STATISTICAL DESIGN VERIFICATION

Sharad C. Seth
Department of Computer Science
University of Nebraska, Lincoln, Nebraska 68588

Vishwani D. Agrawal
Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Design verification is a process that verifies a design against a given behavioral description. An often used procedure is to verify that the design produces correct result for certain test inputs. It is, however, rarely practicable to cover the input space exhaustively. Therefore, a basic question that should be answered is that if no design fault has been detected after certain test cases have been applied, then how close is the design to a correct design.

In this paper the design to be verified is viewed as similar to a communication system in which design faults are assumed to be the only causes of error. We also assume that the decision about correctness of response can be made on the basis of the behavioral description or with the help of a simulator. The closeness of a design to the correct design is estimated in terms of how closely its information transmission rate matches that of the correct design. The confidence in the estimate of the information transmitting capability increases fastest when the statistical information produced by the design during test is highest. Hardware and software examples are given.

1. INTRODUCTION

Design verification is the process that verifies a design against a given behavioral specification. If the design could be automatically generated from its specification there would be no need to verify it (as long as the process automating design generation is itself verified!) However, until such time as fully automated design of complex systems becomes a possibility, design verification in one form or another will continue to be essential for exposing design errors or gaining more confidence in correctness.

Formal methods of design verification have been proposed for both software and hardware designs (see Elspas et al [1] and Cory & vanCleemput [2] for good tutorial articles.) Most notable amongst correctness proof techniques for programs is the *inductive assertion* method based on the work of Floyd [3] and Hoare [4]; Gries [5] provides a good illustrative example. The technique has also been extended to hardware verification by using the notion of *symbolic execution*, Darringer [6].

Formal methods are at one end of the spectrum, representing potentially powerful tools with theoretical underpinnings. However, after almost two decades of continual refinements they are still not in wide use; DeMillo et al [7] argue that they may *never* become very popular. At the other end of the spectrum is the commonly used method of testing a design by verifying that it produces correct results for certain test inputs. Several guidelines for test selection have been proposed (Goodenough & Gerhardt [8], Howden [9], and DeMillo et al [10].)

Unfortunately, neither testing nor correctness proofs guarantee correctness of the design ([7], Gerhart & Yelowitz [11].) Noting this Duran & Wiorkowski [12] have proposed a statistical measure of the degree of validity of a program. The measure is derived from tests which *fail* to uncover an error. A similar measure, the expected number of remaining software errors, can be obtained by fitting test results to a statistical model for software error detection proposed by Goel [13]. This model assumes that the number of errors detected in the testing interval $[0,t]$ is a random variable distributed according to a nonhomogeneous Poisson process.

The present paper shares the underlying premise of [12] and [13] namely that guaranteeing total correctness of complex systems is an elusive goal. However, by design verification tests it may be possible to achieve an arbitrarily high level of confidence. The design to be verified is viewed as similar to a communication system in which design faults are assumed to be the only causes of errors. Simulation results on two examples of software and hardware, respectively, show that the rate of increase in the confidence level is greatest when the statistical information produced by the design during test is highest.

0731-3071/82/0000/0393$00.75

2. NEED FOR STATISTICAL FORMULATION

Consider the simple example of a two-input OR gate. In order to guarantee that it has not been replaced by any other gate, among NOR, AND or NAND, suppose one applies three test inputs 00, 01 and 10. If the outputs of our design of the OR gate are 0,1 and 1, respectively, we feel confident that the function implemented is not a NOR, AND or NAND. But we still cannot guarantee that the designed gate is not an EXCLUSIVE-OR gate. Thus we should apply one more test, 11, to make sure that our design is not an EXCLUSIVE-OR. The tests are exhaustive now. More complex designs may not be completely verified even with the exhaustive set of patterns if the operation is pattern sensitive.

This simple example illustrates that the tests that are generated for certain fault models may leave many other faults undetected. For large circuits, exhaustive testing is impossible. Enumeration of all possible fault models and generation of corresponding tests is also difficult. For such cases statistical methods provide a possible solution.

