W. W. Peterson

M. O. Rabin

On Codes for Checking Logical Operations

Abstract: Two types of codes for checking logical operations digit by digit on two vectors of binary digits are studied. The first type attaches a check symbol to each vector of binary digits and requires that the check symbol for the logical function of two vectors can be determined from the check symbols of the two input vectors. The second type of coding is ordinary block coding into vectors of binary digits, with the added requirement that the coded vectors be processed digit by digit.

The constraints on the codes resulting from the assumptions for the coding system are studied by typical algebraic arguments. It is shown that for both types of coding and for all nontrivial logical functions of two variables, except "exclusive or" and its complement, there is no system of checking simpler than duplication. For "exclusive or" and its complement, group alphabets can be used, and for the block coding these are the only codes which can be used.

Introduction

Checking the operation of a two-input, one-output logical device such as an "and" circuit requires checking apparatus which has three inputs (the inputs and output of the circuit to be checked) and one output (the result of the check). Thus, the check requires that the equipment be more than doubled for the simple detection of errors.

If the problem were to transmit one binary digit of information, checking would require that the bit be repeated, and thus would require at least either twice as much time or twice as much equipment. If, on the other hand, many binary digits are to be transmitted, it is possible to check much more economically. For example, a "parity check" symbol added to a sequence of arbitrarily many binary digits enables single-error detection, and the Hamming code permits correction of any single errors and detection of double errors with the addition of relatively few parity-check digits.[1]

This suggests consideration of checking a number of similar logical operations simultaneously, which is the subject of this paper. The general approach is to define a type of coding and then study the consequences of the requirement that the coding has to be compatible with the processing. The types of coding considered must be chosen carefully to avoid misleading results. For example, a device which detects single errors in the output but which misses double errors in the output resulting from the failure of a single circuit component could hardly be considered "single-error-detecting."

The two general types of coding considered are illustrated in Figs. 1 and 2. In the first case, blocks of information are processed in the usual way and in addition check symbols, one for each block, are processed separately to give a check symbol for the result. For this system, there is no restriction assumed on the type of check symbols or the method of processing, except that it be compatible with the processing of the information. In the second case, it is assumed that blocks of k binary digits to be processed are coded into blocks of n binary digits, which are processed digit by digit (either serially or in parallel). The n-digit output is decoded into a k-digit sequence.

It is known that the parity-check-type binary codes used in transmission of information[1,2] can also be used for checking the logical operation "exclusive or" and its complement. For all other nontrivial logical operations, all coding schemes considered require complete duplication of equipment for single-error detection, triplication for single-error correction, et cetera.

The second type of coding described above was considered by Elias,[3] and he obtained the principal results for this type of coding. The theory is carried out here in somewhat more detail, thus giving a clearer concept of the constraints on error-checking codes for logical operations.

Check-symbol coding

Let X, Y, Z denote vectors or sequences of k binary digits, and x_i, y_i, z_i the i^{th} digits in the respective vectors. The "and" operation on vectors means digit-by-digit "and,"

that is, $X \cdot Y = Z$ means that for each i, $x_i \cdot y_i = z_i$. For example, if $k=3$, $X=(1, 0, 1)$, $Y=(0, 1, 1)$, then $X \cdot Y = (0, 0, 1)$.

Now consider check-symbol coding, which is represented in Fig. 1. Two sequences X and Y are combined to form a new sequence $X \cdot Y$. To the first input X is associated a check symbol which is denoted $a(X)$ to indicate that it is a function of X, i.e., that different sequences may have different check symbols. Similarly, to the second input Y is associated $b(Y)$, a check symbol which may be a different function, and to the output Z is associated $c(Z)$. The check symbols $a(X)$ and $b(Y)$ are put into a checking device, and the output is supposed to be the check symbol which matches the output of the logical device being checked, i.e., $c(Z)$. This condition for compatibility of the checking with the logical operation on vectors may be expressed

$$a(X) * b(Y) = c(X \cdot Y),$$

where $*$ represents the operation done by the checking device.

