Stephen J. P. Todd
Glen G. Langdon, Jr.
G. Nigel N. Martin

A General Fixed Rate Arithmetic Coding Method for Constrained Channels

This paper extends the result of earlier work on the application of arithmetic codes to the constrained channel problem. We specifically present a general length-based fixed rate implementation technique which performs the arithmetic coding recursions during each channel time unit. This technique is superior to an earlier unpublished code for general constrained channels. The approach permits the design of codes for sophisticated channel constraints.

1. Introduction

The constrained channel has been studied by Shannon [1]. References [2, 3] show how a fixed rate length-based (L-based) arithmetic code can be applied to the problem of coding to a constrained channel, where the constraints are described by a channel finite state machine (CFSM). We assume familiarity with a companion paper [3] which provides references to other approaches to channel coding and which contains some background material on arithmetic coding. We employ the same terminology and system of notation as [3].

In saturation magnetic recording, the media are divided into channel time units. The events are the occurrence or nonoccurrence of a flux change, so the elementary symbols are said to consist of **0**'s or **1**'s. The well-known run-length-limited (RLL) constraints are called (d,k) constraints when there must be at least d and no more than k **0**'s between successive **1**'s. Codes for (d,k) constraints are used to maximize information on the channel and synchronize a clock (provide self-clocking).

In [2] an L-based channel code for general constraints was described. Also developed in [2] and published in [3] was a simpler version applicable only to (d,k) codes.

Many nontrivial constrained channel problems place restrictions in addition to the (d,k) constraint. Such generalized (d,k) problems include a charge-limited constraint [4] or desired spectral null. In this paper, we extend the simpler approach of [3] to the general case, resulting in a more economical realization than that in [2]. For a description of L-based and P-based (probability-based) arithmetic compression codes, see [5]. The foundation for the application of arithmetic codes to the constrained channel appears in [3]; however, we provide a brief review.

Arithmetic coding transformations require a symbol ordering. Let symbol $\xi + 1$ denote the symbol following ξ in the ordering. If symbol γ precedes ξ in the ordering, let $\gamma < \xi$. Let ω denote the last symbol in the ordering. Let $p(i,\xi)$ denote the probability of symbol ξ in state i. Let $P(i,\xi)$ denote the cumulative probability of the symbols preceding ξ in the ordering:

$$P(i,\xi) = \sum_{\gamma<\xi} p(i,\gamma).$$

(In what follows, we loosely identify $P(i,\xi)$ with an augend and $p(i,\gamma)$ with an addend.)

Let channel string u be a sequence of symbols ξ drawn from channel alphabet $\{\alpha, \cdots, \xi, \cdots, \omega\}$. The constraints do not allow all possible sequences of the channel alphabet. Shannon [1] defines the constraints by a channel finite state machine (CFSM) whose next state function T when fed

string u with the CFSM in its initial state 1 yields CFSM state $T(1,u)$, which we simply call $T(u)$ when the initial state is understood.

As in [3], coding to a constrained channel corresponds to an expansion operation on the source data, and recovering the source data corresponds to a compression operation. It is convenient to describe the operations in terms of the compression operation. A channel string u is transformed back into a data string $F(u)$ by a recursion on u. Let $u.\xi$ denote string u with symbol ξ concatenated to it. Then

$$F(u.\xi) = F(u) + D(u,\xi),$$

where quantity $D(u,\xi)$ is called the augend and is represented as

$$D(u,\xi) = M(T(u),X(u),\xi) \times 2^{-E(u)},$$

where $2^{-E(u)}$ represents an integral shift and $M(T(u),X(u),\xi)$ is called the augend factor. The value $X(u)$ is an internal recursion variable which takes on a finite set of values. In L-based arithmetic codes X is determined by a recursion which is viewed as a sum of lengths, or more precisely as the channel time modulo C, where C is the cardinality of the set of values of X. In P-based arithmetic codes the X recursion is reminiscent of a product of probabilities. The L-based arithmetic codes use augends of the form $M(i,X,\xi)$, where i is a typical CFSM state, by table lookup. The design of L-based codes amounts to determining a table of augends which satisfy a consistency test. L-based arithmetic codes for channels are capable of a rational rate J/C, where C channel time units carry J bits of information.

An example of a code more complex than (d,k) codes is a (2,7,8) charge-constrained code using the general fixed rate approach. In the general case [2] of more complex codes the size of the table of augends is

$$\textit{Augend table size} = n \times C \times (|\xi| - 1), \tag{1}$$

where $|\xi|$ is the cardinality of the set of channel symbols (typical symbol ξ), n is the cardinality of the set of states (typical state i), and C is the denominator of the rational fixed rate J/C. The multiplier involving the cardinality $|\xi|$ of the symbol alphabet $\alpha, \cdots, \omega$ is decreased by one because any augend $M(i,X,\alpha)$ corresponding to the first symbol α has value 0.

Let the transformation of channel string u to a data string $F(u)$ have recursion variables $F(u)$, $\ell(u)$ (duration or length of u in unit-time symbols) and CFSM state $T(u)$. We replace $\ell(u)$ by $E(u)$ and $X(u)$ such that

$$J \times \ell(u)/C = (E(u) + X(u))/C,$$

where $E(u)$ is an integer and $X(u)$ is $0, 1, \cdots,$ or $C - 1$. To transform next symbol ξ we have

$$F(u.a) = F(u) + M(T(u),X(u),\xi) \times 2^{-E(u)} \tag{2}$$

$$E(u.a) = \lfloor (\ell(u) + \ell(\xi)) \times J/C \rfloor, \tag{3a}$$

$$X(u.a) = ((\ell(u) + \ell(\xi)) \times J) \bmod C, \tag{3b}$$

where $\lfloor\ \rfloor$ denotes the integer part. Equation (3) constitutes the length recursion. Martin [2] provides a procedure for the calculation of the augend factors $M(i,X,\xi)$, which is essentially an approximation to $P(i,\xi) \times 2^{-X/C}$.