In statistical testing the presence of a fault corresponds to a non-zero probability of error in the result. Thus the fault plays a role similar to that of noise in a communication channel. We can, therefore, analyze the verification process using the concepts of information theory. This viewpoint has been used for studying statistical test generation for digital circuits [14].

3. INFORMATION THEORETIC ANALYSIS

In this section we present an information theoretic analysis for determining the probability of error from a faulty *processor* and the confidence in correctness when no errors are observed. The term "processor" is used for any information processing hardware or software.

Communication and Computing. A communication channel transmits information while a computing device processes information. By processing we mean that the supplied information is manipulated and certain relevant information is extracted. In general, the processed information is less than the input information. Throughout this paper the term *information* refers to the statistical information as defined by Shannon [15].

Figure 1(a) shows a communication channel which, at the input, is supplied with information at the rate of H_i bits per unit time (for any specified time units). At the output H_o bits correspond to that portion of the input information which can be correctly inferred from the output of the channel. Because of the noise, it is not possible to receive everything as sent and therefore, H_o is, in general, less than H_i. Channel capacity is defined as [15],

$$C = \max_{H_i} H_o. \quad (1)$$

Next consider an information processor which can be a digital circuit or a software program. Figure 1(b) shows an information processor without any fault. Its operation

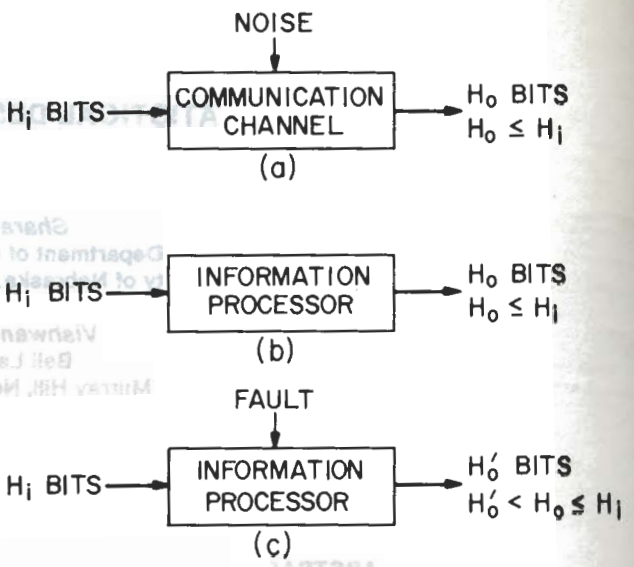

Fig. 1 Communication channel and information processor.

is assumed to be free from noise as well and the output is always correct. Yet, as explained above, the output H_o may be less than the input H_i due to the loss of information in processing. Figure 1(c) shows the same information processor with a detectable fault. Now the information output H'_o is further reduced because some of the outputs will be incorrect due to the fault. For a given input the correct design will produce an unique output. In the case of a faulty design, only the correct outputs are supposed to be carrying any information. It is, therefore, assumed that we have some way to check the outputs during verification.

In general, the output information of an information processor depends upon: 1) processor characteristics and 2) input source (or input pattern generator). However, the output information capacity of a processor can be defined, in the same way as for a communication channel, by Equation (1). This capacity depends only on the processor characteristics.

In the information theoretic framework we need not assume absence of memory in the processor. Memory in the processor will produce outputs that are correlated. As the separation between two outputs increases, their correlation reduces. Such correlation also exists in most information sources which, for example, produce clusters of symbols forming words and sentences rather than random letters.