This is an extremely general check-symbol coding scheme. A different code is allowed at each input and at the output. The check symbols need not be processed in any special way; in fact, no assumption has been made as to the character of the check symbols except that they be compatible with the logical operation. Even with these very general assumptions, the following result holds:

• *Theorem 1*

If for a given choice of functions $a(X)$, $b(Y)$, and $c(Z)$, and the operation $*$, if the check-symbol coding described above detects all single errors, the number of different check symbols $a(X)$ is 2^k. The same holds for $b(Y)$.

Proof: Detection of single errors requires that whenever only one of the digits in the output vector is incorrect, the error can be detected. This implies that if Z and Z' differ in exactly one position, $c(Z) \neq c(Z')$. Let X and X' be two different k-bit vectors, and suppose that they differ in the i^{th} position. Let D_i denote a vector which has a one in the i^{th} position and zeros in all other positions. Then $X \cdot D_i$ and $X' \cdot D_i$ differ in exactly one position. Therefore,

$$c(X' \cdot D_i) \neq c(X \cdot D_i),$$

and

$$a(X') * b(D_i) \neq a(X) * b(D_i).$$

Clearly then $a(X') \neq a(X)$, and the 2^k different k-bit input vectors thus have pairwise unequal check symbols; this completes the proof.

If in particular the check symbols are m-bit vectors, then $m \geqq k$, so that the attempt to check for single errors involves as many calculations as the operation to be checked, or in other words, is no simpler than complete duplication.

There are a total of sixteen different logical functions of two variables. Six of these ($f(x, y) = 1, 0, x, y, \bar{x}$, and $\bar{y}$) are trivial because they depend upon only one or neither

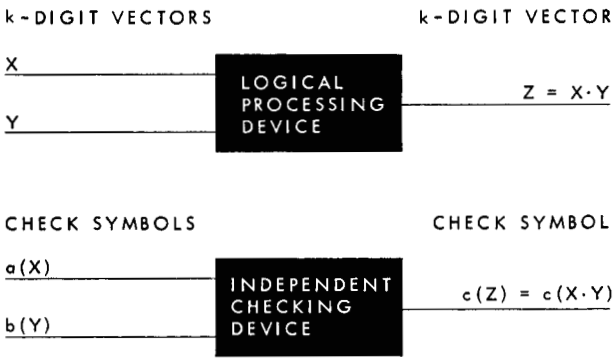

Figure 1 **Check-symbol coding.**

of the arguments. Two others, "exclusive or" ($f(x, y) = x \cdot \bar{y} \cup \bar{x} \cdot y$) and its complement ($f(x, y) = x \cdot y \cup \bar{x} \cdot \bar{y}$), can be checked using parity digits in error-detecting and correcting codes of the type used in information transmission.[1,2] The remaining eight functions are:

$f(x, y) = x \cdot y$, $x \cdot \bar{y}$, $\bar{x} \cdot y$, $\bar{x} \cdot \bar{y}$, $x \cup y$, $x \cup \bar{y}$, $\bar{x} \cup y$, and $\bar{x} \cup \bar{y}$.

Theorem 1 (and the method of proof) applies to all nontrivial logical functions except "exclusive or" and its complement.

Somewhat stronger results can be obtained if it is assumed that (1) the same type of encoding is used at both inputs and output, i.e., $a(X) = b(X) = c(X)$ for all X, and (2) both "and" and "exclusive or" are to be checked, i.e., there are two types of checking devices, one to be used with an "and" device and one with an "exclusive or" device. The symbol $\oplus$ will be used to denote "exclusive or" of vectors of binary digits.

• *Theorem 2*

Let S be a set of symbols for which two operations $\diamond$ and $\oplus$ are defined, and let $C(V)$ be a function which attaches an element of S to each vector V, in such a way that:

$$C(V_1 \oplus V_2) = C(V_1) \oplus C(V_2)$$

and

$$C(V_1 \diamond V_2) = C(V_1) \diamond C(V_2).$$

Then the number of distinct elements of S which appear as check symbols of some V is a power of 2, say 2^m, and there is a subset of m of the coordinates of V such that $C(V)$ depends only upon these coordinates and is different for any two vectors which differ in any of these coordinates.

Note that it has not been assumed that the operations $\cdot$ and $\oplus$ defined on the set S of check symbols are actually multiplication or any kind of addition. However, the assumptions that $C(V_1 \oplus V_2) = C(V_1) \oplus C(V_2)$ and $C(V_1 \diamond V_2) = C(V_1) \diamond C(V_2)$ make these operations have the properties of addition and multiplication, as will appear in the proof, and hence the notation is natural.

