NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

ISBN 0-315-56392-3

PROBABILISTIC FAULT LOCATION IN COMBINATIONAL LOGIC NETWORKS BY MULTISTAGE BINARY TREE CLASSIFIER - ALGORITHM DEVELOPMENT, IMPLEMENTATION RESULTS AND EFFICIENCY

by

George E. Fares

A Thesis Submitted to the

School of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Sciences

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering

Faculty of Engineering

University of Ottawa

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this thesis. I authorize the University of Ottawa to lend this thesis to other institutions or individuals for the purpose of scholarly research .

George E. Fares

I further authorize the University of Ottawa to reproduce this thesis by photocopying or by other means, in total or part, at the request of other institutions or individuals for the purpose of scholarly research .

George E. Fares

Contents

List of Figures

ACKNOWLEDGEMENTS

My deepest gratitude with appreciation to Dr.Sunil R. Das for giving me the benefit of his specialized knowledge and advice. I wish to thank him for being an understandable supervisor and a very good friend .

A special thanks to : The Electrical Engineering Department, Professors and graduate students of the University of Ottawa for providing the academic environment, the atmosphere, the discussion and help for excellent milieu to work on this thesis .

In my family I wish to thank: My father Elias for being a very hard working man during the war in my homeland Lebanon, and who worked really hard to help and encourage me, whenever possible to accomplish my ambition in life ; My mother Jamal for all the care and love towards my engineering profession; To my brother, my sisters, and their families, a special thank for all the patient support and help during my study.

At home, my wife Nada encouraged me and gave good suggestions while she spent a part of our Honeymoon typing the first draft of the thesis . I thank my wife for her patience during the time I was working on this thesis.

ABSTRACT

The basic philosophy of random test generation for digital circuits is extended in this thesis to fault location in combinational logic networks using the concept of multistage binary tree classifier. The suggested approach can solve the fault location problem of combinational networks for both single and multiple faults.

The proposed approach has two parts:

PART I is to construct a multistage binary tree classifier (preconstructed tree), each nonterminal node of which contains a feature set to control the input sequence, and a threshold level for the decision to be made to go along the subtrees. Each terminal node has the information to indicate the location of a specified fault.

PART II is the testing stage. It has the information of the preconstructed tree. There can be only two possible outcomes at each stage of decision making. The statistics of the circuit under test provides a specified threshold which helps to make the decision whether to go to the left or to the right along the subtree, the process being followed iteratively until finally a terminal node is arrived at where it represents the location of a specific fault.

However, the process of constructing the multistage binary tree classifier (phase 1) is rather complex in general, but once it is successfully completed, all the testing and location of faults (phase 2) can be performed very easily.

Chapter 1

INTRODUCTION

1.1 OVERVIEW

Today's more complex digital circuits demand stringent testing procedures . The testing of digital circuits requires two major points to be taken into consideration:

- Accuracy of testing (confidence measure).
- Testing time (cost).

Figure 1.1 illustrates graphically the relation between these objectives . Based on these two objectives , a number of testing methods have evolved over the years .

The first step in developing a testing strategy involves identifying the type of faults [15,16] . Two broad fault types can be identified for digital systems : one is the soft fault caused by software problems or by the signals used to operate the units. The other is the hard fault which comprises the physical failures caused by the interconnection failures between the components, or from the failures of the components themselves.

The most common fault model for digital circuits in use is the stuck fault model [3] of which combinational networks are the main target for our approach . Figure 1.3 represents a simple example of a combinational circuit with input line A stuck-at-zero (A-s-0) . The tests which detect this fault are defined by the logical expression $f_{A-s-0} = B$.

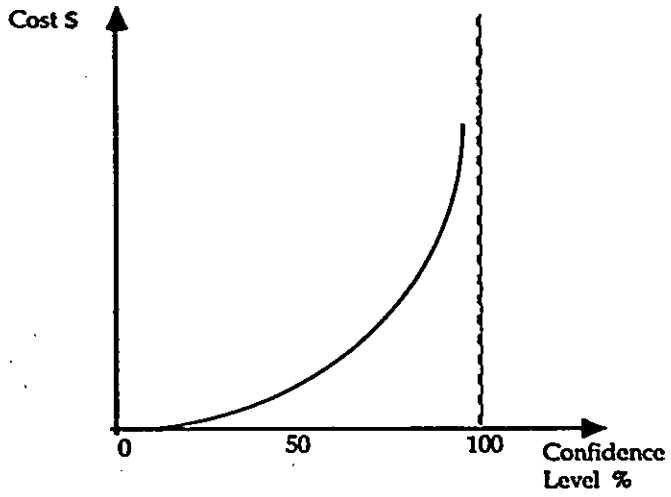


Figure 1.1: Relation between cost and confidence level .

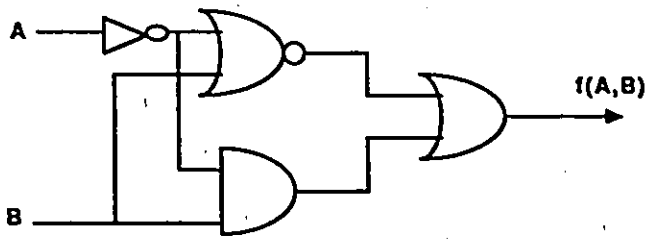


Figure 1.2: A simple example of a digital circuit .

Our goal is to solve the problem of fault testing and diagnosis which consist of solving both the fault detection problem and the fault location problem . Various approaches have been developed in the last two decades for the solution of the fault detection and location problems of combinational networks .

We can classify the different solution approaches to the fault testing problem in two major categories [16,22] : deterministic and probabilistic (Figure 1.2). There are several methods available for solving deterministically the test generation problem for combinational circuits, viz.

* Fault table methods (FTM).
(Yields a minimum set of diagnostic tests)
* Path-sensitizing and equivalent normal form methods (PSENF) .
(Works for the class of Tree-Like and some non-Tree-Like circuits)
* Karnaugh map and tabular methods (KM-TM).
(Using FTM with Karnaugh map concept)
* Equivalent-normal-form and Karnaugh map methods (ENFKM).
(A heuristic, systematic procedure derived from PSENF)
* Boolean difference methods (BD).
(Detecting single or multiple faults without any fault tables or maps)
*Spoof methods (SPOOF).
(Detecting single or multiple faults without any fault tables or maps)
And so on ... .

By using these methods, the amount of computations required to generate the test sets can be extremely large, particularly for more complex LSI or VLSI circuits.

However, probabilistic methods which come under the category of random testing

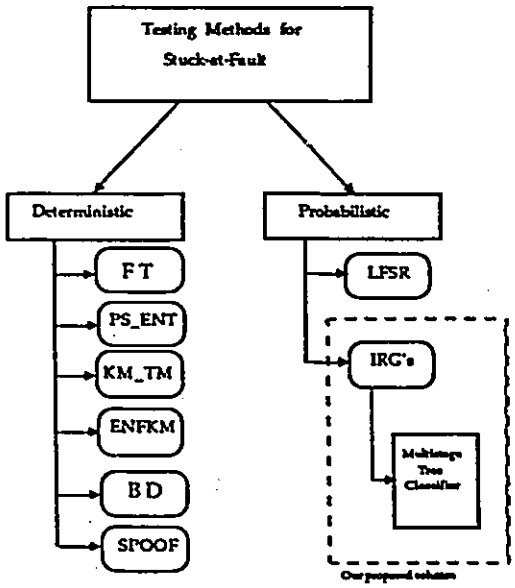


Figure 1.3: Different methods for testing stuck-at-faults .

[1,2,19,21] provides a solution to the difficulties arising out of the deterministic methods, and can reduce the cost of computation in general. Two major approaches for test pattern generation in probabilistic testing are : i) random test generation where the input patterns are generated by a linear feedback shift register (LFSR), and ii) to use independent random generators (IRGs) to create the input patterns.

In this thesis, a method based on multistage tree classifier has been used for random testing of digital combinational circuits using IRGs. The concept of comparing the output of the circuit under test (CUT) with the output of a known good unit (gold unit) is the basis of this method. Since both the circuits are fed by the same random test patterns, we can detect the faulty circuit by observing (comparing) their outputs Fi (Figure 1.4) .

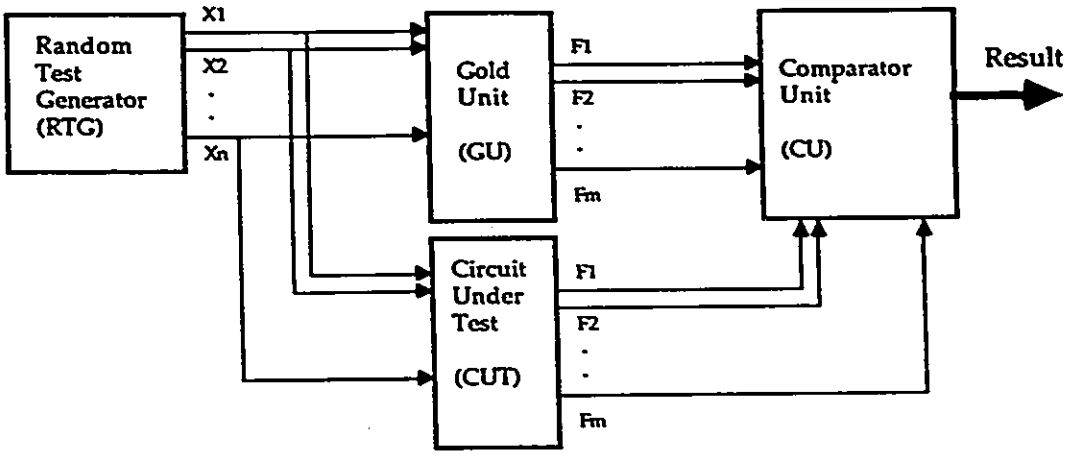


Figure 1.4: The block diagram of probabilistic method of test generation .

1.2 THE CONRTIBUTION

In this study we basically extend the concept of random testing for detecting a faulty logic circuit to locate the faults in faulty combinational networks by using a multistage binary tree. By selecting specific input probabilities (called feature set or feature vector, which in our case is the probability of the input being logical 1 in each of the input ports of the CUT) at each stage of the tree, the proposed method will have a good prospect for reducing the classification time in the following stages.

1.3 THE METHOD

The proposed method consists in classifying or grouping the faults; however, the proposed data structure for the solution of the problem is the binary decision tree. In the grouping approach using a binary tree, there are only two possible outcomes for each node : information, with a link to the left member or a link to the right member. From the statistics of the outputs (the mean of output distribution) of a circuit under test, we can decide either to go through the left member or the right member based on the region of a specified threshold (obtained by minimizing the error probability assigned to this node). This process is repeated until the test generation reaches a terminal node

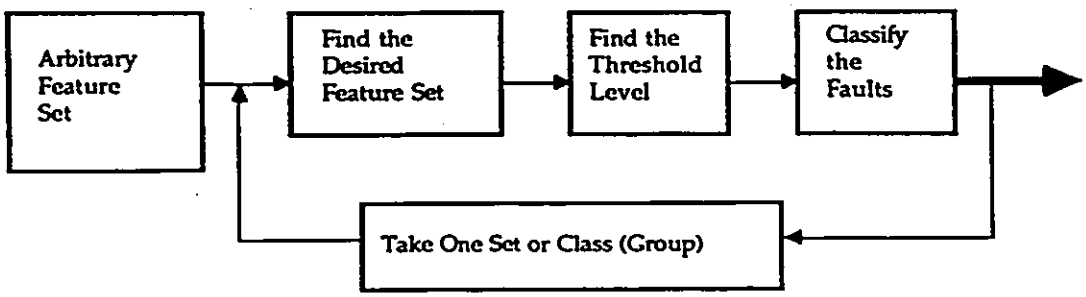


Figure 1.5: The block diagram for the tree construction phase .

where it represents a specified fault.

To obtain a solution of the fault location problem in combinational networks, two phases are identified :

1 . The construction phase .

2 . The testing phase .

To obtain an optimal input probabilities for good construction, the concept of fault distance is introduced [11,13,17] . The concept is based on the probability density function of the distribution of the outputs under consideration [7,8,9,]. If the fault distance calculated from the observable outputs cannot satisfy a prespecified value, we have to repeat the process with new selection of the input probabilities until we find a set which can satisfy the prespecified condition. From this acceptable selection, we can classify the faults under consideration, and find a suitable threshold for making decision in the testing stage.

Figure 1.5 gives the block diagram of the construction phase of the binary classifier tree. The testing phase can be graphically presented as in Figure 1.6 .

The material of this thesis is arranged as follows :

Chapter 1 provides a general overview of the fault detection and location problems in logic networks, and the solutions approaches for such problems .

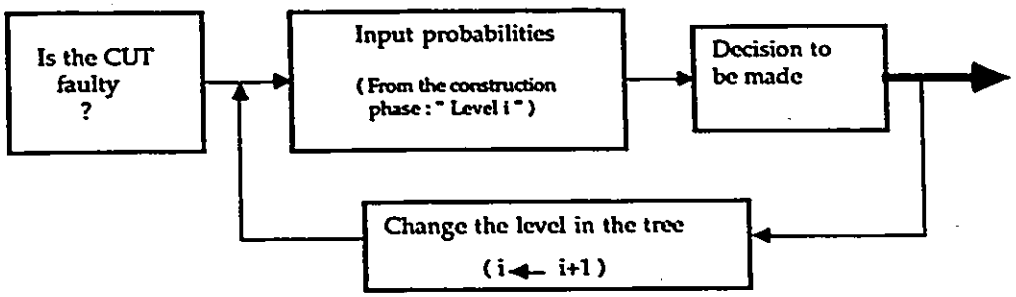


Figure 1.6: The block diagram for testing phase .

Chapter 2 presents the concept of multistage binary tree in probabilistic test generation .

Chapter 3 introduces a known concept of probability theory and discusses how we can use it in our approach . A method of stochastic approximation to find the feature set along with application are also presented. The chapter includes examples and the developed algorithms .

The binary tree and node information are covered in **Chapter 4**. The basic concept of binary decision, rules, grouping, and threshold calculation for decision making are explained in this chapter. Algorithms and some examples are presented as well .

Chapter 5 provides the structure (flowcharts and algorithms) for a complete construction of multistage binary tree classifier. Application and computational results are again included .

The testing phase (flowcharts and algorithms) is presented in **Chapter 6**. The preconstructed tree of Chapter 5 is used to provide the results of probabilistic fault location . Application and computational results are included .

Finally discussions and conclusions are presented in **Chapter 7** .

Chapter 2

MULTISTAGE DECISION TREE

2.1 INTRODUCTION

This chapter deals with the overall form of the solution, the emphasis being on methods and associated tools that are suitable for testing combinational networks using a decision tree classifier .

Here we find that the design description for the method of testing may be described in terms of three domains, namely :

1. The concept of multistage decision tree .
2. The information that each node contains .
3. The algorithm of testing .

We will summarize the concept of binary tree and focus on the structure of the testing level, since it is most important for our purpose .

2.2 CONCEPT OF BINARY TREE

A conventional description of a binary tree [24] is that there can be only two possible outcomes at each node, except for the terminal node which is the final result . A single element called the root represents the top node of the tree, and is said to be the father

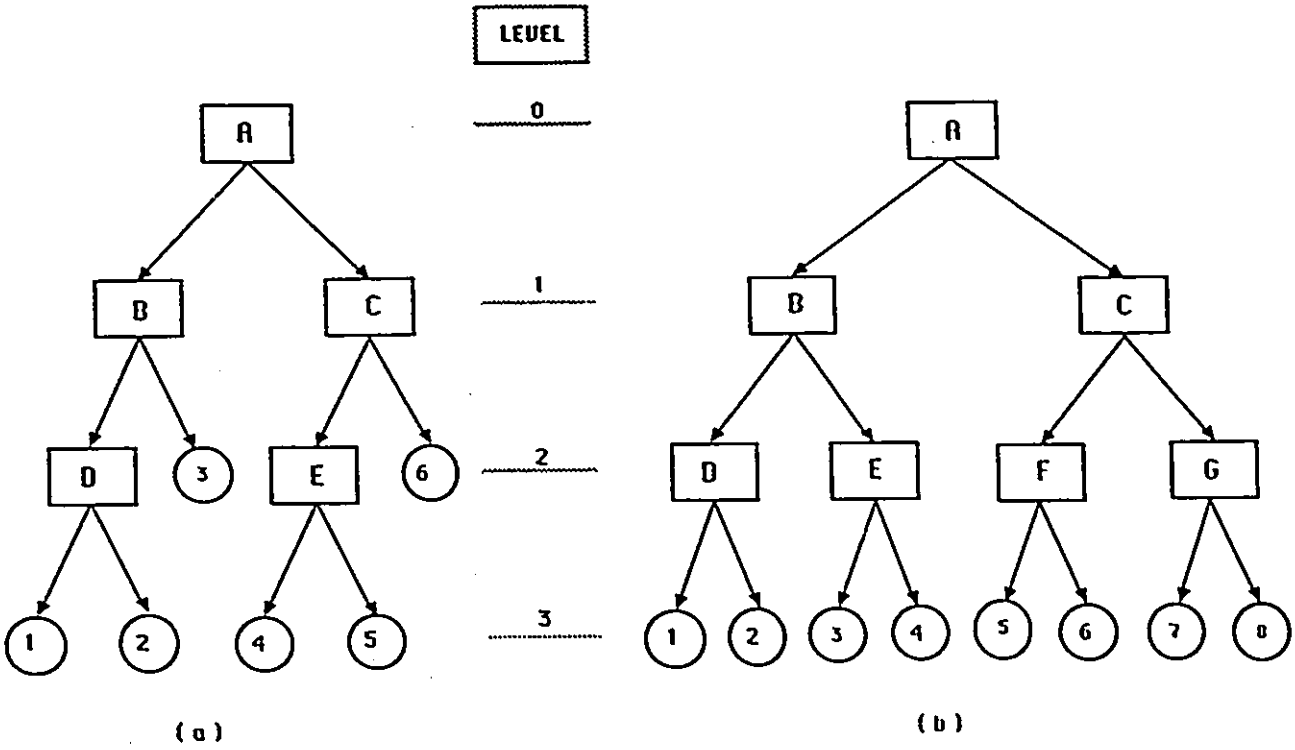


Figure 2.1: The structure of : (a) binary tree . (b) complete binary tree .

of two disjoint subsets . These two subsets are called the left and right subtrees, each of which is in itself the father of the left and right sons (or the root of a binary tree) . If a node that has no sons, it is called the terminal node .

One important thing in a binary tree is the level of a node ; the root node of the tree is in level 0 , then level 1 contains the sons . The level of each of following node is always one more than the level of the father . A complete binary tree of N levels is one in which all the nodes in level N have no sons, and all fathers in level N-1 have two sons; otherwise, it is called a binary tree .

□**EXAMPLE 2.1 :**

Suppose we have a binary tree of 3 levels. Figure 2.1 (a) and (b) illustrate respectively, the structures of a *binary tree* and *completed binary tree.*

The convention that we are going to use in this thesis is:

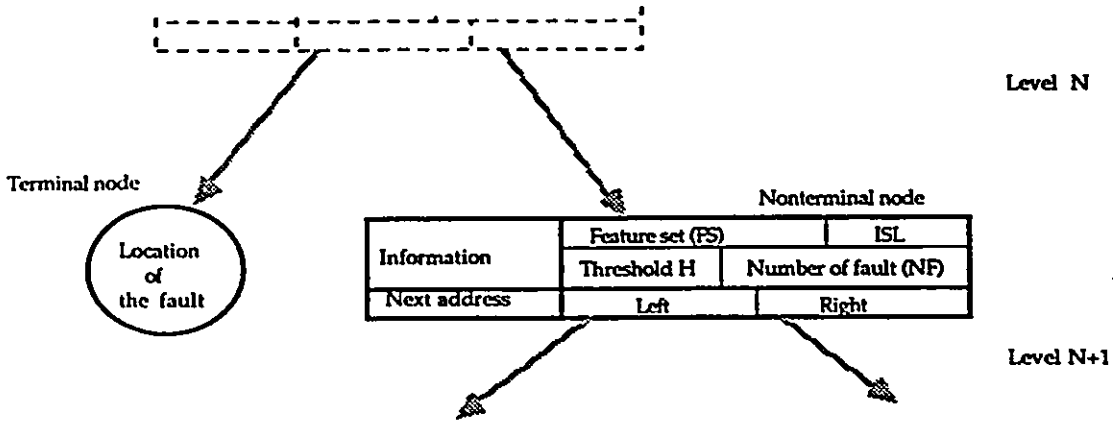


Figure 2.2: The fields of the multistage binary tree nodes (terminal and nonterminal node) .

□ represents a nonterminal node of the binary tree.

○ represents terminal node of the binary tree.

* Node A is the root of the tree and the father of the sons B and C.

* Nodes 1,2 are the sons of node D, and they are terminal nodes.

Fig. 2.1 (a)represents a typical binary tree . □

2.3 INFORMATION ON NODES

In a tree representation, the item in each level is called a node and contains two major fields, the information, and the next address. If we consider each of these fields, we represent their construction as:

⋆**The information field :** holds : (i)- commands for generating the input data, such as the feature set$[P_{(x_i)}]$ and the input sequence length (ISL). (ii)- commands to analyze the circuit output, such as the threshold level H used for the decision to be made to proceed either along the left or right subtree, and the number of faults under study

(NF) to specify if the node under consideration is terminal or nonterminal . However, OVERLAP faults (a faults cannot be well identified by a specified decision rule) are considered as part of the faults under study.

* The next address field contains the address of the next node of the next level. Such address is known as a pointer.

Figure 2.2 (a) and (b) illustrate the construction of terminal and nonterminal nodes in each level of a binary tree.

□**EXAMPLE 2.2:**

Suppose that we have a 7-input combinational circuit with eight possible fault classes (single or multiple). Figure 2.3 shows a sample binary tree constructed by our method to locate faults in a CUT (circuit under test) .□

2.4 METHOD OF TESTING

After the decision tree is constructed, the testing procedure can be applied to circuit under test, and can be implemented stage by stage. Each nonterminal node contains the threshold level that helps us to decide if we are to follow leftsubtree or rightsubtree until we reach the terminal node.

Assuming that we have the setup as shown in Figure 2.4 , an algorithm for testing can now be described as follows:

Algorithm

The testing can be done using the following steps:

1. • Generate the input sequence randomly with feature set $[P_{(x_1)} = P_{(x_2)} = ... = P_{(x_n)} = 0.5\ ]$, where n is the number of input lines .

2. • Feed the two units (CUT and GU) with the same input data, (see Figure 2.4) .

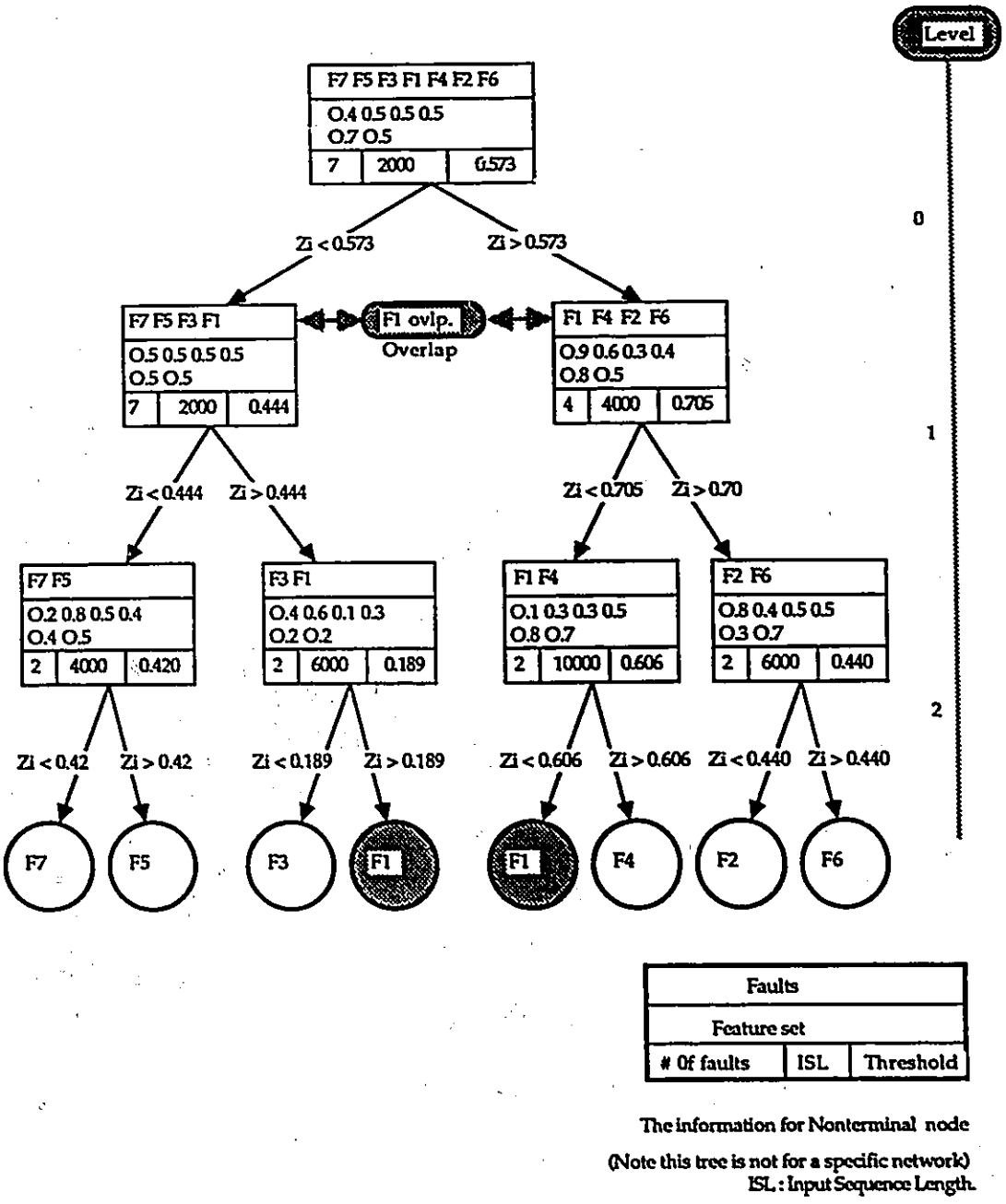


Figure 2.3: A sample of binary tree with 7 possible faults and six inputs .

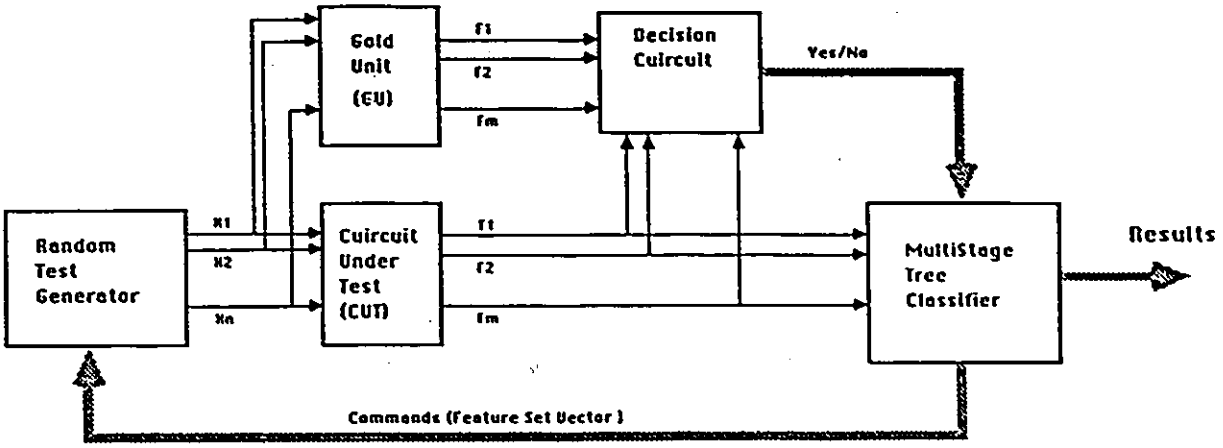


Figure 2.4: The block diagram for the testing method using random bit streams .

3. • Check both sets of outputs by using checking circuit (EX-OR gates or their simulation).

4. • Observe the result from the checking circuit after a long sequence of input data is applied (Input Sequence Length ISL = 10,000 , for example), then check :

   IF NO THEN :

   - The circuit is not faulty ;
   - Go to Step 9(a) .

   IF YES THEN :

   - The circuit is faulty ;
   - Go to Step 5 .

5. • Get from the root of the tree classifier the feature set vector $[P_{(x_1)}, P_{9x_2)}, ..., P_{(x_n)}]$, and the threshold level H .

6. • Generate random input data (with respect to the new feature set) .

7. • Observe the statistical mean Z's of the outputs of the circuit under test; then check :

IF Z less then the threshold ,

THEN GO TO leftsubtree ;

ELSE GO TO rightsubtree .

8. • Check the number of faults in the new node.

IF greater than 1 ,

THEN :

- Get the feature set vector ;
- Generate input data;
- Go to step 2;

ELSE :

- Get the information of the terminal node;
- GO to step 9 (b);

9. • Stop with the result that,

(a) ◇ The circuit is not faulty;

(b) ◇ The circuit is faulty (with the fault location).

If we assume that the tree is well constructed, we can find the faults very quickly, because the needed testing stage is almost proportional to:

$$Number\ of\ Stages(NS) = \log_2 m = \frac{\log_{10} m}{\log_{10} 2} \approx (\frac{1}{0.3}) \log_{10} m. \qquad (2.1)$$

where m is the number of possible faults.

To be more general, if we assume that we have an N_{ov} overlap faults, Eq.(2.1) becomes :

$$Number\ of\ Stages(NS) \approx (\frac{1}{0.3}) \log_{10}(m + N_{ov}). \tag{2.2}$$

The overall flowchart of testing can now be represented as in Figure 2.5 .

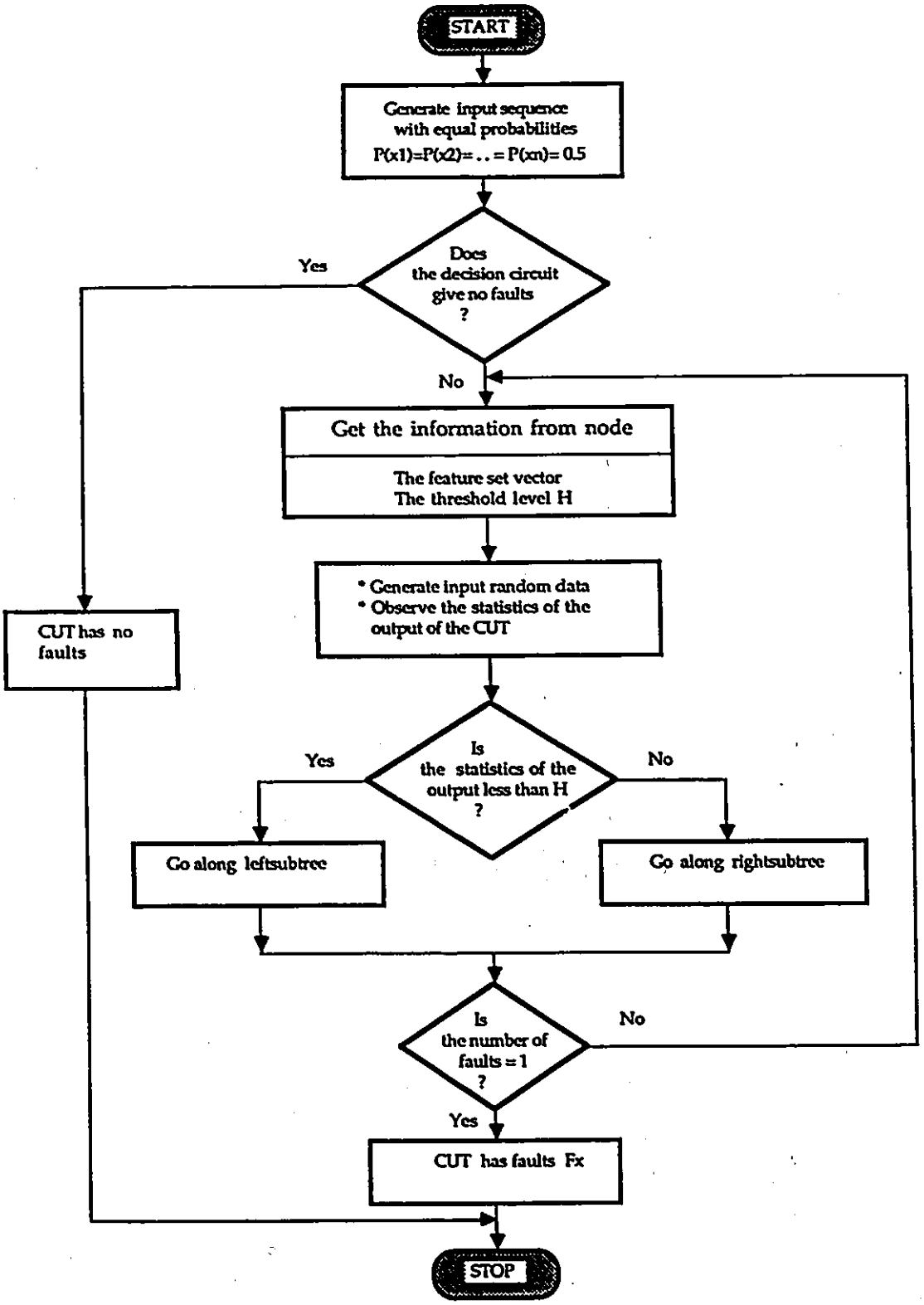


Figure 2.5: The flowchart for the testing method .

Chapter 3

THE THEORETICAL STUDY

3.1 INTRODUCTION

Many decisions have to be made that involve uncertainties. Here the random phenomenon arises from our unwillingness to carry out the deterministic method. In our approach, probability and statistics can provide the model to help us making decisions such as locating faults in a circuit that has to be tested within an assigned level of confidence.

We shall begin with a review of the basic concepts of the theory of probability [18,20] which forms the basis for describing our approach.

3.2 PROBABILITY ASPECT

Of the different distributions in statistical applications, the normal distribution is perhaps the most important one for our consideration.

Let us begin by defining the concept of the central limit theorem Figure (3.1).

