components $u^1(i)$ satisfy

$$u^1(i) = \sum_{j=1}^{M} p(i, j) u^0(j) \geq \sum_{j=1}^{M} p(i, j) \min u^0(k) = \alpha,$$

and by repeating this,

$$u^k(i) \geq \alpha^k.$$

Pick k so large that

$$\alpha^k > \max_j u^0(j).$$

Then

$$u^k(i) > \max_j u^0(j), \qquad i = 1, \cdots, M,$$

which contradicts the inequality (29). Therefore, no solution $\mathbf{u}^0$ exists, which completes the proof.

We show next that if (28) holds with equality, then the augends are determined by (21) and the equalities in (22). This, then, is a singular case, and r fractional bit augends exist only for special values for $\ell(k)$.

Theorem 4
If (28) holds with equality, then every solution to (27) or, equivalently, to (22) also satisfies these with equality.

Remark
This theorem is of fundamental importance, above all in codes with integer-length code words, because it permits a description of the code words by a formula. See the next section.

Proof
Suppose $\mathbf{u}$ satisfies (27) and for some k

$$u(k) > p(k, 1)u(1) + \cdots + p(k, M)u(M).$$

Then by Lemma 1 and (28),

$$\sum_{k=1}^{M} u(k) > \sum_{k,i=1}^{M} p(k, i)u(i) = \sum_{i=1}^{M} u(i),$$

which is impossible. Hence $\mathbf{u} = \mathbf{Pu}$.

For the associated value,

$$B(x(i)) = u(i)2^{x(i)}, \tag{30}$$

(26) holds with equality, and from (21) and (16)

$$A(x, N - 1) = \sum_{i=0}^{N-2} B(x(i))2^{-y(x(i),i)}. \tag{31}$$

From (22) then,

$$A(x, k) \geq \sum_{i=0}^{k-1} B(x(i))2^{-y(x(i),i)}, \quad k = 1, \cdots, N - 1, \tag{32}$$

which with (31) implies that (32) holds with equality.

The next theorem implies that the inequality (28) is in effect a decodability criterion in an analogous sense to Kraft inequality.

Theorem 5
Let the numbers $\ell(k)$ have at most q fractional bits and let

$$\sum_{k=0}^{N-1} 2^{-\ell(k)} < 1.$$

Then, for some r, augends $A(x, k)$, $k = 1, \cdots, N - 1$, exist, each having at most r fractional bits, such that (21) and (22) hold, and correct decoding is obtained.

Proof
The set of solutions to the inequality $\mathbf{u} \geq \mathbf{Pu}$, unless empty, always includes the special solution

$$\mathbf{u} = \text{col}\,(1, \cdots, 1),$$

which gives rise to the particular values

$$B^0(x) = 2^x, \qquad \text{all } x \text{ in } X. \tag{33}$$

These, in turn, by (21) lead to the special values for the largest augends:

$$A^0(x, N - 1) = 2^x - 2^{z(x,N-1)-y(x,N-1)} = 2^x(1 - 2^{-\ell(N-1)}). \tag{34}$$

The numbers $B^0(x)$ may also be used with the equalities in (22) to give recursively the other augends:

$$A^0(x, k + 1) = A^0(x, k) + 2^{-\ell(k)} \cdot 2^x, \qquad k < N - 2. \tag{35}$$

With these special augends, decodability is ensured, because (22) holds for $k < N - 2$ with equality, while for $k = N - 2$ we get from (34) and (35)

$$\begin{aligned} &A^0(x, N - 1) - A^0(x, N - 2) \\ &\quad = 2^x\left(1 - 2^{-\ell(N-1)} - \sum_{i=0}^{N-3} 2^{-\ell(i)}\right) \\ &\quad = 2^x \cdot 2^{-\ell(N-2)} + 2^x\left(1 - \sum_{i=0}^{N-1} 2^{-\ell(i)}\right) \\ &\quad > B^0(z(x, N - 2))2^{-y(x,N-2)}, \end{aligned}$$

where we used (16). However, these augends may be irrational numbers. Our plan is to prove the theorem by demonstrating that in a neighborhood about these numbers r-fractional bit augends exist such that decodability still results.

Before proceeding with the proof, compare these special "ideal" augends,

$$A^0(x, k + 1) = 2^x \sum_{i=0}^{k} 2^{-\ell(i)}, \tag{36}$$

with the corresponding terms

$$2^x \cdot P(k + 1) \tag{37}$$

appearing in Elias' coding, as in Eq. (5). The augends in both (36) and (37) may be viewed as being generated by the code element undergoing a fractional shift. It is important to observe that the augends should be dependent on the chosen lengths rather than the symbol probabilities. This increases the flexibility in developing codes by facilitating the replacement of code words by formulas, as was the case earlier in Example 1. Also, by letting the Kraft inequality hold strictly, we may be able to select the augends with fewer fractional bits and thereby trade implementation complexity for performance.