Let F and M be binary numbers. The purpose of variable E is to maintain proper alignment between the bits of F and M. We shift F left one bit for each increase by 1 in the value of E, and thus the initial proper alignment is maintained. The value of the shift after iteration $u.\xi$ is $E(u.\xi) - E(u)$. The shift can be determined from $X(u)$ and $\ell(\xi)$ as follows:

$$E(u.\xi) - E(u) = \lfloor ((J \times \ell(\xi)) + X(u))/C \rfloor .$$

Similarly, the recursion on X is

$$X(u.\xi) = ((J \times \ell(\xi)) + X(u)) \bmod C.$$

Shifting and alignment, as well as the carry-over problem resulting from the addition of $D(u,\xi)$ to $F(u)$, are covered in [5].

2. Two approaches to defining the CFSM for (d,k) channels

To describe the constraints for a (d,k) channel, consider first the view that the CFSM consists of one state and $(k - d + 1)$ symbols. These symbols are a result of parsing the binary-valued channel strings into an "extended" alphabet of channel phrases. The duration of a phrase extends beyond one channel time unit. In (d,k) channels the phrases are of the form **001, 0001,** etc., where the phrases are of length $d + 1$ through length $k + 1$.

In the general approach to (d,k) codes using phrases, the number of nonzero augends in the table for channel parsing CFSM's and rate J/C is

$$\textit{parsing CFSM table size: } C \times (|\xi| - 1) = C \times (k - d). \tag{4}$$

In a second approach let the (d,k) channel be described by two symbols and a $(k + 1)$-state CFSM with states $1, 2, \cdots, k + 1$. The (d,k) constraints admit binary decisions, e.g., "flux change" or "no flux change" as in magnetic recording, and are such that the constrained CFSM always returns to the same "home" state following a flux change. We denote the two-symbol version of the CFSM to be a unit-time CFSM. Thus a one-state $(k - d + 1)$-symbol parsing CFSM is equivalent to the two-symbol $(k + 1)$-state unit-time CFSM, where state 1 is the "home" state reached following a flux change. In general the table size is

$$(k + 1) \times C \times (2 - 1) = (k + 1) \times C.$$

Binary alternatives occur at states $d + 1, d + 2, \cdots, k$. There are $k - d$ information-carrying states; for the other channel states the channel symbol is a certainty. The number of nonzero augends, using Eq. (1), is now

$$(k - d) \times C. \tag{5}$$

Thus, a one-state, $(k - d + 1)$-symbol parsing CFSM and its equivalent $(k + 1)$-state, two-symbol unit-time CFSM have the same augend table size under the general L-based approach in [2], as seen by comparison of Eqs. (4) and (5). See the Appendix for a general approach to converting a CFSM governing channel constraints for multiduration comeasurable symbols (the symbol lengths are rational multiples of each other) to a unit-time CFSM.

3. Unit-time fixed rate channel coding

For the special case of (d,k) channels, Martin [2, 3] showed that the number of augends could be reduced to C by properly arranging the calculations. As seen in [3], the augends are independent of the $k - d$ states. Upon analysis, it is seen that the technique works when the CFSM has two properties:

P1. The CFSM can be converted to a 1-state channel parsing CFSM.
P2. The 1-state CFSM has no more than one symbol of a given duration.

In this section, we remove these restrictions in order to implement a code using the simpler unit-time CFSM and reduced table size.

• *Recursions based on* $W^{-\ell(u)}$

The advantage in [3] is achieved by keeping a recursion variable which approximates the value $W^{-\ell(u)}$, where W is the growth factor (the largest real root of a determinant equation of Shannon [1]) and $\ell(u)$ is the length of channel string u. We extend the use of this recursion variable, call it $R(u)$, to both the P-based and L-based approaches for the general fixed-rate case. We update the recursion variable each channel time unit and calculate an addend, which is added or not to the data string $F(u)$ as dictated by the relevant unit-time CFSM transition.

To understand the technique, first, note that the CFSM cannot "loop" among intermediate states. A home state must be reached once a phrase has been parsed from the channel string. Let state I be the initial home state. Order the alphabet of phrases according to symbol duration or length. Thus, let the alphabet of channel phrases be ordered $\alpha, \beta, \cdots, \omega$, where $\ell(\alpha) < \ell(\beta) < \cdots < \ell(\omega)$. The symbol ordering extends naturally to give a lexicographical ordering on all channel strings.

The transformation of channel string u to data string $F(u)$ follows Martin's idea [3] that of all long channel strings [of some length suitably longer than $\ell(u)$], a portion $F(u)$ precede u in the lexicographic ordering. Now suppose the next symbol to be encoded is α. Since α is the first symbol in the ordering, the portion of channel strings preceding $u.\alpha$ is still $F(u)$, so the augend has value 0.

Instead, suppose the next symbol to be encoded is β. Observe that continuations of channel string $u.\alpha$ will precede continuations of $u.\beta$ in the ordering. $F(u)$ accounts for channel strings beginning with a prefix less than u, but not for strings whose prefix is $u.\alpha$.