If $\overline{X}$ is the mean of a random sample of size L taken from a population S having the mean μ and the finite variance σ^2, then, the value of a random variable Z whose distribution function approaches that of standard normal distribution as $L \rightarrow \infty$ [18] :

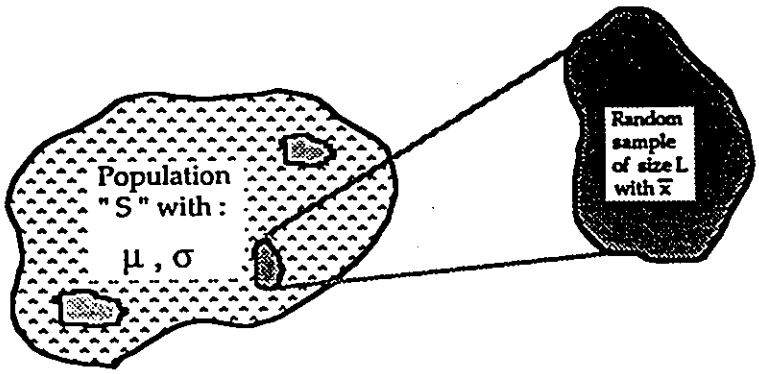


Figure 3.1: The concept of central limit theorem .

$$Z = \frac{(\overline{X} - \mu)}{\sqrt{\frac{\sigma^2}{L}}} \tag{3.1}$$

$$However : \overline{X} = \mu + Z\sqrt{\frac{\sigma^2}{L}} \tag{3.2}$$

Therefore, if $L \to \infty$, then $\overline{X} \approx \mu$. Hence $\overline{X}$ has a standard normal distribution .

The mean of the random variable is:

$$\overline{X} = \frac{1}{L}\sum_{i=1}^{L} x_i \approx \mu \tag{3.3}$$

under the condition that $L \to \infty$.

The variance of sample is given by :

$$Sample(Var)^2 = \frac{1}{(L-1)}\sum_{i=1}^{L}(x_i - \overline{X})^2 \approx \frac{1}{L}\sum_{i=1}^{L}(x_i - \mu)^2 \tag{3.4}$$

Then, for large L, the probability density function is defined as:

$$f_{(x)} = \frac{1}{\sqrt{2\pi\sigma_2}} e^{(-\frac{(x-\mu)^2}{2\sigma^2})} \tag{3.5}$$

The approximation of the theory

Here we shall discuss the approximation for calculating the mean μ and the variance σ^2

for observable data. The calculation of μ does not pose any problem . From Eq.(3.3), we have:

$$Approximation\ of\ \mu = \frac{1}{L}\sum_{i=1}^{L} x_i, \qquad or \qquad \mu \approx \frac{1}{L}\sum_{i=1}^{L} x_i \tag{3.6}$$

On the other hand, the calculation of $Sample(Var)^2$ is rather difficult, if we directly use Eq.(3.4). Instead, we will be going to use the algebraic equivalent form:

$$Sample(Var)^2 = \frac{1}{(L-1)}\sum_{i=1}^{L}(x_i^2 - 2x_i\overline{X} + \overline{X}^2) \tag{3.7}$$

which can be approximated (in our case) as :

$$Sample(Var)^2 = \frac{1}{L}[\sum_{i=1}^{L} x_i - L(\overline{X})^2] \tag{3.8}$$

under the conditions that : L tending to infinity, $(x_i)^2 = x_i$, because $x_i = 0$ or 1 only . Then :

$$Sample(Var)^2 = \frac{1}{L}\sum_{i=1}^{L} -(\overline{X})^2 = \mu - \mu^2 \tag{3.9}$$

in the case that L is the size of random sample, which is taken from a population that have mean μ and variance $Sample(Var)^2$. x_i is the value of a random variable whose distribution has the mean μ for samples from infinite populations; the variance of this distribution is $Sample(Var)^2/L$; therefore, the variance in our case should be approximated as:

$$\sigma^2 \approx \frac{\mu - \mu^2}{L} \tag{3.10}$$

Let us see what happens when this theory is applied to a combinational circuit. Each logic circuit consists of an input vector $X = (x_1, x_2, ..., x_n)$ where n is the number of input lines, and $x_i = 0$ or 1, where $1 \leq i \leq n$. We therefore might have 2^n such possible vectors , $1 \leq i \leq n$. The probability of a logic signal x_i is expressed as:

$$P(x_i) = P(x_i = 1) \tag{3.11}$$

If an input x_i is produced with a probability of $P(x_i)$, this will always mean that $\overline{x_i} = 1$ with probability $(1 - P(x_i))$. For clearer understanding, let us assume that the n-input circuit has only a single output F. Obviously, the output signal is a function of the input vector X : $F(X) = f_{(x_1,x_2,...,x_n,k)}$ where k represents the length of input sequence, $1 \leq k \leq L$.

We define Z_0 as the mathematical expectation of the output as :

$$Z_0 = E[f_X] = E[f_{(x_1,x_2,...,x_n,k)}]. \tag{3.12}$$

The average of the output over the input sample size L is denoted by the mean Z_L, where Z_L is defined as :

$$Z_L = \frac{1}{L}\sum_{1}^{L} f_{(X)} = \frac{1}{L}\sum_{k=1}^{L} f_{(x_1,x_2,...,x_n,k)}. \tag{3.13}$$

In switching theory the mathematical expectation Z_0 can be calculated by using the assigned input probability vector $(P(X_1), P(X_2), ..., P(X_n))$ of the Boolean function of the circuit .

In general, for a large circuit, the calculation of Z_0 may be rather difficult; however, if L is sufficiently large , the value of Z_0 can be closely approximated as:

$$Z_0 \approx \frac{1}{L}\sum_{k=1}^{L} f_{(x_1,x_2,...,x_n,k)} \tag{3.14}$$

Since Z_L is defined as the average of the outputs F over the input sequence length L (random distribution), then the limit of this normal distribution must be the normal distribution of Z_L, and Z_L will be a random variable. However, if we apply the theory of the central limit theorem, for very long input sequence, we can approximate the probability density function of the random variable Z_L as:

$$n_{(Z_L)} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{1}{2}\frac{(Z_L - Z_0)^2}{\sigma^2})} \tag{3.15}$$

where the mean and variance are , respectively :

$$Z_0 = \frac{1}{L}\sum_{k=1}^{L} F(x_1, x_2, ..., x_n, k) \tag{3.16}$$

$$\sigma^2 \approx \frac{1}{L}\sum_{i=1}^{L}(Z_{L,i} - Z_0)^2 \tag{3.17}$$

and $Z_{L,i}$ is the average for the ith input sequence of length L.

In the approximation that we used for this particular distribution of the outputs under consideration, the variance σ^2 can be obtained as:

$$\sigma^2 \approx \frac{1}{L}(Z_0 - (Z_0)^2). \tag{3.18}$$

Typically, the mathematical expectation Z_0 of a designed combinatorial network can be calculated by using the probability transformation rules (tabulated as formulas) of logic functions [25], and the Boolean function of the circuit . Thus Z_0 can be obtained readily . However, the average Z_L of the outputs is observable and measurable from the output data of the network under test. Later on, we will give a relationship between the calculated and measured data .

Let us now tabulate the output probability P_F of some logic gates, assuming that the input variables are mutually independent. P_F can be calculated from the probabilities of all the minterms for which F = 1 . Table 3.1 provide the formulas for probabilities of output variables for some standard logic gates .

TABLE 3.1 Probability Transformation Formulas ($P_i = P_{(i=1)}$)

Logic Gate	Boolean Function	Output probability P_F
NOT	F = $\overline{a}$	$P_F = 1 - P_a$
NOR	F = $\overline{a+b+c}$	$P_F = (1-P_a)(1-P_b)(1-P_c)$
OR	F = a+b+c	$P_F = 1-(1-P_a)(1-P_b)(1-P_c)$
AND	F = a.b.c	$P_F = P_a.P_b.P_c$
NAND	F = $\overline{a.b.c}$	$P_F = 1 - P_a.P_b.P_c$
EX-OR	F = $\overline{a}b + a\overline{b}$	$P_F = P_a + P_b - 2P_aP_b$
EX-NOR	F = $ab + \overline{a}\overline{b}$	$P_F = 1-(P_a + P_b - 2P_aP_b)$

□EXAMPLE 3.1

Consider the combinational circuit shown in Figure 3.2 .

The output function can be expressed as:

$$f_{(A,B)} = \bar{A}B + A\bar{B}$$

The value of the mathematical expectation Z_0 of the output $f_{(A,B)}$ can be calculated from the definition as:

$$Z_0 = E[f_{(A,B)}] = E[\bar{A}B + A\bar{B}]$$

$$Z_0 = (\bar{A}=0)(B=0)P_{(\bar{A}=0)}P_{(B=0)} + (\bar{A}=0)(B=1)P_{(\bar{A}=0)}P_{(B=1)}$$
$$+(\bar{A}=1)(B=0)P_{(\bar{A}=1)}P_{(B=0)} + (\bar{A}=1)(B=1)P_{(\bar{A}=1)}P_{(B=1)}$$

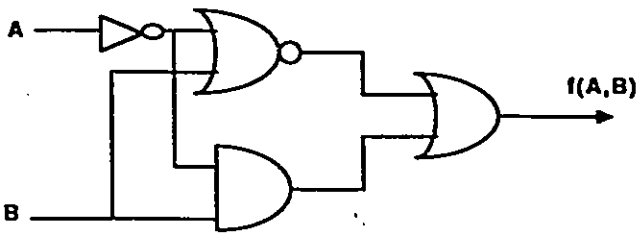


Figure 3.2: A simple combinational circuit .

$$+(A=0)(\bar{B}=0)P_{(A=0)}P_{(\bar{B}=0)}+(A=0)(\bar{B}=1)P_{(A=0)}P_{(\bar{B}=1)}$$
$$+(A=1)(\bar{B}=0)P_{(A=1)}P_{(\bar{B}=0)}+(A=1)(\bar{B}=1)P_{(A=1)}P_{(\bar{B}=1)}$$

An easy way for calculating the expectation is to take all the combinations of minterms in the sum-of-products of the function. In our case: $\bar{A}B$: 00 01 10 11 and $A\bar{B}$: 00 01 10 11 .

Therefore, if we assume that:

$$P_{(A=1)} = P_{(\bar{A}=0)} = a, \text{ and } P_{(B=1)} = P_{(\bar{B}=0)} = b,$$

or $$P_{(A=0)} = P_{(\bar{A}=1)} = 1-a, \text{ and } P_{(B=0)} = P_{(\bar{B}=1)} = 1-b,$$

we have :

$$Z_0 = 1.0.(1-a)(1-b)+0.1.a.b+1.0.(1-a)(1-b)+1.1.(1-a).b$$
$$+0.0.(1-a).b+0.1.(1-a)(1-b)+1.0.a.b+1.1.a.(1-b)$$

or $$Z_0 = (1-a)b+a(1-b) = b-ab+a-ab$$

or $$Z_0 = a+b-2ab$$

or $$Z_0 = 2a-2a^2 \qquad \text{for : a = b .}$$

However, if we use the formulas from the Table 3.1, Z_0 can be expressed as:

$$Z_0 = E[\bar{A}B+A\bar{B}] = P_{\bar{A}B}P_{A\bar{B}}$$
$$\equiv \text{AND(NOT(A),B) + AND(A,NOT(B))}$$

or $$Z_0 = (1-a).b+a.(1-b)$$

or $Z_0 = a + b - 2ab...$ for : $a \neq b$

or $Z_0 = 2a - 2a^2.....$ for : $a = b.$ □

3.3 FAULT DISTANCE THEORY

Let us start by the definition of the word Feature in our study .

The word feature here is the feature of the input sequence which is not directly used for the pattern recognition purposes . It will mean (in our case) the probability vector : $[P_i(x_1), P_i(x_2), ..., P_i(x_n)]$ of the inputs, where n is the number of the input lines and i is the ith approximation of the best feature set. Therefore, for the pattern classification for fault diagnosis, the measurement of Z_L has been used.

Furthermore, it should be noted that class separability is the separation of two or more classes, and the feature selection becomes the process of choosing of those features which are most effective in showing fault separability in Z_0 space, where faults represent classes in our study .

The most important aspect of the study is to find the feature set : $[P_i(x_1), P_i(x_2), ..., P_i(x_n)]$ which maximizes some distance measure between two or more faults in the Z_0 space.

EUCLIDEAN DISTANCE (denoted by E-distance) may be used as the distance measure, which is explained by the following example:

□ EXAMPLE 3.2

Let us consider the same circuit of Figure 3.2. Assume as shown in Figure 3.3 that we have fault F_1 with line A stuck-at-zero, and fault F_2 with line B stuck-at-one.

To find the feature set to maximize the distance between these two faults, let us consider the distance in the Z_0 space in the form of E-distance.

From the circuit we have:

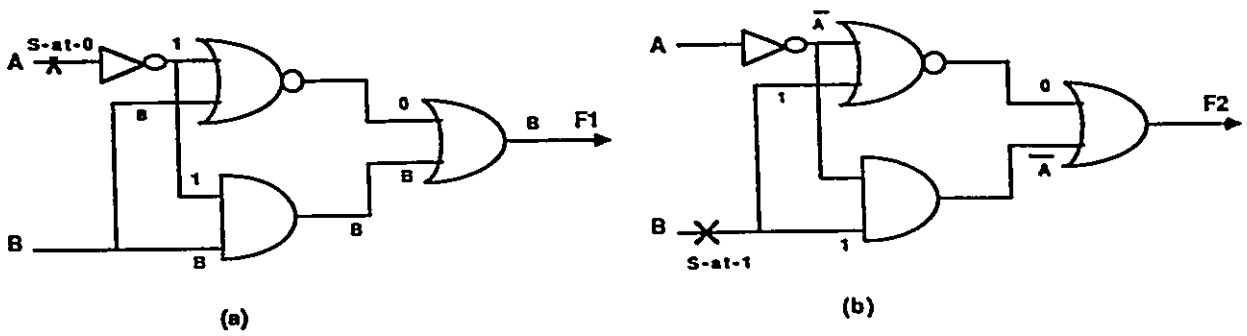


Figure 3.3: The circuit for Example 3.2 with line A-s-a 0, and line B-s-a 1 .

$$F_1 = B \qquad \text{and} \qquad F_2 = \bar{A}$$

If we assume the feature : $[P_{(A=1)}, P_{(B=1)}] = [a, b]$, then the expectations Z_0 of the faults are:

$$Z_0(F_1) = b \qquad \text{and} \qquad Z_0(F_2) = 1 - a$$

Hence the E-distance between $Z_0(F_1)$ and $Z_0(F_2)$ is :

$$E - distance = \sqrt{[Z_0(F_1)]^2 + [Z_0(F_2)]^2} \tag{3.19}$$

$E - distance = \sqrt{b^2 + (1-a)^2}$ with the contraints :

$$0 \leq a \leq 1 ;$$

$$0 \leq b \leq 1 .$$

If we solve this constrained maximization problem, we will have a = b = 0, or a = b = 1, which i the same if we started with the feature $P_{(A=1)} = P_{(B=1)}$.

(In our approximation, the starting point is: $P_{(X_1)} = P_{(X_2)} = ... = P_{(X_n)} = 0.5$)

On the other hand, if we assume the feature $[P_{(A=1)} = P_{(B=1)}] = [a, a]$, then

$$Z_0(F_1) = a \qquad \text{and} \qquad Z_0(F_2) = 1 - a$$

Then the E-distance will be :

$$E - distance = \sqrt{a^2 + (1-a)^2}$$

$$E - distance = \sqrt{2a^2 + 1 - 2a}, \text{ with the contraint that } 0 \leq a \leq 1.$$

E-distance will have the maximum value if we select a = 0 or 1; hence the feature will be $[P_{(A=1)}, P_{(B=1)}] = [0, 0]$ and $[P_{(A=0)}, P_{(B=0)}] = [1, 1]$.

□

This example suggests that the assumption of equal probabilities may be good enough in many circuits. For large circuits, it is evident that the feature is long and difficult to calculate , and the more the number of fault classes, the more difficult it gets. Therefore, this assumption will minimize the calculation complexity for finding the desired feature in many cases.

A major disadvantage of using the E-distance is that it has no direct relation with the error probability, or with the means (Z_0's) and variances (σ^2's) of the density functions. So our goal is to find such a distance between two classes (faults), which can be represented by the distance between two density functions corresponding to the density functions of Z_L's of the two fault classes (since Z_L is a random variable under the random input) . Intuitively, it seems that larger fault distances among faults may result in low testing error. BHATTACHARYYA DISTANCE (denoted by B-distance) [11,13] is used here for classification, because of its convenient relation with the error probability where BAYES' rule is applied for the fault separability.

Some interesting properties for B-distance which may be useful in our study are :

i - B-distance is always a positive quantity ,0 ≤B-distance ≤ ∞ .

ii - B-distance does not obey the triangle inequality .

iii - If we have: α a set of system parameters with π_1 a priori probability, and β as another set with π_2, and B-disance (α) > B-distance(β), then there exists $\Pi(\pi_1, \pi_2)$ as a set of a priori probabilities for which the error condition is :

$$\epsilon(\alpha, \pi) < \epsilon(\beta, \pi) \ .$$

For the particular case where (Z_L's) are assumed to have normal density functions, B-distance measure depends on (Z_0's) and (σ^2's) of the density functions. Therefore, the problem is to minimize the error probability by finding a specified feature set $[P_i(x_1), P_i(x_2), ..., P_i(x_n)]$ that maximizes the distance (B-distance) between the two fault classes .

3.3.1 BAYE S' CLASSIFIER

In some statistical cases, if an outcome of an experiment depends on the outcomes of various intermediate stages, BAYES' rule is very effective to simplify the study.

To illustrate, suppose that there are two faults : F_1 with output expectation value Z_1, and F_2 with Z_2, with respect to a prespecified feature set $[P_{(A)}, P_{(B)}]$, if we assume that A,B are the inputs of the circuit under study . Also assume that F_1 has P_1 as a priori probability (that the fault exists), and F_2 has P_2.

Using Eq.(3.5), the average values $Z_L(F_1)$ and $Z_L(F_2)$ of the outputs of F_1 and F_2 have the probability density functions (pdf), respectively, as follows (assuming L is of fixed length).

$$f_{(Z_L(F_1))} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{1}{2}\frac{[Z_L(F_1)-Z_1]^2}{\sigma^2})} \tag{3.20}$$

$$f_{(Z_L(F_2))} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{(-\frac{1}{2}\frac{[Z_L(F_2)-Z_1]^2}{\sigma^2})} \tag{3.21}$$

where :

$$\sigma_1^2 = \frac{1}{L}\sum_{i=1}^{L}(Z_{1,i} - Z_1)^2 \approx \frac{1}{L}(Z_1 - Z_1^2). \tag{3.22}$$

$$\sigma_2^2 = \frac{1}{L}\sum_{i=1}^{L}(Z_{2,i} - Z_2)^2 \approx \frac{1}{L}(Z_2 - Z_2^2). \tag{3.23}$$

where : L tending to ∞ .

The pdf $f_{(Z_L(F_1))}$ and $f_{(Z_L(F_2))}$ can be graphically represented as in figure 3.3 .

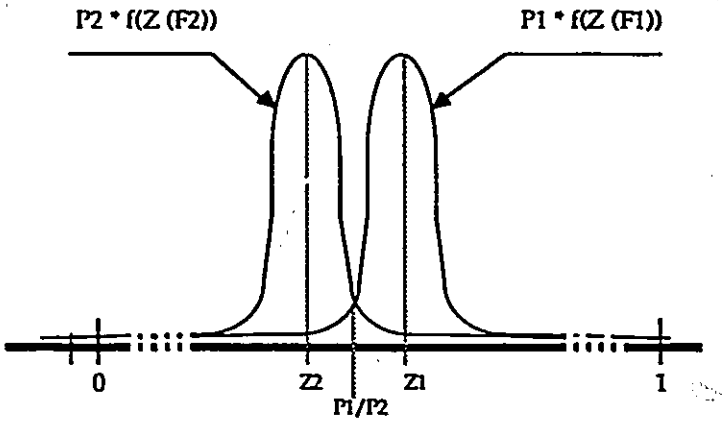


Figure 3.4: pdf of faults F_1 and F_2.

Before deciding on the decision rule, we should define the meaning of $P(\alpha \mid \beta)$ notation. The notation $P(\alpha \mid \beta)$ is used for the conditional probability of an event α relative to a sample space β , or given that event β has occurred.

If we have Z as an observation vector (which is a set of numbers obtained through an observation or a measurement process) and if we need it to determine whether Z belongs to fault F_1 or F_2 , the decision rule which is based simply on probabilities may be written as follows:

$$If \quad P(F_2 \mid Z) \geq P(F_1 \mid Z), \quad then \quad Z \in F_2; \quad else \quad Z \in F_1 \tag{3.24}$$

The conditional probabilities $P(F_i \mid Z)$ may be calculated from P_i (the a priori probabilities). Therefore $P(Z \mid F_i)$ or the conditional density functions can be written as:

$$P(F_i \mid Z) = \frac{P(Z \mid F_i)P_i}{P(Z)} \tag{3.25}$$

Since Eq.(3.17) could be written as:

$$\frac{P(Z \mid F_2)P_2}{P(Z)} \geq \frac{P(Z \mid F_1)P_1}{P(Z)} \tag{3.26}$$

the decision rule of Eq.(3.17) can be expressed as:

$$If \quad P(Z \mid F_2)P_2 \geq P(Z \mid F_1)P_1, \quad then \quad Z \quad \in F_2; \quad else \quad Z \in F_1 \tag{3.27}$$

or :

$$If \quad \frac{P(Z \mid F_2)}{P(Z \mid F_1)} \geq \frac{P_1}{P_2}, \quad then \quad Z \quad \in F_2; \quad else \quad Z \in F_1 \tag{3.28}$$

$$where \qquad P(Z \mid F_1) = f_{(Z_L(F_1))} \qquad and \qquad P(Z \mid F_2) = f_{(Z_L(F_2))} \tag{3.29}$$

This decision is optimum in the sense that it is based on the full statistics of the measurements. This is the well known is BAYES' classifier .

3.3.2 ERROR DERIVATION

In general, through this decision rule or any other decision rule it is almost impossible to lead to perfect classification. In order to analyze the confidence level of a decision making, we must calculate the probability of error, which is the probability that Z is assigned to the wrong class. Then the testing error probability [8] can be expressed as:

$$\varepsilon = Prob(error) = Prob(error \mid F_1)P_1 + Prob(error \mid F_2)P_2 \tag{3.30}$$

Whenever a sample belongs to F_1 or F_2, an error occurs whenever $Z \in$ the regions of domain ξ_2 or ξ_1 of F_2 or F_1, respectively .

Therefore:

$$\varepsilon = Prob(Z \in \xi_1 \mid F_1)P_1 + Prob(Z \in \xi_2 \mid F_2)P_2 \tag{3.31}$$

To solve this equation, we use the integration of the density functions of F_1 and F_2; therefore, by the help of Figure 3.3 , we have :

$$\varepsilon = P_1 \int_0^{P_1/P_2} P(Z \mid F_1)dz + P_2 \int_{P_1/P_2}^{1} P(Z \mid F_2)dz \tag{3.32}$$

In general:

$$\varepsilon = P_1 \int_{-\infty}^{P_1/P_2} P(Z \mid F_1)dz + P_2 \int_{P_1/P_2}^{+\infty} P(Z \mid F_2)dz \tag{3.33}$$

Or :

$$\varepsilon = P_1\varepsilon_1 + P_2\varepsilon_2. \tag{3.34}$$

where $\varepsilon_1, \varepsilon_2$ are the errors of misclassifying samples from faults F_1 and F_2,respectively.

3.3.3 BHATTACHARYYA DISTANCE (Calculation and Observation)

Let us study the fact that we cannot obtain a closed-form expression for the error probability. We may seek either an approximate expression, or an upper and/or lower bound on the error probability.

If we use Chernoff bound theory, the probability of error may be bounded and expressed as:

$$\varepsilon \leq (P_2)^{1-s}(P_1)^s e^{[-\mu(s)]}. \tag{3.35}$$

where s is a real number, F being the random class vector samples, F_1 and F_2 being the two elements of random vector F, and $\mu(s)$ is the threshold of the solutions (for given ε). Then :

$$\varepsilon = (P_2)^{1-s}(P_1^s \int_\xi P(Z \mid F_2)^{1-s} P(Z \mid F_1)^s dz \tag{3.36}$$

where ξ is the total range of the domain of Z.

The selection of 0.5 for s, which still gives the upper bound of ε, bypasses the problem of finding the optimum s as well as the relative sensitivity of Z around the optimum value. Then Eq.(3.36) becomes:

$$\varepsilon \leq (P_1P_2)^{1/2} \int_\xi P(Z \mid F_2)^{1/2} P(Z \mid F_1)^{1/2} dz \tag{3.37}$$

$$or \qquad \varepsilon \leq (P_1P_2)^{1/2} e^{[-\mu(1/2)]} \tag{3.38}$$

$$where \qquad \mu(1/2) = -Ln \int_\xi [P(Z \mid F_2)P(Z \mid F_1)]^{1/2} dz \tag{3.39}$$

This $\mu(1/2)$ is the well known Bhattacharyya distance and it will be the fault distance in our study.

In our approach, $\mu(1/2)$ for the normal distributions of Eq.(3.15) and Eq.(3.16) can be expressed as:

$$B_{cal} = B - distance_{1,2} = \mu(1/2) = \frac{1}{4}\frac{(Z_1 - Z_2)^2}{\sigma_1^2 + \sigma_2^2} + \frac{1}{2}Ln\frac{\frac{1}{2}(\sigma_1^2 + \sigma_2^2)}{\sigma_1\sigma_2}. \qquad (3.40)$$

where Z_1, Z_2 and σ_1^2, σ_2^2 can be obtained by the help of Eqs.(3.11), (3.16), and(3.17), respectively .

From Eq.(3.31), we can find a relation between the testing error probability and the B-distance as follows:

$$Ln(\varepsilon) \leq Ln[(P_1P_2)^{1/2}e^{(-\mu(1/2))}] \leq (\frac{1}{2})Ln(P_1P_2) - \mu(1/2) \qquad (3.41)$$

$$Or \qquad \mu(1/2) \leq \frac{1}{2}Ln(P_1.P_2) - Ln(\varepsilon) \leq \frac{1}{2}Ln\frac{P_1.P_2}{\varepsilon^2} \qquad (3.42)$$

If we take the equality condition only, the B-distance between the two distributions of the two fault classes F_1 and F_2 is :

$$B_{req} = B - distance_{1,2} = \mu(1/2) = \frac{1}{2}Ln\frac{P_1.P_2}{\varepsilon^2} \qquad (3.43)$$

The conclusion that we arrive at is that B-distance has link between the confidence level (CL) of the testing and the feature set selection.

Given a specified test confidence level CL, we can relate it to the limiting error probability ε as : $CL \geq 1 - \varepsilon$.

We must select a feature set having an observable or measurable fault distance (B-distance) satisfied by Eq.(3.43), or greater than that value; otherwise, another feature set should be selected.

Note that from Eq.(3.40) we have the calculated distance to be denoted by B_{cal}, and from Eq.(3.43) we have the required B-distance to be denoted by B_{req} .

Then the condition to study is :

If $B_{cal} \geq B_{req}$

then, accept the feature set as the required feature,

else find another feature set to satisfy the condition.

We are going to use a stochastic approximation method for the solution of this problem.

3.3.4 METHOD OF CHOOSING THE FEATURE SET

The process of choosing the feature set (in previous case) is straightforward except at the decision point where the assumed feature under test does not satisfy the condition in Eq.(3.43) .

One way of choosing a new feature set is to use the discretion and the previous experimental results. Another way of solving this problem of finding the desired feature set satisfying Eq.(3.43) is to use any stochastic gradient method, as for example, the method of stochastic approximation [14,26]

Suppose the desired B-distance should satisfy the condition in Eq.(3.36). Let the approximated (calculated B-distance) at the ith stage be represented as B_{i-cal} , where:

$$B_{i-cal} = f(P_{i(x_1)}, P_{i(x_2)}, ..., P_{i(x_n)}) \tag{3.44}$$

Remembering that B_{i-cal} is a function of the feature under test at the ith stage, we want the root of the equation :

$$B_{i-cal} = f(P_{i(x_1)}, P_{i(x_2)}, ..., P_{i(x_n)}) = B_{req} = B_{min} \tag{3.45}$$

where B_{min} is the desired B-distance satisfying Eq.(3.43). Using the stochastic approximation techniques extended by Kiefer and Wolfowitz [14] , we can form from Eq.(3.45) the regression function F_i as:

$$F_{(P(x_1),P(x_2),....,P(x_n))} = (B_{i-cal} - B_{min})^2 = F_i \tag{3.46}$$

Our problem now is to find the root of F_i .

If the theory of Kiefer and Wolfowitz is used, the present estimate and the new observation $P_i(x_j)$ is given by:

$$P_{i+1}(x_j) = P_i(x_j) - \Gamma_i \frac{\delta F_i}{\delta P_i(x_j)} \tag{3.47}$$

where $1 \leq j \leq n$ and Γ_i is a sequence satisfying the convergence conditions :

$$(a) : \lim_{i \to \infty} \Gamma_i = 0, \qquad (b) : \sum_{i=1}^{\infty} \Gamma_i = \infty, \qquad (c) : \sum_{i=1}^{\infty} \Gamma_i^2 < \infty \tag{3.48}$$

Eq.(3.48a) allows the process to settle down in the limit, Eq.(3.48b) ensures that there is enough corrective action to avoid stopping short of root, Eq.(3.48c) guarantees the variance of the accumulated noise to be finite. More generally, a sequence of the form $\Gamma_i = (\frac{1}{i})^k$ where $1 \leq k \leq \frac{1}{2}$, satisfies Eq.(3.48 a,b,c), and it is not the only possible sequence .

Hence $\frac{\delta F_i}{\delta P_i(x_j)}$ can be obtained by experimentation as :

$$\frac{\delta F_i}{\delta P_i(x_j)} = \frac{[F_{(P_i(x_j)+a_i)} - F_{(P_i(x_j)-a_i)}]}{2a_i} \tag{3.49}$$

where a_i is a sequence satisfying the conditions :

$$(a) : \lim_{i \to \infty} a_i = 0, \qquad (b) : \sum_{i=1}^{\infty} (a_i)^2 \leq \infty, \qquad (c) : \sum_{i=1}^{\infty} (\frac{\Gamma_i}{a_i})^2 < \infty \tag{3.50}$$

Note that Eq.(3.50c) is to cancel the accumulated noise effects .

Therefore, we measure the derivative experimentally and modify $P_{i+1}(x_j)$ as :

$$P_{i+1}(x_j) = P_i(x_j) - \Gamma_i \frac{[F_{(P_i(x_j)+a_i)} - F_{(P_i(x_j)-a_i)}]}{2a_i} \tag{3.51}$$

Then, for an n-input combinational network, Eq.(3.47) can be extended and gives the algorithm of the form :

$$\begin{bmatrix} P_{i+1}(x_1) \\ P_{i+1}(x_2) \\ \cdot \\ \cdot \\ \cdot \\ P_{i+1}(x_n) \end{bmatrix} = \begin{bmatrix} P_i(x_1) \\ P_i(x_2) \\ \cdot \\ \cdot \\ \cdot \\ P_i(x_n) \end{bmatrix} - \Gamma_i * \begin{bmatrix} \frac{\delta F_i}{\delta P_i(x_1)} \\ \frac{\delta F_i}{\delta P_i(x_2)} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\delta F_i}{\delta P_i(x_n)} \end{bmatrix} \tag{3.52}$$

The vector $[\frac{\delta F_i}{\delta P_i(x_1)}, \frac{\delta F_i}{\delta P_i(x_2)}, ..., \frac{\delta F_i}{\delta P_i(x_n)}]$ can be obtained from Eq.(3.49) as :

$$\begin{bmatrix} \frac{\delta F_i}{\delta P_i(x_1)} \\ \frac{\delta F_i}{\delta P_i(x_2)} \\ \cdot \\ \cdot \\ \cdot \\ \frac{\delta F_i}{\delta P_i(x_n)} \end{bmatrix} = \begin{bmatrix} \frac{F(P_i(x_1)+a_i, P_i(x_2), ..., P_i(x_n)) - F(P_i(x_1)-a_i, P_i(x_2), ..., P_i(x_n))}{2a_i} \\ \frac{F(P_i(x_1), P_i(x_2)+a_i, ..., P_i(x_n)) - F(P_i(x_1), P_i(x_2)-a_i, ..., P_i(x_n))}{2a_i} \\ \cdot \\ \cdot \\ \cdot \\ \frac{F(P_i(x_1), P_i(x_2), ..., P_i(x_n)+a_i) - F(P_i(x_1), P_i(x_2), ..., P_i(x_n)-a_i)}{2a_i} \end{bmatrix} \tag{3.53}$$

The algorithm in Eq.(3.52) and Eq.(3.53) can be used to get the feature set : $[\hat{P}(x_1), \hat{P}(x_2), ..., \hat{P}(x_n)]$ will minimize the F function in Eq.(3.39) , which means :

$$\begin{bmatrix} \hat{P}(x_1), \\ \hat{P}(x_2), \\ \cdot \\ \cdot \\ \cdot \\ \hat{P}(x_n) \end{bmatrix} \longrightarrow \begin{bmatrix} P(x_1), \\ P(x_2), \\ \cdot \\ \cdot \\ \cdot \\ P(x_n) \end{bmatrix}, \qquad as \qquad i \to \infty \tag{3.54}$$

Although this procedure for finding the desired feature set is straightforward, the experimentation needed for the calculations of Eq.(3.53) are quite involved .

3.4 ALGORITHM

If we follow the process of Section 3.2, we can easily form (in the case of two faults i and j) an algorithm for finding the features set .

This algorithm can be described as :

Algorithm

Step1 : Set the following theoretical data :

ϵ : The error probability .

P_i and P_j : The a priori probabilities of F_i and F_j, respectively . (in our case we assume that the faults have equal probabilities of appearing as $\frac{1}{NF}$(NF= Number of faults)

$[P_0(x_1), P_0(x_2), ..., P_0(x_n)]$: The arbitrary feature vector of the input sequence (in our case, the starting point for the vector is : [0.5, 0.5,...., 0.5]).

Step2 : Find the B$-distance_{i,j}$ from Eq.(3.43) which is the theoretical $B_{req(i,j)}$ Let us say it is the minimum B-distance B_{min}.

Step3 : Simulate the combinational network with all the possible faults.

Step4 : This step consists of several substeps as :

(i) Generate the input random sequence with respect to the prespecified feature vector.

(ii) Observe the means Z_i and Z_j experimentally with the help of Eq.(3.16) .

(iii) Calculate the variances σ_i^2 and σ_j^2 from Eq.(3.18) .

Step5 : Calculate Bhattacharyya distance (B_{ij-cal}) from Eq.(3.40) .