For each x, select $A(x, N - 1)$ as the number obtained when $A^0(x, N - 1)$ is truncated to r fractional bits and 2^{-r} is added to the result. Then by (34),

$$A(x, N - 1) = 2^x(1 - 2^{-\ell(N-1)}) + \mu(x), \tag{38}$$

where $0 \leq \mu(x) \leq 2^{-r}$. Next, by Lemma 2 below,

$$B(x) = 2^x + \delta(x), \tag{39}$$

where $\delta(x)$ tends toward zero as $\mu_r(x)$ and r tend toward zero. In order to satisfy (22) define recursively $A(x, k + 1)$ for $k = 0, \cdots, N - 3$ as the number obtained when

$$A(x, k) + B(z(x, k))2^{-y(x,k)}$$

is truncated to r fractional bits, 2^{-r} is added to the result, and where $A(x, 0) = 0$. Then, clearly,

$$\begin{aligned} A(x, k + 1) = A(x, k) &+ 2^{z(x,k)-y(x,k)} \\ &+ \delta(z(x, k))2^{-y(x,k)} + \epsilon(x, k), \end{aligned} \tag{40}$$

where the error $\epsilon(x, k)$ satisfies $0 < \epsilon(x, k) \leq 2^{-r}$. The inequalities (22) now surely hold for $k = 0, \cdots, N - 3$.

To check the last inequality in (22), we calculate from (38)-(39):

$$A(x, N-1) - A(x, N-2) = 2^x\left(1 - 2^{-\ell(N-1)} - \sum_{i=0}^{N-3} 2^{-\ell(i)}\right) + \mu(x) - \sum_{i=0}^{N-3} \delta(z(x, i) 2^{-y(x,i)} + \epsilon(x, i)).$$

Because

$$2^x\left(1 - 2^{-\ell(N-1)} - \sum_{i=0}^{N-3} 2^{-\ell(i)}\right) = 2^x(2^{-\ell(N-2)} + \epsilon) = 2^{z(x,N-2)-y(x,N-2)} + \epsilon \cdot 2^x,$$

where

$$\epsilon = 1 - \sum_{i=0}^{N-1} 2^{-\ell(i)},$$

and

$$2^{z(x,N-2)} = B(z(x, N-2)) - \delta(z(x, N-2)),$$

we get

$$A(x, N-1) - A(x, N-2) = B(z(x, N-2)) 2^{-y(x,N-2)} + \eta(x), \tag{41}$$

where

$$\eta(x) = \epsilon \cdot 2^x + \mu(x) - \sum_{i=0}^{N-2} \delta(z(x, i)) 2^{-y(x,i)} - \sum_{i=0}^{N-3} \epsilon(x, i).$$

Now, pick r so large that $\eta(x) \geq 0$, which can be done, because all terms in $\eta(x)$ other than $\epsilon \cdot 2^x$ go to zero as r grows to infinity. Then clearly (22) holds even for $k = N - 2$. This completes the proof.

For Lemma 2 to follow we write (21) as a vector equation

$$\mathbf{u} = \mathbf{G}\mathbf{u} + \mathbf{v}, \tag{42}$$

where $\mathbf{u}$ as above denotes the column vector $\text{col}(u(1), \cdots, u(M))$,

$$u(i) = B(x(i)) \cdot 2^{-x(i)},$$

and

$$\mathbf{v} = \text{col}\,(v(1), \cdots, v(M)),$$

$$v(i) = A(x(i), N-1) \cdot 2^{-x(i)}.$$

The matrix $\mathbf{G} = \{g(i, j)\}$ is given by

$$g(i, j) = 0 \quad \text{if} \quad z(x(i), N-1) \neq x(j)$$
$$= 2^{-\ell(N-1)} \quad \text{otherwise}. \tag{43}$$

Lemma 2
If (28) holds and not all $\ell(k)$ are zero, then the matrix $\mathbf{I} - \mathbf{G}$ in (42) has inverse.