We need to know what portion of channel strings begin with prefix $u.\alpha$. This value for the augend $M(u,\beta)$ to be added to $F(u)$, is the addend $A(u,\alpha)$. The question of how to determine this value has been neatly answered by Theorem 4 in Shannon [1]; of all channel strings beginning in initial state I, fraction

$$A(u,\alpha) = (B(T(u.\alpha))/B(1)) \times W^{-\ell(u.\alpha)} \tag{6}$$

have prefix $u.\alpha$. In Eq. (6), state $T(u.\alpha)$ is determined by the CFSM next-state function T acting on string $u.\alpha$ from initial channel state I. For convenience, let the eigenvector component for state I be unity, so that augend

$$M(u,\beta) = A(u,\alpha) = B(T(u.\alpha)) \times W^{-\ell(u.\alpha)}. \tag{7}$$

Now suppose instead that the symbol to be encoded following u is γ. We must now account for the portions of the channel strings whose prefixes are $u.\alpha$ and $u.\beta$. Equation (7) takes care of the first term, and the second term is taken care of by adding

$$A(u,\beta) = B(T(u.\beta)) \times W^{-\ell(u.\beta)}. \tag{8}$$

These equations hold for both the parsing CFSM and the unit-time CFSM. Were there more than three symbols in the parsed alphabet, this process would continue. For each symbol $\gamma < \xi$, where ξ is to be encoded, an addend term $A(u.\gamma)$,

$$A(u.\gamma) = B(T(u.\gamma)) \times W^{-\ell(u.\gamma)}, \tag{9}$$

is added to $F(u)$. Thus to transform channel string $u.\xi$, following the transformation of channel string u to $F(u)$, we have

$$F(u.\xi) = F(u) + \sum_{(\gamma<\xi)} B(T(u.\gamma)) \times W^{-\ell(u.\gamma)}. \tag{10}$$

Equation (10) is independent of the number of symbols in the channel string alphabet for the parsing CFSM, which is the basis for our generalization. There is one distinct addend factor $B(T(u.\gamma))$ for each home state $T(u.\gamma)$. The other addend factor $W^{-\ell(u.\gamma)}$ is approximated by a recursion variable. The recursion may be handled in an L-based or P-based manner, as we explain later.

• General L-based unit-time codes

Recursion variable for L-based approach

In the L-based approach, $W^{-\ell(u.\gamma)}$ is approximated by use of the rate $J/C < \log W$. Value W^{-1} is approximated by $2^{-J/C}$, so that $W^{-\ell(u.\gamma)}$ is approximately $2^{-\ell(u.\gamma)\times J/C}$. Denote $2^{-\ell(u)\times J/C}$ by $R'(u)$, which is not yet the desired recursion variable:

$$W^{-\ell(u)} \simeq R'(u) = 2^{\ell(u)\times J/C}. \tag{11}$$

To obtain $R'(u.\gamma)$ from $R'(u)$, we have

$$R'(u.\gamma) = R'(u) \times 2^{-\ell(\gamma)\times J/C}. \tag{12}$$

In L-based codes, the recursion variable is $R = -\log R'$. Taking logs of both sides of Eq. (12), we have

$$R(u.\gamma) = R(u) + \ell(\gamma) \times J/C, \tag{13}$$

which is the length recursion characteristic of L-based arithmetic channel codes. Now

$$W^{-\ell(u.\gamma)} \simeq 2^{-R(u.\gamma)} = 2^{-X(u.\gamma)} \times 2^{E(u.\gamma)}, \tag{14}$$

where the value of $R(u.\gamma)$ is split into an integer part $E(u.\gamma)$ and a fraction part $X(u.\gamma)$; see Eq. (3).

To implement Eq. (10), we replace factor $W^{-\ell(u.\gamma)}$ by the right side of Eq. (14):

$$F(u.\xi) = F(u) + \sum B(T(u.\gamma)) \times 2^{-X(u.\gamma)} \times 2^{-E(u.\gamma)}.$$

Next we replace the factor $B(T(u.\gamma)) \times 2^{-X(u.\gamma)}$ above with a value $m(i,X)$ obtained by table lookup on i (one of n channel states) and X (a value $0, 1, \cdots, C-1$). The remaining factor $2^{-E(u.\gamma)}$ performs a data handling function. The addend factor $m(i,X)$ is "scaled" by multiplication by $2^{-E(u.\gamma)}$, which amounts to an integer right shift. This right shift aligns addend factor $m(i,X)$ to the right end of $F(u)$. In practice, this data handling function is more easily performed by a left shift of $F(u.\gamma)$ by the amount $E(u.\gamma) - E(u)$. The result is the same—the proper relative position of the looked-up addends $m(i,X)$ and the right end of $F(u)$. The L-based embodiment of Eq. (10) now yields

$$F(u.\xi) = F(u) + \sum_{\gamma<\xi} m(i(u.\gamma),X(u.\gamma)) \times 2^{-E(u.\gamma)}. \tag{15}$$

If there are n states in the parsing CFSM, then a table of $n \times C$ addend factors $m(i,X)$ is sufficient to implement Eq. (15). This gives the desired result of making the table size independent of the cardinality of the parsed symbol alphabet for the parsing CFSM.