Step6 : Compare the experimental result (Step2) with the theoretical result (Step4) and make the decision as:

if $B_{ij-cal} \geq B_{ij-req}$

then accept the feature vector as the required feature set;
else use stochastic approximation method to form
the new feature vector, then go to Step4.

If after several trials the stochastic approximation does not give a good result, then we should start with a new arbitrary feature vector other then [0.5,0.5,...,0.5], and then go to Step4.

The algorithm is simple up to Step 6 where the assumed feature vector under test does not satisfy the condition of Eq.(3.43). Then the input sequence needed for the calculation of Eq.(3.52) and Eq.(3.53) can be quite large. Two inportant facts should be taken into consideration. One is the time for generating the sequence and simulating the circuit and the second is the rate of convergence which might be rather slow.

3.5 m-FAULT CASE

From the two fault case the procedure can be extended to the multiple fault case.

Assume we having a combinational network with m possible faults. Let us assume that these faults can be grouped into two classes A and B with class A containing k faults, and class B containing (m-k) faults. Let the total error probability be ϵ (specified) and the pairwise error probability be ϵ_{ij} between faults i and j, respectively.

Then the most important condition to consider is :

$$\epsilon \leq \sum_{i=1}^{k} \sum_{j=k+1}^{m} \epsilon_{ij} \leq \sum_{i=1}^{k} \sum_{j=k+1}^{m} B_{ij} \tag{3.55}$$

where B_{ij} is the B-distance between two faults i and j from the two classes, respectively. Because ϵ is already specified as (1-CL) where CL is the confidence level, then Eq.(3.55)

becomes :

$$\epsilon \leq \sum_{i=1}^{k} \sum_{j=k+1}^{m} B_{ij} \tag{3.56}$$

For the stochastic approximation, the function to be formed is:

$$F^1 = (\sum_{i=1}^{k} \sum_{j=k+1}^{m} B_{ij} - \varepsilon)^2 \tag{3.57}$$

Then algorithms in the form of Eq.(3.56) and Eq.(3.57) can be formed to minimize F^1 and to get the desired feature set vector. These procedures may give rise to a problem which will be evident in the following example.

□EXAMPLE 3.3

Let us consider a combinational network with five possible faults (i.e, m = 5). Let those five faults be grouped into two classes, Class 1 with 2 faults (k = 2); Class 2 with 3 faults (m-k = 2). Then Eq.(3.57) becomes:

$$F^1 = [(B_{13} + B_{14} + B_{15}) + (B_{23} + B_{24} + B_{25}) - \epsilon]^2. \tag{3.58}$$

where ϵ is the specified error probability. As we know, in general B-distance does not satisfy the triangle inequality, so in minimizing F^1, the situations in Figure 3.4(a,b) may result .

Moreover, the use of the binary decision classifier with the assumed grouping will give very poor result.

On the other hand, if the result gives rise to the situation as depicted in Figure 3.5(a), the binary decision will give good result.

One way to ensure that the situation of Figure 3.5(a) will result is to define two functions F^2 and F^3 as:

$$F^2 = B_{12} \tag{3.59}$$

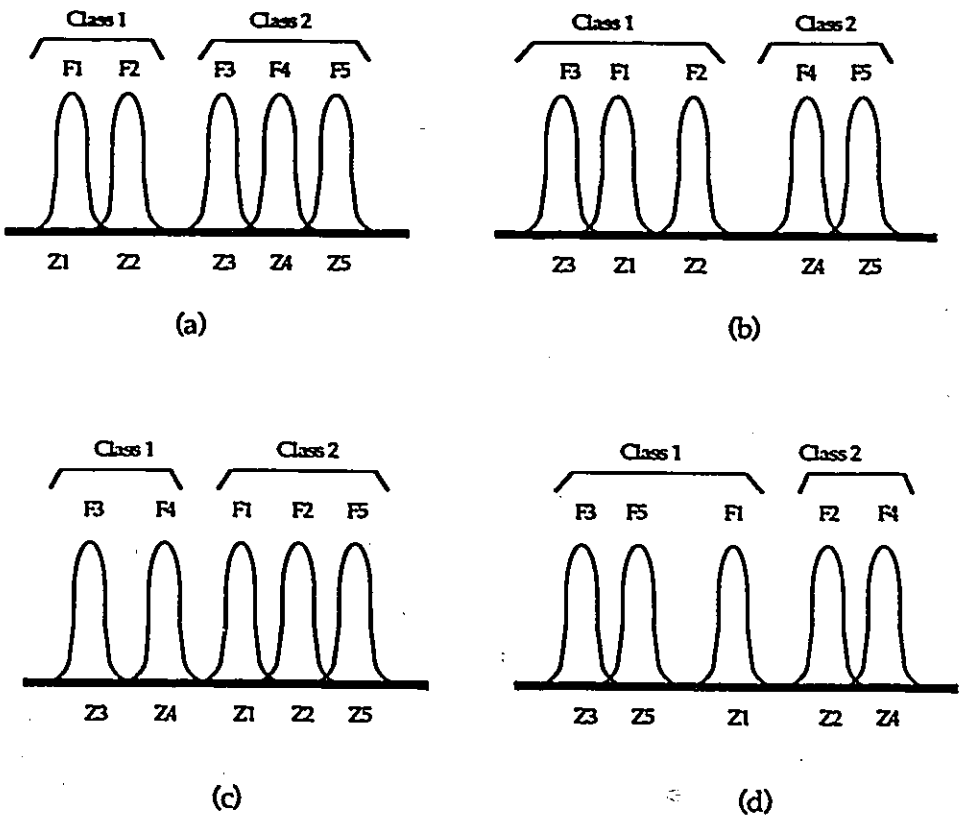


Figure 3.5: Four possibilities for the pdf's distribution of Example 3.3 .

$$F^3 = (B_{34} + B_{35} + B_{45}) \tag{3.60}$$

Now, if we minimize the functions F^1, F^2, and F^3 simultaneously, it is expected that the situation as in Figure 3.4(a) is ensured. The problem of minimization can be solved by minimizing F^1 first, then F^2, and finally F^3, and repeating the process until the convergence is reached. □

Unfortunately, there are two major disadvantages of this method :

(1) The time required for the calculations is quite long. There is no general proof of convergence for such a minimization problem. Therefore, for a given problem, it is hopefully expected that the minimization problem will converge.

From the preceding discussions, it is obvious that the process of finding the desired feature is quite involved, and it can be formulated as in Eq.(3.42) and Eq.(3.43). How-

ever, once the feature vector is obtained, the method of testing and locating the faults becomes rather simple.

Figure 3.5 shows the flowchart of the overall process of finding the desired feature vector.

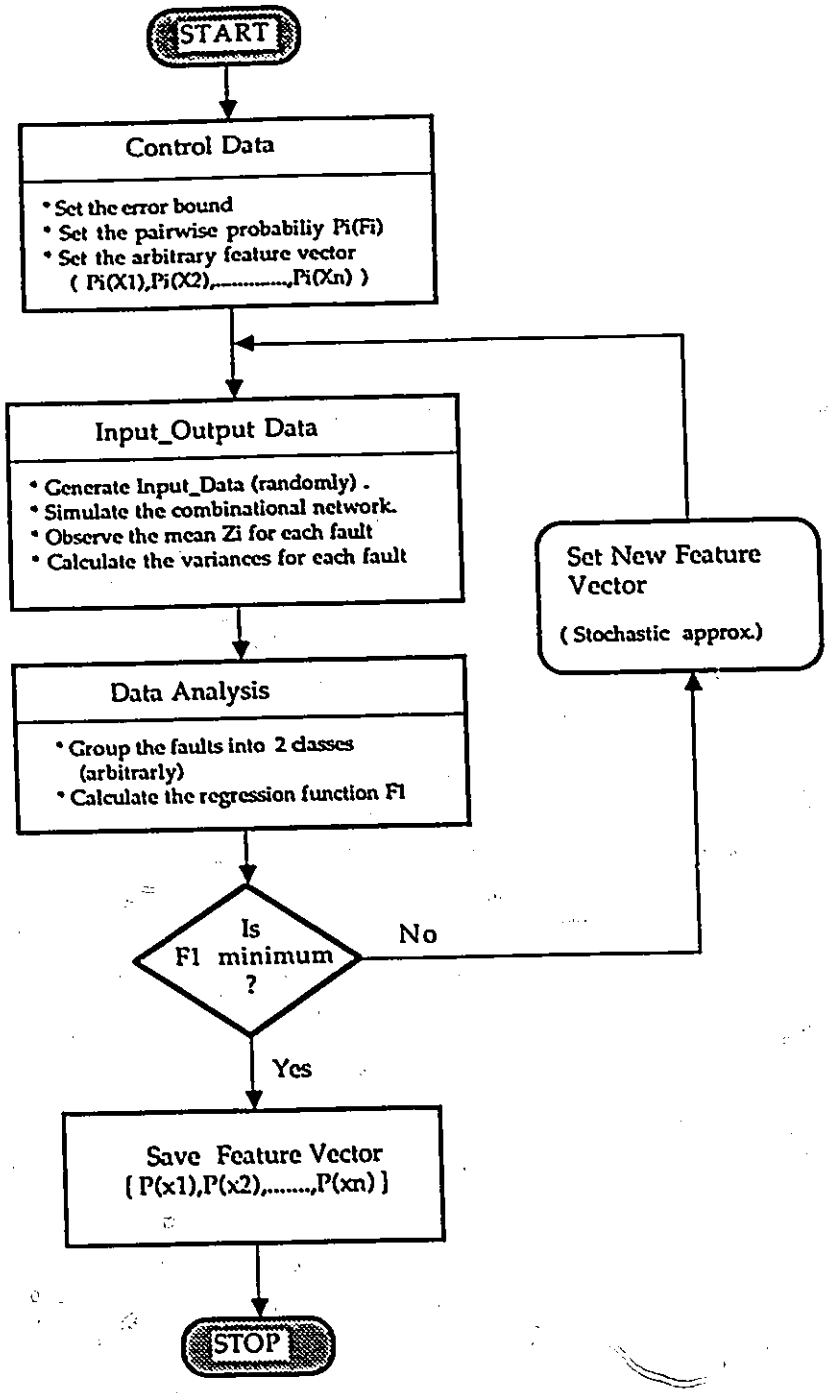


Figure 3.6: The flowchart for finding the desired feature vector.

Chapter 4

TREE CLASSIFIER THEORY

4.1 INTRODUCTION

The present study on faults is based on the use of binary trees which is comprised of two phases. The first phase is to build a binary tree and the second phase is to traverse the tree.

In this chapter, we focus our attention to building the tree. It centers around classification of the faults, based on the feature vector which has been found to be very effective in the classification. We define the rules of grouping the faults with respect to the threshold which can be obtained from the theory of error calculation.

The binary classifier tree has the property that the number of faults in each left or right subtree is less than the number of faults in the root of those subtrees, until the last node that has the fault to be located.

We will use the theory of probability to calculate an optimal value of the threshold and finally present an example to illustrate the theory.

4.2 FAULT CLASSIFICATION

Assuming the different fault classes are available, the only problem is the development of the fault classification rule. For our purpose some of the well known pattern recognition

techniques [4-6] can be used. In general, all the different classification rules do not lead to the solution without producing different error probabilities. But to approach more to the reality of measurement, it is expected that the technique which is more measure-matched for a given application will lead to the desired goal. Here the binary decision tree has been adapted to this multi-class classification problem.

4.2.1 DECISION RULE

The starting point of the classification will be established in the root of the tree and the classification will be done with the help of a threshold H. Each successive mean of each fault is then compared to H. If it is smaller, it goes to the left subtree; if it is greater, it goes to the right subtree; and if it matches, we call it an overlap fault to be defined later.

The threshold H is obtained by minimizing the error probability assigned to the node of this threshold. For the accuracy of the overall classification, we note that the size of tolerable error probability at each node should be preassigned. So to obtain an optimal error probability of the whole classifier, we shall have the optimal assignment of error probability at each node of the tree.

□EXAMPLE 4.1

For a given combinational network, assume we have seven possible faults: F1,F2,F3,F4,F5,F6, and F7, with the means: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7, respectively. Assume that the threshold is 0.5 ; therefore , the probability density function of the normal distribution of the outputs can be sketched as shown in Figure 4.1 . □

When an unknown fault F_i with mean Z_i is to be classified, the decision can be used

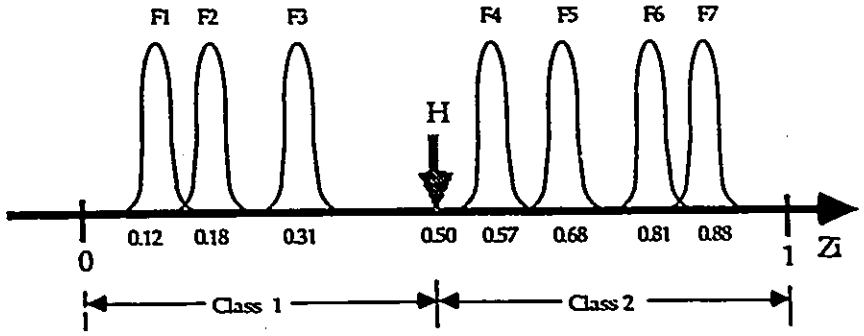


Figure 4.1: pdf of the 7 output faults of Example 4.1 (with H = 0.50).

as:

$$If\ Av\ F_i(whichisZ_i) < 0.5,\quad then\ F_i \in Class1,\quad else\ F_i \in Class2. \qquad (4.1)$$

The method of evaluating the threshold H of such a node is based on the theory of error probability of the node (the calculation will be explained later in the chapter) .

4.2.2 OVERLAP FAULTS

DEFINITION : A fault F_i is called an overlap fault if it cannot be well identified by a specified decision rule.

To be more specific, the definition can be explained by the following example :

□EXAMPLE 4.2

Assume that the threshold level in Example 4.1 is 0.57 rather than 0.50 ; then the pdf of the normal distribution is as shown in Figure 4.2 . The fault F4 is hard to be identified by this decision rule which means that this is an overlap fault. The decision process can then be represented by the tree structure in Figure 4.3 .

Since F4 cannot be well defined by the threshold H, overlap occurs in the next stage of classification. The more the overlap occurs, the more will be the number of stages in the decision tree; so, to find an optimal threshold in each stage is very important.□

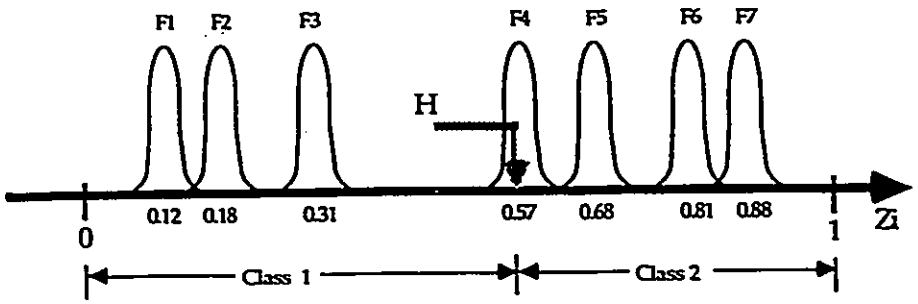


Figure 4.2: pdf of the 7 output faults of Example 4.1 (with H = 0.57).

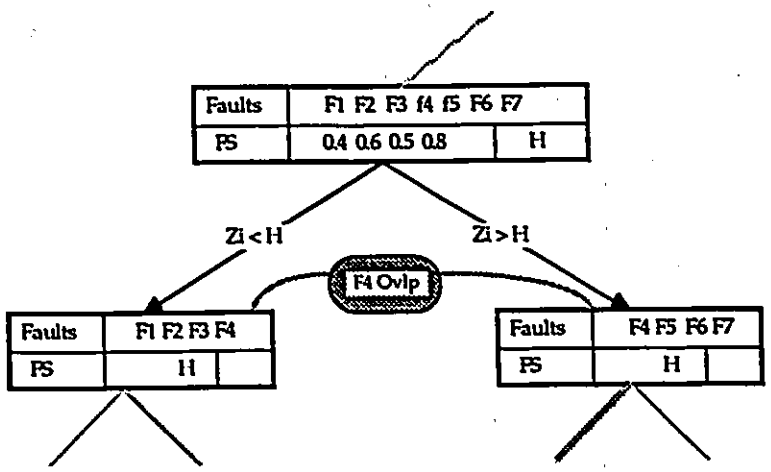


Figure 4.3: Tree representation in case of overlap fault .

4.3 THRESHOLD (H) CALCULATION

Associated with the decision rule, the probability of error is the most effective measure for the evaluation of the threshold H. In general, the calculation of error probability is very difficult. To illustrate the concept for the derivation of the optimal threshold, we treat the case of two fault classes first and then extend the result to the case of multiple faults.

4.3.1 CASE OF TWO FAULTS

Assume that we have two fault classes F1 and F2 with the information from the preceding Section 3.3.1 . The measurements have the probability density functions (recall from Eq.(3.20) to Eq.(3.23)) as :

$$f_{(Z_L(F_1))} = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(Z_L - Z_1)^2}{2\sigma_1^2}} \tag{4.2}$$

$$f_{(Z_L(F_2))} = \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(Z_L - Z_2)^2}{2\sigma_2^2}} \tag{4.3}$$

where the variances, respectively, are :

$$\sigma_1^2 = \frac{1}{L}(Z_1 - Z_1^2) \tag{4.4}$$

$$\sigma_2^2 = \frac{1}{L}(Z_2 - Z_2^2) \tag{4.5}$$

Assume also that H is the optimal threshold with a minimum testing error probability. Then the error probability can be computed as:

$$\varepsilon = P_1 \int_H^1 \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(Z_L - Z_1)^2}{2\sigma_1^2}} dz + P_2 \int_0^H \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(Z_L - Z_2)^2}{2\sigma_2^2}} dz. \tag{4.6}$$

By assuming $\gamma = \frac{(Z - Z_i)}{\sigma_i}$ we can write Eq.(4.6) as :

$$\varepsilon = P_1 \int_{\frac{H - Z_1}{\sigma_1}}^{\frac{1 - Z_1}{\sigma_1}} \frac{1}{\sqrt{2\pi}} e^{-\frac{\gamma^2}{2}} d\gamma + P_2 \int_{\frac{-Z_2}{\sigma_2}}^{\frac{H - Z_2}{\sigma_2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{\gamma^2}{2}} d\gamma \tag{4.7}$$

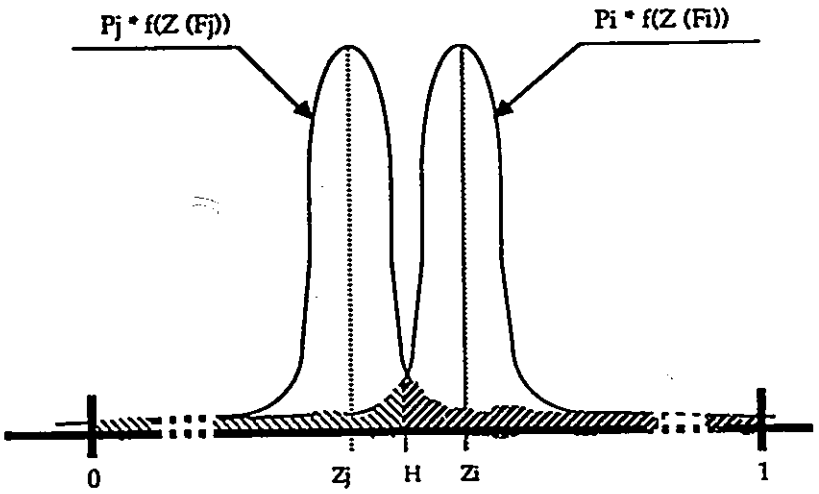


Figure 4.4: Figure 3.4 reproduced .

Substituting the values of σ_1 and σ_2 from Eq.(4.4) and Eq.(4.5) in Eq.(4.7) , we have :

$$\varepsilon = P_1 \int_A^B \frac{1}{\sqrt{2\pi}} e^{-\frac{\gamma^2}{2}} d\gamma + P_2 \int_C^D \frac{1}{\sqrt{2\pi}} e^{-\frac{\gamma^2}{2}} d\gamma \tag{4.8}$$

where :

$$A = \frac{(H-Z_1)}{\sqrt{(Z_1-Z_1^2)/L}}, \quad B = \frac{(1-Z_1)}{\sqrt{(Z_1-Z_1^2)/L}},$$
$$C = \frac{(-Z_2)}{\sqrt{(Z_2-Z_2^2)/L}}, \quad D = \frac{(H-Z_2)}{\sqrt{(Z_2-Z_2^2)/L}}, \text{ and } \quad \gamma = \frac{(Z-Z_i)}{\sqrt{(Z_i-Z_i^2)/L}}$$

To find an optimal value of H, we differentiate ε with respect to H and equate to zero. Then we find the value of H from this equation with the restriction that H will be within Z_1 and Z_2 .

Hence :

$$\frac{d\varepsilon}{dH} = \frac{-P_1}{\sqrt{2\pi}} e^{[-\frac{1}{2}\frac{(H-Z_1)^2}{(Z_1-Z_1^2)/L}]} \frac{\delta}{\delta H}\left[\frac{(H-Z_1)}{\sqrt{(Z_1-Z_1^2)/L}}\right] + \frac{P_2}{\sqrt{2\pi}} e^{[-\frac{1}{2}\frac{(H-Z_2)^2}{(Z_2-Z_2^2)/L}]} \frac{\delta}{\delta H}\left[\frac{(H-Z_2)}{\sqrt{(Z_2-Z_2^2)/L}}\right] \tag{4.9}$$

Now if we make : $\frac{d\varepsilon}{dH} = 0$, immediately we have :

$$P_1 \frac{1}{\sqrt{Z_1-Z_1^2}} e^{[-\frac{1}{2}\frac{(H-Z_1)^2}{(Z_1-Z_1^2)/L}]} = P_2 \frac{1}{\sqrt{Z_2-Z_2^2}} e^{[-\frac{1}{2}\frac{(H-Z_2)^2}{(Z_2-Z_2^2)/L}]} \tag{4.10}$$

Taking logarithms of both sides, we have:

$$\frac{1}{2}\frac{(H-Z_2)^2}{(Z_2-Z_2^2)}-\frac{1}{2}\frac{(H-Z_1)^2}{(Z_1-Z_1^2)}=\frac{1}{L}Ln(\frac{P_2}{P_1}\sqrt{\frac{Z_1-Z_1^2}{Z_1-Z_1^2}}). \tag{4.11}$$

To simplify calculation , let :

$$(i): Z_2-Z_2^2=a; \qquad (ii): Z_1-Z_1^2=b; \qquad (iii): Ln(\frac{P_2}{P_1}\sqrt{\frac{Z_1-Z_1^2}{Z_1-Z_1^2}})=c \tag{4.12}$$

Then Eq.(4.9) can be written as :

$$\frac{(H-Z_2)^2}{2a}+\frac{(H-Z_1)^2}{2b}=c. \tag{4.13}$$

or :

$$(a+b)H^2-2(bZ_2+aZ_1)H+(aZ_1^2+bZ_2^2-abc)=0. \tag{4.14}$$

Therefore, with the constraint $Z_1 < H < Z_2$, we can easily solve Eq.(4.14) to find the optimal threshold H .

In the limiting case, if we consider the length of the input sequence L to be very large (i.e. L$\rightarrow \infty$), Eq.(4.11) can be approximated as:

$$\frac{1}{2}\frac{(H-Z_2)^2}{(Z_2-Z_2^2)}-\frac{1}{2}\frac{(H-Z_1)^2}{(Z_1-Z_1^2)}=0. \tag{4.15}$$

which, with Eq.(4.12), becomes :

$$\frac{(H-Z_2)^2}{2a}-\frac{(H-Z_1)^2}{2b}=0. \tag{4.16}$$

Solving this equation, the optimal threshold H with a minimum testing error probability (in the case of two fault classes F1 and F2), becomes :

$$H=\frac{Z_2+Z_1\sqrt{\frac{Z_2-Z_2^2}{Z_1-Z_1^2}}}{1+\sqrt{\frac{Z_2-Z_2^2}{Z_1-Z_1^2}}} \tag{4.17}$$

From Eq.(4.17), we have obviously :

$$Z_1 < H < Z_2 \tag{4.18}$$

For any assigned input sequence length L, we can easily find an optimal threshold H by solving Eq.(4.11) .

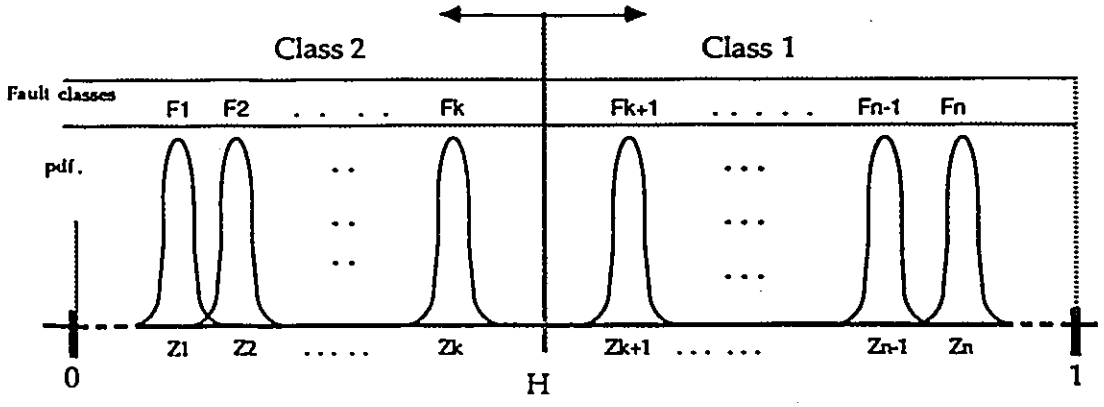


Figure 4.5: pdf for the n-fault case .

4.3.2 CASE OF n FAULTS

For the case of n fault classes ($n > 2$) , after grouping, we have two classes. Class 1 has k faults and Class 2 has (n-k) faults (Figure 4.5) .

The total error calculation can be done as:

$$\varepsilon_T = \sum_{i=1}^{k} \frac{P_i}{\sqrt{2\pi}} \int_{\frac{H-Z_i}{\sigma_i}}^{\frac{1-Z_i}{\sigma_i}} e^{-\frac{\gamma^2}{2}} d\gamma + \sum_{j=k+1}^{n} \frac{P_j}{\sqrt{2\pi}} \int_{\frac{-Z_j}{\sigma_j}}^{\frac{H-Z_j}{\sigma_j}} e^{-\frac{\gamma^2}{2}} d\gamma \tag{4.19}$$

where : σ_i and σ_j can be obtained from :

$$\sigma_\lambda = \sqrt{\frac{1}{L}(Z_\lambda - Z_\lambda^2)} \tag{4.20}$$

By putting $\frac{\delta \varepsilon_T}{\delta H} = 0$, we have :

$$\sum_{i=1}^{k} P_i \frac{1}{\sqrt{Z_i - Z_i^2}} e^{-\frac{1}{2}\frac{(H-Z_i)^2}{\sigma_i^2}} = \sum_{j=k+1}^{n} P_j \frac{1}{\sqrt{Z_j - Z_j^2}} e^{-\frac{1}{2}\frac{(H-Z_j)^2}{\sigma_j^2}} \tag{4.21}$$

Eq.(4.21) is not very easy to solve for large k and n, and no closed form can be obtained. But by using stochastic approximation or any such similar algorithms, an optimal H can always be obtained, if we denote:

$$(a) : x < H < y; \quad (b) : x = Max.|_{i=1}^{k} Z_i; \quad (b) : y = Min.|_{j=k+1}^{n} Z_j \tag{4.22}$$

So, the starting point of the algorithm can be reasonably selected so that it converges rapidly.

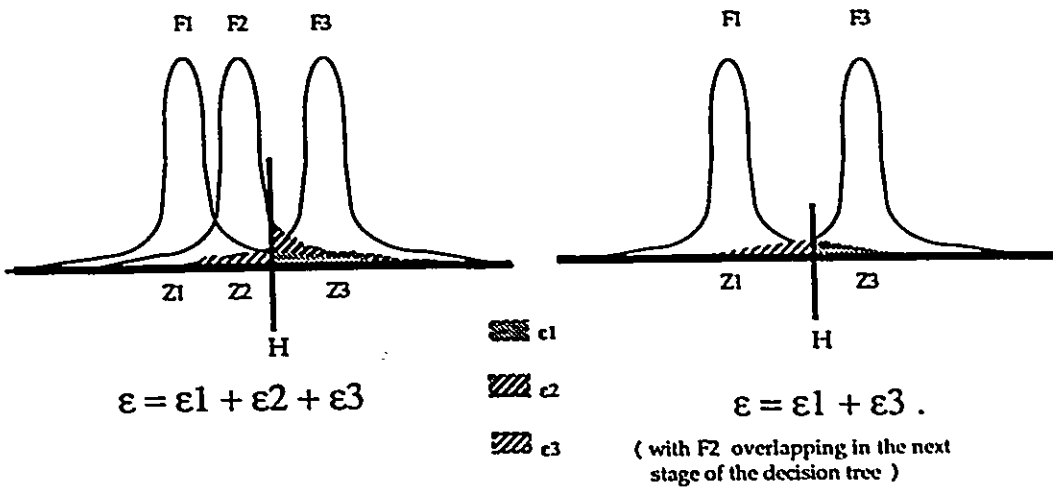


Figure 4.6: The effect of overlapping faults .

4.4 LARGE ERROR CALCULATION

Now, having the threshold H from Eq.(4.11), we can use Eq.(4.19) to calculate the probability of error ε_T .By the decision rule, if ε_L is higher than the preassigned error bound, the decision with the obtained threshold will not provide good performance . Therefore, we propose two methods for solving this problem.

The first method is called overlap solution method . By looking at the problem, we propose to take the fault which has the nearest mean to the threshold H out of consideration (nearest mean has the higher error probability), and to let it overlap in the next level of the tree; then we find the decision threshold again. This process can be repeated until the total error probability is less than the preassigned error. Figure 4.6 shows the effect of overlapping on the error probability.

The second method is based on the limiting case of Eq.(4.20). By increasing the input sequence length L, we can reduce the variances of faults. This will also reduce the total error probability.

By recalling Eq.(4.20), we have :

$$\sigma_i^2 = \frac{1}{L}(Z_i - Z_i^2) = f_{(Z_i,L)}. \tag{4.23}$$

Therfore , Eq.(4.19) becomes :

$$\epsilon_T = f_{(Z_i,Z_j,L)} \tag{4.24}$$

If $L \rightarrow \infty$, then $\epsilon_T \rightarrow 0$.

This two methods can be applied together to obtain a very low error probability .

4.5 GROUPING TECHNIQUE FOR FAULT CLASSES

Until this point, we are close to defining the algorithm for building the decision tree. The only problem left is to find a way out of spreading the faults at a particular node under consideration into two groups, with respect to the feature set selected at this node. Then Eqs.(4.19) and (4.21) become useful to find if the calculated total error probability satisfies the prespecified total error.

As illustration, let us consider the study of a particular tree node having P faults under consideration. Obviously, the possible combinations of classification in two groups of faults is:

$$Possible\ Combination(PC) = (2^{P-1} - 1) \quad possibilities. \tag{4.25}$$

It has been shown in Chapter 3 that if we arbitrarily spread the faults into two classes and find the feature which will satisfy the conditions of Eqs.(3.58), (3.59), and (3.60), the feature will be optimal for this grouping and this grouping can be used for the threshold calculation. But as already mentioned, the procedure for finding the feature satisfying conditions of Eqs.(3.58), (3.59), and (3.60) is quite involved. Now if we assume that we find the feature which will satisfy only the condition in Eq.(3.58), then any of

the situations as shown in Figure(3.4) might result. A need for a regrouping is hence required before the threshold calculation.

The grouping technique is rather easy and straightforward. After the arbitrary grouping of the faults into two groups, the optimal feature is obtained which will satisfy only the conditions in Eq.(3.58). Once it is done, an algorithm to select one good regrouping can be given as follows:

One way is that the fault classes are regrouped according to their average values (mean Z_i). The fault classes which will have their mean values less than 0.5 are grouped in one class and the other fault classes into the other class. Then, from Eq.(4.21), we can obtain the threshold H. Furthermore, from Eq.(4.19), we obtain the total error probability for this regrouping.

This regrouping will not be complete until the condition satisfies the preassigned error probability; otherwise, we find the fault class whose mean value is nearest to 0.5, let it overlap in the next stage of the tree, then recalculate the error probability, and check whether or not it satisfies the preassigned error probability . This is continued every time we overlap a fault until we succeed.

That this grouping procedure has an intuitive justification is evident from Eq.(4.23). Since the variance of the average of the outputs is related to the average value, the maximum variance will occur when the average value tends to move either towards zero or towards 1. The value of variance will be zero when the average value is either 0 or 1. Thus it is reasonable to group all the faults having average values less than 0.5 to one group and all others having the average values greater than 0.5 to the other group. (Note here that the faults having average values equal to 0.5 will be overlapped.) In this situation, the threshold value obtained will be near 0.5. Since the fault classes which has the average value near to 0.5 will have the most variance value, it is expected that they will introduce the most error; thus in the case of selecting overlap faults, these will

be selected first.

As far as the distribution of the average values of the outputs for all the fault classes to be grouped are well spread between 0 and 1, the above method may have good performance; otherwise, in the case where the average values are within less than 0.5 or greater than 0.5, then the following procedure may be used.

First, try another arbitrary grouping for the feature selection so that the average values of the outputs of the fault classes spread over the range 0 to 1. If with several trials this cannot be obtained, the following procedure is next used .

Assuming that the node under consideration has P faults, after several arbitrary groupings, let us denote Z_{min} to be the minimum average value and Z_{max} to be the maximum average value. Then the average of these limits is :

$$Z_{Av} = \frac{Z_{min} + Z_{max}}{2} \tag{4.26}$$

Therefore, among the fault classes, all the fault classes having the average output values less than Z_{Av} are to be grouped in Group 1, and the others in Group 2. However, to reduce the calculation probability, we use again the overlap technique, in which the nearest output average value to Z_{Av} causes the overlap fault. It is obvious that this guess about the overlap technique may not always be correct.

4.6 THE ALGORITHM FOR TREE CONSTRUCTION

So far we have been explaining the process of completing the tree building phase. Therefore, a grouping algorithm is needed to efficiently determine the best classification with respect to the feature set and preassigned error probability. In our notation, a grouping algorithm and a flowchart representation together make up a classification procedure which is the mechanism for solving the construction problem.