Proof: Let D_i denote a vector which has a one in the i^{th} position and zeros elsewhere, and let 0 denote a vector of

all zeros. Then, $C(D_i)$ may be unequal to $C(0)$ for some, say m, of the vectors D_i and equal to $C(0)$ for the remaining k-m vectors. There will be no loss of generality in assuming that the coordinates are arranged so that:

$C(D_i) \neq C(0)$ if $i \leqq m$

$C(D_i) = C(0)$ if $i > m$.

If $0A$ is defined as $C(0)$ and $1A$ is defined as A, for A belonging to S,

$C(aV) = aC(V)$,

where $a=0$ or 1. Then any vector $V=(v_1, v_2, \ldots v_k)$ can be represented as follows:

$V = v_1D_1 \oplus v_2D_2 \ldots\ldots\ldots\ldots \oplus v_kD_k$

and, therefore,

$C(V) = v_1 \cdot C(D_1) \oplus v_2 \cdot C(D_2) \ldots\ldots\ldots \oplus v_mC(D_m)$.

The $m+1^{st}$ to k^{th} components do not appear because $C(D_i)=C(0)$ for these components, and $C(X) \oplus C(0) = C(X \oplus 0) = C(X)$ for any vector X. Thus, $C(V)$ depends only upon the first m components of V and not at all upon the last k-m components.

If V does not have all zeros in its first m positions, and if the i^{th} position contains a "one," then

$C(V \cdot D_i) = C(V) \cdot C(D_i)$

and since

$V \cdot D_i = D_i$,

$C(D_i) = C(V) \cdot C(D_i)$.

Now if $C(V)=C(0)$, then $C(D_i) = C(0) \cdot C(D_i) = C(0 \cdot D_i) = C(0)$ contrary to the assumption that $C(D_i) \neq C(0)$ for the first m positions. Therefore, $C(V) \neq 0$.

Now suppose V_1 and V_2 differ in at least one of the first m positions. Then, $V_1 \oplus V_2$ is not all zeros in these positions, and hence

$C(V_1 \oplus V_2) \neq C(0)$.

Then

$C(V_1) \oplus C(V_2) \neq C(0)$

and

$C(V_1) \neq C(V_2)$,

for if they were equal,

$C(V_1) \oplus C(V_2) = C(V_1) \oplus C(V_1) = C(V_1 \oplus V_1) = C(0)$.

Thus any two vectors which differ in the first m coordinates have different check symbols and the check symbol depends only upon the first m coordinate. There must then be 2^m different check symbols, one for each possible configuration of the first m coordinate.

Since "exclusive or" can be defined in terms of "and" and "not," the theorem holds if an attempt is made to check "and" and "not" rather than "and" and "exclusive or," or similarly, "or" and "not." If a constant input vector of all 1's is allowed, "and" and "not" can both be defined in terms of any *one* logical function except "and," "or," "exclusive or," the negation of "exclusive or," or the six trivial ones. Therefore, an attempt to check any *one* of these remaining six using the same code at both inputs and the output results in the same conclusion.

More general codes

Two types of more general coding systems are check-symbol codes which are capable of correcting errors or detecting multiple errors, and block codes capable of detecting or correcting errors. In either case, it is necessary to classify errors so that the types of errors to be corrected or detected can be specified. If only sequences of binary digits are used, then errors can be classified as single, double, triple, et cetera, according to how many digits are incorrect. In general there is no such simple classification and, therefore, in the remainder of this paper only sequences of binary digits will be considered.

If the failure of a single component in the circuits doing the logical operations can cause two or more digits in the answer to be in error, then "single-error detection" codes might fail to detect the failure of a single component. To insure against this, it will be assumed that the vectors of binary digits are processed digit by digit, and thus one error in processing can affect only one digit. The only alternative appears to be to study the circuits themselves, which is beyond the scope of this paper.

With digit-by-digit processing, no stronger results can be obtained for check-symbol coding (illustrated in Fig. 1) than for general codes (illustrated in Fig. 2), and since for sequences of binary digits the check symbol and information together can be considered the block in the general block coding, check-symbol coding is a special case of block coding. Therefore, the general block coding with digit-by-digit processing is considered next.

