STASTICAL DESIGN VERIFICATION

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

Vishwanath 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 to what extent the design is correct. This is a question that has been posed many times in the past years.

In this paper the design to be verified is viewed as a model of a communication system in which design faults are assumed to be the only causes of error. We also assume that the developer of the design has been able to discover all the design faults that can be 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 a model of a communication system in which design faults are assumed to be the only causes of error. We also assume that the developer of the design has been able to discover all the design faults that can be detected after certain test cases have been applied, then how close is the design to a correct design.

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 automatically generating the design is itself error-free). 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 design (see Elesap et al [4] and Cory & van Geijn [5] 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 Floyd [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] Gerhardt & Yelovitz [11] note this Duran & Wibrowski [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 simulating test runs to a statistical model for software error detection proposed by Good [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.
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, and 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 [10].

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 [16].

Figure 1(a) shows a communication channel which as at the input is supplied with information at the rate of bits per unit time (for any specified unit time). At the output H0 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 H1 is, in general, less than H0. Channel capacity is defined as [16].

\[ C = \max_{P(x)} H_0 \]

Next consider an information processor which can be a digital circuit or a software program. Figure 1(b) shows an information processor without any faults. Its operation is assumed to be free from noise as well and the output is always correct. Yet, as explained above, the output H0 may be less than the input H0 due to the loss of information in processing. Figure 1(c) shows the same information processor with a detectable fault. Now the information output H0 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 output H0. 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 single 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 binary patterns that can be applied to the input of the circuit. For example, an N input combination circuit will have 2N patterns in its input alphabet. An input source or a pattern generator is then constructed

394
by assigning probabilities to each pattern. The information generated by this source is (16)
\[ I = \sum_{i=1}^{N} p_i \log_2 \frac{1}{p_i} \text{ bits per pattern}, \]

where \( p_i \) is the probability of selecting the \( i \)-th pattern.
For a sequential circuit, the input space consists of \( 2^{2^N} \) pattern sequences where the circuit has \( N \) primary inputs and \( M \) memory elements. This is because each of the \( 2^N \) patterns can be applied after setting the circuit to any one of the \( 2^M \) 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 \( 2^M \)). 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 circuit 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 \( q_j \), then the output information is given by
\[ I = \sum_{j=1}^{2^M} q_j \log_2 \frac{1}{q_j} \text{ bits per pattern}. \]

Notice that \( M \) can be greater than \( N \), i.e., the output space can be larger than the input space. Yet the output information \( I \) cannot exceed \( I \). As an illustration consider a 3-bit binary decoder with three input and eight output lines. Even though the output space contains \( 2^3 = 2^8 \) patterns, only eight of these have non-zero probabilities of occurring. Thus \( I \) will not exceed 3 bits per pattern. In this case \( I = I \).

The output information \( I \) 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 \( I = C = 3 \) bits per pattern when each of the eight possible patterns are equally likely to occur. If a circuit can produce \( K \) as \( 2^M \) output patterns, then \( C \) can also be calculated as
\[ C = \log_2 K. \]

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 memory 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 \( e \) 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, whereas \( R \) can at most be equal to \( C \). This rate will be reduced to \( (1-e) R \) under a fault. The fault-free circuit can produce \( 2^N \) sequences of length \( T \), each with almost equal probability \( 1/B \). 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 \( 2^{N-eT} \). Since only correct outputs are assumed to carry information, we have

\[ \text{Prob. (test passed)} = \frac{2^{N-eT}}{2^N} = 2^{-eT}. \]

Notice that in the presence of a fault (i.e., \( e>0 \)) 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 reductions 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 \( e \) as a random variable with the probability density \( p(e) \). Then from (6), the probability of the circuit passing the test is obtained as

\[ \text{Prob. (test passed)} = \int_0^\infty 2^{-eT} p(e) \, de. \]

Consider the specific case where, before this test was run, \( e \) is assumed to be uniformly distributed in the interval \([0, \max_e]\). Then the above equation reduces to

\[ \text{Prob. (test passed)} = \int_0^{\max_e} 2^{-eT} \, de = \frac{1}{\max_e}, \]

which is the probability density \( p(e) \) uniformly distributed in the interval \([0, \max_e]\).

Also, \( \text{Prob. (test passed)} = \frac{1}{R} \int_0^{\max_e} 2^{-eT} \, de = \frac{1}{R} \max_e \]

is the absence of any prior knowledge about the probability density \( p(e) \) it might be reasonable to assume that \( e \) is uniformly distributed in the interval \([0, 1]\). Using this assumption we proceed to determine the a posteriori distribution of \( e \) given that the circuit passed a test of length \( T \). Using Bayes' rule [18], we have

\[ p(e \mid \text{test passed}) = \frac{p(e) \cdot \text{Prob. (test passed)}}{\text{Prob. (test passed)}}. \]

Using (3) and the previous result, we get

\[ p(e \mid \text{test passed}) = 2^{N-eT} R T \int_0^\infty 2^{-eT} \, de = 2^{N-eT} R T. \]

This conditional density function decreases exponentially with \( e \) where the rate of exponential decrease is determined by \( R T \).