Input and Output Spaces. For a simple communication channel the input and output spaces may consist of the letters of the alphabet in some encoded form. An information source is defined by assigning some statistical properties to the letters like their probabilities of occurrence, correlations, etc. Now consider a digital circuit. Its input alphabet will consist of all the binary patterns that can be applied to the input of the circuit. For example, an N input combinational circuit will have 2^N patterns in its input alphabet. An input source or a pattern generator is then constructed

by assigning probabilities to each pattern. The information generated by this source is [15]

$$H_i = -\sum_{i=1}^{2^N} p_i \log_2 p_i \text{ bits per pattern}, \quad (2)$$

where p_i is the probability of selecting the *ith* pattern. For a sequential circuit, the input space consists of 2^{N+S} pattern sequences where the circuit has N primary inputs and S memory elements. This is because each of the 2^N patterns can be applied after setting the circuit to any one of the 2^S states. An input sequence consists of a pattern (one among 2^N) preceded by a pattern sequence that sets the circuit to a particular state (one among 2^S). Often, however, a pattern in a sequence is applied to the machine whose state is the same as that left by the previous pattern. In such sequences, the dependence on the starting state becomes less and less the longer is the sequence length.

For a circuit with M output lines, the output space contains 2^M patterns. For any input pattern generator if the output patterns are produced with probabilities q_1, q_2,...,then the output information is given by

$$H_o = -\sum_{i=1}^{2^M} q_i \log_2 q_i \text{ bits per pattern}. \quad (3)$$

Notice that M can be greater than N, i.e., the output space can be larger than the input space. Yet the output information H_o cannot exceed H_i. As an illustration consider a 3-bit binary decoder with three input and eight output lines. Even though the output space contains $2^8 = 256$ patterns, only eight of these have non-zero probabilities of occurring. Thus H_o never exceeds 3 bits per pattern. In this case $H_o = H_i$.

The information output H_o will be equal to the output capacity C as given by Eq. (1) when each output pattern that can be produced occurs with equal probability. Thus the 3-bit decoder will produce $H_o = C = 3$ bits per pattern when each of the eight possible patterns are equally likely to occur. If a circuit can produce $k \leq 2^M$ output patterns, then C can also be calculated as

$$C = \log_2 k. \quad (4)$$

Information Loss Due to Fault. As pointed out earlier, in our statistical framework, we define the presence of a fault by a nonzero probability of error in the output. An actual fault may be a wrong connection, a wrong type of gate, sensitivity of circuit to noise, a wrong statement in software or any other artifact which can produce an incorrect result. In the case of an output error, the information loss is equal to the information that would be required to correct this error. We will assume that of each bit of information that the output produces, a fraction α is lost due to a fault.

Error Probability. We will assume that the output rate of the fault-free circuit is R bits per pattern for a given input source, where R can at most be equal to C. This rate will be reduced to $(1-\alpha)R$ under a fault. The fault-free circuit can produce 2^{RT} sequences of length T, each with almost equal probability [15]. This is a well-known result in information theory, also referred to as McMillan's theorem, and its accuracy improves as the length T of the sequences becomes large. In the presence of the fault the corresponding number of sequences will be reduced to $2^{(1-\alpha)RT}$. Since only correct outputs are assumed to carry information, we have

$$\text{Prob. \{test passed} \mid \alpha \} = \frac{2^{(1-\alpha)RT}}{2^{RT}} = 2^{-\alpha RT} \quad (5)$$

Notice that in the presence of a fault (i.e., $\alpha > 0$) the probability of correct output (or, equivalently, the probability of missing the fault) goes to zero as T is increased. Also, this probability will reduce faster, the larger the rate R of transmission.

The reduction α in each bit of output information may be different for different faults. Also, its value may be dependent upon the probabilities assigned to various input patterns for test generation. We will, therefore, treat α as a random variable with the probability density $p(\alpha)$. Then from (5), the probability of the circuit passing the test is obtained as

$$\text{Prob. \{test passed\}} = \int_0^1 2^{-\alpha RT} p(\alpha)\, d\alpha$$

Consider the specific case where, before this test was run, α is assumed to be uniformly distributed in the interval $[0,\alpha_{max}]$. Then the above equation reduces to

$$P(T,\alpha_{max}) = \text{Prob. \{test passed} \mid \alpha \text{ uniform in } [0, \alpha_{max}]\}$$