Proof
Because the state transition map $z(-, N-1)$ has inverse, $\mathbf{G}$ has one element, $2^{-\ell(N-1)}$, in each row and each column. Therefore, $\mathbf{G}^k$ has one element, $2^{-k\ell(N-1)}$, in each row and each column. Because $2^{-\ell(N-1)} < 1$, $\mathbf{G}^k \to 0$ as $k \to \infty$. The inverse of $\mathbf{I} - \mathbf{G}$ is given by

$$(\mathbf{I} - \mathbf{G})^{-1} = \lim_{k\to\infty} (\mathbf{I} + \mathbf{G} + \cdots + \mathbf{G}^k).$$

Consider now the performance of the arithmetic code. Consider a set of ideal lengths $\ell'(i)$, and approximate them to q fractional bits, rounded up. Each length differs at most by 2^{-q} from the ideal, and the average symbol length L for the code is therefore bounded by

$$L \leq \sum_i p(i)\ell'(i) + 2^{-q}. \tag{44}$$

The first sum is the source entropy. Therefore, by making q sufficiently large, the mean per-symbol length can be made arbitrarily close to entropy.

A somewhat better way to determine the length parameters is to select q and then find $\ell(k)$ as the solution to the optimization problem:

$$\min_{\ell(i)} \sum_{i=0}^{N-1} p(i)\ell(i) \tag{45}$$

subject to the inequality (28). For $q = 0$, Huffman's algorithm solves this problem.

Integer-valued length parameters

The special case of arithmetic codes where the length parameters are positive integers is of particular interest because the finite state machine degenerates to a 1-state machine and thereby disappears. We emphasize that within the arithmetic codes the length parameters being integers by no means implies that the resulting codes are concatenation codes, as in Example 1. Indeed, nothing in the above theory states that these lengths must be given by the number of bits in the code words, i.e., the augends $A(x, k) \triangleq A(k)$.

Because the lengths of binary Huffman codes satisfy (28) with equality, Theorem 4 implies that there is only one set of augends defined by (21) and the equalities in

(22). Specifically, supremum $B \triangleq B(x) = 1$, and

$$A(k) = \sum_{i=0}^{k-1} 2^{-\ell(i)}, \qquad k > 0$$

$$A(0) = 0. \tag{46}$$

In the special case where the symbols are so ordered that $i < j$ implies $p(i) \geq p(j)$, so that $\ell(i) \leq \ell(j)$, this formula reduces to one given by Gilbert and Moore [6]. If we put $A(0) = .0 \cdots 0$, $\ell(0)$ zeros, the addition in (6) and (7) degenerates to a concatenation, and the code apart from the irrelevant scaling is just a Huffman code.

To conclude this section we describe briefly how ordinary prefix-concatenation codes are represented as arithmetic codes in which the decodability criteria (21) and (22) are satisfied. By reordering of the alphabet, if necessary, we may further assume that the code words satisfy $0 = a(0) < a(1) < \cdots < a(N-1)$, when they are regarded as binary integers. These inequalities hold because of the prefix property. Clearly, the opposite is not true, i.e., the magnitude order does not imply the prefix property. Now, define the augends

$$A(k) = a(k) \cdot 2^{-\ell(k)}.$$

Then, because $a(N-1)$ has $\ell(N-1)$ bits,

$$A(N-1) \leq 1 - 2^{-\ell(N-1)}.$$

Then $B(0) \triangleq B$, as a solution to (21), satisfies

$$B \leq 1.$$

By the prefix property,

$$A(k) \geq A(k-1) + 2^{-\ell(k-1)}, \qquad k = 1, \cdots, N-1,$$

and, hence,

$$A(k) \geq A(k-1) + B \cdot 2^{-\ell(k-1)},$$

which verifies (22).

If in addition the code is compact, all the preceding inequalities hold with equality.

Code design

Code design consists of assigning values to the length parameters $\ell(i)$ and the augends $A(x, k)$. The problem of finding the values for the length parameters was already discussed in Eq. (45), although no elegant algorithm was given for the solution. The number of fractional bits in $\ell(i)$ is chosen so as to achieve close approximation to the entropy.

The augends, then, are to be found to satisfy (21) and (22). The fact that the augends have at most r fractional bits, r to be chosen as small as possible, permits a simplification of (21) and (22), which we derive next. First, (22) is seen to be true if and only if

$$A(x, k+1) - A(x, k) > B_r(z(x, k))2^{-y(x,k)}, \qquad k < N-1, \tag{47}$$

where $B_r(z)$ denotes the truncation of $B(z)$ to r fractional bits.

Turning then to (21) we can expand iteratively

$$B(x) = A(x, N-1) + A(z(x, N-1), N-1)2^{-y(x,N-1)} + \cdots, \tag{48}$$

from which $B_r(x)$ can be obtained as a sum of finitely many terms.