The intuitive notion of building confidence in the design by testing can now be stated more precisely in terms of the conditional distribution of \( e \). In particular,
Fig. 2 Maximum intended test size for maximum error in test length.

\[
Q(x) = \text{Prob.} \{e \leq x \text{ | test passed}\} = \int_0^x \left(1 - \frac{1}{2} \cdot \frac{1}{\sqrt{T}} \cdot \frac{2^{\alpha T}}{\alpha T} \right) \cdot \left(\begin{array}{c}
\alpha T + 1
\end{array}\right)^{\alpha T} \text{d}T
\]

where \( \alpha T + 1 \) 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

\[
R = T \log_2 T - \sum_{i=1}^{k} x_i \log_2 x_i \text{ bits per test,}
\]

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

\[
R = T \log_2 T - \sum_{i=1}^{k} x_i \log_2 x_i \text{ bits.}
\]

Once \( R \) is computed for a test sequence, the value of \( \alpha \) can be obtained from Fig. 2. For example, if \( R = 10^4 \), with 99.9 percent probability \( \alpha \) is no greater than \( 10^{-4} \). When \( \alpha \) 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, 0, 1, or \(-\cdot\), referring to unimplemented, complemented, or not used, respectively. Similarly, the output is specified by another 29 by 6 array (OR matrix) whose element, \( q_{j} \), indicates whether the jth product term is included in the jth output, or \( q_{j} = 0 \) if jth 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 \( O \geq \log_2(10) \times 3.22 = 3.22 \text{ bits per pattern}. \)

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

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 "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 3.22 bits per pattern. For 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.

3) Simulation. Using the AND and OR matrices as described above, the program could compute the FPA'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 FPA.

c) Fault Injection. The type of faults studied were assumed to produce an error as a single entry among the two matrices that popped 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, \ldots} such that it is different from the correct value. Similarly, for the OR-matrix the faulty value would be picked from {1, 0}.

For completely random patterns, using Eq. (8), the information output was obtained as 1.59 bits/pattern. For each type of pattern, one hundred fault samples were generated and for every single fault the pattern generation was continued until either the fault was detected or the number (T) of patterns corresponded to \(N = 10,000\). At this point, if the fault remained undetected it was checked for redundancy by simulating all \(2^{10}\) 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 non-redundant faults, as observed in this experiment, is shown in Fig. 4. Notice that for \(N = 10,000\) 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 \(T = 10,000 / 0.52 = 3,000\). With the same number of random patterns, since \(N = 3,000\), 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 exercises corresponds to altering a single entry in the matrix specification of the design. While such faults are realistic, there are many other types of faults associated with various stages of the design. The reason for selecting these faults is 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 \(n_{fp} \). Although the values of \(n\) 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 values of \(n\) than those with larger \(n\). The distribution of \(n\) 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 \(N = 10,000\), we can conclude that \(n < 0.001\). There are only faults with a value of \(n\) less than 0.001 might be left undetected.

5. EXAMPLE OF SOFTWARE VERIFICATION

Consider the FORTRAN program [16] that characterizes triangles for given integer-valued lengths (A, B, and C) of sides:

```
INTEGER A,B,C
READ 10,A,B,C
10 FORMAT (I3)
5 IF (A.EQ.B.OR.B.EQ.C) GOTO 100
PRINT 50
50 FORMAT (1H,'LENGTH OF TRIANGLE NOT IN 1 ORDER')
STOP
100 IF (A.EQ.B.OR.B.EQ.C) GOTO 500
A=A+1
B=B-1
C=C+1
D=D+1
IF (A.NE.C) GOTO 200
PRINT 150
150 FORMAT (1H,'RIGHT ANGLED TRIANGLE')
STOP
200 IF (A.EQ.B) GOTO 500
PRINT 550
550 FORMAT (1H,'OBTUSE ANGLED TRIANGLE')
STOP
300 PRINT 350
350 FORMAT (1H,'ACUTE ANGLED TRIANGLE')
STOP
500 IF (A.GE.B.OR.A.GE.C) GOTO 600
PRINT 530
530 FORMAT (1H,'ISOSCELES TRIANGLE')
STOP
600 PRINT 650
650 FORMAT (1H,'Equilateral Triangle')
STOP
END
```

In reference [16] six tests are given which were derived from path analysis. It is shown in [16] 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 [16]. 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.B.OR.B.GE.C) GOTO 100
```

These nine tests are quite different from the original program.
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 \( R_{\text{max}} \) can be estimated independently, the curve in Fig. 6 can be used to provide an estimate for \( R_T \) (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 \( e^{-x} \) where the test length \( T \) is analogous to the time parameter \( t \). The failure rate \( \lambda \) is then equal to \( e^{-x} \). Thus, \( R_{\text{max}} \) may be estimated from the system constraint on maximum permissible false rate.

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

REFERENCES