Consistency test

One of the most important aspects of this paper is the consistency test which addend factors $m(i,X)$ must satisfy. In [3] we introduced the notion of a code space of finite strings which at each iteration is subdivided into as many parts as there are next symbols. Since we are inverting the operations, the code space of compression codes corresponds here to the data strings. Here we employ the term "data" space. Channel strings which are continuations of channel string u are mapped to the data space between $F(u)$ and $F(u+1)$. For representability, the subdivision operation on the data space cannot leave a gap. In L-based (length-oriented) codes, the size of the data space which is subdivided, in order to determine the gap, is difficult to calculate precisely. After encoding prefix u, the left (lower) side of the data space is $F(u)$. The extent of the right (upper) boundary of the data space used by continuations of u is $F(u.\omega\omega\cdots)$, i.e., at each symbol position one adds the largest augend. Thus, $F(u.\omega\omega\cdots)$ is a never-ending sum. For channel codes, we must ensure that there is no gap in the data space, i.e., that $F(u.\omega\omega\cdots)$ is equal to or larger than $F(u+1)$. If $F(u.\omega\omega\cdots)$ is less than $F(u+1)$, then we have a gap in the data space. Data strings whose values lie in this gap [between $F(u.\omega\omega\cdots)$ and up to but not including $F(u+1)$] are not representable. Let $D(u,\omega\omega\cdots)$ be defined as $F(u.\omega\omega\cdots) - F(u)$, where $F(u.\omega\omega\cdots)$ is calculated by first obtaining $F(u)$ by transforming string u and then by adding a succession of augends to $F(u)$. In other words $D(u,\omega\omega\cdots)$ must equal or exceed the difference $F(u+1) - F(u)$ to prevent a gap, where $F(u+1)$ is obtained by transforming string $u+1$.

A consistency test guarantees that there is no gap. The test is discussed in [2, 3]. The test on augends M is

$$M(j,X,\gamma+1) - M(j,X,\gamma) \le M(T(j,\gamma),(J\times\ell(\gamma)+X) \bmod C,\omega+1) \times 2^{-\lfloor (J\times\ell(\gamma)+X)/C \rfloor}.$$

The left-hand side is the difference between the augends of adjacent symbols γ and $\gamma+1$, hence is an addend $m(i,X)$. The right-hand side of this inequality is described as follows. The value $M(j,X,\omega+1)$ is an approximation (from below) to the value of $D(u,\omega\omega\cdots) = D(u,\omega+1)$; see [3]. Let u applied to the CFSM in its initial state and initial value for the internal variable leave the CFSM in state j and internal value X. Now $D(u,\omega\omega\cdots)$ becomes $F(\omega\omega\cdots/j,X)$. We approximate $F(\omega\omega\cdots/j,X)$ from below simply by taking a finite number of terms.

For our unit-time algorithm, we modify this test to use addends directly. We still perform the consistency test only on the home states of the parsing CFSM and the parsed phrases. We must also reformulate the test.

Consider the addend $m(i,X)$. This corresponds to a subinterval size on the unit line. From state i, we have a set α,β, etc., of allowed channel (extended alphabet) symbols. Denote this set (which depends on i) as $\xi(i)$, with typical member γ:

$$m(i,X) \le \sum_{\gamma\in\xi(i)} m(T(i,\gamma),(J\times\ell(\gamma)+X) \bmod C) \times 2^{-\lfloor (J\times\ell(\gamma)+X)/C \rfloor}. \tag{16}$$

The right-hand side of Eq. (16) is a lower bound (we take a finite number of terms) on $F(\omega\omega\cdots/i,X)$ starting in state i with internal length variable value X.

See Section 4 for the mechanization of the encoder and decoder transformations. Section 5 generalizes the transformations to include symbols of the same length by applying their addends in order in a single channel time unit.

- *P-based unit-time arithmetic codes*

The fixed rate L-based arithmetic channel code extends to fixed rate P-based codes. Earlier we described the basics of the approach for both L-based and P-based codes. In the P-based approach, we implement Eq. (10) differently. Let the recursion variable be $R'(u)$. Find a suitable q-bit fractional value $Q > W^{-1}$. Let recursion variable $R'(u)$ be the product of repeated multiplication by Q:

$$W^{-\ell(u)} \simeq R'(u) = Q^{\ell(u)}. \tag{17}$$

Since $R'(u)$ must have a fixed precision, the product

$$R'(u.\gamma) = \langle R(u) \times Q \rangle, \tag{18}$$

must be appropriately rounded. The notation $\langle x \rangle$ means that the result x is rounded to q-bit precision. To implement Eq. (10), when we reach an intermediate state of the unit-time CFSM transition to state $T(u.\gamma)$ under symbol γ, an addend is required. We must form the product $B(T(u.\gamma)) \times R(u.\gamma)$ and add the result to $F(u)$.

The data handling aspect is no different from ordinary P-based arithmetic compression codes [5]. The left-shifting of F is controlled by a corresponding realignment to F whenever the value in R' is renormalized. The renormalization is a consequence of new leading zeros introduced by the product of Eq. (18).

4. Mechanization of the L-based unit time code

In this section we describe the operations involved in the fixed rate L-based approach for constrained channels with Property P2. We discuss the steps performed during encoding/decoding, then the method to calculate the addends, followed by an example. The next section extends the technique to codes without Property P2.

- *Implementation*

We describe the channel coder and decoder for generalized run length limited codes. The hardware described can be used to implement any CFSM for which there is at most one symbol of a given length from a given state.