For a given n-input combinatorial network with the set of all possible faults, the following algorithm is used for the construction of the decision tree

ALGORITHM

Step 1: From the feature set algorithm in Chapter 3, select the desired feature vector :

$$[P^*_{(x_1)}, P^*_{(x_2)}, ..., P^*_{(x_n)}].$$

Step 2: Generate randomly the input, with respect to the desired feature vector. Then from the simulated network under study, observe the average of the outputs (the mean Z_i) . Calculate the variance with respect to the input sequence length L.

Step 3: Regroup the faults based on their averages with 0.5 level, or

with : $[\frac{(Z_{min}+Z_{max})}{2}]$ level.

Step 4: Obtain the threshold H using any method to solve Eq.(4.21).

With this threshold, calculate the total error probability ε_T from Eq.(4.19).

Step 5: The decision to be made here is :

Is the total error probability ε_T is less than the preassigned error probability ?

If yes , then go to Step 7 ;

else , go to Step 6 .

Step 6: Two possible solutions are :

(1) Take off the fault of nearest average to the threshold H (overlap problem),

or (2)increase the input sequence L to reduce the variances.

Then go to Step 4.

Step 7: Obtain two nodes for the tree with node 1 having the faults of Group 1, and node 2 having the faults of Group 2. Then observe if nodes have only one fault.

If yes , then stop and the decision finished ;

else , go to Step 1 .

The overall flowchart for the decision tree can be represented as in Figure 4.6 . The algorithm is simply the interactive application of a classification rule based on the total error probability criterion. It is important to note that at each node with an overlap fault the levels of the tree will increase, and create a need for more classification time.

The results from the overall study for the building phase of the decision tree can be illustrated in the following example.

□ EXAMPLE 4.3

Assume that a 4-input combinational network has a nine possible faults. After grouping (using the feature vector 0.4, 0.7, 0.8, 0.5, for example) we get these faults into 2 classes, Class 1 containing 5 faults: F1, F2, F3, F4, and F5, while Class 2 containing 4 faults : F6, F7, F8,and F9, as shown in Figure 4.6(a). H_1 is obtained as the optimal threshold and the decision stage is as in Figure 4.6(b).

If we find that the calculated total error probability ϵ_T by this decision rule is higher than the preassigned error probability, we should find the fault which has the higher error and let it overlap in the following stage of the tree. Suppose, in our example , the fault F5 yields the maximum error probability. So F5 is taken out of consideration and the decision threshold is found as H_2. Figure 4.9 illustrates the solution in the case of

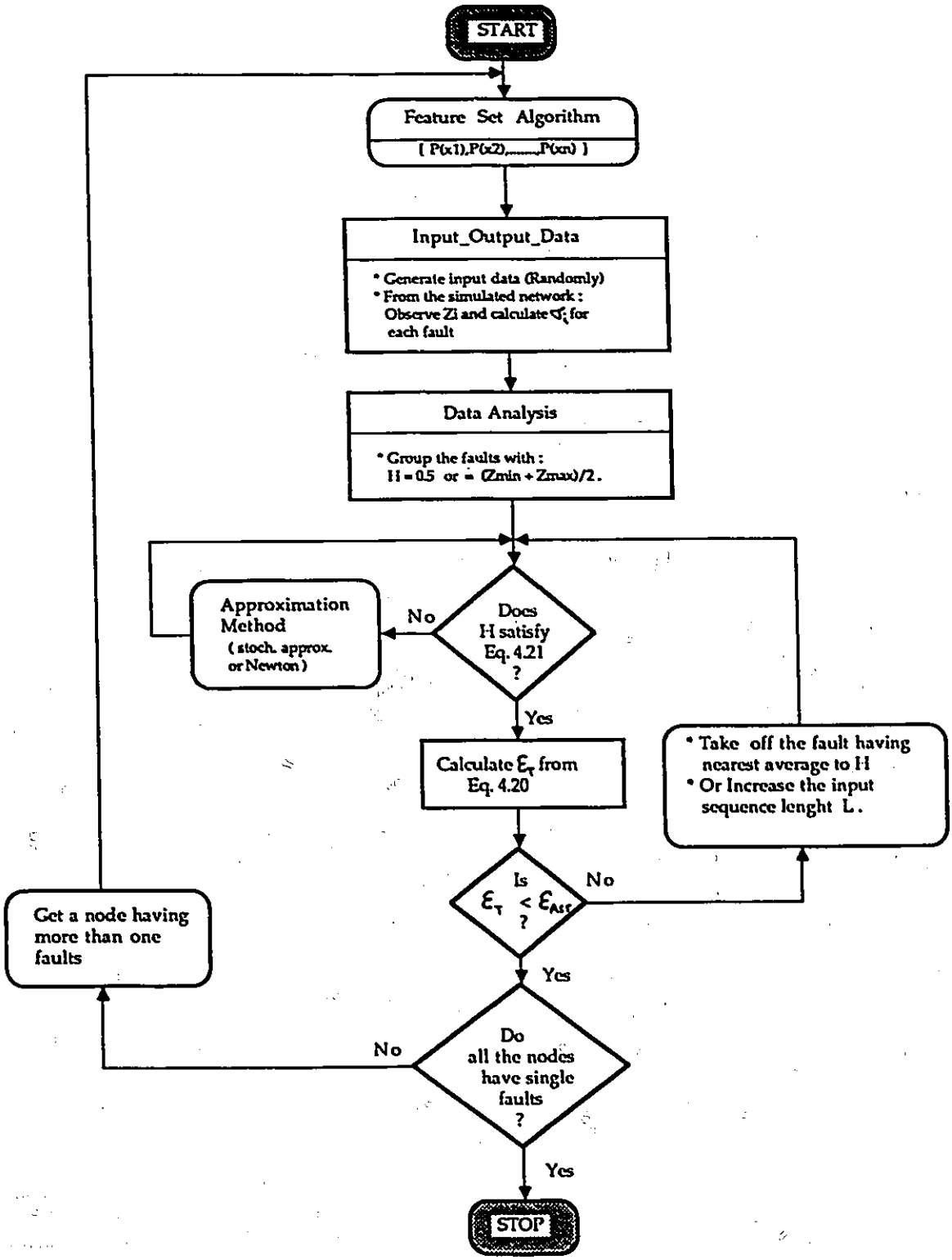
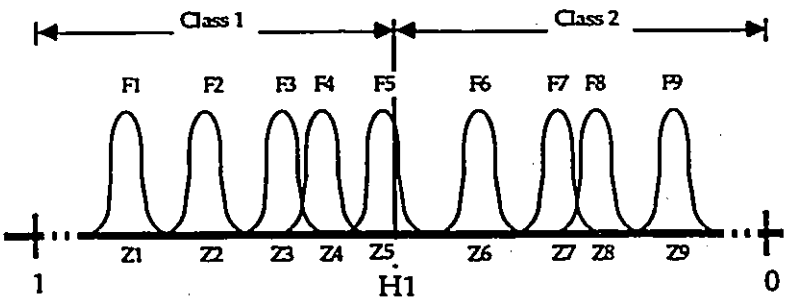
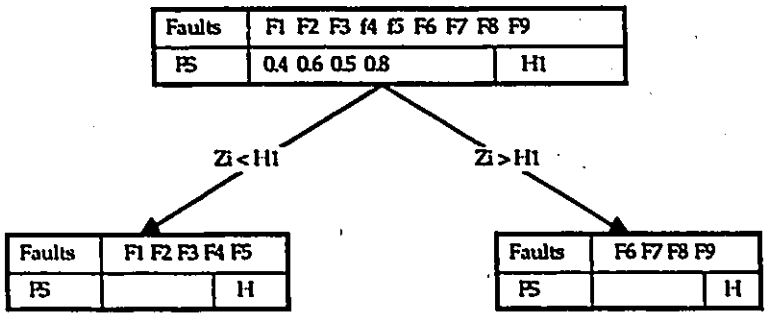


Figure 4.7: The flowchart for the decision tree .

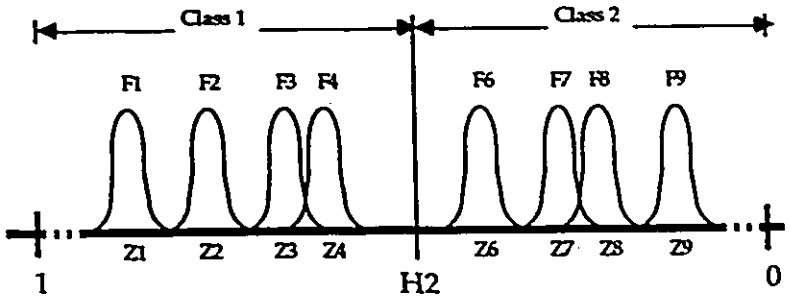


(a)

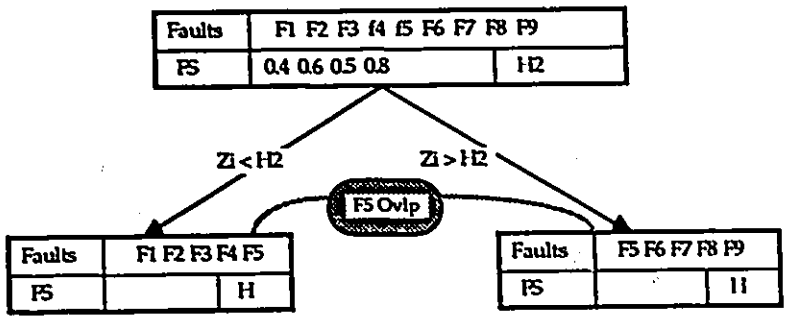


(b)

Figure 4.8: (a): pdf representation, and (b): decision tree representation for Example 4.3 .



(a)



(b)

Figure 4.9: (a):pdf representation, and (b): decision tree representation, with fault F5 out of consideration(F5 is overlapped) .

large error probabilities .

If the error as calculated is lower than the error bound, we build up the tree . Thus we create two nodes and then we check these nodes if they have one fault each . If the answer is yes, we stop; if not, we apply the above procedures of the algorithm to the node which needs further expansion .

Figure 4.10 shows the decision tree generated for our Example .

Note that the balancing of the tree reduce the average classification time. To illustrate this fact, let us consider the previous example with 9 faults. If in the first

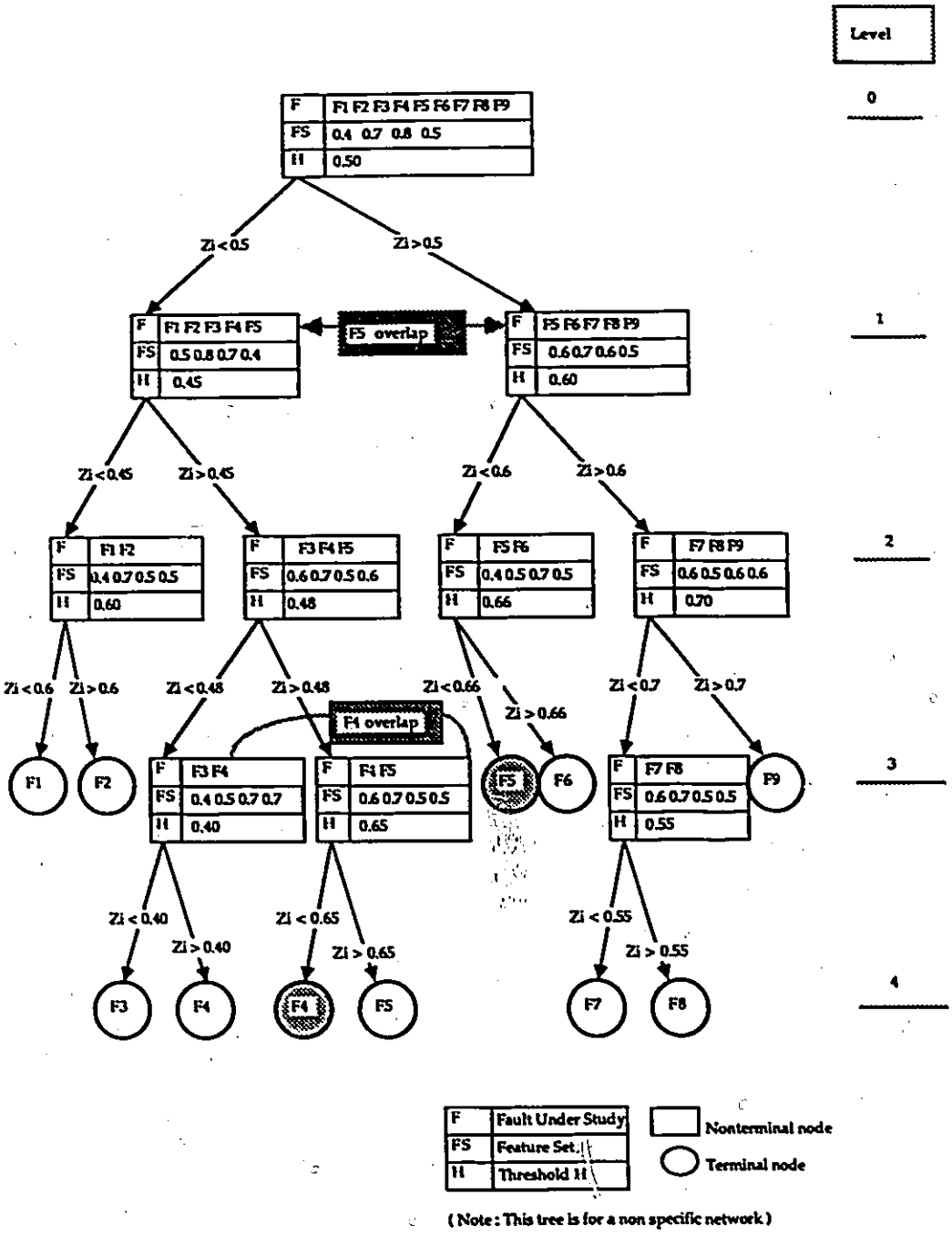


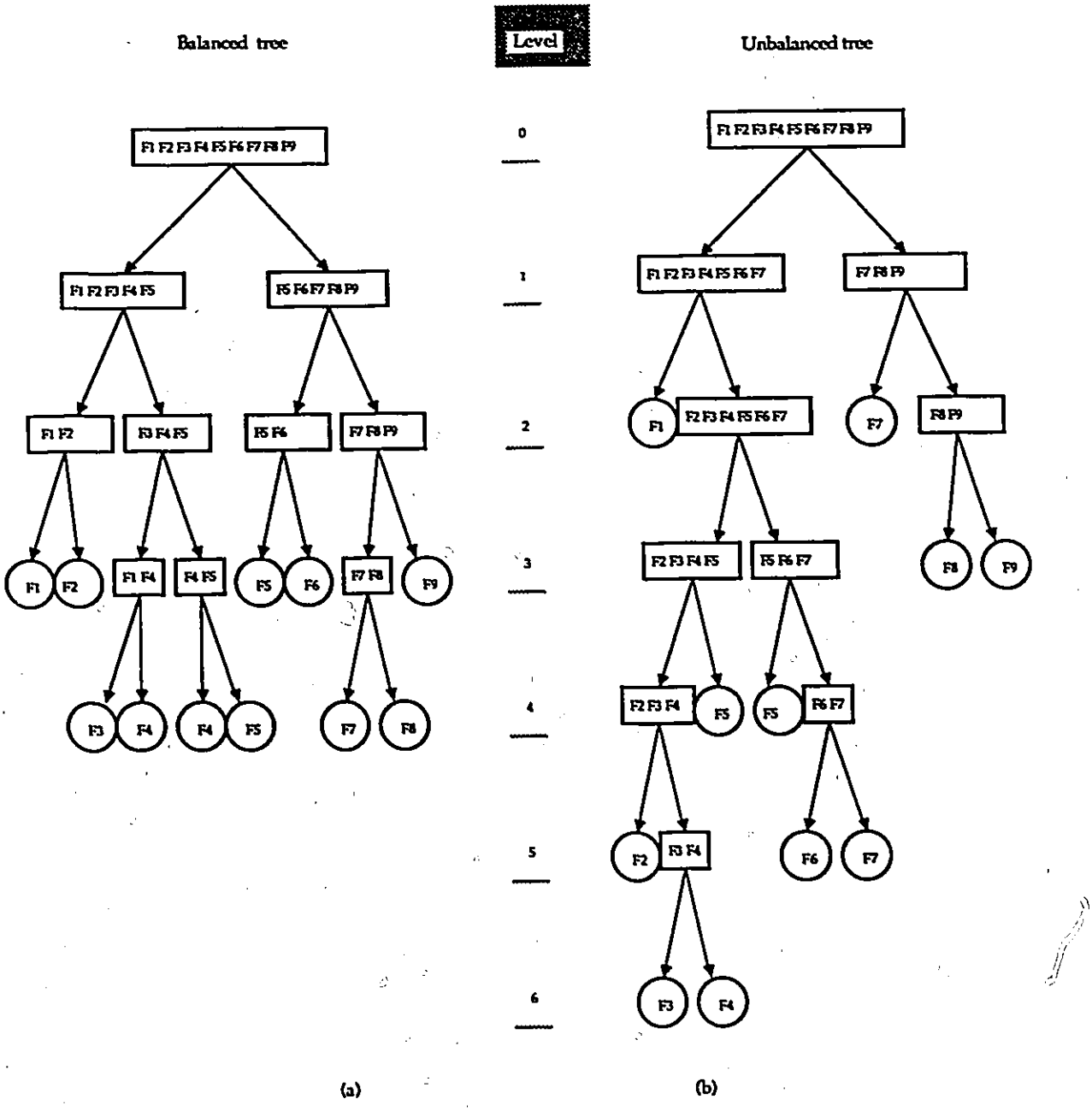
Figure 4.10: The overall decision tree representation for Example 4.3 .

classification , we had Class 1 with seven faults rather than five faults, it needs at least three levels for the classification to be completed with balance and without overlap. However , from Figure 4.9 , it is easy to see that for detecting fault F9 we need six classification levels with unbalanced tree (case b) rather than 4 levels with the balanced tree (case a). □

Once the decision tree is obtained from the building phase, the testing procedure which is the search phase can be implemented as explained in Chapter 2. If the tree is well constructed (balanced and less overlapping), we can find the faults very quickly, because the needed number of testing stages is almost proportional to :

$$No\ of\ Testing\ Level(NTL) \approx \log_2 n = \frac{\log_{10} n}{\log_{10} 2} \tag{4.27}$$

where n is number of possible faults of the network under test .



(a) (b)

Figure 4.11: (a): Illustration of balanced tree, and (b): illustration of unbalanced tree .

Chapter 5

STRUCTURE AND IMPLEMENTATION OF THE TREE CONSTRUCTION PHASE

5.1 INTRODUCTION

In this chapter we focus our attention on the structure and implementation of the tree construction phase - tree classifier - which has been developed in the preceding chapter. We now develop algorithms and provide flowcharts for the implementation of this structure and show how easily they can be realized in software.

For our purpose it is convenient to split the discussion of our implementation into two major categories which correspond closely to the major aspects of the programming structure .

1 - Algorithm :The way of defining the action of an operation. It specifies how to complete the solution for any given problem.

2 - Flowchart :The interface between the programming language and the algorithm of the solution method. In another words it is the graphical representation of a proposed solution method.

Figure 5.1 shows some of the standard symbols.

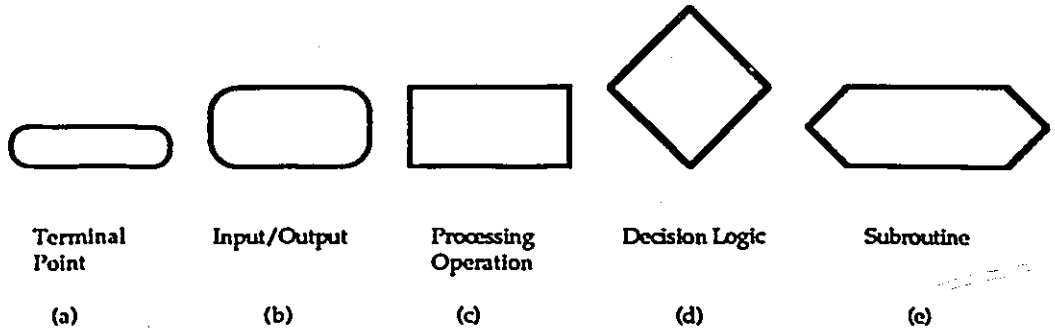


Figure 5.1: Some standard flowchart symbols .

5.2 THE OVERALL PERFORMANCE

The overall performance will have three major stages :

STAGE 1 : Input data - Input data from and for the circuit under study -.

STAGE 2 : Data analyses - Classification of faults - .

STAGE 3 : Output data - Inserting data in tree - .

Figure 5.2 illustrates the overall arrangement of those stages. A set of procedures or subroutines forms the structure of each of them . These procedures are sufficient to define the programming logic (Appendix A).

Note that we used PASCAL as high level programming language in the implementation because of its flexibility in realizing tree structures [2].

We are mainly concerned in the following sections about the algorithm and the flowchart of each step.

5.3 PROCEDURES OF LEVEL 1

Level 1 is a set of starting procedures; it consists of input data and the data analysis procedure (Const-Tree).

In the input data, we have to set : the error permissible for our design (Error-Ass), the input sequence length (ISL), and the number of possible faults of the circuit under study (Fault).

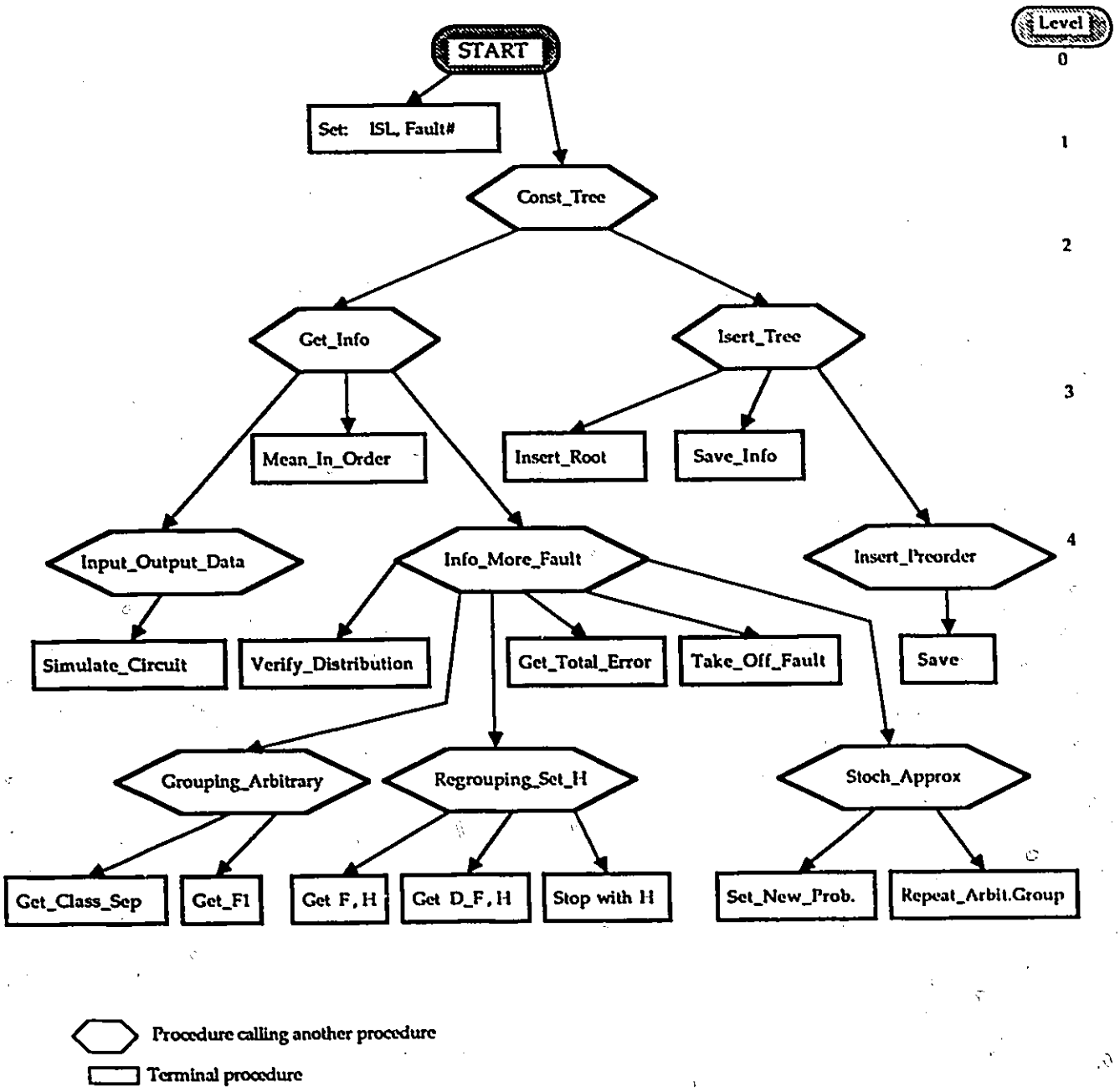


Figure 5.2: The overall diagram of the tree building phase .

Const-Tree deals with the data analysis for constructing the tree or in other words, classifying the faults. This procedure can be implemented by the following algorithm. Note here that each step starting with a "□" means that it is defined earlier as an algorithm, and with a "•" means it could be done inside the procedure itself.

CONST-TREE ALGORITHM .

1 • Assign a reference number for each fault (Fi) (Figure 5.3).

2 □ Get the information for the classification aspect .

3 • Assign the root node of the tree .

4 • Insert the information in the root node .

5 • Save the information in output file .

6 □ Insert in the tree (Insert-Preorder) means that the first analyses should be in the root, then left node, then right node .

For example, the seven nodes are constructed recursively in the order 1,2,....,7 as shown in Figure 5.5 . Figure 5.4 shows the structure of the nonterminal and terminal nodes; refer to the pdf distribution of the faults . Finally, Figure 5.6 illustrates the flowchart of the previous algorithm .

5.4 PROCEDURES OF LEVEL 2

This level has two procedures :

(1) Getting the information that is used for classification (Get-Info).

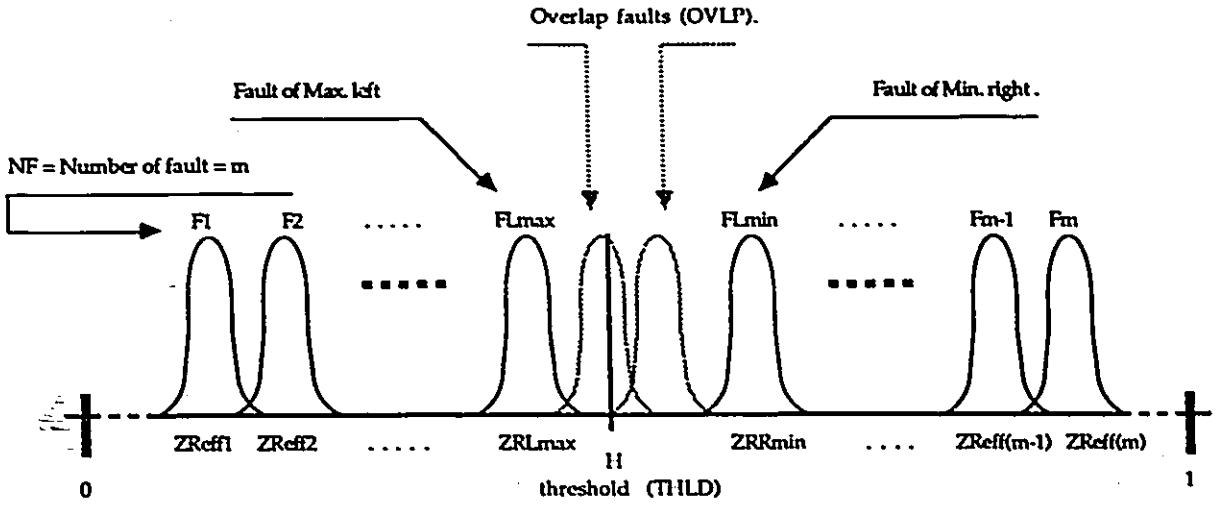
(2) Inserting preorder in the tree (Insert-Preorder) .

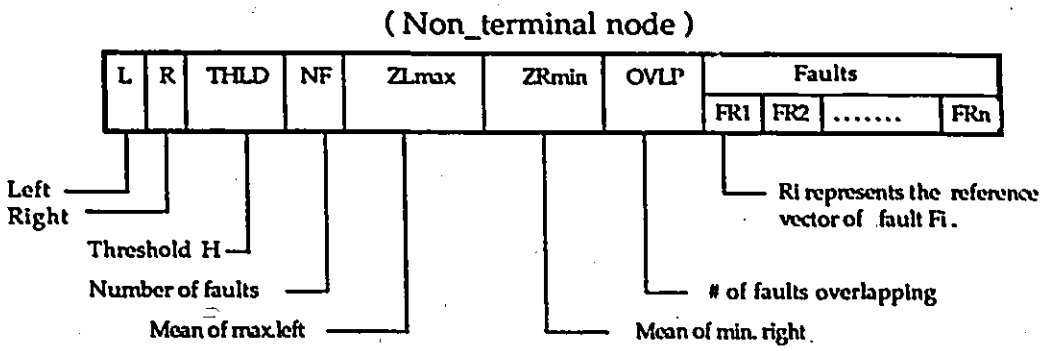
To be more precise, we are going to develop the structure of each one of the above .

5.4.1 GET-INFORMATION (Get-Info).

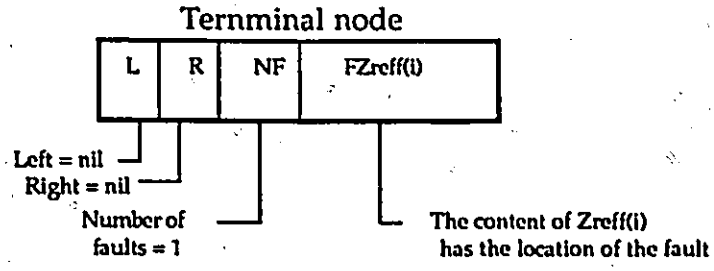
The steps forming this procedure will be as follows :

1 • Select arbitrary feature set $[P_0(x_1), P_0(x_2), ..., P_0(x_N)]$.





(a)



(b)

Figure 5.3: Reference names for computing purpose .

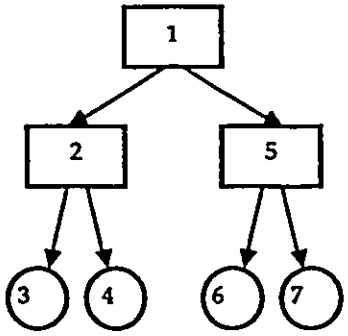


Figure 5.4: Preorder construction of the tree .

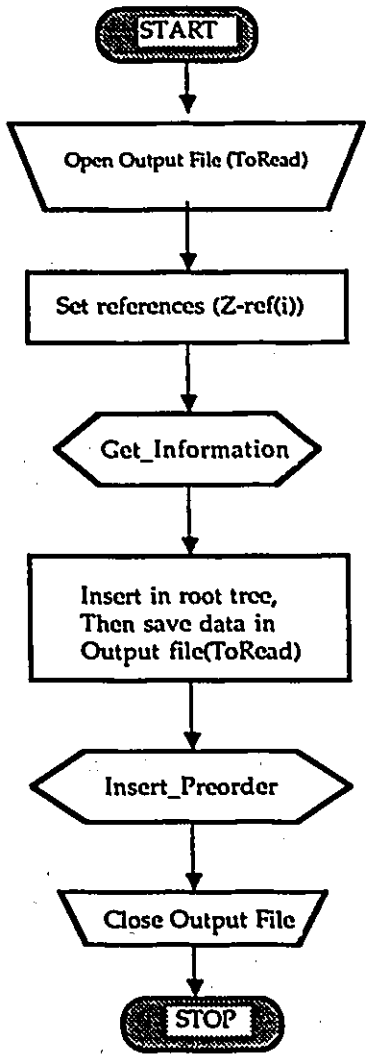


Figure 5.5: The 1st level procedure .

In our case we assign the same probability to all the inputs as [0.5,0.5,....,0.5] .

2 □ Generate input data, then observe the average output data (Input-Output-Data),

X-in[I] as an input, Mean[I] as an average output .

3 □ Select the average output, Mean[I] in order from lower to higher (Mean-in-Order) .

4 • Verify the condition of not having a limit problem with Mean[of first fault] = 0

or mean[of last fault] = 1. If we have one of these two conditions,

then go to Step 2 ;

else go to Step 5 .

5 • Get the information for a node of more than one fault.

The flowchart of forming the (Get-Info) procedure is given in Figure 5.7 .

5.4.2 INSERT IN THE TREE

This procedure is to save the data in tree form that will be needed for the testing phase.

It consists of the following phases :

1 • Insert information in the root node .

2 • Save the information in output file.

What we need to save for the root node is :

The input sequence length (ISL).

The number of faults (NF).

The order of faults by their reference numbers (Z-Ref).

The threshold level (H) for decision making.

This result can appear as :

10,000 ⟸ Input Sequence Length

6 ⟸ Number of faults

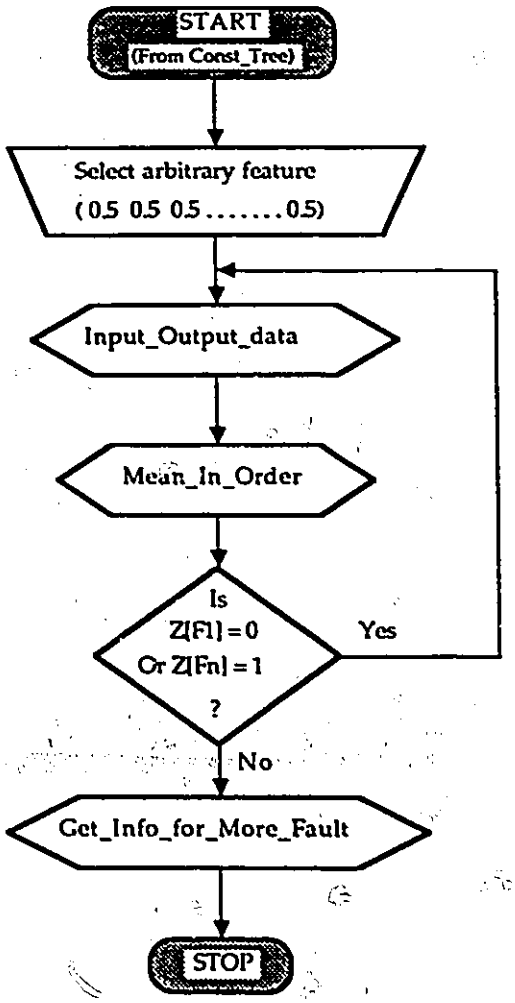
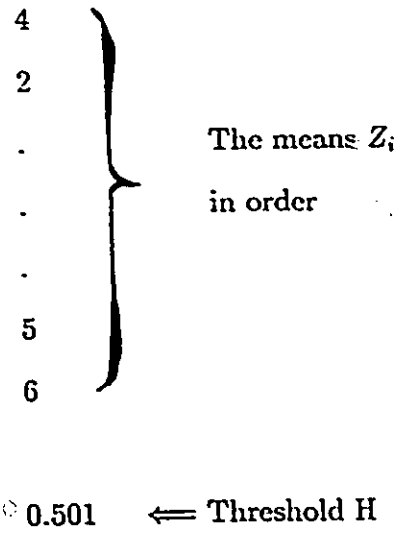


Figure 5.6: Flowchart for Get-Info procedure .