$$= \frac{1}{\alpha_{max}} \int_0^{\alpha_{max}} 2^{-\alpha RT}\, d\alpha$$

$$= \frac{1-2^{-\alpha_{max} RT}}{\alpha_{max} RT \ln 2}. \quad (6)$$

Also,

$$\text{Prob. \{test passed\}} = P(T,1) = \frac{1-2^{-RT}}{RT \ln 2}.$$

In the absence of any *a priori* knowledge about the probability density $p(\alpha)$ it might be reasonable to assume that α is uniformly distributed in the interval $[0,1]$. With this assumption we proceed to determine the *a posteriori* distribution of α given that the circuit passed a test of length T. Using Bayes' rule [16], we have

$$p(\alpha \mid \text{test passed})\, d\alpha$$

$$= \frac{\text{Prob. \{test passed} \mid \alpha\}\, p(\alpha)\, d\alpha}{\text{Prob. \{test passed\}}}$$

Using (5) and the previous result, we get

$$p(\alpha \mid \text{test passed}) = \frac{2^{-\alpha RT}\, RT \ln 2}{1-2^{-RT}}.$$

This conditional density function decreases exponentially with α where the rate of exponential decrease is determined by RT.

The intuitive notion of building confidence in the design by testing can now be stated more precisely in terms of the conditional distribution of α. In particular,

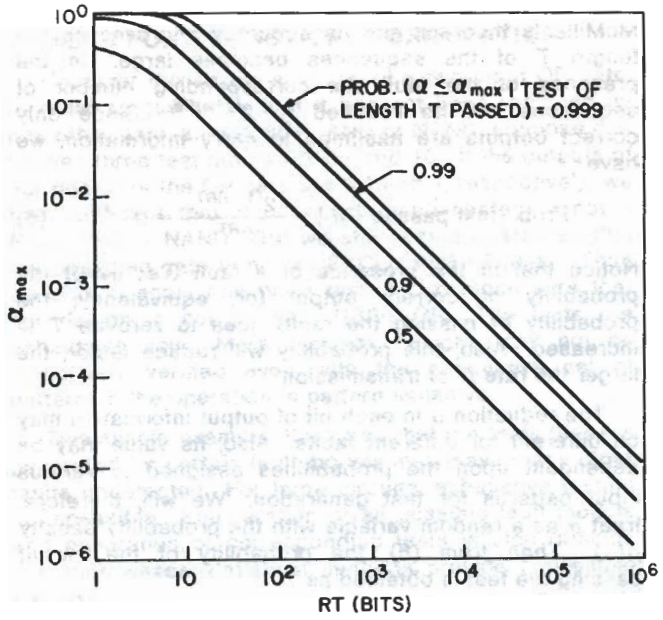

Fig. 2 Maximum untested fraction (α_{max}) of information as a function of test length.

$$Q(\alpha_{max}) = \text{Prob.}\{\alpha \le \alpha_{max} \mid \text{test passed}\}$$

$$= \int_0^{\alpha_{max}} p(\alpha \mid \text{test passed})\, d\alpha$$

$$= \frac{RT \ln 2}{1-2^{-RT}} \int_0^{\alpha_{max}} 2^{-\alpha RT}\, d\alpha$$

$$= \frac{1-2^{-\alpha_{max} RT}}{1-2^{-RT}} \qquad (7)$$

Figure 2 shows α_{max} as a function of RT for $Q(\alpha_{max}) =$ 0.999, 0.99, 0.9, and 0.5. For a test of length T, the information rate R can be empirically determined from the formula:

$$R = \lim_{T\to\infty} \sum_{i=1}^{k} \frac{x_i}{T} \log_2 \frac{T}{x_i} \text{ bits per test},$$

where x_i is the number of times the *ith* output occurs and k is the total number of possible outputs. In practice, a sufficiently large value of T will give adequate accuracy. Thus

$$RT = T \log_2 T - \sum_{i=1}^{k} x_i \log_2 x_i \text{ bits}. \qquad (8)$$

Once RT is computed for a test sequence, the value of α_{max} can be obtained form Fig. 2. For example, if $RT = 10^5$, with 99.9 percent probability α_{max} is no greater than 10^{-4}. When α_{max} is unity we can say that our design has no apparent relationship to the correct design.