The hardware generates a channel string as a sequence of symbols **0** and **1**. We let variable t remember the channel unit time modulo C. The unique symbol of length $t + 1$ from state s is generated as t **0**'s followed by one **1**.

We discuss the coder, then the decoder, and finally the sizes of tables and arithmetic needed to implement particular codes.

The channel coder

The channel coder consists of the following:

Four registers **i, t, F,** and **X,** respectively, for the variables
- i (about 4 bits)
- t (about 4 bits)
- F (about 10 bits)
- X (about 2 bits)

Two constants: J, C (about 3 bits each)

Two ROM arrays:
- $T^*(s,n)$ (about $16 \times 16 \times 4$ bits)
- $m(s,X)$ (about $16 \times 4 \times 8$ bits)

- **i** is set by table lookup from T^*.
- **t** is set by addition of 1 or setting to 0.
- **F** is set by subtraction of a value from m and by shifting.
- **X** is set by addition of constant J modulo constant C.

The coder generates one channel bit for each major iteration. It consumes one source bit synchronously for C of each J iterations. The coder algorithm is as follows:

Step 1 Initialize variables as follows: F with the leading $|F|$ bits of source data, set $X = J$, $i = 1$, $t = 0$.

Step 2 Check for end of source data. If it is the end, go to step 6.

Step 3 Compare F with $m(T^*(i,t),X)$.
If $F \geq m(T^*(i,t),X)$
then
output **0**
$F = F - m(T^*(i,t),X)$
$t = t + 1$
else
output **1**
$i = T^*(i,t)$ (update new value of i from old value)
$t = 0$.

Step 4 Add J to X. The result of the sum mod C is retained as the new value of X, and the carry governs a shift. If a carry occurs, then shift F left one bit and shift the next bit of source data into the right of F. (The bit shifted out of F will always be 0.) This addition may be carried out using an adder. (Alternatively, since step 4 cycles and is independent of the data values, a circular shift register of the correct length could be preloaded with the correct new values of X and carry.

Step 5 Go to step 4.

Step 6 Handle end of source data, i.e., handle the *end effect*.

One option is to let the coder run for additional cycles until it has consumed Q dummy bits of source. These bits are set to 1. Q is code dependent.

The channel decoder

The channel decoder has the same registers as the encoder. It consumes one channel bit for each major iteration. It generates one tentative reconstructed source bit at the rate of J for each C iterations. With the algorithm as we describe it here, the reconstructed source must be buffered to handle certain overflows.

The decoder algorithm is as follows:

Step 1 Initialize $F = 0, X = J, i = 1, t = 0$.

Step 2 Check for end of channel data. If it is the end, go to step 6.

Step 3 Read next channel bit. If **0**
then
$F = F + m(T^*(i,t),X)$
$t = t + 1$
else
$i = T^*(i,t)$
$t = 0$.
An overflow may occur in the addition to F. In this case, 1 is added to the rightmost bit of the tentative reconstructed source. This addition may cause propagation up to the top of this string.

Step 4 Add J to X mod C. If there is a carry, shift F left one bit, introducing a 0 in the right. The leftmost bit shifted out is concatenated to the right of the reconstructed source.

Step 5 Go to step 2.

Step 6 Handle the end effect in a way compatible with the encoder. For the method described above, decode until the end of channel string is reached, and ignore the trailing Q bits of reconstructed source.

Sizes

Sizes of the registers and tables depend on the code to be implemented. Let the CFSM defining the model have n states and the longest symbol be of length k.

J/C is a rational approximation from below to the rate W of the code. The sizes needed for J and C depend on this approximation.

- **i** must be large enough to define the states, $1 \le i \le n$.
- **t** must be large enough to describe symbol length so far, $0 \le t < k$.
- **T*** has $n \times k$ entries, each large enough to hold next state j, $0 \le j \le n$.
- **X** must be large enough to hold $0 \le X < C$.
- **m** requires $n \times C$ entries; i.e., each requires approximately $-\log 2[((\log W) - J/C)/\log W]$ bits, where the number in square brackets is the *inefficiency* of the approximation J/C to W.
- **F** requires approximately two more bits than **m**.

As an example, consider the charge-constrained (2,7,8) code. There are 14 states ($n = 14$), so register **i** has 4 bits. The maximum symbol length is 8, so register **t** is 4 bits. Table **T*** has 14×8 entries, each of 4 bits. Rate J/C is $\frac{1}{2}$, so register **X** is one bit. Growth rate W is 1.415, so channel capacity $\log W$ is 0.501 information bits per channel time unit. The inefficiency is $\simeq 0.001/0.5$ or 0.002. Register **F** should therefore have about $9 + 2 = 11$ bits.

We have great freedom in what code to choose, as any code is implementable. We may often choose a weaker code with greater theoretical capacity $\log W$ but use the same J/C. Thus we are working with greater inefficiencies and can use fewer bits in **F**, **m**, and the adder.

For example, a (2,8,8) code has theoretical capacity 0.510. At rate 0.5 this gives inefficiency about (0.010/0.5) or 0.02. This requires an **F** register size of about 8 bits instead of 11 bits for (2,7,8).

- *Computing the table lookup data*

Basic steps

We compute the table lookup data in the following steps. They are based on data from the given (parsed form) CFSM. Computation of m involves experimental iteration on a value "scale."

*Computation of T**:

$T^*(i,t) = T(i,\xi)$, where there is a symbol ξ of length $t + 1$ from state i.
$= 0$, otherwise.