An approach to finding the augends is suggested by the analysis following Theorem 5. We guess a value for r such that 2^{-r} is about

$$1 - \sum_{i=0}^{N-1} 2^{-\ell(i)},$$

and pick $A(x, N-1)$ in (21) so that $B(x) = 2^x$. By (48), then, $B_r(x)$ is calculated exactly. The other augends $A(x, k)$ for $k < N-1$ are determined recursively from (47) as small as possible with r fractional bits. If the last inequality holds, we have found a valid set of augends. If not, increase r and try again. Theorems 4 and 5 guarantee that for r large enough a valid set will result. A more systematic and likely faster way is to convert (47) and (48) for each r into integer equalities and inequalities, and use the extended Euclidean algorithm [7] to find a solution if one exists.

We illustrate the code design and some of the practical issues involved by means of Example 2. With the length parameters given in Example 2 we have

$$\sum_{i=0}^{3} 2^{-\ell(i)} = .990578.$$

Because the sum above is very close to 1, Theorems 4 and 5 imply that the neighborhoods about the "ideal" augends are not large, and we must expect to select r on the order of 7; i.e., $2^{-r} \approx \epsilon \approx .01$.

Denote the state .0 by 0 and .1 by 1. The state transitions and outputs as given by Eq. (8) are in the table below.

$z(x, k)/y(x, k)$ (49)

$x \backslash k$	0	1	2	3
0	1/3	1/1	0/4	1/5
1	0/2	0/0	1/4	0/4

From Eq. (48),

$$\begin{pmatrix} B(0) \\ B(1) \end{pmatrix} = \begin{pmatrix} A(0,3) \\ A(1,3) \end{pmatrix} + \begin{pmatrix} A(1,3)\cdot 2^{-4} \\ A(0,3)\cdot 2^{-5} \end{pmatrix} + \begin{pmatrix} A(0,3)\cdot 2^{-9} \\ A(1,3)\cdot 2^{-9} \end{pmatrix} + \cdots, \tag{50}$$

while Eq. (47) reads as follows:

$$\begin{aligned} &A(0,1) > B_r(1)2^{-3} \\ &A(0,2) > A(0,1) + B_r(1)2^{-1} \\ &A(0,3) > A(0,2) + B_r(0)2^{-4} \\ &A(1,1) > B_r(0)\cdot 2^{-2} \\ &A(1,2) > A(1,1) + B_r(0) \\ &A(1,3) > A(1,2) + B_r(1)2^{-4}. \end{aligned} \tag{51}$$

After a bit of trial and error $A(0, 3)$ and $A(1, 3)$ are picked as (in binary)

$$\begin{aligned} &A(0,3) = .1110111 \\ &A(1,3) = 1.01011 \end{aligned} \tag{52}$$

which by (50) give $B_8(0) = 1.0$ and $B_8(1) = 1.01011111$. The other augends can then be chosen easily by (50) and (51) as given by (9).

Summary

Material in the second section was intended to provide perspective for variable rate noiseless coding from the viewpoint of arithmetic coding. The fundamental parameters in arithmetic coding are the noninteger-valued length parameters and the similar augends, which are combined by a shift and addition to recursively grow the code string from either end for LIFO or FIFO coding. The decoding is based on magnitude comparison, and it is virtually instantaneous without the prefix property, which is replaced by the more general magnitude order. The notion of a sequential machine was introduced to remember the fractional shift amount. For integer length codes, the number of states reduces to just one. For Elias' code, in the idealized formulation, the number of states can become arbitrarily large. This has forced earlier practical implementations of it to introduce some way of systematically terminating the code strings. The sequential machine formulation is quite general as it also provides a common basis for understanding the differences between the arithmetic codes of Rissanen [4] and Pasco [5].

The per-symbol length of arithmetic codes can be made as close to the entropy as desired. This is achieved by increasing the precision of fractional lengths, instead of performing alphabet extensions.

Finally, a self-contained, widely applicable, and fairly thorough decodability analysis (Theorem 2) has been made with results which permit construction of invertible codes with any desired realizable per-symbol length. The analysis applies to any scheme whose encoding mechanism can be formulated by Eqs. (6) or (7), (recursive shift-and-add process) and for which Eq. (11) can be used for decoding. Since these hold for prefix codes, Eqs. (21) and (22) of Theorem 2 apply to prefix codes, as well as to Elias' [1], Rissanen's [4] and Pasco's [5] codes. What remains open, however, are algorithms for a fast and convenient determination of the design parameters in which the desired compression can rapidly be balanced with the complexity of the implementation of the code. The decodability analysis and the proof of Theorem 5 [see Eqs. (36) and (37)] demonstrate the importance of the sum

$$\sum_{i=0}^{k-1} 2^{-\ell(i)}$$

as a code element. When the generalized Kraft inequality (Theorem 3) is satisfied with equality, this is the only choice for symbol k. Theorem 5 has a semi-constructive proof showing how augend tables can be designed.