Constraints on general block codes with digit-by-digit processing

The coding system to be considered is illustrated in Fig. 2. The k-digit first input $X=(x_1 \ldots x_k)$ is coded into an n-digit vector $U=(u_1 \ldots u_n)$. The second input $Y=$

Figure 2 **Block coding with digit-by-digit processing.**

$(y_1 \ldots y_k)$ is coded into a second n-digit vector $V=(v_1 \ldots v_n)$. Then U and V are combined digit by digit to form an n-digit output vector $W=(w_1 \ldots w_n)$. The assumption "digit by digit" means mathematically that for each i:

$w_i=f_i(u_i, v_i)$.

Note that it is not even assumed that the same function, or rule of combination, is used for every digit. Finally, the vector W is decoded into a k-bit vector Z which is supposed to be

$Z=X \cdot Y$.

It will also be assumed that in the absence of noise, the decoding is one-to-one, i.e., Z is a one-to-one function of W, and therefore W is a single-valued function of Z.

Any code which maps k-digit vectors of binary digits into n-digit vectors can be transformed into a code which maps the vector of all zeros into the vector of all zeros, and which for all practical purposes is completely equivalent to the original code. This can be accomplished by simply locating all the digit positions which are ones in the n-digit vector which is the code for the k-digit vector of all zeros, and complementing these digit positions in all the n-digit coded vectors. Since every code is equivalent to a code of this type, *only codes which map the vector of all zeros into the vector of all zeros will be considered from here on.*

• *Theorem 3*

If $u_i(0)=v_i(0)=w_i(0)=0$ *and* $f_i(u_i(X), v_i(Y))=w_i(X \cdot Y)$, *then either* $w_i(X)=0$ *for all vectors* X, *or* $u_i(X)=v_i(X)=w_i(X)$ *for all* X, *and* $f_i(u_i(X), v_i(Y))=u_i(X) \cdot v_i(Y)$.

Proof: **(a)** If $u_i(Z)=0$, then $u_i(Z)=u_i(0)$ and $f(u_i(Z), v_i(Z))=w_i(Z)=f(u_i(0), v_i(Z))=w_i(0 \cdot Z)=0$. Therefore, if $w_i(Z)=1$, $u_i(Z)=1$. Similarly, if $w_i(Z)=1$, $v_i(Z)=1$.

(b) If for any vector, say X_0, $w_i(X_0)=1$, then

$f_i(u_i(0), v_i(0))=w_i(0 \cdot 0)=f_i(0, 0)=0$

$f_i(u_i(0), v_i(X_0))=w_i(0 \cdot X_0)=f_i(0, 1)=0$

$f_i(u_i(X_0), v_i(0))=w_i(X_0 \cdot 0)=f_i(1, 0)=0$

$f_i(u_i(X_0), v_i(X_0))=w_i(X_0 \cdot X_0)=f_i(1, 1)=1$.

Therefore $f_i(u_i(X), v_i(Y))=u_i(X) \cdot v_i(Y)$ if for any X_0, $w_i(X_0)=1$, and from this point on the latter notation will be used.

(c) If a vector of all 1's is represented by 1, and if $w_i(X_0)=1$, then $u_i(1) \cdot v_i(X_0)=w_i(X_0 \cdot 1)=1$, and hence $u_i(1)=1$. Similarly, $v_i(1)=1$, and it follows that $w_i(1)=1$.

(d) Finally, $u_i(X) \cdot v_i(1)=u_i(X)=w_i(X \cdot 1)=w_i(X)$. Similarly, $v_i(X)=w_i(X)$.

This completes the proof.

In other words, the constraint that the coding must be compatible with the "and" operation requires that the processing of the check symbols be done by an "and" operation also and that the same code be used at both inputs and the output.

Again the theorem can be generalized. It is true for all nontrivial logical operations that if any one of the three codes (either input code or the output code) is given, the others are determined. The relationships are given in Table 1.