Computation of m (four steps):

Step 1. Choose some scale factor "scale."

Step 2. Set $m(j,X) = \lfloor \text{scale} \cdot B_j \cdot 2^{-X/C} \rfloor$. (19)

Step 3. Check the consistency of these values for m. We wish

$$m(j,X) \le \sum_{\gamma} m(T(j,\gamma),(J \times \ell(\gamma) + X) \bmod C) \times 2^{-\lfloor (J \times \ell(\gamma) + X)/C \rfloor}, \quad (20)$$

where γ ranges over all phrases in the parsing CFSM alphabet. [See Eq. (16).]

Step 4. If the check succeeds for all j,X, we are done. If not, we retry with a larger scale. Theoretically this iteration should terminate for "scale" sufficiently large.

The iteration to find m may be modified with the aim of finding a set of smaller values of m following the techniques of Todd and Langdon [6].

Example

Consider the finite state machine with four states 1, 2, 3, and 4 and three symbols, α, β, and γ, of length 2, 3, and 4, respectively. The transitions are given in the following matrix.

from state j	*to state* $T(j,\xi)$			
	1	*2*	*3*	*4*
1	–	α	β	γ
2	α	–	γ	β
3	–	β	α	–
4	α	β	γ	–

We compute a rate and find eigenvector components. We find

$W = 1.42065$, rate $= 0.50655$
$B(1) = 1.00000$
$B(2) = 1.01680$
$B(3) = 0.70289$
$B(4) = 1.02266$

Given this rate, we choose $J = 1$ and $C = 2$. We now compute values for m from Eq. (19), and the "check" from Eq. (20). With scale = 32 we have the following:

j	*X*	*m*	*check*
1	0	32	31.5 *
1	1	22	22.75
2	0	32	32.5
2	1	23	22.75 *
3	0	22	22.5
3	1	15	15.5
4	0	32	33
4	1	23	22.75 *

We see that inequality Eq. (16) fails for three values of m. We try scale = 64 to get

j	*X*	*m*	*check*
1	0	64	64.25
1	1	45	45.5
2	0	65	65.5
2	1	46	46.25
3	0	44	45
3	1	31	31.75
4	0	65	66
4	1	46	46.5

5. General CFSMs with symbols of same duration

In Section 4, we dealt only with constrained channels whose channel symbol alphabet permitted only one parsed phrase of a given length (Property P2). To generalize the above hardware for a general CFSM where the symbols have co-measurable length and more than one symbol of the same length (Property P2 does not hold), we operate as follows.

Let there be $G(t)$ symbols of length $t + 1$. The gth CFSM symbol of length $t + 1$ (sym(t,g)) will be generated in the channel string as t **0**'s followed by symbol g. T^* is now indexed by state i, length count t, and g. We replace step 3 in the coder by an iteration:

```
do g = 1 to G(n)
        if m(T*(i,t,g),X) < F
        then go to gfound
        F = F − m(T*(i,t,g),X)
        end of "do" loop
output channel symbol 0 /* fell out of the loop as F large */
        t = t+1
        go to end_step_3
gfound:
        output channel symbol g
        i = T*(i,t,g)
        t = 0
        go to end_step_3
```

We replace step 3 in the decoder by an iteration:

```
        collect channel symbol gg
        if gg = 0 then num = G(t); else num = gg − 1
        do g = 1 to num
                F = F+m(T*(i,t,g),X)
                end of "do" loop
        if gg = 0
        then t = t+1
        else do
            i = T*(i,t,gg)
            t = 0
            end
```

6. Summary

We have extended Martin's original L-based fixed rate unit-time (d,k) channel code approach [2, 3]. The extension is in two directions. First the simpler unit-time approach now can be used on any CFSM-defined constrained channel with symbols of co-measurable durations. We do this by causing the addend to depend only on the parsing CFSM home state. Second, the above improvement for fixed rate L-based arithmetic codes for channels extends to fixed rate P-based arithmetic codes for channels. The major advantage of the unit-time technique is evident when applied to an L-based (2,7,8) code. The method in [2] requires a table size, per Eq. (1), of

$14 \times 5 \times 2 = 140$ entries.

Our approach only requires $14 \times 2 = 28$ entries for the addend table. Moreover, the code is more easily realized in digital system components. For channel constraints yielding a rate where the value of C is large, a P-based approach may offer the simpler implementation.

Appendix: Conversion to unit-time CFSM

We show here how to trade fewer channel states for more channel symbols and vice versa. The basis of the unit-time

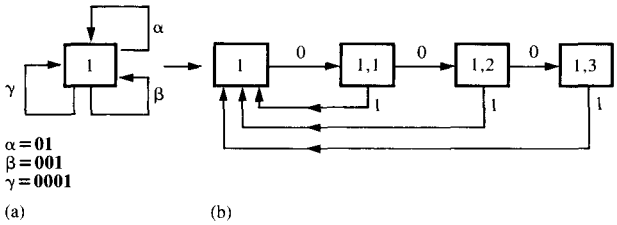

Figure 1 Conversion of parsing 1-state CFSM to unit-time CFSM: (a) parsing CFSM; (b) unit-time CFSM.