Acknowledgment
The authors are grateful for the encouragement provided them by Patrick E. Mantey and R. Bruce Brodie. This paper probably would not have been written had they not foreseen the promise of this work and stimulated the collaboration of the coauthors.

Appendix: Symbol definitions

$A(x, k)$	"real" augend; it is a state-dependent code word of symbol k.
$A^0(x, k)$	"ideal" augend based on ideal $B^0(x)$, $A^0(x,k) = 2^x \sum_{i=0}^{k-1} 2^{-\ell(i)}$
a, b	initial states for the finite state machine
$B(x)$	the supremum of the values attained by the code strings $C(x, s)$ interpreted as numbers. It is also the supremum of $C(x, s)$ when s is a string of symbols $(N - 1)$.
$B^0(x)$	"ideal" limit 2^x
$C(s)$	code string for strings grown to the right

$\bar{C}(s)$	code string for string s grown to the left, which is the dual of code string $C(s)$
FIFO	first-in, first-out; the code string is recursively encoded to the right, then recursively decoded from the left.
F(ℓ)	fractional part of ℓ
G	$N \times N$ matrix whose elements $g(i, j)$ are zero unless $z(x(i), N-1) = x(j)$, in which case it is $2^{-\ell(N-1)}$
I	$M \times M$ identity matrix
I(ℓ)	integer part of ℓ
k	a symbol of alphabet $\langle 0, 1, \cdots, k, \cdots, N-1\rangle$
$K(i, j)$	set of indices k which take state $x(i)$ to state $x(j)$: $\{k \mid z(x(i), k) = x(j)\}$
$\ell(k)$	length parameter of source symbol k
$L(s)$	total length, the sum of all lengths $\ell(s_i)$ where $s = s_1, s_2, s_3, \cdots$
LIFO	last-in, first-out; the code string is recursively encoded to the left, and then recursively decoded from the left.
M	number of states in state space X
$m(i)$	number of occurrences of symbol i in string s
N	number of source symbols
P	$M \times M$ matrix of element $p(i, j) = \sum_{k \in K(i,j)} 2^{-\ell(k)}$
$P(k)$	cumulative probability; $P(0) = 0, P(k) = \sum_{i=0}^{k-1} p(i)$
$p(k)$	probability of occurrence of symbol k
q	maximum number of fractional bits in $\ell(k)$
r	maximum number of bits in fractional part of augends $A(x, k)$
$\bar{s}$	reversal of string s
s'	post fix substring of source string s
T	a multiplication factor corresponding to a shift of length L, $L = -\log T$ or $T = 2^{-L}$
$u(i)$	$B(x, i) \cdot 2^{-x(i)}$
v	column vector whose ith component is $A(x(i), N-1) \cdot 2^{-x(i)}$
X	state space of the finite state machine
$x(i)$	states $[x(i) \in X]$, where X is ordered
$x(k)$	fractional part of $\ell(k)$
$y(k)$	integer part of $\ell(k)$
$Y(s)$	integer part of total length $L(s)$
$z(x(s), k)$	next-state function; present state $x(s)$ and input k.
$z^*(a, s)$	n-fold composite of $z(x, k)$ starting at state a.
λ	empty string, used to express initial conditions for finite state machine
$0, 1, \cdots, k, \cdots, (N-1)$	source symbol alphabet

References

1. N. Abramson, *Information Theory and Coding*, McGraw-Hill Book Co., Inc., New York, 1963.
2. J. P. M. Schalkwijk, "An Algorithm for Source Coding," *IEEE Trans. Info. Theory* **IT-18,** 395 (1972).
3. T. M. Cover, "Enumerative Source Encoding," *IEEE Trans. Info. Theory* **IT-19,** 73 (1973).
4. J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," *IBM J. Res. Develop.* **20,** 198 (1976).
5. R. C. Pasco, "Source Coding Algorithms for Fast Data Compression," Ph.D. thesis, Dept. of Electrical Engineering, Stanford University, CA (1976).
6. E. N. Gilbert and E. F. Moore, "Variable-length Binary Encodings," *Bell. Syst. Tech. J.* **38,** 933 (1959).
7. D. Knuth, *The Art of Computer Programming*, Vol. 2, Addison-Wesley Publishing Co., Reading, MA, 1969.

Received January 31, 1978; revised May 17, 1978

The authors are located at the IBM Research Division laboratory, 5600 Cottle Road, San Jose, California 95193.