Table 1 **Relationship between input and output codes.**

Logical operation being checked	*Relation between codes*
1. $Z=X \cdot Y$ 2. $Z=X \cup Y$ 3. $Z=X \oplus Y$ (exclusive or) 4. $Z=\overline{X \oplus Y}$	$u_i(X)=v_i(X)=w_i(X)$
5. $Z=X \cdot \overline{Y}$ 6. $Z=X \cup \overline{Y}$	$u_i(X)=w_i(X)$ $v_i(X)=\overline{w_i(\overline{X})}$
7. $Z=\overline{X} \cdot Y$ 8. $Z=\overline{X} \cup Y$	$u_i(X)=\overline{w_i(\overline{X})}$ $v_i(X)=w_i(X)$
9. $Z=\overline{X} \cdot \overline{Y}$ 10. $Z=\overline{X} \cup \overline{Y}$	$u_i(X)=v_i(X)=\overline{w_i(\overline{X})}$

It is also true that the operation on the coded digits must be the same as the operation being checked except for the complement of "exclusive or." The proofs for the other cases are similar to the proof given for "and." Furthermore, for Operations 1, 5, 7, and 9 in the Table,

$w_i(X) \cdot w_i(Y)=w_i(X \cdot Y)$

while for 2, 6, 8, and 10,

$w_i(X) \cup w_i(Y)=w_i(X \cup Y)$.

The following theorem completes the characterization of codes for checking "exclusive or" or its complement:

• *Theorem 4*

Let $C(Y)$ *be a function which takes on the values 0 and 1 such that* $C(X)*C(Y)=C(X \oplus Y)$. *Then the operation* $*$ *is "exclusive or," and* $C(X)$ *is a parity check on some subset of the components of* X. *If* $C(X)*C(Y)=C(X \oplus Y)$, *then again* $C(X)$ *is a parity check on some subset of the coordinates of* X, *and* $*$ *is "exclusive or" if an even number of positions are included, the complement of "exclusive or" if an odd number of positions are involved.*

The proof will be given only for "exclusive or." The proof for its complement follows the same lines but is slightly more involved. Let X_0 be a vector for which $C(X_0)=1$. By hypothesis, $C(0)=0$. Then

$$0*0=C(0)*C(0)=C(0\oplus 0)=C(0)=0$$

$$0*1=C(0)*C(X_0)=C(0\oplus X_0)=C(X_0)=1$$

$1*0=1$ similarly, and finally

$$1*1=C(X_0)*C(X_0)=C(X_0\oplus X_0)=C(0)=0\ .$$

Therefore the operation $*$ corresponds to "exclusive or." Now suppose $C(D_i)=s_i$, where D_i is again the vector with a "one" in the i^{th} position and zeros elsewhere. Then since:

$$V=v_1D_1\oplus v_2D_2\oplus v_3D_3\oplus \ldots\ldots\oplus v_kD_k\ ,$$

$$C(V)=v_1s_1\oplus v_2s_2\oplus v_3s_3\oplus \ldots\ldots\oplus v_ks_k\ .$$

This can be considered the definition of a parity check, where $s_i=1$ for the coordinates included in the parity check.

Each digit in the block of n digits to be processed satisfies the hypotheses of Theorem 4 if "exclusive or" or its complement is to be checked. Thus parity checks, and in fact *only* parity checks can be used to check these operations with the type of coding assumed.

If $Y_1\cdot Y_2=Y_1$, we will say Y_1 is contained in Y_2, or $Y_1\subset Y_2$. This will be the case if, and only if, every position which contains a "one" in Y_1 also contains a "one" in Y_2.

The next theorems completely characterize codes for checking the other eight nontrivial logical operations:

• *Theorem 5A*

Let $C(Y)$ be a function which takes on the value 1 or 0, and such that $C(Y_1\cdot Y_2)=C(Y_1)\cdot C(Y_2)$. Then there exists a vector Y_0 such that $C(Y)=1$, if and only if $Y_0\subset Y$. Conversely, for any vector Y_0 the function $C(Y)$ defined to be one if and only if $Y_0\subset Y$ has the property $C(Y_1\cdot Y_2)=C(Y_1)\cdot C(Y_2)$.

Proof: Let Y_0 be the vector which results from combining by the "and" operation all the vectors Y such that $C(Y)=1$. Then $C(Y_0)=1$, and from the definition of Y_0 it is clear that $Y_0\subset Y$ if $C(Y)=1$. Also, if $Y_0\subset Y$, then $Y_0\cdot Y=Y_0$, $C(Y_0)\cdot C(Y)=C(Y_0)$, and since $C(Y_0)=1$, $1\cdot C(Y)=1$, and hence $C(Y)=1$.