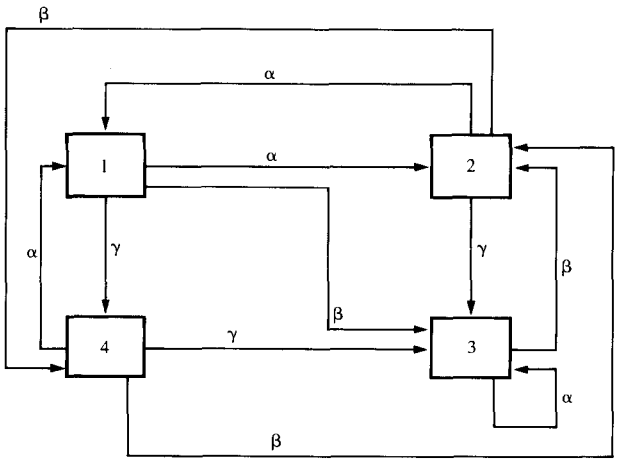

Figure 2 Generalized (1,3) constrained channel with four home states.

Table 1 Converting parsing CFSM to unit-time CFSM.

1. Begin with new home state $i,0$ ($i = 1$ initially).
2. Begin with longest parsed phrase ω, of length $\ell(\omega)$, and generate *intermediate states* $i,1$, $i,2$, $\cdots$, $i,\ell(\omega - 1)$. Intermediate state $i,\ell(\omega - 1)$ goes to state $T(i,\omega)$. For symbols ξ, where $\ell(\xi) < \ell(\omega)$, new intermediate states are not generated. Rather, for each symbol ξ, the new transition implemented is from state $i,\ell(\xi) - 1$ to home state $T(i,\xi),0$.
3. Continue the process of step 2 above with other states. Beginning with the longest allowed symbol first, generate the intermediate states.
4. After the transitions of the allowed symbols from state i of the parsing CFSM have been converted, if the last home state has not been converted, go to step 1 to handle next home state $i + 1$.

conversion is that neither the encoder nor the decoder is operating on the CFSM which defines the code. Code definition is more easily done with fewer channel states on an "extended" alphabet of parsed phrases. Following the code design (see Section 5), we allow the encoder/decoder to operate on a machine with simpler symbols, each of unit duration, but with more states. The original CFSM is the parsing CFSM, and the new CFSM is a unit-time CFSM. In order to take advantage of an easier code design process, while also taking advantage of an easier implementation method, we must be able to convert between the equivalent parsing CFSM and the unit-time CFSM.

In this section we show how (d,k) constrained channel CFSMs with n home states in the parsing CFSM can be systematically converted to unit-time CFSMs. This is a preparatory step to a realization with a fixed rate J/C L-based arithmetic code with augend table size of $n \times C$ entries instead of $n \times C \times (|\xi| - 1)$ entries.

We first describe the conversion process of a one-state $(k - d + 1)$-symbol (symbols of multiple unit duration) parsing CFSM to a $(k + 1)$-state two-symbol (each of unit duration) unit-time CFSM. See Fig. 1. Consider the 1-state (1,3) channel whose three symbols are denoted α, β, and γ, shown in Fig. 2. This is exactly the transformation used by Martin in his unit-time code for (d,k) codes.

We can describe the states of the unit-time CFSM with two components, the first of which is the last home state. The second component is the distance, in channel time units, from the occurrence of that state. The unit-time CFSM is defined if we define all its states and state transitions.

We keep existing home state *1* and call it *1,0*. Begin with the longest symbol γ first. Symbol γ is converted to its unit-time equivalent of symbol **0001**, which has a length $\ell(\gamma) = 4$. We create $\ell(\gamma) - 1 = 3$ intermediate states: state *1,1*, state *1,2*, and state *1,3*. The next state function for the parsing CFSM is T, and $T(1,\gamma)$ is state *1,0*. So from state $1,\ell(\gamma) - 1$, the transition to $T(1,\gamma)$ is drawn. Now consider the next shortest symbol β, which is **001**. We begin in state *1,0*, and deal with each unit-time symbol. First symbol **0** already takes the CFSM to state *1,1*. For the second **0**, we travel two channel time units from the home state to intermediate state *1,2*, which is state $1,\ell\beta - 1$. The third symbol is **1**, which returns the CFSM to home state *1,0*. The trajectory for symbol α = **01** is handled the same way; i.e., when state $1,\ell(\gamma) - 1$ is reached, the next symbol (**1**) takes the unit-time CFSM to state $T(1,\alpha) = 1,0$.

The rules for the unit-time CFSM transitions governing each parsed phrase α from state i are as follows. As long as state $i,\ell(\gamma) - 1$ has not been reached, the transition is by unit-time symbol 0 from state i,j to $i,j + 1$. The transition from state $i,\ell(\gamma) - 1$ is by unit-time symbol 1 to home state $T(i,\alpha),0$.

We have described the process in such a way that it is easily generalized to parsing channel CFSMs which have more than one "home" state. The rules for the general case are shown below in Table 1. The method works because no parsed phrase is permitted to be the prefix of another parsed phrase.

Example

We show a generalized (1,3) parsing channel CFSM in Fig. 2. The CFSM has four home states, *1*, *2*, *3*, and *4*. The first state *1* is converted to intermediate states as shown in Fig. 3.

Other parsing CFSM home state transitions are similarly converted to unit-time CFSM transitions. In Fig. 4 we show how state *3* is converted. In state *3*, symbol γ is not allowed, so that there is no intermediate state *3,3*.

For state *2*, all three symbols are allowed, so that three intermediate states are generated. For state *4*, all three symbols are allowed, so that three intermediate states are generated. Thus, the total number of states in the unit-time CFSM is 15: the 4 home states plus 11 intermediate states.