4. EXAMPLE OF HARDWARE VERIFICATION

An FPLA (Field Programmable Logic Array) was considered as an example of a hardware design. This FPLA is described in a table in reference [17] (see Fig. 3). It has sixteen inputs from which 29 product terms are generated. The portion of the table that describes the product terms is a 29 by 16 array (AND matrix) whose element p_{ij} indicates how the *jth* input is used in the *ith* product term. In particular, p_{ij} can have any of the three values, H, L or $-$, referring to uncomplemented, complemented or not used, respectively. Similarly, the output is specified by another 29 by 8 array (OR matrix) whose element, $q_{ij} = A$ if *ith* product term is included in the *jth* output, or $q_{ij} = \cdot$ if *ith* product term is not used in the *jth* output. A closer examination of this table reveals that although the circuit has eight output lines, only ten output patterns are possible. Thus the maximum information output is $C = \log_2(10) = 3.32$ bits per pattern.

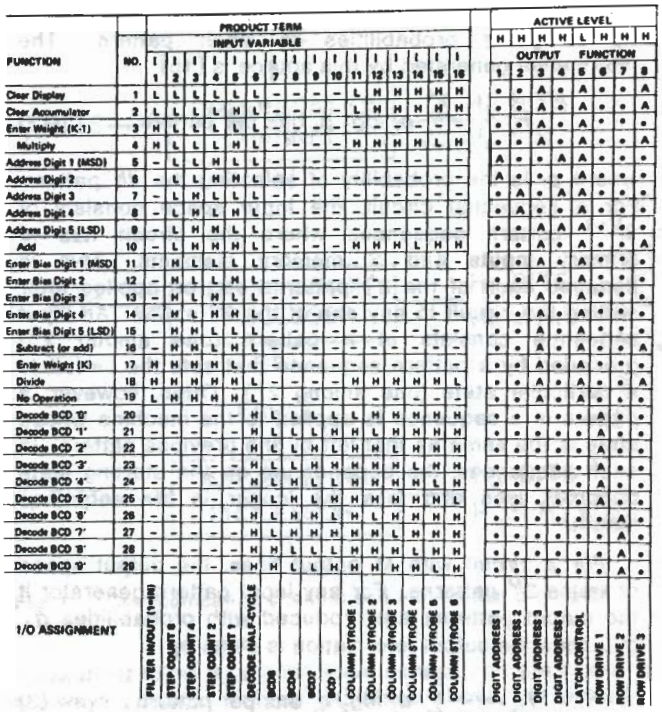

Fig. 3 Description of FPLA from page 90 of reference [17].

In order to evaluate the effectiveness of different types of tests, a Pascal program was written which could perform the following tasks:

a) *Pattern Generation.* Two types of input patterns could be generated by the program. The first type of patterns were completely random. The input bits were set to "0" or to "1" depending upon whether computer generated random numbers in the interval [0,1] were "less than" or "not less than" 0.5. The second type of patterns were those that maximized the output information to 3.32 bits per pattern. First all 2^{16} input patterns were simulated and classified into ten sets such that the patterns in a set produced the same output. Then for generating a test pattern an output was obtained using a random number generator which produced an integer uniformly distributed in [1,10]. Now

from the input set of this selected output, one pattern was randomly selected as a test pattern.

b) *Simulation.* Using the AND and OR matrices as described above, the program could compute the FPLA's output for any given input pattern. If any entries in the matrices were changed to study a faulty behavior, then the simulator would compute the output of the faulty FPLA.

c) *Fault injection.* The type of faults studied were assumed to produce an error in a single entry among the two matrices. The program would first pick an element in the matrices randomly (using a random number generator to determine the indices of the element). If the element belonged to the AND-matrix then its faulty value would be picked randomly from $[H, L, -]$ such that it is different from the correct value. Similarly, for the OR-matrix the faulty value would be picked from $[A, \cdot]$.