3□ Complete the construction of the tree .

The flowchart for this process is shown in Figure 5.8 .

5.5 PROCEDURES OF LEVEL 3

Two sets of procedures for forming this level are given below .

SET 1 consists of the following procedures :

(a) □ Generating input data and obtaining output results (Input-Output-Data) .

(b) • Arranging the output averages (Mean-In-Order) .

(c) □ Data analysis for the information of more faults (Info-More-Fault) .

SET 2 consists of the following procedures :

(a) • Saving the data by inserting in the root (Save-Info) .

(b) □ Inserting the rest of the node in preorder form (Insert-Preorder) .

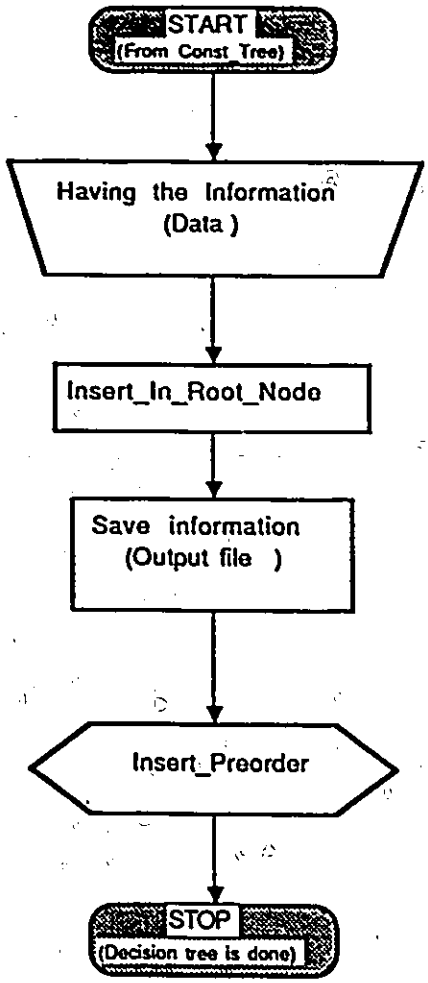


Figure 5.7: Flowchart used for Const-Tree procedure .

5.5.1 INPUT-OUTPUT DATA

This procedure can be realized in the following steps :

1 • Assign a reference for each output.

2 • Assuming that we have the input probability vector, generate the input data by using one of the following two methods :

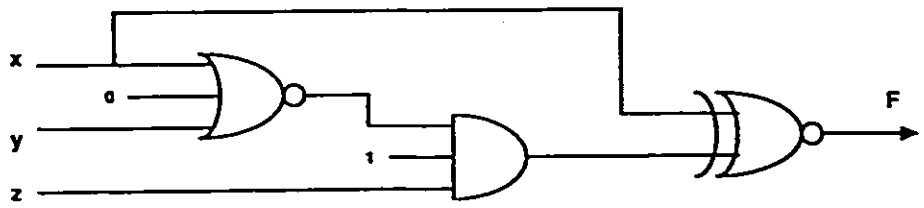
- Standard PASCAL random generator function , or
- procedure called Coding-Decoding input data (as explained later on).

3 • Simulate the network under consideration (Figure 5.10).

Our simulation consists of a set of functions as shown in Table 5.1.

Table 5.1 Simulation of Logic Gates .

Gate	Condition for output F
NORG	If x+y+z = 0, then F = 1 ; else F = 0
ORG	If x+y+z = 0, then F = 0 ; else F = 1
ANDG	If x+y+z = 3, then F = 1 ; else F = 0
NANDG	If x+y+z = 0, then F = 1 ; else F = 0
NOTG	If x = 1, then F = 0 ; else F = 1
EX-ORG	If x = y, then F = 0 ; else F = 1
EX-NORG	If x = y, then F = 1 ; else F = 0



F = EX-NORG(x,ANDG(NORG(x,0,y),1,z)).

Figure 5.8: Example with the simulation functions .

4 • Repeat Steps 2 and 3 until the preassigned input length ISL is completed.

5 • Obtain the output average Z_i .

The flowchart of the above algorithm is presented as in Figure 5.9.

Because of the constraint imposed by storage location that we need to use to save (ISL * N-input) line matrix of logic representation, we decided to use coding and decoding of data which consists of representing each decimal number in 2 bits length rather than 16 bits.

But this method requires tremendous amount of time for generating, coding, and decoding data length (10,000 points for example). Therefore, in our approach we used the standard random generator in PASCAL to eliminate the time problem. But we have to keep in mind that the sequences of numbers generated by a computer are deterministic, each number other than the first depends on the number that precedes it. However, this is sufficient for our purpose . Therefore, from our experimental results, if we assign a 0.60 probability for the input X_i for length of 10,000 , we get approximately 5500 to 6500 1's out of 10,000 .

The preceding result explains that our input data will be generated with ±5% error

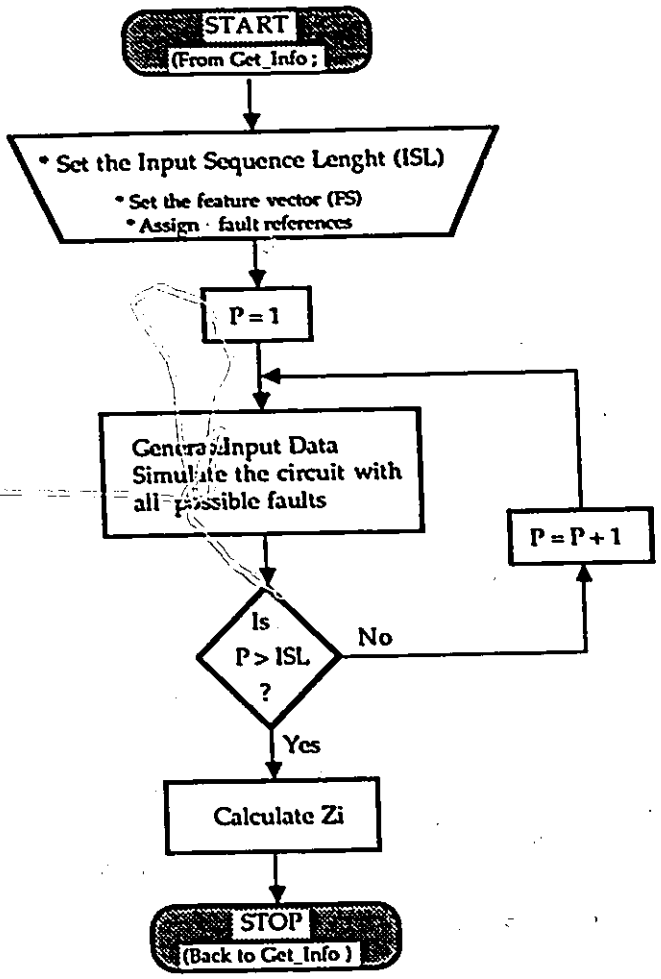


Figure 5.9: Input-output data flowchart.

in faster time (as compared to the time of first method) .

The logic representative (PASCAL) of this method is as follows :

If random(ISL) $\leq$ Abs ($P(x_i)$ * (ISL))

then input $X_i = 1$;

else input $X_i = 0$.

The random (ISL) is obtained from the following equation:

$$N_{i+1} = (a * N_i + C) \text{ Mode m}$$

with the constraints that N, a,and C are ≥ 0 ; and $m > N_0$, a, and C ; N_i and N_{i+1} are the previous and the next numbers, respectively ; a, c, and m are arbitrary numbers. Modules (m) determines the range of the random numbers which should be fairly large.

5.5.2 MEAN-IN-ORDER

From previous explanations, we have a reference number for each fault. Assume the following example with 6 faults:

i =	1	2	3	4	5	6
mean Z[i]	Z[1]	Z[2]	Z[3]	Z[4]	Z[5]	Z[6]
Z-Ref[i]	1	2	3	4	5	6
Value	0.5	0.S	0.1	0.5	0.9	0.3

To select those means in order form, we should follow Steps 1 to 7 as :

1 . Save Z[i] and its reference i .

2 . Verify if $Z[i] > Z[i+1]$, then switch means and references as :

Z[i] becomes Z[i+1] ;

Z[i+1] becomes Z[i] ;

Z-Ref[i] becomes Z-Ref[i+1] ;

Z-Ref[i] becomes Z-Ref[i+1] ;

3 . Repeat Step 2 (m-1) times where m = number of faults = 6 in this example .

4 . Repeat Step 3 (m) times.

5. Calculate the variance for the ordered means .

The result after this arrangement for the same example is :

i =	1	2	3	4	5	6
mean Z[i]	Z[1]	Z[2]	Z[3]	Z[4]	Z[5]	Z[6]
Z-Ref[i]	3	6	1	4	2	5
Value	0.1	0.3	0.5	0.5	0.8	0.9

For example, if we have i=2, then Z-Ref[2] = 6 means that we have 0.3 as value of mean Z[2], which refer to fault number 6 from the original information.

Graphically, the flowchart of this procedure can be illustrated as in Figure 5.12 .

5.5.3 INFO-MORE-FAULTS

We now present the most important procedure for constructing the tree . This procedure consists of the following steps :

1 • Verify if the distribution of $Z_i's$ covers 0 to 1 .

2 □ Group the faults arbitrarily into two groups.

3 □ Regroup the faults and obtain the threshold level H .

4 • Calculate the total error probability ϵ .

5 • Verify the condition :

If error calculated > error assigned.

then go to Step 6 ;

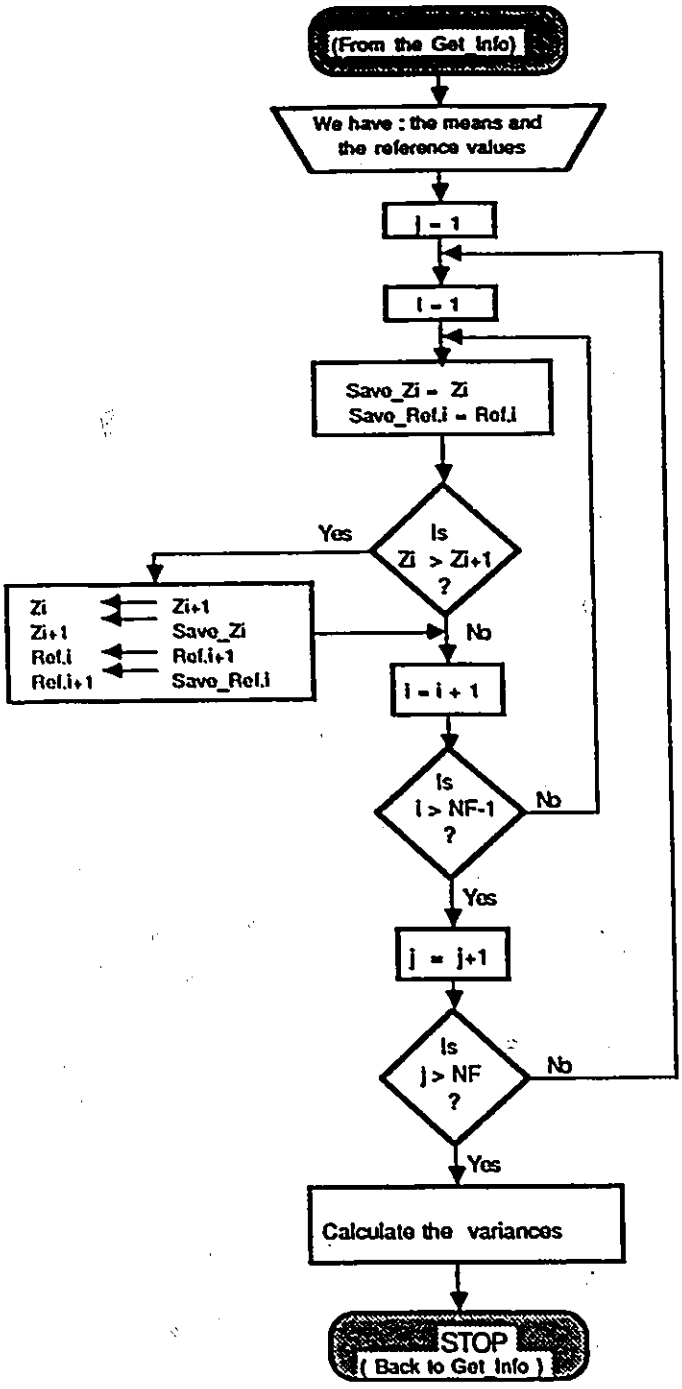


Figure 5.10: The flowchart for Mean-In-Order.

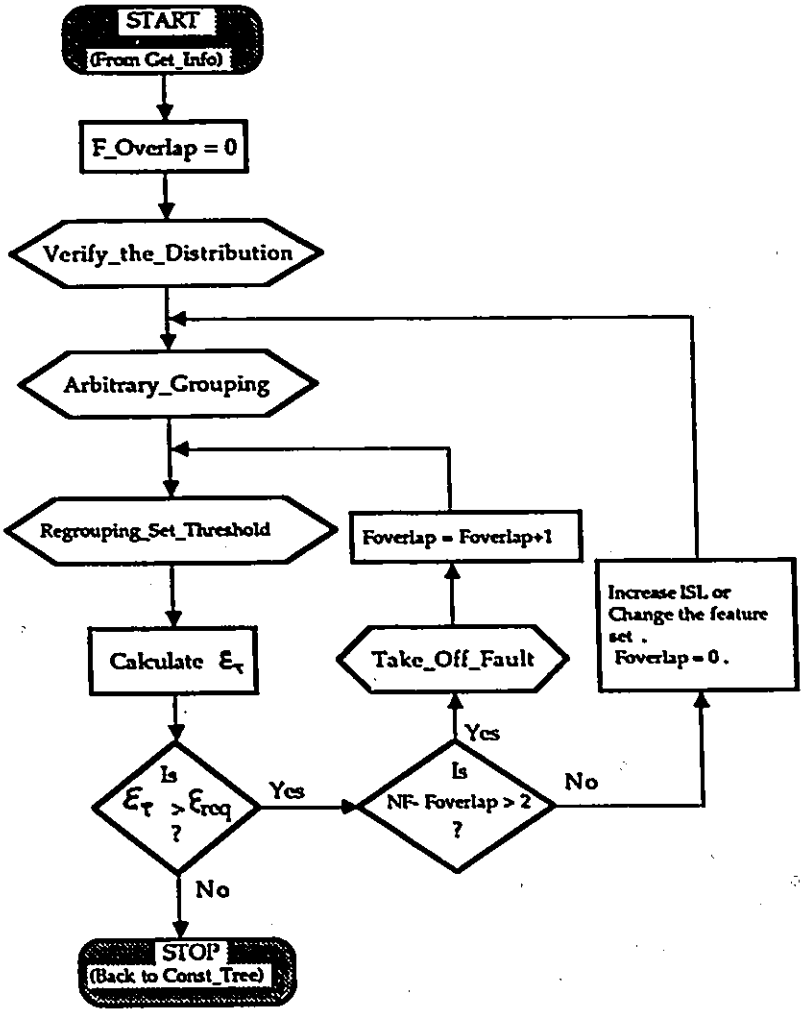


Figure 5.11: Flowchart for the procedure : Info-More-Fault.

else go to Const-tree .

6 □ Take off faults (overlap) if fault overlap > 2, or increase the input sequence length, or change input probability vector if :

Number of faults - Number of overlap faults = 2.

7 • Repeat from Step 3.

Figure 5.12 represents graphically the flowchart of the processes .

5.5.4 INSERT IN THE TREE (Insert-Preorder)

This is done recursively in order of father node first, then left child, then right child .

The algorithm is given as follows :

1 • Get the set of faults under study which is the class of the root node (from Section 5.3 : Algorithm : Step 6). For starting, X is the class one of the root .

2 • Check if the number of faults is greater than one. If yes, then go to Step 4 ; else go to Step 3.

3 • Insert the information in terminal node and save the data in the output file.

4 □ Get the input and output information (Get-Info) as in the previous section.

5 • Insert the information in a new left node, then save it in the output file. Then this new node becomes the root for Step 1. Start again with X = 1.

6 • Insert the information in new right node, then save it in the output file. Then this new right node becomes the root for Step 1. Start again with X = 2.

This process will continue until all the terminal nodes have each only one fault.

In graphical form the flowchart becomes as in Figure 5.12 .

5.6 PROCEDURES OF LEVEL 4

We now present the set of procedures of level 4 as :

(a) i- Group the fault arbitrarily into two classes.

ii- Regroup by setting the threshold level H.

(b) Take off the closest fault to the threshold or increase ISL.

(c) Use stochastic approximation to obtain the desired feature set .

5.6.1 GROUPING ARBITRARY

Grouping the faults arbitrarily can be refined in the following algorithm .

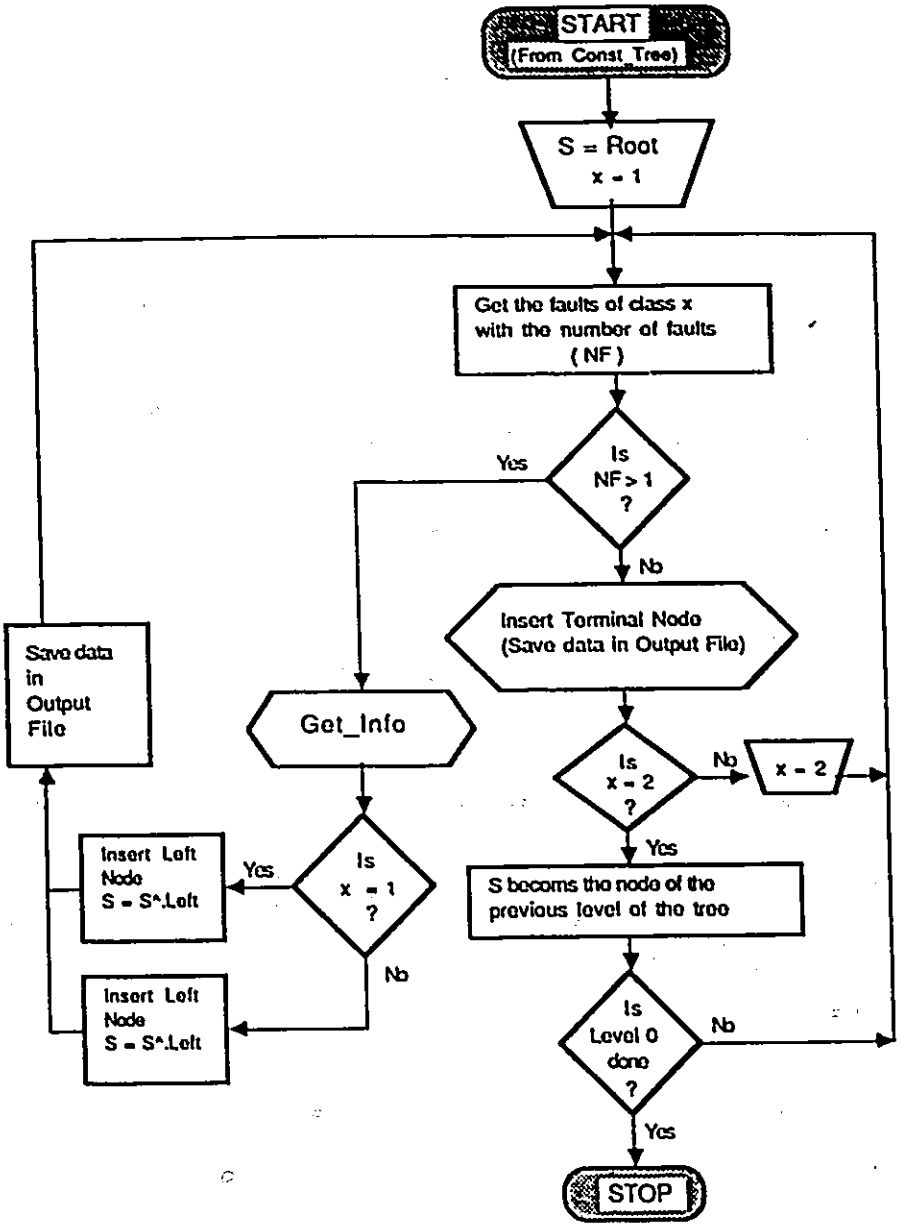


Figure 5.12: Flowchart for inserting preorder in binary tree.

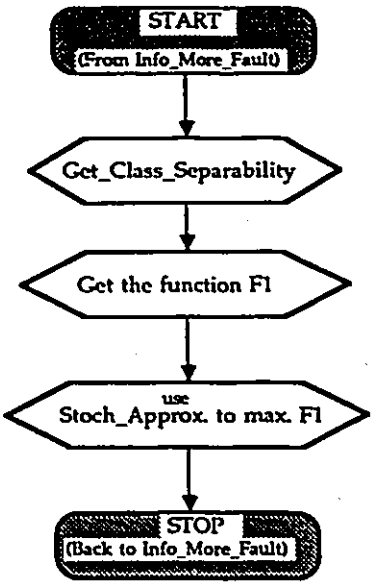


Figure 5.13: Flowchart for arbitrary-grouping procedure .

1 • Get the class separability by setting the threshold H to 0.5

or to $(Z_{max} + Z_{min})/2$, and finding the group classes.

2 • Calculate F^1 .

3 □ Use stochastic approximation method to minimize F^1 .

5.6.2 REGROUPING AND SET-THRESHOLD H

The algorithm can be illustrated as:

1 • Calculate Eq.(4.21) with H from the arbitrary grouping .

2 • Does H satisfy Eq.(4.21) ? : If yes, stop ; if not, go to Step 3.

3 • Calculate the derivative of Eq.(4.21) as :

By recalling Eq.(4.21), we can form the H function as :

$$f(H) = \sum_{i=1}^{k} P_i \frac{1}{\sqrt{Z_i - Z_i^2}} e^{-\frac{1}{2}\frac{(H-Z_i)^2}{\sigma_i^2}} - \sum_{j=k+1}^{n} P_j \frac{1}{\sqrt{Z_j - Z_j^2}} e^{-\frac{1}{2}\frac{(H-Z_j)^2}{\sigma_j^2}} \qquad (5.1)$$

(See Eq.(4.21) in Chapter 4 for details)

Then assuming :

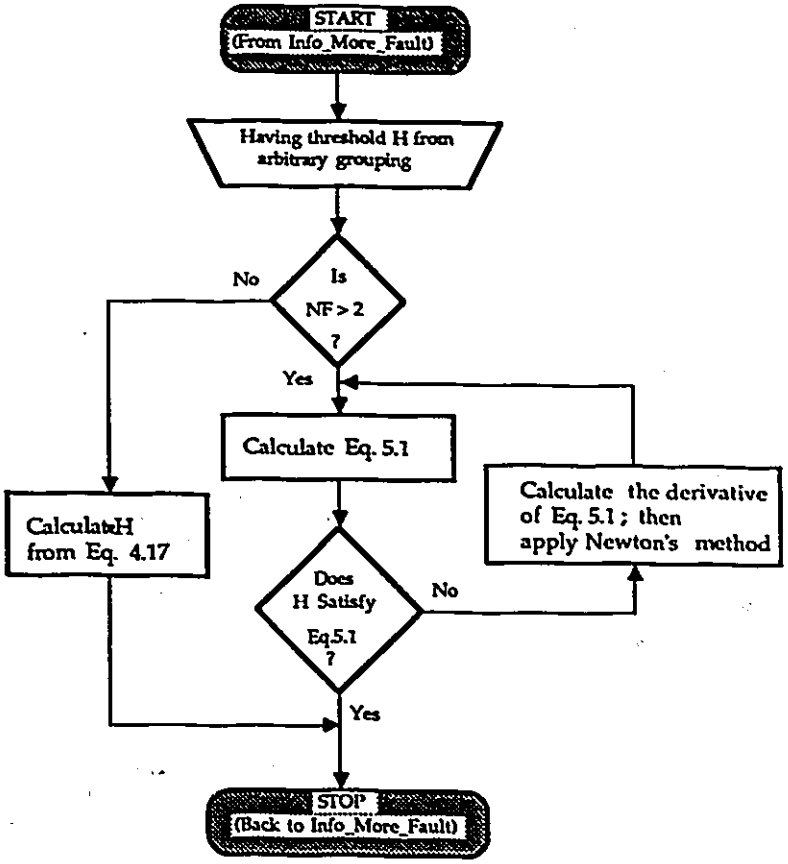


Figure 5.14: Flowchart for setting threshold H .

$$A = \frac{P_i}{\sqrt{Z_i - Z_i^2}}\ , \qquad B = \frac{P_i}{\sqrt{Z_j - Z_j^2}}\ ,$$
$$C = \frac{L}{2(Z_i - Z_i^2)}\ , \text{ and} \qquad D = \frac{L}{2(Z_j - Z_j^2)}\ ,$$

the derivative of f(H) becomes :

$$f'(H) = -\sum_{i=1}^{k} 2AC(H - Z_i)e^{-C(H-Z_i)^2} + \sum_{j=k+1}^{n} 2BD(H - Z_j)e^{-D(H-Z_j)^2} \qquad (5.2)$$

4 • Apply Newton's method to find a new threshold H by using its standard equation as :

$$H_{i+1} = H_i - \frac{f(H_i)}{f\prime(H_i)} \qquad (5.3)$$

5 • Repeat from Step 2.

5.6.3 STOCHASTIC APPROXIMATION

Only one procedure at this level which called stochastic approximation method is used to select the feature set vector. The algorithm can be given as:

1 □ Set a new probability for the feature by using Kiefer and Wolfowitz method.

2 • Generate input data and obtain output averages (Z's) .

3 • Group the means in ordered form .

4 • Calculate F^1_{i+1} .

5 • Verify if F^1_{i+1} is less then F^1_i (the previous function) :

If yes, repeat from Step 1 ;

if not, save the feature set of F^1_i .

5.7 EXAMPLE AND APPLICATION .

An example of application of the tree classifier method can be provided by building a binary tree to locate faults in the ALU (Arithmetic Logic Unit) SN54LS181 (see Figure 5.15). We illustrate this application for construction by computer simulation in PASCAL on an IBM-PC .

Because of storage limitation, consider only 7 primary input lines X_i where i = 1, 2, 3,...,7 , with a total of 22 lines having either stuck-at-one or stuck-at-zero types of faults. From those 22 lines we have 28 different output functions as tabulated in Table 5.2 .

TABLE 5.2 : 28 Different Output Functions with faults s-a-0/1 .

Fault[i]	Line #,stuck-at-0/1	Fault[i]	Line #,stuck-at-0/1
1	(1,0); (10,0)	15	(8,0)
2	(1,1)	16	(8,1)
3	(2,0); (11,0)	17	(9,0); (15,0)

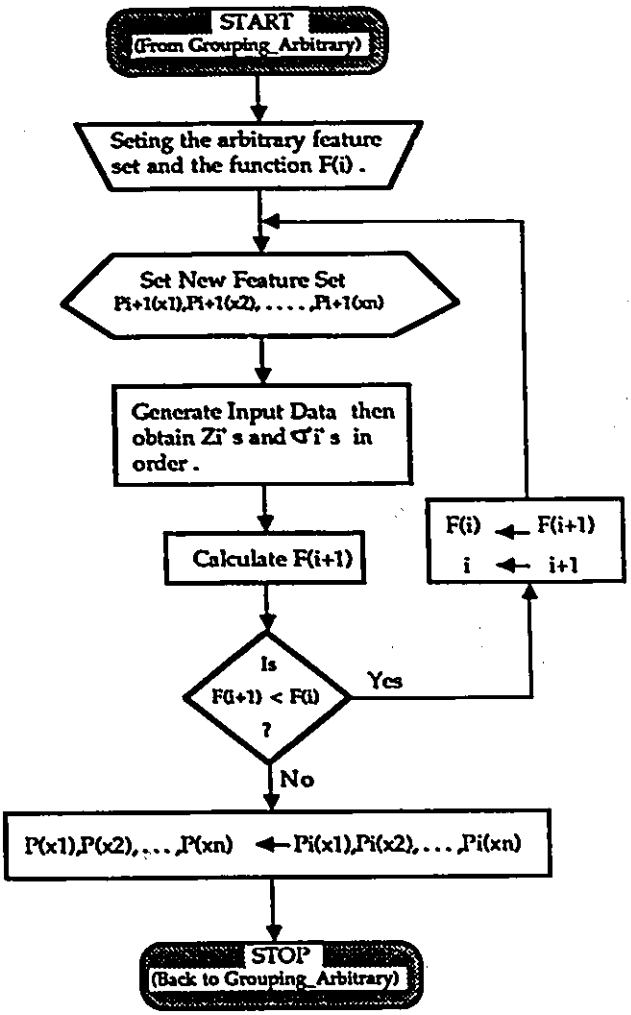
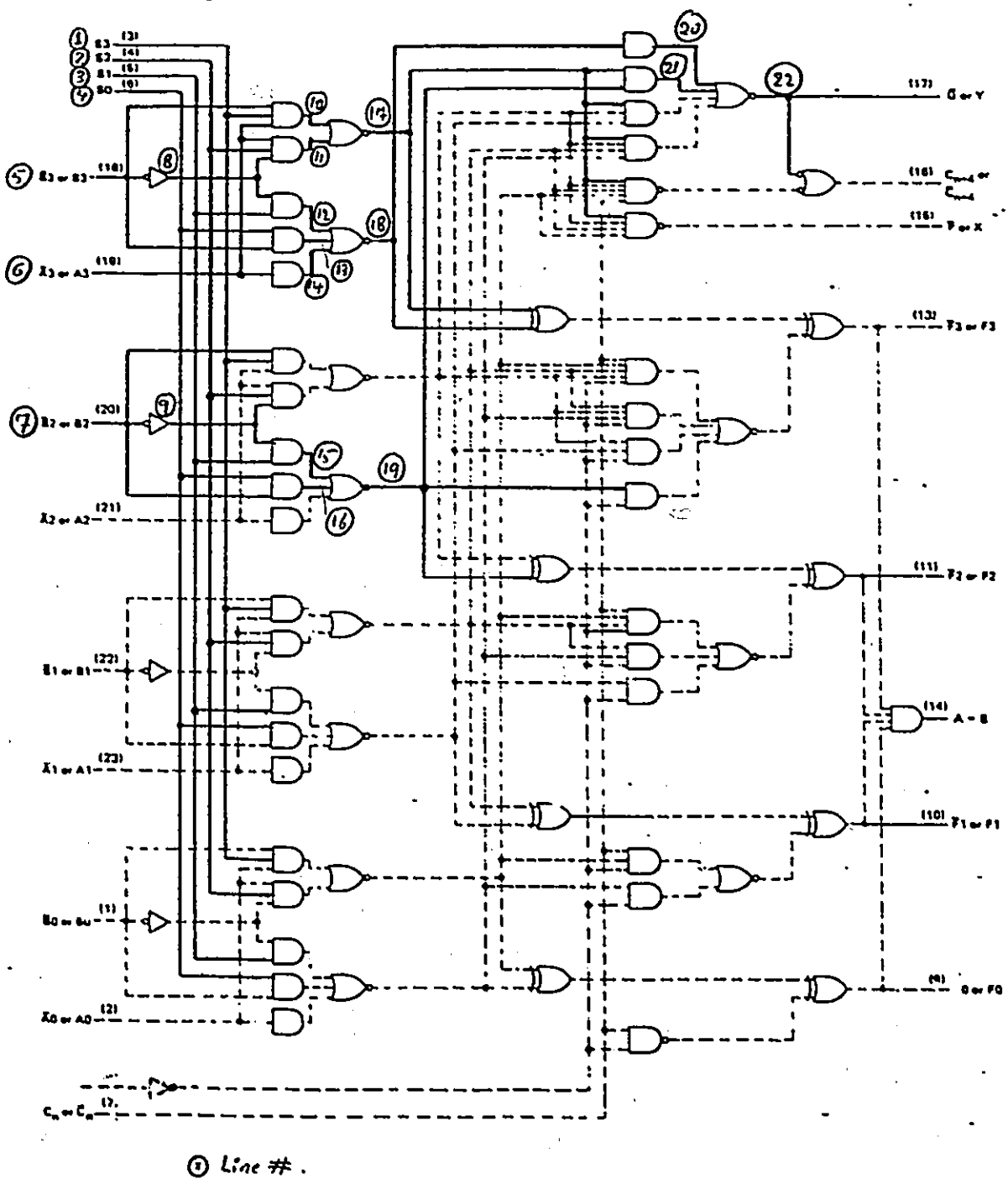


Figure 5.15: Flowchart for stochastic approximation .

functional block diagram



① Line #.

Figure 5.16: The Arithmetic Logic Unit (ALU) SN54LS181 .

4	(2,1)	18	(9,1)
5	(3,0)	19	(10,1); (11,1); (15,1); (16,1); (17,0); (19,0); (21,0)
6	(3,1	20	(12,0)
7	(4,0)	21	(12,1); (13,1); (14,1); (18,0); (20,0)
8	(4,1)	22	(13,0)
9	(5,0)	23	(14,0)
10	(5,1)	24	(16,0)
11	(6,0)	25	(17,1)
12	(6,1)	26	(19,1)
13	(7,0)	27	(18,1); (20.1); (21,1); (22.0)
14	(7,1)	28	(22,1)

Two major programming results were obtained and are described below .

1 •Results with Fixed Feature Set

The simplest technique to reduce the time in the construction phase is through fixing the input feature set vector to equal probability [$P(x_i) = 0.5$]. Figure 5.17 shows the flowchart of this approach. Based on this approach, we have the results shown in Figures 5.18 and 5.19 with 28 faults under consideration .