The procedure of Table 1 need not require Property P2. That is, there can be two parsed phrases of the same length. Such is the case where three levels of magnetic recording are permitted. The parsed phrases must have distinct unit-time transitions. The method of Table 1 generates a unit-time CFSM provided that no parsed phrase is the prefix of another phrase from the same home state.

The design advantage can be explained in terms of the previous example. We calculate the growth factor W and the 4 associated eigenvector components $B(1)$, $B(2)$, $B(3)$, and $B(4)$ for the 4-state parsing CFSM. This is a simpler task than for the equivalent 14-state unit-time CFSM. In our technique, only W and the home state eigenvector components need be known. Next we convert the 4-state CFSM to a 14-state CFSM, which is easier to realize.

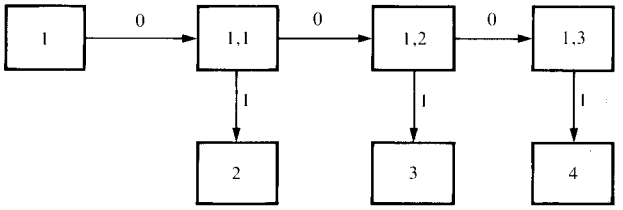

Figure 3 Conversion of state 1 of 4-state parsing CFSM to unit-time CFSM intermediate states.

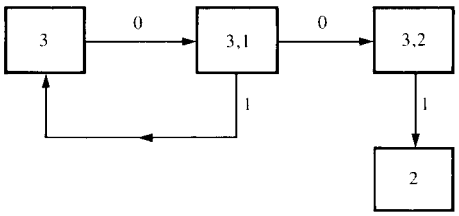

Figure 4 Conversion of state 3 of 4-state parsing CFSM to unit-time CFSM intermediate states.

References

1. C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.* **27,** 379–423 (July 1948).
2. G. Nigel N. Martin, "Range Encoding for Discrete Noiseless Channels," unpublished internal memorandum, IBM UK Scientific Centre, March 1980.
3. G. Nigel Martin, Glen G. Langdon, Jr., and Stephen J. P. Todd, "Arithmetic Codes for Constrained Channels," *Research Report RJ-3381*, IBM San Jose Research Laboratory, January 21, 1982.
4. S. J. Hong and D. L. Ostapko, "Codes for Self-Clocking, AC-Coupled Transmission: Aspects of Synthesis and Analysis," *IBM J. Res. Develop.* **19,** 358–365 (July 1975).
5. G. Langdon, "Tutorial on Arithmetic Coding," *Research Report RJ-3128*, IBM San Jose Research Laboratory, May 6, (1981).
6. Stephen J. P. Todd and Glen G. Langdon, Jr., "Augend Computations for Arithmetic Channel Codes," *IBM Tech. Disclosure Bull.* **25,** 1127–1129 (August 1982).

Received January 22, 1982; revised September 29, 1982

Glen G. Langdon, Jr. *IBM Research Division, 5600 Cottle Road, San Jose, California 95193.* Dr. Langdon received the B.S. from Washington State University, Pullman, in 1957, the M.S. from the University of Pittsburgh, Pennsylvania, in 1963, and the Ph.D. from Syracuse University, New York, in 1968, all in electrical engineering. He worked for Westinghouse on instrumentation and data logging from 1961 to 1962 and was an application programmer for the PRODAC computer for process control for most of 1963. In 1963 he joined IBM at the Endicott, New York, development laboratory, where he did logic design on small computers. In 1965 he received an IBM Resident Study Fellowship. On his return from Syracuse University, he was involved in future system architectures and storage subsystem. During 1971, 1972, and part of 1973, he was a Visiting Professor at the University of Sao Paulo, Brazil, where he developed graduate courses on computer design, design automation, microprogramming, operating systems, and MOS technology. The Brazilian computer called Patinho Feio (Ugly Duckling) was developed by the students at the University of Sao Paulo during his stay. He is author of *Logic Design: A Review of Theory and Practice,* an ACM monograph, and coauthor of the Brazilian text *Projecto de Sistemas Digitais;* he has recently published *Computer Design.* He joined the IBM Research laboratory in 1974 to work on distributed systems and later on stand-alone color graphic systems. He has taught graduate courses on logic and computer design at the University of Santa Clara, California. He is currently working in data compression. Dr. Langdon received an IBM Outstanding Innovation Award for his contributions to arithmetic coding compression techniques. He holds eight patents.

G. Nigel N. Martin *IBM United Kingdom Laboratories Ltd., Hursley House, Hursley Park, Winchester, Hampshire S021 2JN, England.* Mr. Martin joined IBM Information Services in 1968 as a systems programmer. In 1973, he joined the IBM United Kingdom Scientific Center, where he worked with others to provide a relational database interface to IMS. He took a year at Warwick University in 1978 to research data access techniques. He is currently working on data channels at Hursley. Mr. Martin received a B.A. from Cambridge University and an M.A. in 1968.

Stephen J. P. Todd *IBM United Kingdom Scientific Centre, Athelstan House, St. Clement Street, Winchester, Hants S023 9DR, England.* Mr. Todd joined the IBM Scientific Center in 1971 and has worked there since except for a two-year assignment at the Research Division in San Jose, California. During this assignment, he researched text processing and coding theory. Most of his other work with IBM has been on relational data base systems, with some work on automated offices and image processing. He is currently working on graphics. Mr. Todd received a B.A. in mathematics from Oxford University, England, in 1968, and an M.A. in 1969.