For completely random patterns, using Eq. (8), the information output was obtained as 1.89 bits/pattern. For each type of pattern, one hundred fault samples were generated and for every sample fault the pattern generation was continued until either the fault was detected or the number (T) of patterns corresponded to $RT = 10{,}000$. At this point if the fault remained undetected it was checked for redundancy by simulating all 2^{16} input patterns. If the fault turned out to be redundant a new fault sample was picked and pattern generation was repeated. The fraction of undetected nonredundant faults, as observed in this experiment, is shown in Fig. 4. Notice that for $RT = 10{,}000$ maximum information output patterns left 6 percent faults undetected while 10 percent were left by the completely random patterns. The number of patterns in the case of maximum information output was $T = 10{,}000/3.32 \simeq 3{,}000$. With the sames number of random patterns, since $RT = 3{,}000 \times 1.89 = 5{,}670$, there were 15 percent undetected faults. This result clearly shows the greater efficiency of the maximum information output patterns in uncovering errors. The fault model that we have chosen in this exercise corresponds to altering a single entry in the matrix specification of the design. While such faults are realistic, there can be many other types of faults associated with various stages of the design. The reason for selecting these faults was that they change the design very slightly and so their detection may be considered difficult. The theoretical probability of an undetected fault as computed from (6), is also plotted in Fig. 4 (solid curves) for various values of α_{max}. Although the values of α for these faults appears to be in the range [0, 0.1], their distribution may not be uniform as assumed while deriving (6). For example, there may be a larger number of faults with smaller value of α than those with larger α. The distribution of α in our experiment is dependent upon the fault model and may not be a true representation of a general design verification process.

Figure 2 shows that for $RT = 10{,}000$, we can consider that $\alpha \leq 0.001$. That is, only the faults with a value of α less than 0.001 might be left undetected.

5. EXAMPLE OF SOFTWARE VERIFICATION

Consider the FORTRAN program [18] that

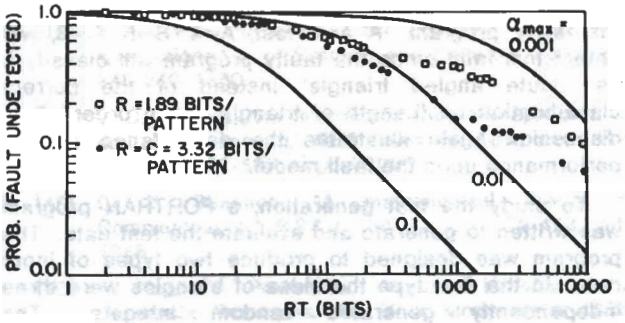

Fig. 4 Fraction of undetected sample faults in a 16-input FPLA for maximum information output rate ($R=C=3.32$ bits/pattern) and for equiprobable random inputs ($R=1.89$ bits/pattern).

characterizes triangles for given integer-valued lengths (A, B, and C) of sides:

```
      INTEGER A,B,C,D
      READ 10,A,B,C
   10 FORMAT (4I10)
    5 IF ((A.GE.B).AND.(B.GE.C)) GO TO 100
      PRINT 50
   50 FORMAT(1H ,'LENGTH OF TRIANGLE NOT IN
     1 ORDER')
      STOP
  100 IF ((A.EQ.B).OR.(B.EQ.C)) GO TO 500
      A=A*A
      B=B*B
      C=C**2
      D=B+C
      IF (A.NE.D) GO TO 200
      PRINT 150
  150 FORMAT(1H ,'RIGHT ANGLED TRIANGLE')
      STOP
  200 IF (A.LT.D) GO TO 300
      PRINT 250
  250 FORMAT(1H ,'OBTUSE ANGLED TRIANGLE')
      STOP
  300 PRINT 350
  350 FORMAT(1H ,'ACUTE ANGLED TRIANGLE')
      STOP
  500 IF ((A.EQ.B).AND.(A.EQ.C)) GO TO 600
      PRINT 550
  550 FORMAT(1H ,'ISOCELES TRIANGLE')
      STOP
  600 PRINT 650
  650 FORMAT(1H ,'EQUILATERAL TRIANGLE')
      STOP
      END
```