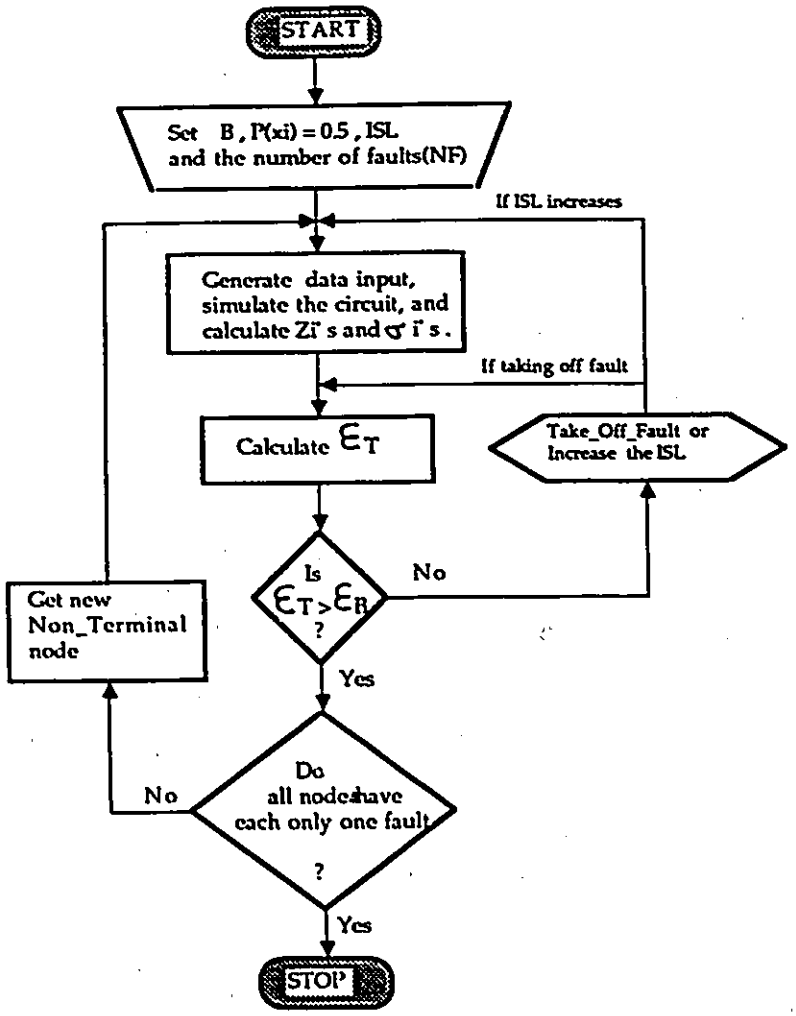


Figure 5.17: Flowchart to construct the multistage binary tree with fixed feature vector [0.5, 0.5, ..., 0.5] .

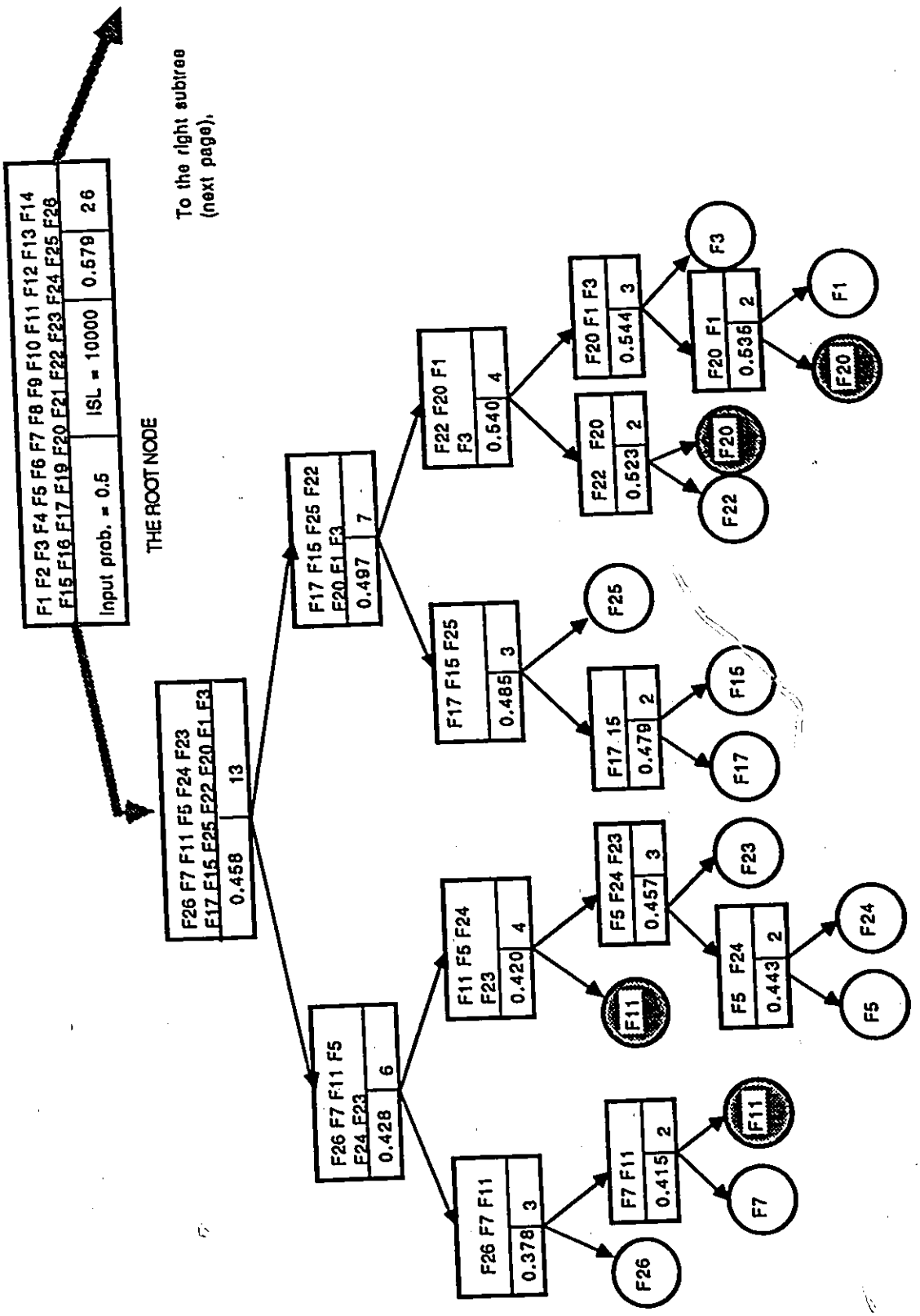
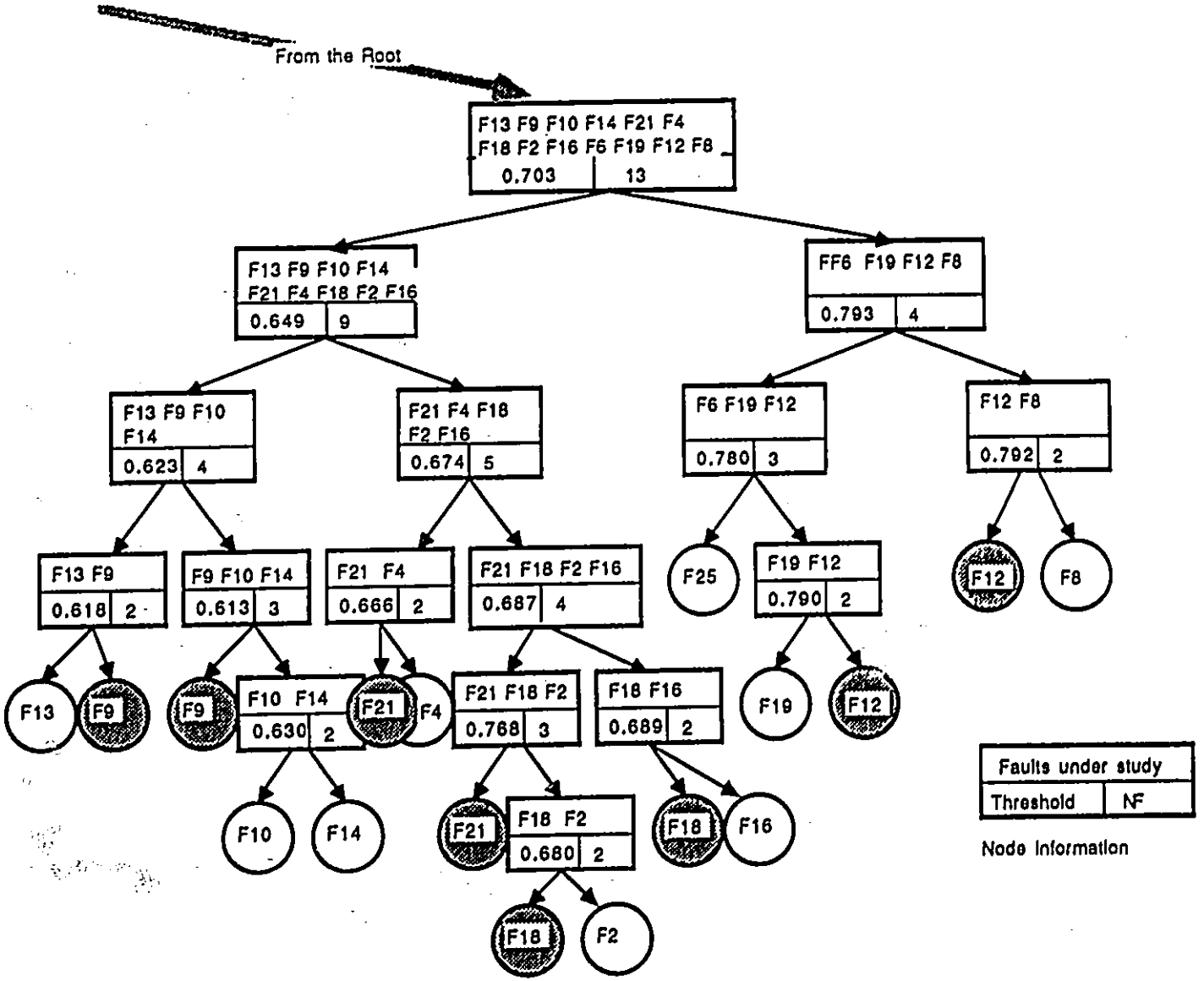


Figure 5.18: Multistage binary tree classifier for the ALU circuit with : 90 % confidence level, 26 possible faults, and 10000 input sequence length .


From the Root
F13 F9 F10 F14 F21 F4
F18 F2 F16 F6 F19 F12 F8
0.703 13
F13 F9 F10 F14
F21 F4 F18 F2 F16
0.649 9
FF6 F19 F12 F8
0.793 4
F13 F9 F10
F14
0.623 4
F21 F4 F18
F2 F16
0.674 5
F6 F19 F12
0.780 3
F12 F8
0.792 2
F13 F9
0.618 2
F9 F10 F14
0.613 3
F21 F4
0.666 2
F21 F18 F2 F16
0.687 4
F25
F19 F12
0.790 2
F12
F8
F13
F9
F9
F10 F14
0.630 2
F21
F4
F21 F18 F2
0.768 3
F18 F16
0.689 2
F19
F12
F10
F14
F21
F18 F2
0.680 2
F18
F16
F18
F2
Faults under study
Threshold NF
Node Information

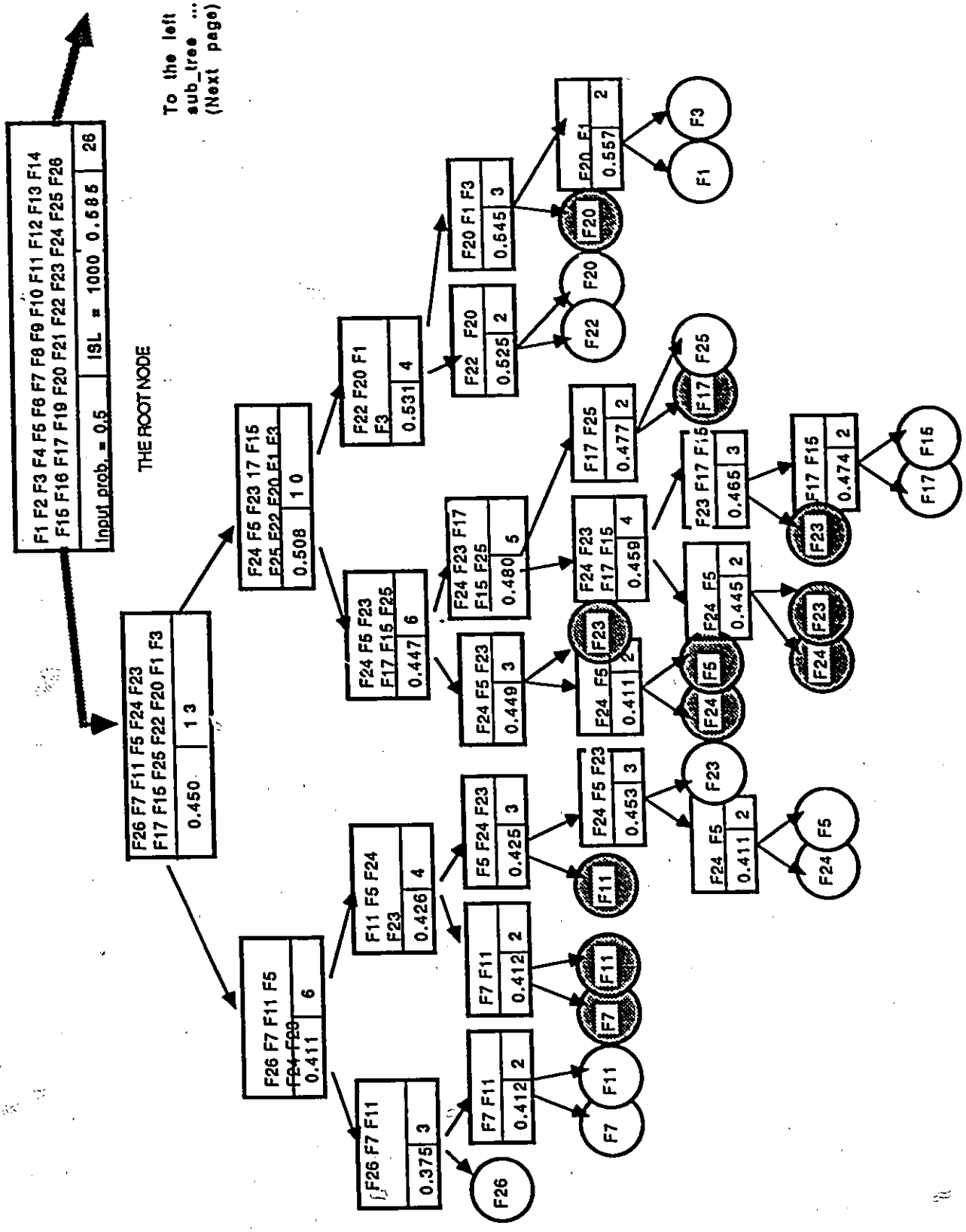
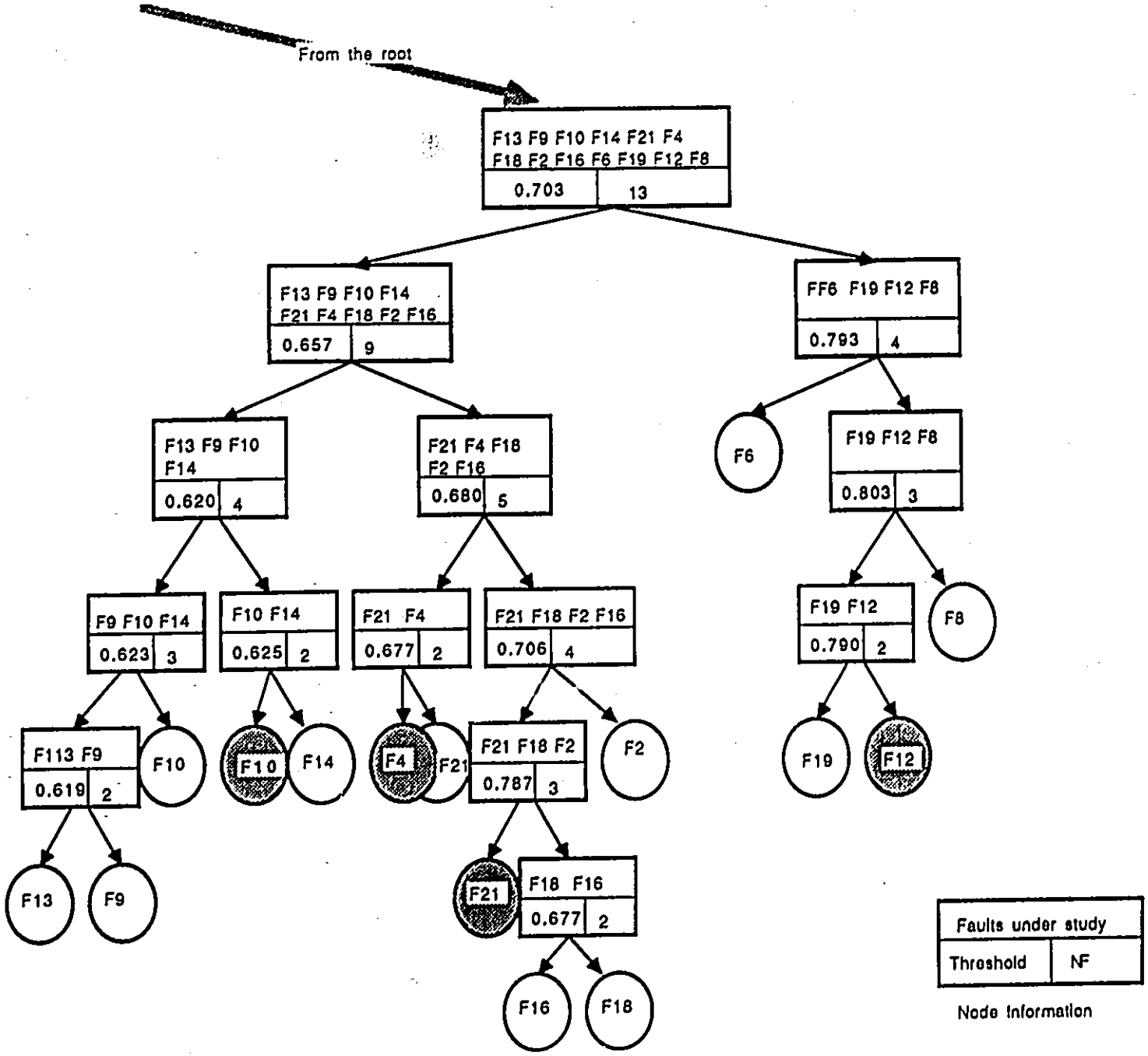


Figure 5.19: Multistage binary tree classifier for the ALU circuit with : 99 % confidence level, 26 possible faults, and 10000 input sequence length .


From the root
F13 F9 F10 F14 F21 F4 F18 F2 F16 F6 F19 F12 F8
0.703 13
F13 F9 F10 F14 F21 F4 F18 F2 F16
0.657 9
FF6 F19 F12 F8
0.793 4
F13 F9 F10 F14
0.620 4
F21 F4 F18 F2 F16
0.680 5
F6
F19 F12 F8
0.803 3
F9 F10 F14
0.623 3
F10 F14
0.625 2
F21 F4
0.677 2
F21 F18 F2 F16
0.706 4
F19 F12
0.790 2
F8
F113 F9
0.619 2
F10
F10
F14
F4
F21
F21 F18 F2
0.787 3
F2
F19
F12
F21
F18 F16
0.677 2
F13
F9
F16
F18
Faults under study
Threshold NF
Node information

2 • Results with Stochastic Approximation

A more efficient technique with longer time for construction is to use the Kiefer and Wolowafitz method of stochastic approximation to obtain the desired feature vector.The following pages illustrate some results.

Figure 5.25 : (a) and (b) shows the relation between the computing time required for the construction phase, the confidence level, and the number of faults under consideration . It also depicts how the computing time will increase if we need higher confidence measure .

Further discussions of these results are provided in Chapter 7. Appendix A has the program listings used for the purpose .

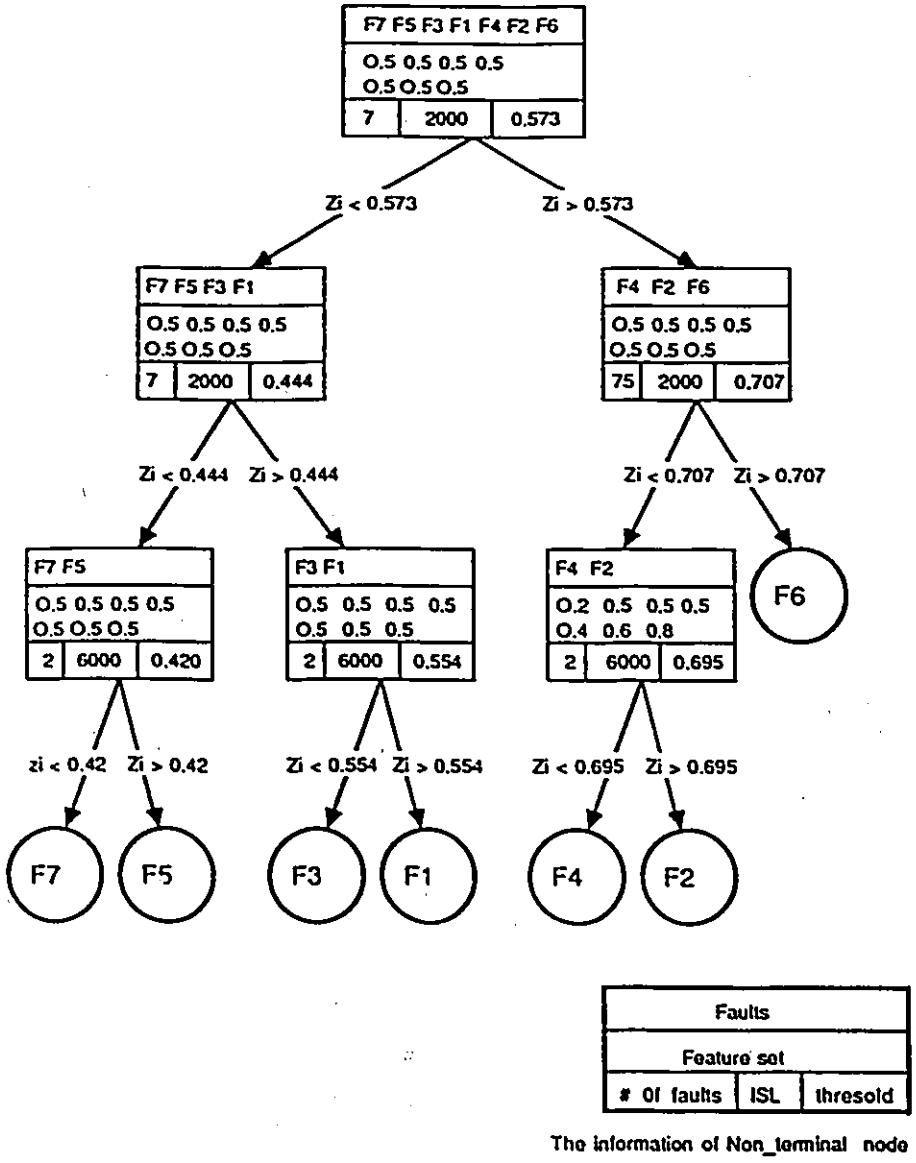


Figure 5.20: Multistage binary tree classifier for the ALU circuit with : 70 % confidence level, 7 possible faults, and 2000 input sequence length .

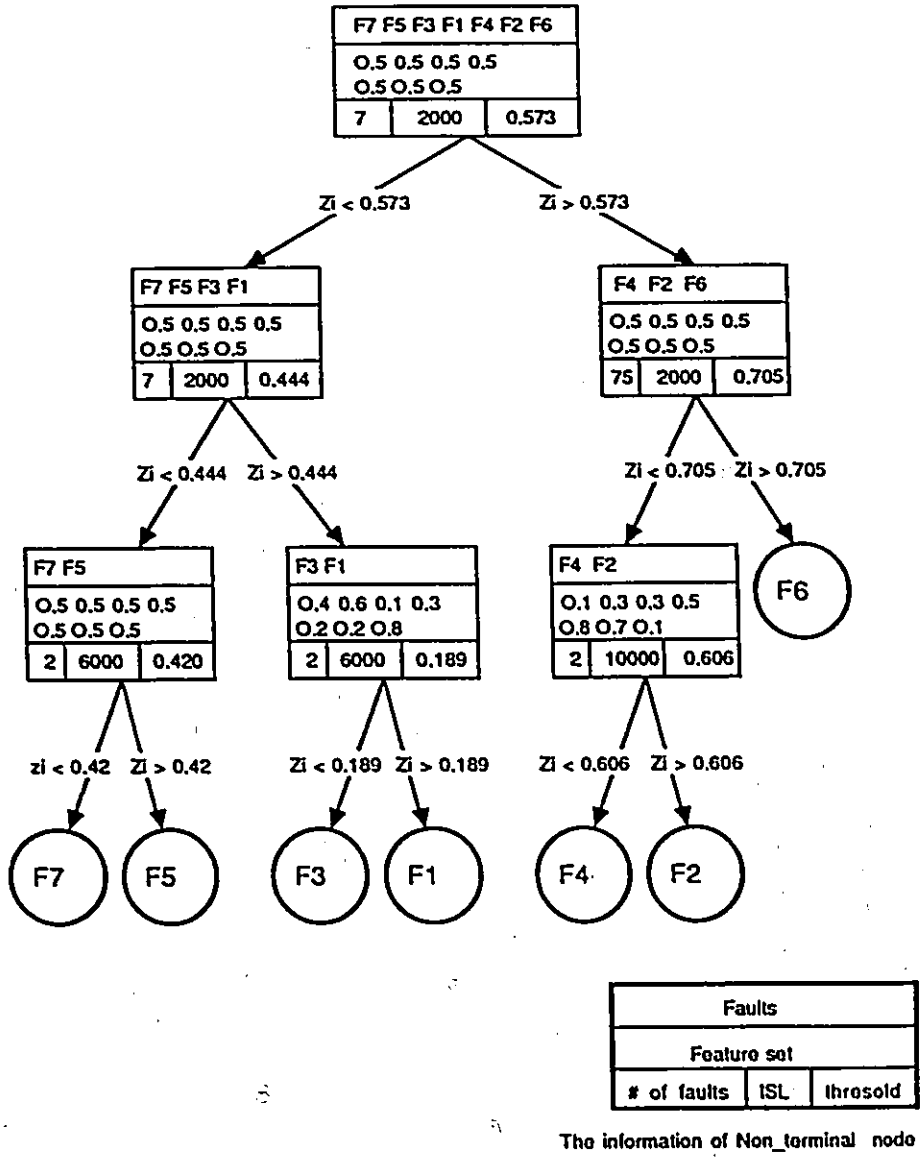


Figure 5.21: Multistage binary tree classifier for the ALU circuit with : 99 % confidence level, 7 possible faults, and 2000 input sequence length .

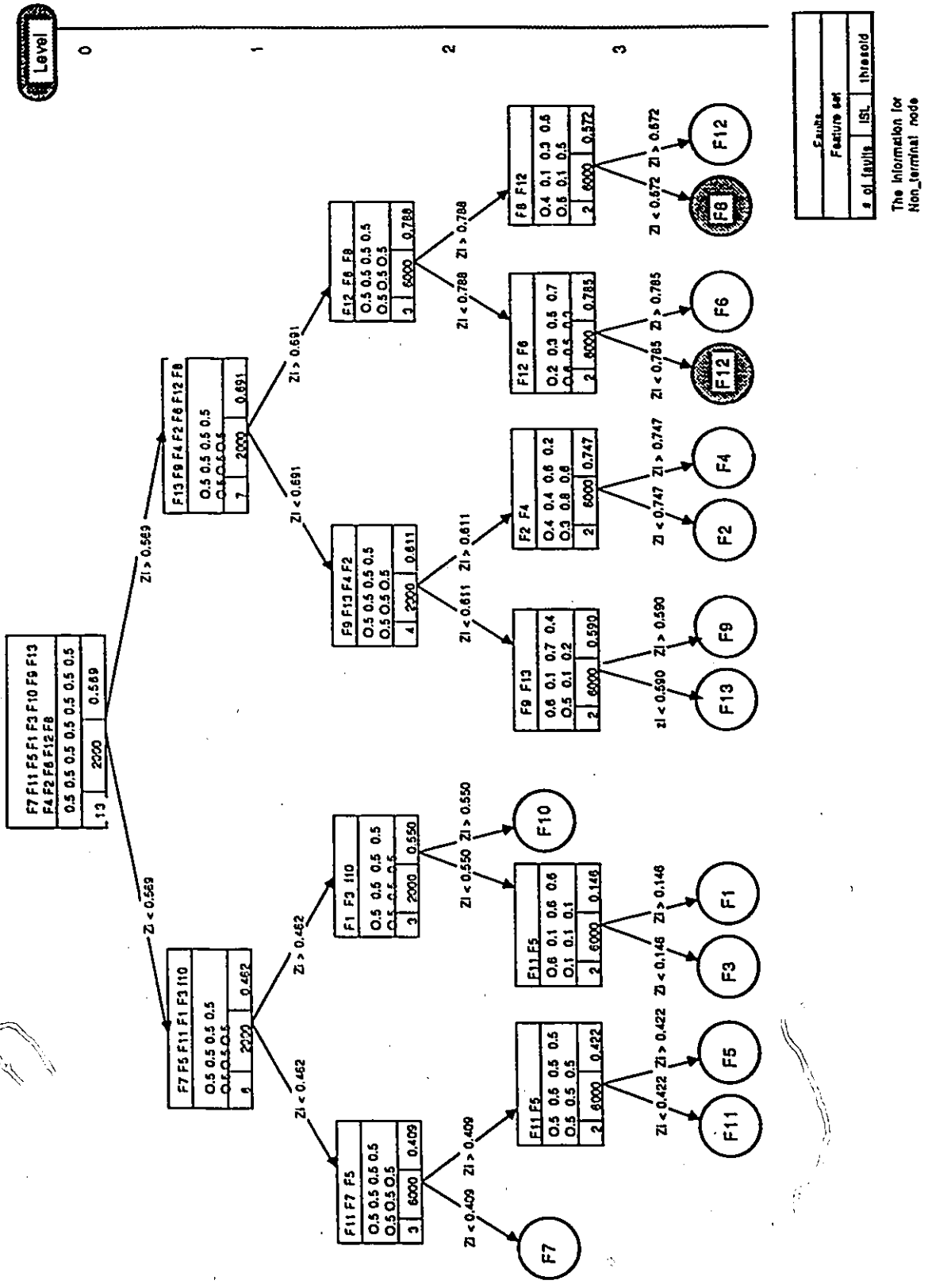


Figure 5.22: Multistage binary tree classifier for the ALU circuit with : 70 % confidence level, 13 possible faults, and 2000 input sequence length .

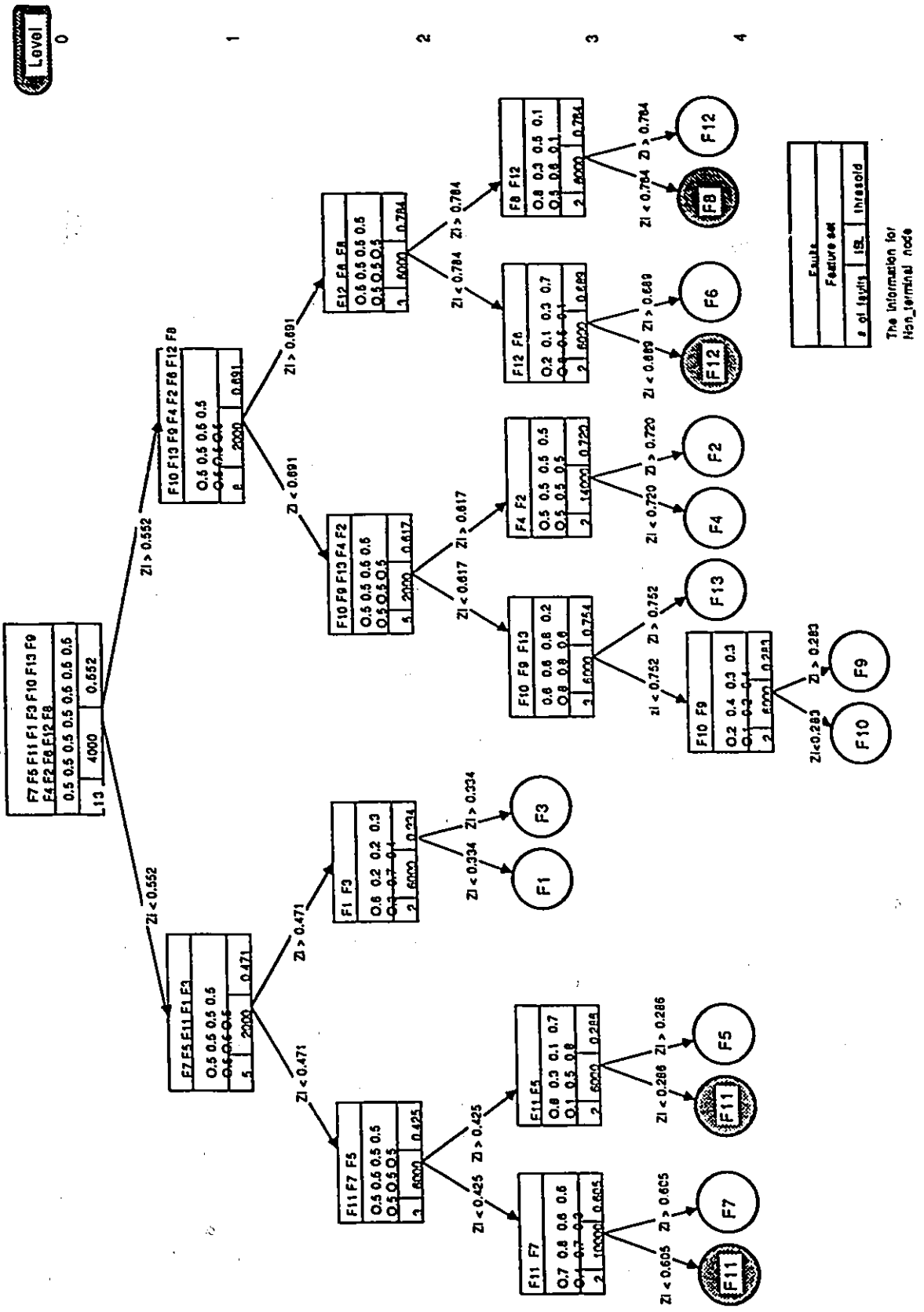
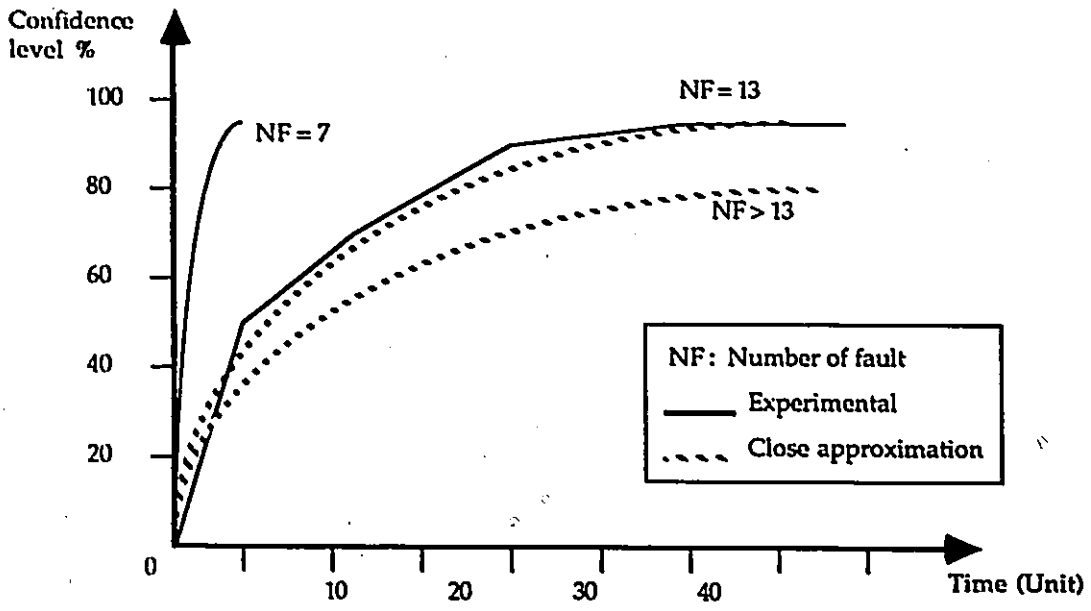


Figure 5.23: Multistage binary tree classifier for the ALU circuit with : 99 % confidence level, 13 possible faults, and 2000 input sequence length .