The converse follows from the observation that $Y_0\subset Y_1\cdot Y_2$ if and only if $Y_0\subset Y_1$ and $Y_0\subset Y_2$. Therefore $C(Y_1\cdot Y_2)=1$ if and only if $C(Y_1)=1$ and $C(Y_2)=1$.

• *Theorem 5B*

Let $D(Y)$ be a function which takes on the values 1 and 0 and such that $D(Y_1\cup Y_2)=D(Y_1)\cup D(Y_2)$. Then there exists a vector Y_0 such that the function $D(Y)=0$ if and only if $Y\subset Y_0$. Conversely, for any vector Y_0, the function $D(Y)$ defined to be zero if and only if $Y\subset Y_0$ has the property $D(Y_1\cup Y_2)=D(Y_1)\cup D(Y_2)$.

The proof is similar to that for Theorem 5A.

• *Theorem 6*

Let $\overline{C(\overline{Y})}=D(Y)$. If $C(Y_1\cdot Y_2)=C(Y_1)\cdot C(Y_2)$ for all choices of Y_1 and Y_2, then $D(Y_1\cup Y_2)=D(Y_1)\cup D(Y_2)$ for all choices of Y_1 and Y_2, and conversely. Also, if Y_{0c} denotes the Y_0 defined in Theorem 5 for the function C, and Y_{0d} is the Y_0 defined for D, $Y_{0d}=\overline{Y_{0c}}$.

Proof: $D(Y_1\cup Y_2)=D(\overline{\overline{Y}_1\cdot\overline{Y}_2})=C(\overline{\overline{Y}_1\cdot\overline{Y}_2})=C(\overline{Y}_1)\cdot C(\overline{Y}_2)=C(\overline{Y}_1)\cup C(Y_2)=D(Y_1)\cup D(Y_2)$. The proof of the converse is similar. The relation between Y_{0c} and Y_{0d} is shown by noting that the assumed relation between C and D implies that for any vector Y, $C(Y)=1$ if and only if $D(Y)=0$. Let $Y_1, Y_2, Y_3, \ldots$ be a list of all vectors such that $D(Y)=0$. Then $\overline{Y}_1, \overline{Y}_2, \overline{Y}_3, \ldots$ is a list of all vectors Y such that $C(Y)=1$. In the proof of Theorem 5, it is shown that Y_{0c} is the result of combining by the "and" operation all vectors Y such that $C(Y)=1$, i.e.,

$$Y_{0c}=\overline{Y}_1\cdot\overline{Y}_2\cdot\overline{Y}_3\ldots\ldots\ldots$$

Similarly,

$$Y_{0d}=Y_1\cup Y_2\cup Y_3\ldots\ldots\ldots$$

and since in general $\overline{\overline{A}\cdot\overline{B}}=A\cup B$, $Y_{0d}=\overline{Y_{0c}}$.

Thus either Theorem 5A or Theorem 5B applies to every checking code at input or output for all non-trivial logical operations except "exclusive-or" and its complement.

Error detection and correction with block codes and digit-by-digit processing

A binary code detects single errors if and only if every pair of unequal k-digit vectors maps into a pair of n-digit vectors which differ in at least two positions, i.e., have minimum distance 2. Minimum distance 3 permits single-error correction or double-error detection. Minimum distance 4 permits simultaneous single-error correction and double-error detection, or triple-error detection, et cetera. Therefore, the questions of error-correcting ability reduce to questions of minimum distance for the code in question.[1] Now the question of how long a code is required to achieve minimum distance d will be considered for each of the ten nontrivial logical operations.