In reference [18] six tests are given which were derived from path analysis. It is shown in [10] that these six tests may not be adequate for detecting errors in the compound logical expressions. To detect such errors, three more tests are added and the set of nine inputs is presented as a stronger test for the program [10]. It is, however, easy to see that even with these nine tests there are faults which would not be detected. For example, if we replace B in statement 5 by C as follows:

```
    5 IF((A.GE.C).AND.(B.GE.C)) GO TO 100
```

Then all nine tests will produce the correct result from

the faulty program. A new test, A=4, B=5, C=3, will detect this fault since the faulty program will classify it as "acute angled triangle" instead of the correct classification as "Length of triangle not in order." This discussion again illustrates the dependence of test performance upon the fault model.

To study the test generation, a FORTRAN program was written to generate and evaluate the test data. The program was designed to produce two types of input data. In the first type the sides of triangles were three independently generated random integers. The information output for these inputs as computed from (8) was 0.84 bits/input. In the second type of test data, an output was chosen randomly from the six possible outputs. A set of 3 integers was then generated to produce this output. For example, if the output was selected to be "RIGHT ANGLED TRIANGLE", then B and C were chosen as two random integers such that $B \geq C$ and then A was computed from $A^2 = B^2 + C^2$. Since there are six possible outputs, the information output for these inputs is $C = \log_2(6) = 2.6$ bits/input.

A set of thirty faults of the types discussed in [10] and the one shown above was considered. All thirty faults were detectable. In a typical pass the program would randomly sample one fault and then continue to generate the specified type of input data patterns until the faulty program keeps on giving the correct result. Also if the fault does not get detected up to $RT = 10{,}000$ the pass is terminated. Results for 100 fault samples for each type of patterns are shown in Fig. 5. Theoretical curves (solid lines) showing the probability of missing a fault as computed from (6) for various values of α_{max} are also plotted. We notice that the experimental data agrees only partially with the theoretical curves. As pointed out in the previous section, this could be because the value of α for the faults might not be uniformly distributed as assumed in the derivation of (6). It is evident here, even more than the previous example, that the maximum information output tests are more efficient.

Even though the abscissa in Fig. 5 is the number of inputs (T) normalized by multiplying the information output rate, the experimental data for the two types of test data differ very significantly. One possible reason for this could be the dependence of α for a fault on the way the inputs are selected. In this case, however, the increase in information output seems to increase the effective value of α making the faults more easily detectable. No fault remained undetected for more than 40 tests. For random inputs, on the other hand (R=0.84 bits/inputs) 4 percent faults were still undetected after about 10,000 test cases had been tried.

6. CONCLUSION

Using the concepts of information theory, this paper provides a non-heuristic basis for design verification. The design faults are characterized by the information loss they produce in the response of the design. Similarly the correct design is characterized by the amount of statistical information in its response. As more tests are conducted, we increase our confidence in the correctness of the design by estimating the fraction

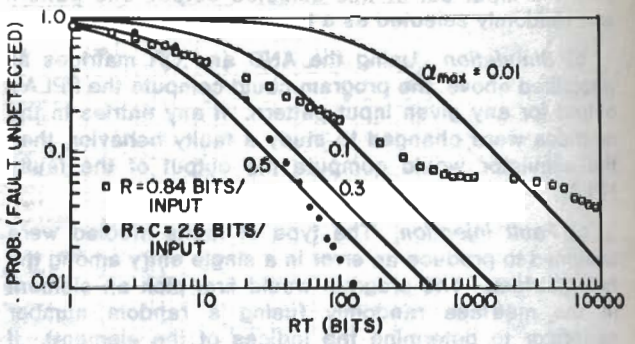

Fig. 5 Fraction of undetected sample faults in a FORTRAN program for maximum information output rate (R=C=2.6 bits/input) and for a lower output rate (R=0.84 bits/input).

of information that could still be in error. Although a complete verification is approached asymptotically, the verification process can be accelerated by selecting the tests that maximize the information content in the response of the design under test.