Conf. level	Computing time for tree construction	
	for 7 faults	for 13 faults
50%	1.5 *	4.5
70%	3.5	11,3
90%	4.2	19.5
99%	7.7	27.2

* The Unit is relative to the speed of the system. (PC or mainframe).

(a)



(b)

Figure 5.24: The relation between the confidence level and the computing time

Chapter 6

STRUCTURE AND IMPLEMENTATION OF TESTING PHASE

6.1 INTRODUCTION

In the previous chapter we developed a program to find the tree classifier. Still, we have not discussed how to use this information in the real world applications. Our program until now have been limited to constructing the tree and storing it in specific file. In this Chapter, we will present the application of our tree classifier method in the second phase of implementation (traverse phase). This implementation illustrates traversing through large amounts of data (data from chapter 5) to find a particular piece of information. As we shall see, because of the two-way decisions that must be made at each point in the process, binary traversing is the most efficient method of traversing. The help of the threshold level H in each node of the tree leads us to locate the fault of the faulty circuit under study.

If we recall the method of testing from Chapter 2, Figure 2.5, the result of traversing our predesigned binary tree is presented in Figure 6.1.

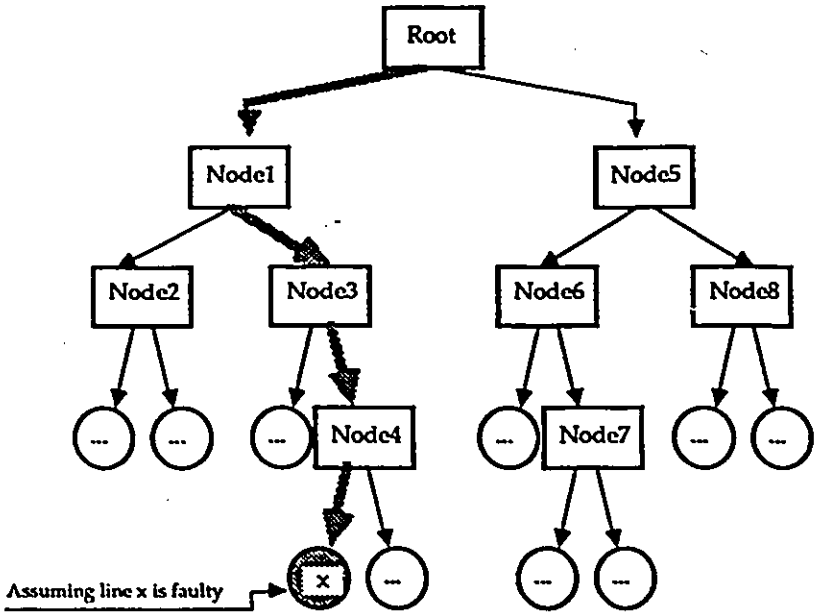


Figure 6.1: Traversing a tree to test if line X is s-a-0 .

6.2 THE OVERALL PERFORMANCE

Traversing a tree is completed by following three major steps:

1 • Get input information which consists of getting information from Chapter 5 .

2 • Do data analysis for the network under test which consists of analyzing information

and making decision along the binary tree.

3 • Get output result which consists of the answer that we are searching for.

Figure 6.2 presents graphically the interconnection of this design in order to form 2 levels of connections.

6.3 PROCEDURES OF LEVEL 1

Level 1 is a collection of two procedures: insertion to form the tree classifier (Insert-Tree-Info), and the decision making to go along the tree (Decision-Making). Each of

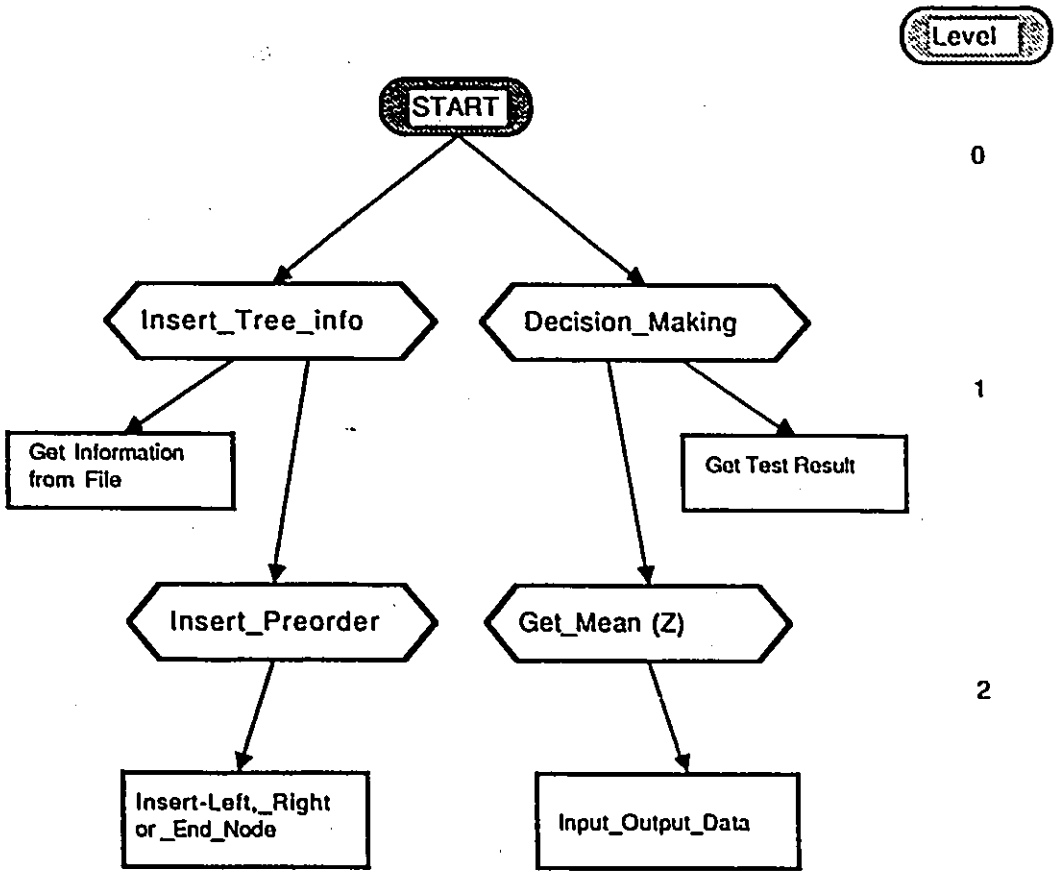


Figure 6.2: The overall performance structure.

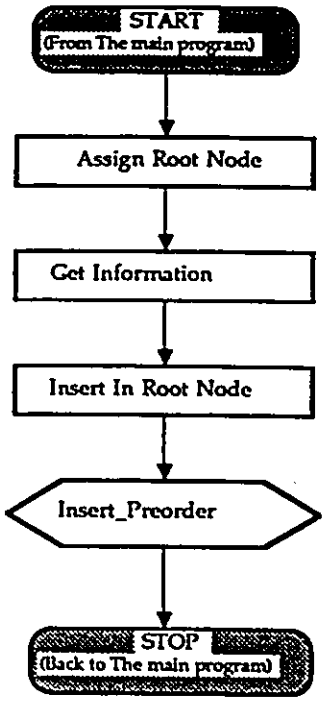


Figure 6.3: Flowchart for Insert-Tree-Info procedure .

these procedures can be implemented as:

6.3.1 INSERT-TREE-INFO

To complete this procedure we have to implement the following algorithm :

1 • Assign the root node for the tree.

2 • Get the information from output file of Chapter 5. Then insert them in root .

3 □ Complete the insertion in preorder form (Insert-Preorder) .

This algorithm can be graphically presented as in Figure 6.3.

6.3.2 DECISION MAKING

Making decision is the most important step that allows us to do the test, if we have a faulty network under test. The algorithm to go along the tree is as follows :

1 • Get the feature set for the input data.

2 □ Generate input data, then obtain the average output Z_0 of the circuit under test.

3 •If $Z_0 = 0$ or 1, then assign a reference number to the result, and go to Step 8 .

else go to Step 4.

4 • Compare Z_0 with the threshold H of the node :

If less, go to left node; else, go to right node.

5 • Check the number of faults in the new node :

If = 2, then read the length of the input sequence(ISL);

If = 1, then read the result reference number from this terminal node, and go to Step 8 .

6 • Get the new feature set.

7 • Get the average Z_0 of the output and go to Step 5.

8 □ Get-Test-Result (case of result number with the number of line and condition of being stuck at 0 or 1) .

Graphically, Figure 6.4 presents the flowchart of this algorithm.

6.4 PROCEDURE OF LEVEL 2

This level consists of two procedures:

1• Insert data in preorder form which means the reading from the output file of Chapter 5.

Then arrange the information in tree form. The insertion method (Algorithm Flowchart) is the same as in Chapter 5. Figure 6.5 illustrates the principle more clearly.

2 □ Get the average of the output by generating the input data to the circuit under study and observing the output response.

This procedure has an algorithm of two steps.

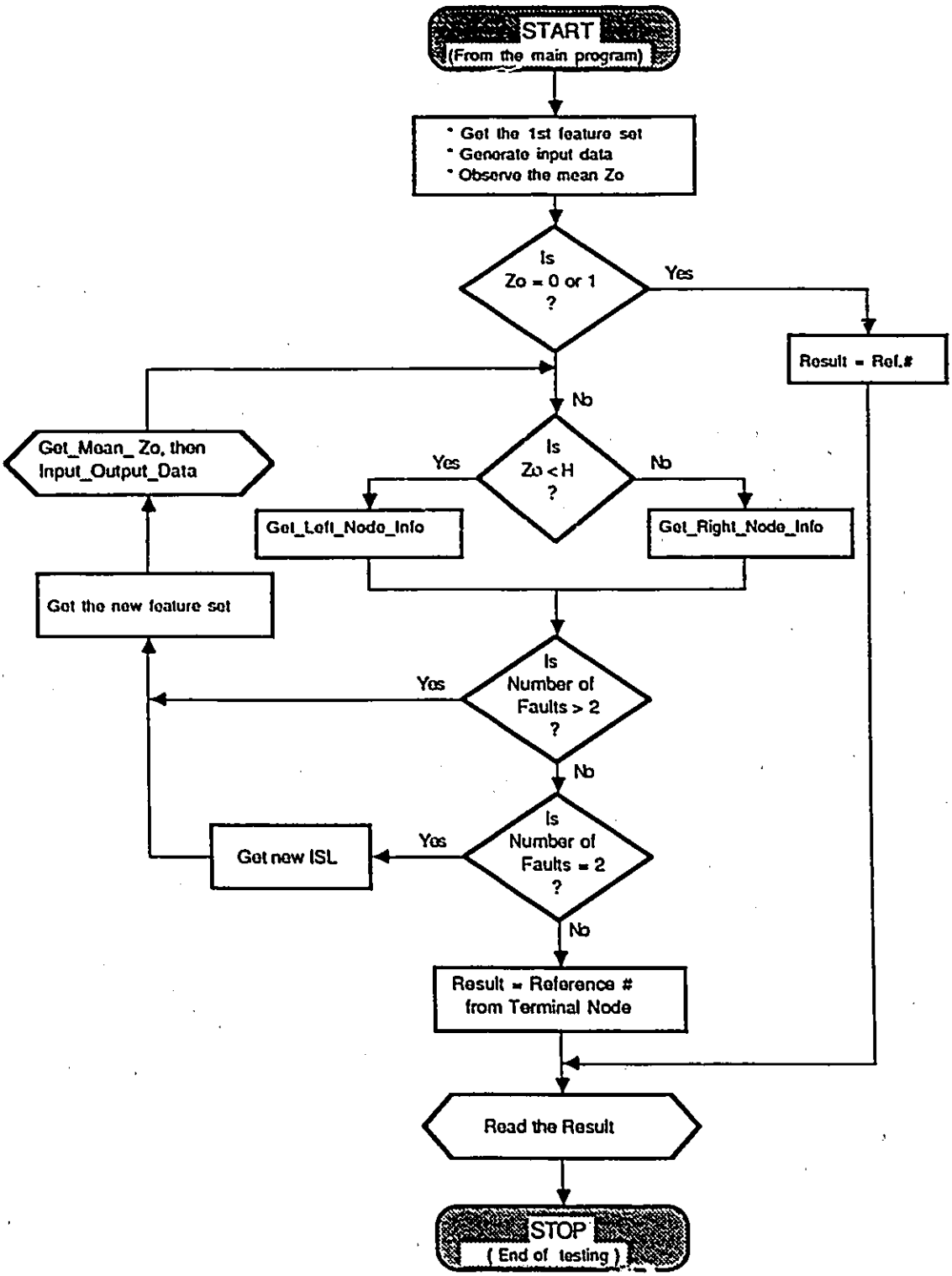


Figure 6.4: Decision-making flowchart.

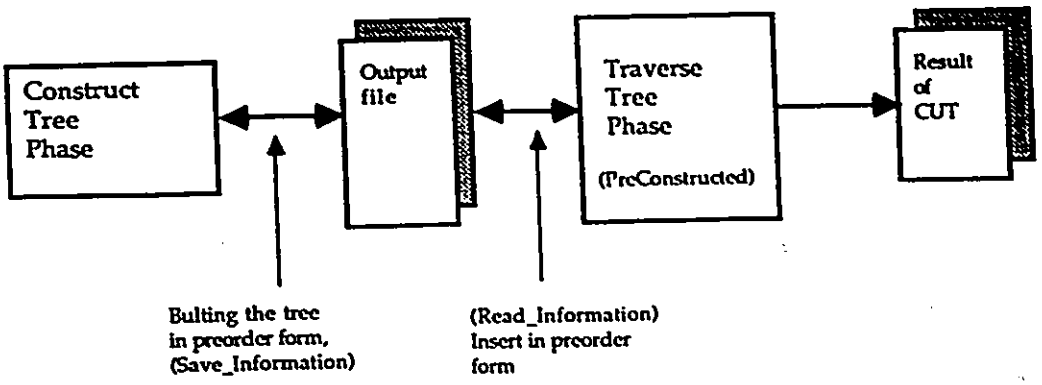


Figure 6.5: Transfer of data.

1 • Generate input output data and observe the mean Z_0 .

2 • Verify if the difference between the mean Z_0 and threshold level H is very small : If yes, repeat Step 1 ; if not, stop .

The flowchart can be graphically presented as in Figure 6.6 .

6.5 APPLICATION

Let us continue with our discussion of the application of these methods to the ALU for which we have a classifier tree in Chapter 5 . Suppose that line 5 is stuck-at-0 (Fig. 6.7). If we run the program of Appendix B and set the information of Fig. 5.8, we can follow the reading on the screen as in Fig. 6.8. However, if we follow the steps on the tree, it is easy to observe the way of solution. We can also observe the number of levels in the tree defining the testing time results .

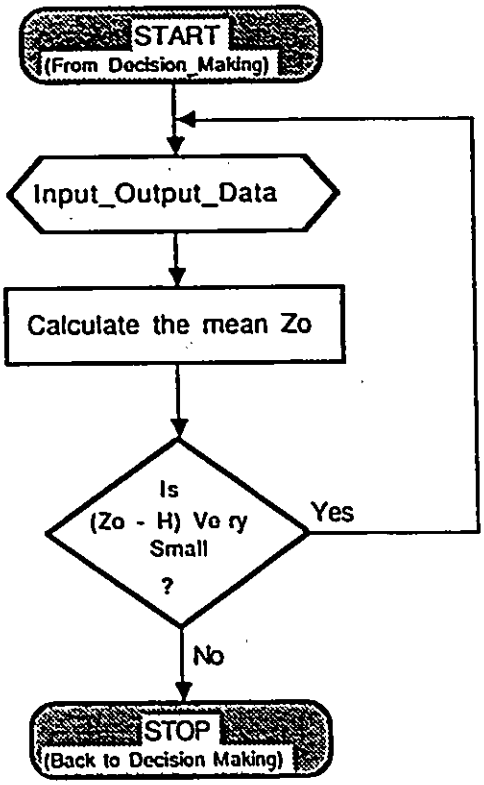


Figure 6.6: The flowchart for Get-Mean Z_0 procedure .

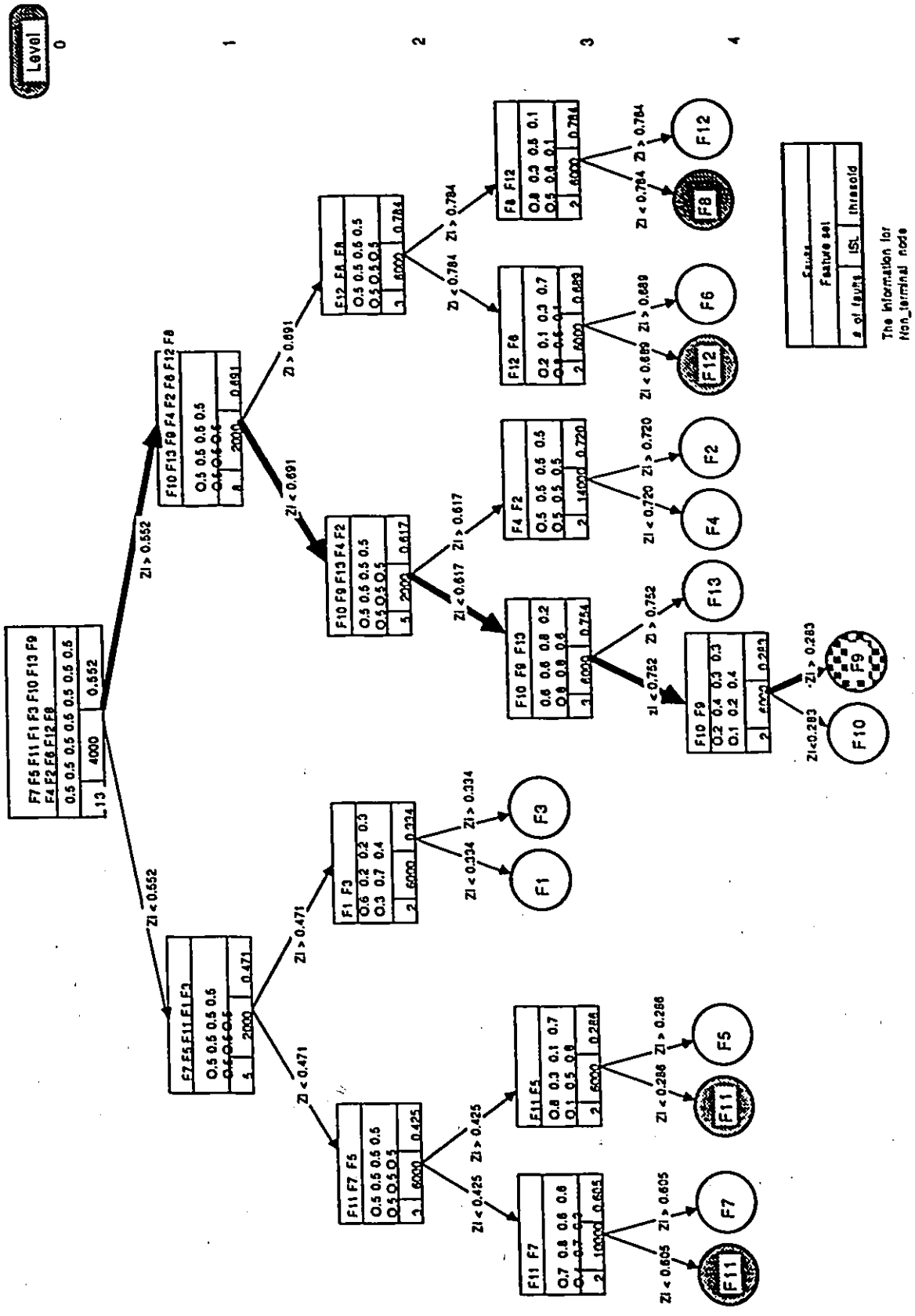


Figure 6.7: ALU with line # 5 stuck-at- 0

```
>

Work file name: a:testalus

Loading A:\TESTALUS.PAS
Compiling
  488 lines

Code:       0200 paragraphs (  8192 bytes), 0B28 paragraphs free
Data:       0018 paragraphs (   384 bytes), 0FC4 paragraphs free
Stack/Heap: 6527 paragraphs (414320 bytes)

Running
 GIVE ME THE NUMBER OF THE CIRCUIT UNDER TEST ..... 9

..... THANK YOU , WELL I AM TESTING IT AND , THEREFORE....

.......... I AM DOING THE Root NODE .....................

..I FINISH FROM LEVEL <0> AND I AM IN LEVEL <1>...

..I FINISH FROM LEVEL <1> AND I AM IN LEVEL <2>...

..I FINISH FROM LEVEL <2> AND I AM IN LEVEL <3>...

..I FINISH FROM LEVEL <3> AND I AM IN LEVEL <4>...

..I FINISH FROM LEVEL <4> AND I AM IN LEVEL <5>...
.....BUT LEVEL <5> HAS NO CALCULATION AND THE ANSWER IS

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::                                                              ::::
::::     THE RESULT FROM THE BINARY DECISION TREE IS :            ::::
::::                                                              ::::
       THE FAULT LOCATION IS THAT Line # 5 ......... S_A_0
::::                                                              ::::
::::                                                              ::::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

>
```

Figure 6.8: Testing phase result for line 5 stuck-at-0.

Chapter 7

CONCLUSIONS

Referring to the theory and application of the proposed method, we denote that the overall performance of the classifier depends on the total error committmentε_T of all faults to be classified at each node of each stage. By the definition of Eq(4.6), ε_T can be expressed as :

$$\varepsilon_T = \sum_{i=1}^{m} P_i \varepsilon_i \tag{7.1}$$

where m represents the number of faults under consideration, and ε_i is the error committment by fault Fi (summation of errors committed by Fi in each stage) .

Until now, the most important thing after a tree classifier is constructed is to restrict the error probability at each nonterminal node below a certain satisfactory bound so as to make the overall error probability no more than the tolerable bound. But the questions to be taken into consideration are : How to find these satisfactory bounds at each node ? If ε_T is the preset total error for the classifier, how to find the error probability tolerable at each nonterminal node ?

Suppose we have a tree classifier with X nonterminal nodes, and each one of them has a probability P_{N_i} , where :

$$P_{N_i} = \sum_{F_i \in Node i} P(F_i) \tag{7.2}$$

The total error committment of the tree classifier is less than or equal to:

$$E_{Total} \leq \sum_{i=1}^{X} P_{N_i} E(N_i) \tag{7.3}$$

(N_i) being the conditional error probability, because misclassification regions of different stages may be overlapped partially. Since, for a given problem, the values of X, P_{N_i}, and E(N_i) are not known a priori, the assignment of the error bound at each of the stages cannot be done analytically before the generation of the tree. If the total error of the tree classifier is less than a certain bound ε_T, how to find the dynamic bound for all terminals is very important. If now we are dealing with nonterminal node of the Kth level of the tree, and we have (X - K) more nonterminal nodes, the error bound for the current node can be :

$$E(N_k) = \frac{1}{(X-K)}(\varepsilon_T - \sum_{i=1}^{k-1} E(N_i)) \tag{7.4}$$

The purpose is to make the error probability at each nonterminal node equal. By fault overlap or controlling of the input length L, we can always have the error probability at nonterminal node N_i less than the conditional error probability $E(N_i)$ as described previously. Here again X being unknown, the problem cannot be solved a priori .

Let us approach the problem by taking some approximation or observation for X as the number of nonterminal node levels, and K as the Kth level. If m is the number of faults under consideration, from Eq.(2.2), $\frac{1}{0.3}log_{10}(m + N_{ov})$ node levels are needed to cover all the classifications (N_{ov} is the number of overlapping faults). This means :

$$X \geq \frac{1}{0.3}\log(m + N_{ov}) \tag{7.5}$$

And Eq.(7.3) becomes :

$$E_{(N_k)} \leq \frac{1}{(\frac{1}{0.3}log(m + N_{ov}) - K)}(\varepsilon_T - \sum_{i=1}^{k-1} E_{(N_i)}) \tag{7.6}$$

For the root node of the tree where K = 0, $E_{(N_k)}$ becomes :

$$E_{(N_0)} \leq \frac{(\varepsilon_T - 0)}{\frac{1}{0.3}log(m + N_{ov}) - 0} = \frac{\varepsilon_T}{\frac{1}{0.3}log(m + N_{ov})} \tag{7.7}$$

If we denote that this probability is equal at each nonterminal node we can define the error probability as :

$$E_{(N_0)} \leq \frac{\varepsilon_T}{\frac{1}{0.3}log(m + N_{ov})} \tag{7.8}$$

at any nonterminal node .

For example, in our case m = 28 faults, and suppose we need 80% confidence level or ($\varepsilon_T = 0.2$); therefore, the data from Table 7.1 approaches $E_{(N)} \leq 0.04$.

TABLE 7.1 : $E_{(N_i)}$ Relative To N_{ov} With m = 28 Faults, $\varepsilon_T = 0.2$.

Tree Case	N_{ov}	# of Levels Eq.(7.5)	$E_{(N)}$
Very well construction	0	5	0.04
Good construction	4	5	0.04
Poor construction	8	5	0.04
Very bad construction	16	6	0.033

An interesting approach to the solution that forms an acceptable constructed tree has the error bound $E_{(N)}$ for each nonterminal node defined as :

$$E_{(N)} \leq \frac{\varepsilon_T}{\frac{1}{0.3}log(m)} \tag{7.9}$$

One way to reduce the average classification time is the balancing of the tree where each child of a nonterminal node has almost equal number of faults after the grouping.

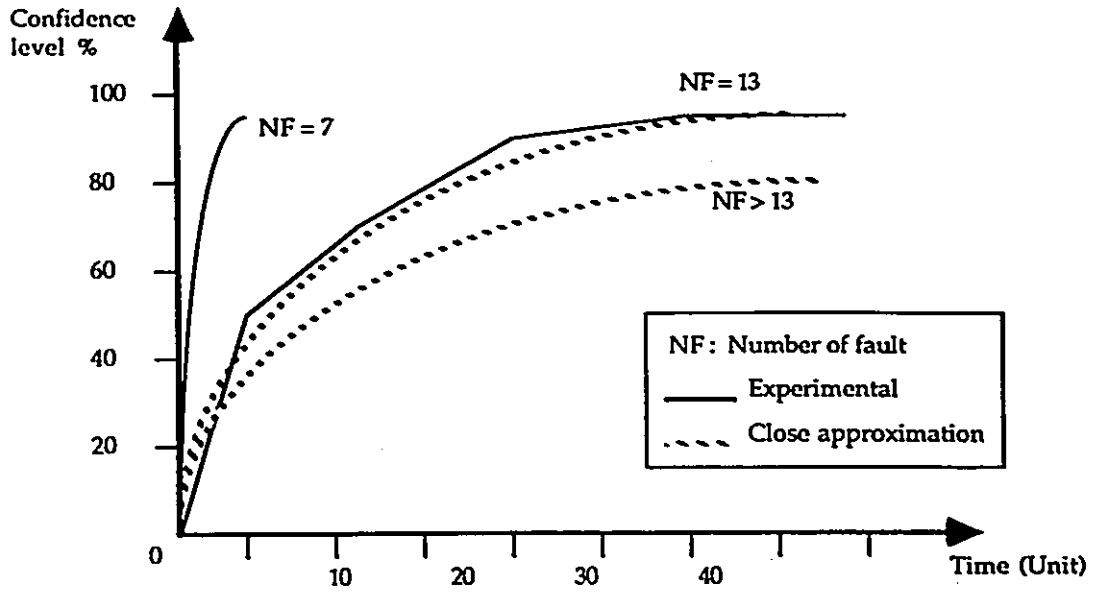


Figure 7.1: Confidence level vs construction time for the multistage. binary tree classifier .

However , this is an important factor to control the stability of the tree levels in relation to Eq.(5.5), which makes the approach of $E_{(N)}$ acceptable .

From the experimental results, it is easy to discover that if we are looking for higher confidence level, the more overlap faults occurs, which increases the classification time and the number of levels of the tree, resulting in a longer testing time (Figure 7.1).

Another important observation that the input sequence length can control the error probability makes it less than the error bound, but the process is quite long to obtain a well constructed tree with less levels for testing phase .

The most important point is the computing time for both phases of constructions and testing. and this is related to the system we use, IBM-PC or the main frame systems.

In the present study we did all the computations with a simulated circuit (CUT and the gold unit), and simulated one random generator for all the N inputs. However, in real applications we can use N generators for the N inputs. This will decrease the time for generating the input data or the computation time .

REFERENCES

[1]• Das, S, R. **Random test generation for LSI/VlSI circuits - a survey,** Proceedings of the 1986 IEEE Workshop on Languages for Automation and Languages for Computer Systems Design, Singapore, August 27-29, 1986, pp.132-138 (IEEE Computer Society Press, Washington, D.C. 1986).

[2]• Das, S. R., Chen, Z., and Jone, W. B. **Random test generation for irredundant combinational logic networks with a high measure of confidence** , Real-Time Systems Symp., Miami Beach, FL, December 1981.

[3]• David, R. and Blanchet, G. **About random fault detection of combinational networks** , IEEE Trans. Comput., Vol.C-25, pp. 659-664, June 1976.

[4]• Devijver, P. A. and Kittler, J. **Pattern Recognition : A Statistical Approach** , Prentice-Hall International, Inc.,1982.

[5]• Fu, K. S., Min, P. J. and Li, T. J. **Feature selection in pattern recognition,** IEEE Trans. System Science and Cybernetics, Vol. SSC-6, pp. 33-39, January 1970.

[6]• Funkuraga, K. **Introduction to Statistical Pattern Recognition** , Academic Press, Inc., New York, NY, 1972.

[7]• Hall, P. and Hegde, C. C. **Martingale Limite Theory and its Application,** Probability and Mathematical Statistics, Academic Press Inc., NY, 1980.

[8]• Hellman, M. E. and Raviv, J. **Probability of error , equivocation and the Chernoff bound** , IEEE Trans. Inform. Theory, Vol.IT-16, pp.368-372, July 1970.

[9]• Hogg, R. V. and Tanis, E. A. **Probability and Statistical Inference** , Macmillan Publishing Co. Inc., 1977.

[10]• Ito, T. **Approximate error bounds in pattern recognition, in Machine**

Intelligence, Vol. 7, Edinburgh University Press. pp. 369-376, Nov. 1972.

[11]• Jain, A. K. **On an estimate of the Bhattacharyya distance** , IEEE Trans. on Systems, Man and Cybernetics, SMC-6, pp.763-766, November 1976.

[12]• Jone, W. B. **Random Testing and Logic Design** , M.S.Thesis, Inst. of Comput. Eng., National Chiao Tung Univ., Hsinchu, Taiwan, ROC, June 1981.

[13]• Kailath, T. **The divergence and Bhattacharyya distance in signal selection** , IEEE Trans. Comm. Technology, Vol. COM-15, pp. 52-60, February 1967.

[14]• Kiefer, J. and Wolfowitz, J. **Stochastic estimation of the maximum of a regression function** , The Annals of Matimatical Statistics, Vol.23, No.3, September 1952.

[15]• Klir, George, J.**Introduction to the Methodology of Switching Circuits,** D. Van Nostrand Co. 1972.

[16]• Lala, Parag, K. **Fault Tolerant and Fault Testable Hardware Design** , Printice-Hall International Inc., London, 1985.

[17]• Matusita, K. **Decision rules based on the distance for problems to fit two samples and estimation** , Ann. Math. Stat., Vol.26, pp.631-640, 1955.

[18]• Miller, I. and Freund, J. E. **Probability and Statistics for Engineers** , Englewood Cliffs, NJ, Prentice-Hall Inc. 1977.

[19]• Page, C.V. **Equivalences between probabilistic and deterministic machines** , IC 9(5), pp.469-520, October 1966.

[20]• Papoulis, A. **Probability, Random Variable, and Stochastic Processes,** New York, McGraw-Hill 1965.

[21]• Parker, K.P. **Probabilistic test generation** , Digital Systems Lab. Stanford Univ. Stanford, Calif., Tech. Rep.18 August 1973.

[22]• Parker, K. P. and McCluskey, E. J. **Probabilistic treatment of general combinational networks** , IEEE Trans. on Computers, pp.668-670, June 1975.

[23]• Schrage, L. **A more portable Fortran random number generator** , ACM Trans. on Math. Software, pp. 132-138, June 1979 .

[24]• Tenanbaum, A.M. and Angenstein, M.J. **Data Structures using Pascal**, Prentice-Hall Inc., 1981.

[25]• Warfield, J. N. **Synthesis of switching circuits to yield prescribed probability relations** , CRSA 6, pp. 303-309, October 1965.

[26]• Wassan, M. T. **Stochastic Approximation** , Cambridge University Press 1969.