Theorem 4 reduces the question of checking "exclusive or" and its complement to that of finding group alphabets with the required error-correcting ability, since a code is a group alphabet if and only if all the digits in the coded vectors are parity checks.[2] A number of such coding systems are described in the literature.[1,2,4,5]

In a code for checking the "and" operation each digit in the coded n-digit vector must satisfy the hypotheses of Theorem 5. Then to achieve distance d, the distance between the vector 0 and the vector $W(D_j)$ must have at least d "ones." (D_j again denotes a k-digit vector with a "one" only in the i^{th} position.) Since the only vector which is contained in D_j is D_j itself, the only type of coded digits $w_i(D_j)$ which will be 1 are those for which Y_0 defined in the theorem is D_j itself. Such a digit will be

1 for any vector whose j^{th} component is 1, since any such vector contains D_j. Thus such a digit $w_i(x)$ a copy of the j^{th} digit of X. To maintain minimum distance d, d such digits are required, i.e., d copies of each digit of X are included among the components of $W(X)$.

Analogous results can be obtained for checking the "or" operation, with exactly analogous proofs. In fact, all codes used in checking all nontrivial logical operations except "exclusive or" and its complement must satisfy Theorem 5, and hence any such code must, if it is to achieve distance d, have at least d copies of each uncoded digit among the digits in the coded vector.

Conclusion

The most important results can be summarized as follows:

(1) For single error-detection codes which consist in the original information with a check symbol, as indicated in Fig. 1, there is no simpler system than making the check symbols duplicates of the information for any nontrivial logical operations except "exclusive or" and its complement. These two operations can be checked by a parity digit.

(2) For general block coding, with the output decoding one-to-one in the absence of errors, and if the coded blocks are processed digit by digit, for "and," "or," "exclusive or," and its complement, the same code must be used at both inputs and at the output (assuming that in each case the sequence of all zeros codes into the sequence of all zeros). For the other six nontrivial logical operations, the input and output codes are closely related. For all logical operations except the complement of "exclusive or," the operation done on coded blocks must be the same as the operation being checked. For "exclusive or" and its complement, each digit in the coded blocks is a parity check on some subset of the digits of the uncoded block. In other words, group codes, and only group codes, can be used to check these two operations.

(3) For the same restrictions on coding as in (2), and for all nontrivial logical operations except "exclusive-or" and its complement, there is no simpler coding system with a specified ability to detect or correct errors than a system in which the coded sequence consists of a number of copies of the uncoded sequence.

Elias stated this last result without proof.[3] He also gave some partial results for the case in which it is not assumed that the decoding at the output is one-to-one in the absence of noise. He pointed out, however, that if the decoding is not one-to-one in the absence of errors, there is some information about the inputs which is not about the logical function of the inputs, in the output. In a sense this requires the decoder to do a part of the logical operation, or at least it leaves some doubt as to whether the "processing" and "coding" operations have really been isolated in the analysis.

It should be noted that the problem considered here is essentially that of checking the operation of logical devices. When the logical devices are combined into a system, an over-all system check may still be possible. For example, an adder can be constructed of "and," "or," and "not" devices which cannot be simply checked, and yet the addition operation as a whole can be checked quite well without complete duplication.[6] Also, a computer programmed to solve a complex mechanical problem frequently can be checked by considering the principles of conservation of energy or momentum, although the individual logical operations in the computer are not checked.

It is interesting to note also that among the few very simple problems considered here, some, namely "exclusive or" and its complement, could be checked easily, while others, like "and," could not be checked short of duplication or repetition of the problem. Larger problems also seem to vary greatly in the degree to which their inherent redundancy or structure aids checking.

Finally, while the results given in this paper and those of Elias are not at all encouraging, it is still possible that some economical way to check simultaneously a number of logical "and" devices might be found by considering the circuits themselves rather than the coding system. In any case, the search for such a system is narrowed considerably.

References

1. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Tech. J.* **29,** 147-160 (1950).
2. D. Slepian, "A Class of Binary Signaling Alphabets," *Bell System Tech. J.* **35,** 203-234 (1956).
3. P. Elias, "Computation in the Presence of Noise," *IBM Journal,* **2,** 346 (October, 1958).
4. A. B. Fontaine and W. W. Peterson, "On Coding for the Binary Symmetric Channel," *Trans. AIEE,* 648-656 (November, 1958).
5. R. R. Kuebler and R. C. Bose, "On the construction of a Class of Error Correcting Binary Signaling Codes." *Inst. of Statistics Mimeograph Series No. 199,* University of North Carolina.
6. W. W. Peterson, "On Checking an Adder," *IBM Journal,* **2,** 166-168 (April, 1958).

Received August 22, 1958