If α_{max} could be estimated independently, the curves in Fig. 2 can be used to provide an estimate for RT (and hence for the test length T) for a given level of confidence. Notice, in this connection, that Eq. (5) has the form of the reliability function $e^{-\lambda t}$ where the test length T is analogous to the time parameter t [19]. The failure rate λ is then equal to $\alpha R \ln 2$. Thus, α_{max} may be estimated from the system constraint on maximum permissible failure rate.

ACKNOWLEDGMENT - The authors thank G. Nagy for his comments and express their appreciation for the help of M. T. Arvay and A. Reno in the preparation of this manuscript.

REFERENCES

[1] B. Elspas, K. N. Levitt, R. J. Waldinger, and A. Waksman, "An assessment of techniques for proving program correctness," *ACM Computing Surveys*, Vol. 4, No. 2, pp. 97-147, June 1972.

[2] W. E. Cory and W. M. vanCleemput, "Developments in verification of design correctness," *Proceedings of 17th Design Automation conference*, Minneapolis, MN., June 23-25, 1980, pp. 156-164.

[3] R. W. Floyd, "Assigning meaning to programs," *Proc. Amer. Math. Soc. Symp. Applied Mathematics*, Vol 19, 1967, pp. 19-31.

[4] C. A. R. Hoare, "An axiomatic basis for computer programming," *Comm. ACM*, Vol. 12, pp. 576-583, October 1969.

[5] D. Gries, "An illustration of current ideas on the derivation of correctness proofs and correct programs," *IEEE Trans. on Software Engineering*, Vol. SE-2, pp. 238-244, December 1976.

[6] J. A. Derringer, "The application of program verification techniques to hardware verification," *Proceedings of 16th Design Automation Conference*, San Diego, CA, June 25-27, 1979, pp. 375-381.

[7] R. A. DeMillo, R. J. Lipton, and A. J. Perlis, "Social processes and proofs of programs and theorems," *Comm. ACM*, Vol. 22, pp. 271-280, May 1979.

[8] J. Goodenough and S. L. Gerhart, "Toward a theory of test data selection," *IEEE Trans. on Software Engineering*, Vol. SE-1, pp. 156-173, 1975.

[9] W. E. Howden, "Functional testing and design abstraction," *J. of Systems and Software*, Vol. 1, pp. 307-313, 1980.

[10] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Computer*, Vol. 11, pp. 34-41, April 1978.

[11] S. L. Gerhart and L. Yelowitz, "Observations of fallibility in applications of modern programming methodologies," *IEEE Trans. on Software Engineering*, Vol SE-2, pp. 195-207, September 1976.

[12] J. W. Duran and J. J. Wiorkowski, "Quantifying software validity by sampling," *IEEE Trans. on Reliability*, Vol. R-29, pp. 141-144, June 1980.

[13] A. L. Goel, "Software Error detection model with applications," *J. of Systems and Software*, Vol. 1, pp. 243-249, 1980.

[14] V. D. Agrawal, "An information theoretic approach to digital fault testing," *IEEE Trans. on Computers*, Vol. C-30, pp. 582-587, August 1981.

[15] C. E. Shannon, "A mathematical theory of Communication," *B.S.T.J.*, Vol. 27, pp. 379-423, July 1948.

[16] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, 1965, p. 112.

[17] J. B. Peatman, *Digital Hardware Design*, New York: McGraw-Hill, 1980, Section 2-12.

[18] C. V. Ramamoorthy, S. F. Ho, and W. T. Chen, "On the automated generation of program test data," *IEEE Transactions on Software Engineering*, Vol. SE-2, pp. 293-300, December 1976.

[19] G. W. A. Dummer and N. B. Griffin, *Electronics Reliability — Calculation and Design*, New York: Pergamon, 1966.