Appendix A

Appendix : The construction phase program

```
{$U+}
{ The program shown here enables us to construct a binary decision
  tree for any combinational network that has 5 input gates . ( we
  have to take the size of the systems memory in consideration). If
  we use m-input rather than 5-input gates , we must do some
  modification to the simulation functions .

  The input data for this program are :
   . The confidence level that we need for our design. For example,
     if our confidence level is 95 % , our data in should be 0.95 .
   . The number of faults under consideration ( Number of faults for
     the root node ) .
   . The input sequence length (which defines the size of the input
     data to  our network ).
   . The network simulation procedure .

  After we set these data , the program will follow the structure
  of Chapter 5 . ..............................................*}


PROGRAM FAULT_LOCATION (INFILE, OUTPUT) ;

CONST
     N_Input    = 7   ; { Number of input line to the network      }
     N_Fault    = 26  ; { Number of possible fault (stuck at 0/1) }


TYPE
     Node    = ^NodeRec ;
     NodeRec= Record
                Left    : Node ;
                Right   : Node ;
                TH      : Real ;       {The threshold level H          }
                NF      : Integer ;    {Number of fault under study    }
                ZLmax   : Integer ;    {Mean Z for Left Max.           }
                ZRmin   : Integer ;    {Mean Z for Right Min.          }
                Overlap: Integer ;     {Number of faults that overlaping}
                F       : Array[1..N_Fault] of Integer ;   {Fault Fi   }
                Pr      : Array[1..N_Input] of Real ;{Input Probability}
              End;


{The followin paragraph contains the global variable for our structure.
 We can define them as :

        Infile2    : Is to save the tree information that we need for
                     the testing phase (TestALU.pas program) .
        X          : Input line .
        Z_Ref      : Fault refference number .
```

```
        PROB_XIN   : The input probability .
        MEAN       : Output average (Zi) .
        VARI       : Output variance .
        BHAT_CAL   : Bhattacharrya calculated distance .
        Error_Ass  : Error required or assigned .
        Error_Cal  : Error calculated .
        L_Error    : Left error from right class fault .
        R_Error    : Right error from left class fault .
        F_THRLD    : Function of threshold H .
        DF_THRLD   : The derivative of F_THRLD function .
        F1         : Function with calculated Bhattacharrya distance .
        OPER_a     : Operators for stochastic approximation .
        OPER_s     : Operators for stochastic approximation .
        THRLD      : The threshold level H .
        N          : Number of input lines.
        Fault      : Number of faults under concideration .
        B          : Increment counter for stoch. approx.
        SEP_CLASS  : Class seperation to group the faults
        ISL        : Input sequence length at each node .
        SAVE_ISL   : The starting ISL .
        Foverlap   : Number of overlap faults .
        Lmax       : Last mean in the left group .
        Rmin       : First mean in the right group .
        Root       : The top node of the tree .                    )


VAR
     Infile1,Infile2 : text ;
     X               : ARRAY [1..N_Input] OF INTEGER ;
     CF1             : ARRAY [1..N_Fault] OF INTEGER ;
     Z_Ref           : ARRAY [1..N_Fault] OF INTEGER ;
     PROB_XIN        : ARRAY [1..N_Input] OF REAL ;
     MEAN            : ARRAY [1..N_FAULT] OF REAL ;
     VARI            : ARRAY [1..N_FAULT] OF REAL ;
     BHAT_CAL        : ARRAY [1..N_FAULT,1..N_FAULT] OF REAL ;
     BHAT_MIN        : ARRAY [1..N_FAULT,1..N_FAULT] OF REAL ;
     Error_Ass,Error_Cal,L_Error,R_Error : Real ;
     F_THRLD,DF_THRLD,F1,F2,OPER_a,OPER_s,THRLD : Real ;
     F1t,N,NX,Fault,B,II,SEP_CLASS,OK,ISL,SAVE_ISL,Foverlap,
     Lmax,Rmin : INTEGER;
     Root : Node ;


(*.. Several functions to simulate the network .....................*

FUNCTION NORG (U;W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 0  THEN NORG := 1  ELSE NORG := 0 ;
  END;
FUNCTION ORG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
```

```
      IF U+W+X+Y+Z = 0  THEN  ORG := 0  ELSE  ORG := 1 ;
  END;
FUNCTION NANDG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 5  THEN  NANDG := 0  ELSE  NANDG := 1 ;
  END;
FUNCTION ANDG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 5  THEN  ANDG := 1  ELSE  ANDG := 0 ;
  END;
FUNCTION NOTG (X : INTEGER ) : INTEGER ;
  BEGIN
      IF X=1 THEN  NOTG := 0  ELSE  NOTG := 1 ;
  END;
FUNCTION EX_ORG (X,Y : INTEGER ) : INTEGER ;
  BEGIN
      IF X=Y THEN EX_ORG := 0  ELSE EX_ORG := 1 ;
  END;
FUNCTION EX_NORG (X,Y : INTEGER ) : INTEGER ;
  BEGIN
      IF X=Y THEN EX_NORG := 1  ELSE EX_NORG := 0 ;
  END;



(*.. Procedure to simulate the ALU logic circuit .....................*)

PROCEDURE SIMULAT_CIRCUIT ;
VAR
   F1 : ARRAY [1..N_Fault] OF INTEGER ;
   A,B,C,D,E,F,G,H,J : INTEGER ;

BEGIN
   FOR J := 1 TO Fault DO
      BEGIN
                CASE Z_Ref[J] OF

(.................. for line (#1,0) or (#10,0) .....................)
1:BEGIN
A := NORG(0,0,0,0,ANDG(1,1,X[6],X[2],NOTG(X[5])));
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]),ANDG(1,1,1,X[4],X[7]));
F1[1] := NORG( 0,0,0,B, ANDG(1,1,1,A,D) );
IF F1[1] = 1 THEN CF1[1] := CF1[1] + 1 ;
END;


(..................... for line (#1,1).....................)
2:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],1,X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
```

```
F1[2] := NORG( 0,0,0,B, ANDG(1,1,1,A,D) );
IF F1[2] = 1 THEN CF1[2] := CF1[2] + 1 ;
END;


{.................. for line (#2,0) or (#11,0).....................}
3:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), 0);
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[3] := NORG( 0,0,0,B, ANDG(1,1,1,A,D) );
IF F1[3] = 1 THEN CF1[3] := CF1[3] + 1 ;
END;

{..................... for line (#2,1) ........................}
4:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],1,
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[4] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[4] = 1 THEN CF1[4] := CF1[4] + 1 ;
END;


{..................... for line (#3,0) ........................}
5:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])));
B := NORG(0,0,0, ANDG(1,1,1,X[4],X[5]), X[6]);
D := NORG(0,0,0,0, ANDG(1,1,1,X[4],X[7]));
F1[5] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[5] = 1 THEN CF1[5] := CF1[5] + 1 ;
END;


{..................... for line (#3,1) ........................}
6:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),1), ANDG(1,1,1,X[4],X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),1), ANDG(1,1,1,X[4],X[7]));
F1[6] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[6] = 1 THEN CF1[6] := CF1[6] + 1 ;
END;


{..................... for line (#4,0) ........................}
7:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]),0, X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]),0);
```

```
F1[7] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[7] = 1 THEN CF1[7] := CF1[7] + 1 ;
END;


{..................... for line (#4,1) ........................}
8:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,1,X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), X[7]);
F1[8] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[8] = 1 THEN CF1[8] := CF1[8] + 1 ;
END;


{..................... for line (#5,0) ........................}
9:BEGIN
A := NORG(0,0,0,0, ANDG(1,1,X[6],X[2],1) );
B := NORG(0,0,X[3],0,X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[9] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[9] = 1 THEN CF1[9] := CF1[9] + 1 ;
END;


{..................... for line (#5,1).........................}
10:BEGIN
A := NORG(0,0,0, ANDG(1,1,1,X[1],X[6]), 0);
B := NORG(0,0,0,X[4],X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[10] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[10] = 1 THEN CF1[10] := CF1[10] + 1 ;
END;


{..................... for line (#6,0).........................}
11:BEGIN
A := 1 ;
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),0);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[11] := NORG( 0,0,0,B,D);
IF F1[11] = 1 THEN CF1[11] := CF1[11] + 1 ;
END;


{..................... for line (#6,1) ........................}
12:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],1), ANDG(1,1,1,X[2],
     NOTG(X[5])) );
B := 0 ;
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[12] := NORG( 0,0,0,0, ANDG(1,1,1,A,D));
IF F1[12] = 1 THEN CF1[12] := CF1[12] + 1 ;
END;
```

```
{..................... for line (#7,0) ......................}
13:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
                    ANDG(1,1,1,1,X[6]));
D := NORG(0,0,0,0,X[3]);
F1[13] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[13] = 1 THEN CF1[13] := CF1[13] + 1 ;
END;

{..................... for line (#7,1).....................}
14:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0,0,X[4]);
F1[14] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[14] = 1 THEN CF1[14] := CF1[14] + 1 ;
END;


{..................... for line (#8,0) ......................}
15:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]),0);
B := NORG(0,0,0, ANDG(1,1,1,X[4],X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[7],
     X[4]));
F1[15] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[15] = 1 THEN CF1[15] := CF1[15] + 1 ;
END;


{..................... for line (#8,1) ......................}
16:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],1) );
B := NORG(0,0,X[3], ANDG(1,1,1,X[4],X[5]), X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],
     X[7]));
F1[16] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[16] = 1 THEN CF1[16] := CF1[16] + 1 ;
END;


{..................... for line (#9,0) or (#15,0).............}
17:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0,0,ANDG(1,1,1,X[4],X[7]));
```

```
F1[17] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[17] = 1 THEN CF1[17] := CF1[17] + 1 ;
END;


{..................... for line (#9,1) ......................}
18:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0,X[3], ANDG(1,1,1,X[4],X[7]));
F1[18] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[18] = 1 THEN CF1[18] := CF1[18] + 1 ;
END;


{......... for line (#10,1) OR (#11,1) OR (#17,0) ..............}
{............or (#15,1) OR (#16,1) OR (#19,0)..................}
19:BEGIN
A := 0 ;
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[19] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[19] = 1 THEN CF1[19] := CF1[19] + 1 ;
END;


{..................... for line (#12,0) ......................}
20:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])));
B := NORG(0,0,0, ANDG(1,1,1,X[4],X[5]), X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[20] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[20] = 1 THEN CF1[20] := CF1[20] + 1 ;
END;


{..... for line (#12,1) 0R (#13,1) OR (#14,1) OR (#18,0) or (#20,0)..}
21:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := 0 ;
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[21] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[21] = 1 THEN CF1[21] := CF1[21] + 1 ;
END;


{..................... for line (#13,0) ......................}
22:BEGIN
```

```
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), 0, X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[22] := NORG( 0,0,0, B, ANDG(1,1,1,A,D));
IF F1[22] = 1 THEN CF1[22] := CF1[22] + 1 ;
END;


{..................... for line (#14,0) ........................}
23:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),0);
D := NORG(0,0,0,ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[23] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[23] = 1 THEN CF1[23] := CF1[23] + 1 ;
END;


{..................... for line (#16,0) ........................}
24:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), 0);
F1[24] := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F1[24] = 1 THEN CF1[24] := CF1[24] + 1 ;
END;

{..................... for line (#17,1) ........................}
25:BEGIN
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F1[25] := NORG( 0,0,0,B,D);
IF F1[25] = 1 THEN CF1[25] := CF1[25] + 1 ;
END;


{..................... for line (#19,1)........................}
26:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
F1[26] := NORG( 0,0,0,B,A);
IF F1[26] = 1 THEN CF1[26] := CF1[26] + 1 ;
END;

                END; {.... END OF CASE ......}
  END; { ....END FOR J...}

END;
```

```
{.. Procedure to generate input data and obtain output information ...}

PROCEDURE INPUT_OUTPUT_DATA ;
VAR  I,P : INTEGER ;
BEGIN
   FOR I := 1 TO FAULT DO CF1[Z_Ref[I]] := 0 ;
   FOR P := 1 TO ISL DO
       BEGIN
          FOR I := 1 TO N DO
              BEGIN
                 IF RANDOM(ISL)+1 <= ABS(Prob_Xin[I] * ISL)
                    THEN X[I] := 1
                    ELSE X[I] := 0 ;
              END;
          SIMULAT_CIRCUIT;
       END;
    FOR I := 1 TO FAULT DO  MEAN[I] := CF1[Z_Ref[I]]/ISL ;
END;



{.. Procedure to group the means in order form ..................}
PROCEDURE MEAN_IN_ORDER ;
VAR
   D_MEAN : REAL ;
   D_Ref,I,J  : INTEGER ;
BEGIN
   FOR I := 1 TO FAULT DO
       BEGIN
          FOR J := 1 TO FAULT-1 DO
              BEGIN
                 D_MEAN := MEAN[J] ;
                 D_Ref  := Z_Ref[J] ;
                 IF MEAN[J] > MEAN[J+1] THEN
                    BEGIN
                       MEAN[J]    := MEAN[J+1] ;
                       MEAN[J+1]  := D_MEAN ;
                       Z_Ref[J]   := Z_Ref[J+1] ;
                       Z_Ref[J+1] := D_Ref ;
                    END;
              END;
       END;
    FOR I := 1 TO FAULT DO VARI[I] := (MEAN[I]-SQR(MEAN[I]))/ISL ;
END;
```

```
(*.. Procedure for experimental Bhattacharrya distance .............*)

PROCEDURE GET_BHAT_CAL ;
VAR
    I,J : INTEGER ;
    R,Z,V : REAL ;
BEGIN
    FOR I:= 1 TO Lmax DO
        FOR J := Rmin TO FAULT DO
            BEGIN
               IF MEAN[I] = MEAN[J]  THEN BHAT_CAL[I,J] := 0
                  ELSE BEGIN
                          IF (VARI[I] = 0) OR (VARI[J]=0)
                             THEN BHAT_CAL[I,J] := 100
                             ELSE BEGIN
                                     R := SQR(MEAN[I] - MEAN[J]) ;
                                     Z := VARI[I] + VARI[J] ;
                                     V := SQRT(VARI[I] * VARI[J]) ;
                                BHAT_CAL[I,J] := (0.25*R/Z)+0.5*LN(0.5*Z/V);
                                   END;
                       END;
            END;
END;



(.. Procedure to find function F1 ..................................)

PROCEDURE GET_F1 ;
VAR
   I,J : INTEGER ;
   FB  : REAL ;
BEGIN
   FB := 0 ;
   FOR I := 1 TO Lmax DO
       FOR J := Rmin TO FAULT DO
              FB := FB + BHAT_CAL[I,J] ;
   F1 := SQR(FB - ERROR_ASS);
END ;



(.. Procedure to find the class separability .......................)

PROCEDURE GET_CLASS_SEP ;
VAR  A : INTEGER ;
BEGIN
   IF FAULT = 2
      THEN BEGIN
              THRLD := (MEAN[1] + MEAN[2]) / 2 ;
```

```
                Lmax := 1 ;
                Rmin := 2 ;
                FOverlap := 0 ;
             END
       ELSE BEGIN
    THRLD := 0 ;
    FOR A := 1 TO FAULT DO  THRLD := THRLD + MEAN[A] ;
    THRLD := THRLD/FAULT ;
    IF (MEAN[1] = THRLD) OR (MEAN[FAULT] = THRLD)
       THEN THRLD := (MEAN[1] + MEAN[FAULT])/2 ;
    A := 1 ;
    WHILE A < FAULT DO
       BEGIN
           IF MEAN[A] > THRLD
              THEN BEGIN
                      SEP_CLASS := A - 1 ;
                      A := FAULT + 1 ;
                   END
              ELSE A := A + 1 ;
       END;
    IF A = FAULT THEN SEP_CLASS := FAULT - 1 ;
    Lmax := SEP_CLASS  ;
    Rmin := SEP_CLASS + 1;
        END;
END;



{.. Procedure to set a new feature set ..............................}

PROCEDURE SET_NEW_PROB (VAR B : INTEGER);
VAR
   A           : ARRAY [1..2] OF REAL ;
   NEW_PROB    : ARRAY [1..N_INPUT] OF REAL ;
   OPER_a,OPER_s,DERIV,OLD_PROB : REAL ;
   i,FF,SV : INTEGER ;
BEGIN
   OPER_s  := 1/EXP(B - 1) ;
   OPER_a  := EXP(-1/3*LN(B)) ;
   FOR NX  := 1 TO N DO
       BEGIN
WRITELN(' I AM SETING THE NEW PROB FOR X_In ',NX);
          OLD_PROB := PROB_XIN[NX] ;
          A[1] := OLD_PROB + OPER_a ;
          A[2] := OLD_PROB - OPER_a ;
          IF A[1] > 1  THEN A[1] := 0.9  ;
          IF A[2] < 0  THEN A[2] := 0.1  ;
          FOR FF := 1 TO 2 DO
              BEGIN
                 PROB_XIN[NX] := A[FF] ;
                 INPUT_OUTPUT_DATA ;
                 MEAN_IN_ORDER ;
                 GET_BHAT_CAL  ;
```

```
                GET_F1 ;
                A[FF] := F1 ;
              END ;
          NEW_PROB[NX] := OLD_PROB - OPER_s * (A[1] - A[2])/(2*OPER_a);
          IF NEW_PROB[NX] >= 1  THEN NEW_PROB[NX] := 0.9 ;
          IF NEW_PROB[NX] <= 0  THEN NEW_PROB[NX] := 0.1 ;
          PROB_XIN[NX] := OLD_PROB ;
       END;
   FOR FF := 1 TO N DO PROB_XIN[FF] :=  NEW_PROB[FF] ;
   FOR FF := 1 TO N DO WRITELN('THE NEW P ',FF,' = ',NEW_PROB[FF]);
END;



{.. Procedure for the stochastic approximation .....................}

PROCEDURE STOCH_APPROX ;
VAR
   SAVE_PROB : ARRAY[1..N_INPUT] OF REAL ;
   B,I       : INTEGER ;
   OLD_F1    : REAL ;
BEGIN
   B := 2 ;
   OLD_F1 := F1 ;
   WHILE B > 0  DO
      BEGIN
         FOR I := 1 TO N DO SAVE_PROB[I] := PROB_XIN[I] ;
         SET_NEW_PROB(B) ;
         INPUT_OUTPUT_DATA ;
         GET_BHAT_CAL ;
         GET_F1 ;
WRITELN(' ');
WRITELN('I AM IN STOCH. APPROX  OLD_F1 = ',OLD_F1:5:5,'   NEW F1 = ',
         F1:5:5);
         IF F1 < OLD_F1
            THEN BEGIN
                    OLD_F1 := F1 ;
                    B := B + 1 ;
                 END
            ELSE BEGIN
                    FOR I := 1 TO N DO  PROB_XIN[I] := SAVE_PROB[I];
                    B := 0 ;
                 END;
      END;
END;



{.. Procedure to group the faults arbitrarily .. ..................}

PROCEDURE ARBITRARY_GROUPING ;
VAR I,S :INTEGER ;
```

```
BEGIN
   GET_CLASS_SEP ;
   GET_BHAT_CAL ;
   GET_F1 ;
writeln('  ');
writeln(':::::::::: i starting stoch.approx for more fault :::::::');
   STOCH_APPROX ;
writeln(':::::::::: i finished stoch.approx for more fault :::::::');
   INPUT_OUTPUT_DATA ;
   MEAN_IN_ORDER;
   GET_CLASS_SEP ;
END;



(.. Procedure to verify if the distribution is done between 0 and 1 ..)

PROCEDURE VERIFY_THE_DISTRIBUTION ;
VAR
   S,I,J : INTEGER ;
BEGIN
   S := 1 ;
   WHILE S > 0 DO
         BEGIN
            IF (MEAN[1] < 0.5) AND (MEAN[FAULT] > 0.5)
               THEN S := 0
               ELSE BEGIN
                       INPUT_OUTPUT_DATA ;
                       MEAN_IN_ORDER ;
                       S := S + 1 ;
                       IF S = 3 THEN S := 0 ;
                    END;
         END;
END;



(.. Procedure to calculate the function of H .....................)

PROCEDURE GET_F_THRLD ;
VAR
   SS,X1,X2,OVER,RX : REAL ;
   I,J : INTEGER ;
BEGIN
   X1  := 0 ;
   FOR I := 1 TO Lmax DO
      BEGIN
         IF MEAN[I] = 0  THEN RX := 0
            ELSE BEGIN
                    OVER := SQR(THRLD-MEAN[I])*ISL ;
                    SS   := MEAN[I] - SQR(MEAN[I]) ;
```

```
                    IF OVER/(2*ss) >= 16 THEN RX := 0
                       ELSE
                          RX :=((1/FAULT)/SQRT(SS))*EXP(-OVER/(2*SS));
                 END;
          X1 := X1 + RX ;
       END;
   X2  := 0 ;
   FOR I := Rmin TO FAULT DO
       BEGIN
          IF MEAN[I] = 1  THEN RX := 0
             ELSE BEGIN
                    OVER := SQR(THRLD-MEAN[I])*ISL ;
                    SS   := MEAN[I] - SQR(MEAN[I]) ;
                    IF OVER/(2*SS) >= 16 THEN RX := 0
                       ELSE
                          RX :=((1/FAULT)/SQRT(SS))*EXP(-OVER/(2*SS));
                 END;
          X2 := X2 + RX ;
       END;
   F_THRLD := X1 - X2 ;
END;



{.. Procedure to calculate the derivative of the function of H ......}

PROCEDURE GET_DF_THRLD;
VAR
   A,B,SS,X1,X2,OVER,RX : REAL ;
   I,J : INTEGER ;
BEGIN
   X1  := 0 ;
   FOR I := 1 TO Lmax DO
       BEGIN
          IF MEAN[I] = 0  THEN RX := 0
             ELSE BEGIN
                    OVER := SQR(THRLD-MEAN[I])*ISL ;
                    SS   := MEAN[I] - SQR(MEAN[I]) ;
                    A := OVER/(2*SS) ;
                    B := (1/FAULT)/SQRT(SS) ;
                    IF A >= 16 THEN RX := 0
                       ELSE  RX := -B*((THRLD-MEAN[I])*ISL/SS)*EXP(-A);
                 END;
          X1 := X1 + RX ;
       END;
    X2  := 0 ;
    FOR I := Rmin TO FAULT DO
       BEGIN
          IF MEAN[I] = 1  THEN RX := 0
             ELSE BEGIN
                    OVER := SQR(THRLD-MEAN[I])*ISL ;
                    SS   := MEAN[I] - SQR(MEAN[I]) ;
                    A := OVER/(2*SS) ;
                    B := (1/FAULT)/SQRT(SS) ;
```

```
                    IF A >= 16 THEN RX := 0
                       ELSE  RX := B*((THRLD-MEAN[I])*ISL/SS)*EXP(-A);
                  END;
          X2 := X2 + RX ;
       END;
   DF_THRLD := X1 + X2 ;
END;



(.. Procedure to find a good H, then to regroup the faults ..........)

PROCEDURE REGROUPING_SET_THRLD;
VAR
   A,H    : ARRAY[1..2] OF REAL ;
   S,I    : INTEGER;
   SAVE_H : REAL;
BEGIN
   S := 1 ;
   WHILE S > 0 DO
      BEGIN
         GET_F_THRLD ;
         IF  ABS(F_THRLD) < 1/100000.
            THEN   S := 0
            ELSE BEGIN
                    S := S + 1 ;
                    IF S = 10 THEN S := 0
                       ELSE BEGIN
                               GET_DF_THRLD ;
                               THRLD := THRLD - F_THRLD/DF_THRLD ;
                            END;
                  END;
       END;
END;



(.. Procedure for the total error calculation ......................)

PROCEDURE GET_TOTAL_ERROR_LR ;
VAR
    I : INTEGER ;
    X,Y,AREA,INTERVAL,L_Error,R_Error,RX,RE : REAL ;
BEGIN
    RE  := 0 ;
    L_Error  := 0 ;
    FOR I := 1 TO Lmax DO
        BEGIN
            IF MEAN[I] = 0  THEN RE := 0
               ELSE BEGIN
```

```
                        X := (THRLD - MEAN[I])/SQRT(VARI[I]) ;
                        Y := (1 - MEAN[I])/SQRT(VARI[I]) ;
                        INTERVAL := ( Y - X )/50 ;
                        AREA := 0 ;
                        WHILE X <= Y DO
                          BEGIN
                             IF ABS(X) >= 8. THEN RX := 0
                                ELSE RX := (EXP(-SQR(X)/2))*INTERVAL ;
                             AREA := AREA + RX ;
                             X := X + INTERVAL ;
                          END;
                        RE := ((1/FAULT)/SQRT(2*PI))*AREA ;
                     END;
            L_Error := L_Error + RE ;
        END;
    RE := 0 ;
    R_Error := 0 ;
    FOR I := Rmin TO FAULT DO
        BEGIN
            IF (MEAN[I] = 1) OR  (MEAN[I] = 0) THEN RE := 0
               ELSE BEGIN
                        X := - MEAN[I]/SQRT(VARI[I]) ;
                        Y := (THRLD - MEAN[I])/SQRT(VARI[I]) ;
                        INTERVAL := ( Y - X )/50 ;
                        AREA := 0 ;
                        WHILE X <= Y DO
                          BEGIN
                             IF ABS(X) >= 8. THEN RX := 0
                                ELSE RX := EXP(-SQR(X)/2)*INTERVAL ;
                             AREA := AREA + RX ;
                             X := X + INTERVAL ;
                          END;
                        RE := ((1/FAULT)/SQRT(2*PI))*AREA ;
                     END;
            R_Error := R_Error + RE ;
        END;
    ERROR_Cal := L_Error + R_Error ;
END;



(.. Procedure to take the closest fault to H out of consideration ..)

PROCEDURE TAKE_OFF_FAULT ;
VAR  LDiff,RDiff,SD : REAL ;
     I,J,CC : INTEGER ;
BEGIN
   IF FAULT - FOverlap = 3 THEN CC := 3
      ELSE IF Lmax = 1 THEN CC := 2
              ELSE IF Rmin = FAULT THEN CC := 1
                      ELSE BEGIN
                             LDiff := THRLD - MEAN[Lmax] ;
                             RDiff := MEAN[Rmin] - THRLD ;
                             IF LDiff < RDiff THEN CC := 1
```

```
                                    ELSE CC := 2 ;
                                END;
          CASE CC OF
1:  BEGIN
       FOverlap := FOverlap + 1 ;
       Lmax := Lmax - 1;
    END;
2:  BEGIN
       FOverlap := FOverlap + 1 ;
       Rmin := Rmin + 1 ;
    END;
3:  BEGIN
       FOverlap := FOverlap + 1 ;
       Lmax := 1 ;
       Rmin := FAULT ;
    END;
          END;
END;



(.. Procedure to find the node information  .....................)

PROCEDURE INFO_MoreFault ;
VAR
   ERROR_BOUND : REAL ;
   I,C,CCC     : INTEGER ;
BEGIN
   CCC := 1 ;
   IF FAULT = FLT  THEN VERIFY_THE_DISTRIBUTION ;
   REPEAT
     FOverlap := 0 ;
     GET_CLASS_SEP ;
     C := 1 ;
     WHILE C > 0 DO
         BEGIN
            REGROUPING_SET_THRLD ;
            GET_TOTAL_ERROR_LR ;
            IF FAULT - FOverlap = 2 THEN ERROR_BOUND := ERROR_ASS
               ELSE ERROR_BOUND := ERROR_ASS /ROUND(LN(Flt)/LN(2)) ;
            IF ERROR_CAL <= ERROR_BOUND
               THEN BEGIN
                       C := 0 ;
                       CCC := 4 ;
                    END
               ELSE BEGIN
                      C := C + 1 ;
                      IF (C = 2) OR (C = 4) THEN ARBITRARY_GROUPING
                         ELSE IF (C = 3) OR (C = 5)
                                 THEN BEGIN
                                         ISL := ISL + SAVE_ISL ;
                                         FOR I := 1 TO N DO
                                               PROB_Xin[I]:= 0.5;
                                         INPUT_OUTPUT_DATA ;
```

```
                                        MEAN_IN_ORDER ;
                                        GET_CLASS_SEP ;
                                    END
                                ELSE IF FAULT-FOverlap > 2
                                        THEN TAKE_OFF_FAULT
                                        ELSE C := 0 ;
                        END;
            END;
    CCC := CCC + 1 ;
    IF CCC < 4 THEN BEGIN
                        FOR I := 1 TO N DO
                              PROB_Xin[I] := 0.1 + RANDOM(8)/10;
                        INPUT_OUTPUT_DATA ;
                        MEAN_IN_ORDER ;
                    END;
      UNTIL CCC >= 4 ;
END;



(.. Procedure to start serching for information .....................)
PROCEDURE GET_INFO ;
VAR I,J1 : INTEGER ;
BEGIN
    FOR I := 1 TO N_Input DO  Prob_Xin[I] := 0.5 ;
    J1 := 1 ;
    REPEAT
          INPUT_OUTPUT_DATA ;
          MEAN_IN_ORDER ;
          IF (MEAN[1]>0) OR (MEAN[FAULT]<1)
             THEN J1 := 0
             ELSE BEGIN
                     J1 := J1 + 1 ;
                     IF J1 = 3 THEN BEGIN
                 WRITELN('...I GENERAT INPUT DATA 3 TIMES AND I
                              STILL GETING');
             WRITELN('  Z[1] AND Z[FAULT] AS ',MEAN[1]:5:3,' ',
                              MEAN[FAULT]);
                                       J1 := 0 ;
                                  END;
                   END;
    UNTIL J1 = 0 ;
    INFO_MoreFault ;
END;



(.. Procedure to save the information in output file.................)

PROCEDURE Save_TreeNode(var S : Node);
var I : integer ;
```

```
Begin
  WRITELN('.....Don't turne me off please, I am still inserting ....');
  WRITELN(INFILE2,S^.NF);
  IF S^.NF = FLT THEN  FOR I := 1 TO FLT DO WRITELN(infile2,S^.F[I]);
  IF S^.NF > 1
     THEN BEGIN
             WRITELN(INFILE2,ISL);
             WRITELN(infile2,S^.TH:5:5);
             FOR I := 1 TO N_INPUT DO WRITELN(infile2,S^.PR[I]:5:5);
          END
     ELSE WRITELN (INFILE2,S^.F[1]);
end;



(.. Procedure to insert information in left subtree ..............)

PROCEDURE INSERT_Left(VAR  q : NODE);
VAR
   S : Node ;
   I : INTEGER ;
BEGIN
   IF q^.Left = NIL THEN
      BEGIN
         NEW(S);
         S^.Left  := NIL ;
         S^.Right := NIL ;
         S^.TH    := THRLD ;
         S^.ZLmax := Lmax ;
         S^.ZRmin := Rmin ;
         S^.Overlap := FOverlap ;
         S^.NF    := FAULT ;
         FOR I := 1 TO S^.NF DO S^.F[I] := Z_Ref[I] ;
         FOR I := 1 TO N_INPUT DO S^.Pr[I] := Prob_Xin[I] ;
         Save_TreeNode(S);
         q^.Left := S ;
  END;
END;



(.. Procedure to insert information in right subtree ..............)

PROCEDURE INSERT_Right(VAR  q : NODE);
VAR
   S : Node ;
   I : INTEGER ;
BEGIN
   IF q^.Right = NIL THEN
      BEGIN
         NEW(S);
         S^.Left  := NIL ;
```

```
          S^.Right := NIL ;
          S^.TH    := THRLD ;
          S^.ZLmax := Lmax ;
          S^.ZRmin := Rmin ;
          S^.Overlap := FOverlap ;
          S^.NF    := FAULT ;
          FOR I := 1 TO S^.NF DO S^.F[I] := Z_Ref[I] ;
          FOR I := 1 TO N_INPUT DO S^.Pr[I] := Prob_Xin[I] ;
          Save_TreeNode(S);
          q^.Right := S ;
      END;
END;



{.. Procedure to insert information in terminal node ..............}

PROCEDURE INSERT_END_NODE(VAR  q : NODE);
VAR
   S : Node ;
   I : INTEGER ;
BEGIN
   NEW(S);
   S^.Left := NIL ;
   S^.Right:= NIL ;
   S^.TH   :=  -1 ;
   S^.ZLmax := -1 ;
   S^.ZRmin := -1 ;
   S^.Overlap := -1 ;
   S^.NF   := 1 ;
   S^.F[1] := Z_Ref[FAULT] ;
   FOR I := 2 TO N_FAULT DO  S^.F[I] := -1 ;
   FOR I := 1 TO N_INPUT DO S^.Pr[I] := -1 ;
   Save_TreeNode(S);
   q := S ;
END;



{.. Procedure to arrange information in preordered form ..............}

PROCEDURE INSERT_PREORDER(VAR  T : NODE);
VAR  I : INTEGER ;
BEGIN
   ISL := SAVE_ISL ;
   FAULT := T^.ZLmax + T^.Overlap ;
   IF FAULT > 1
   THEN BEGIN
          FOR I := 1 TO FAULT DO  Z_Ref[I] := T^.F[I] ;
          GET_INFO ;
          Insert_Left(T) ;
```

```
          INSERT_PREORDER(T^.Left)
         END
   ELSE INSERT_END_NODE(T^.Left);
   FAULT := T^.NF - T^.ZLmax ;
   IF FAULT > 1
   THEN BEGIN
         FOR I := 1 TO FAULT DO  Z_Ref[I] := T^.F[I+T^.ZLmax] ;
         GET_INFO ;
         Insert_Right(T) ;
         INSERT_PREORDER(T^.RIGHT)
      END
   ELSE BEGIN
         FAULT := T^.ZRmin ;
         INSERT_END_NODE(T^.Right);
        END;
END;



(.. Procedure to start constructing the tree .....................)

PROCEDURE CONSTR_TREE ;
VAR  I : INTEGER ;
BEGIN
   ISL := SAVE_ISL ;
   FOR I := 1 TO FAULT DO  Z_Ref[I] := I ;
   GET_INFO ;
   NEW(Root);
   Root^.Left := NIL ;
   Root^.Right:= NIL ;
   Root^.TH := THRLD ;
   Root^.NF := FAULT ;
   Root^.ZLmax := Lmax ;
   Root^.ZRmin := Rmin ;
   Root^.Overlap := FOverlap ;
   FOR I := 1 TO N_FAULT DO  Root^.F[I] := Z_Ref[I] ;
   FOR I := 1 TO N_INPUT DO  Root^.Pr[I] := Prob_Xin[I] ;
   save_TreeNode(Root);  (..... WE HAVE TO FILE THISE DATA ........)
   Insert_Preorder(Root);
END;



(.. THE MAIN PROGRAMM ..........................................)

BEGIN
    Assign(Infile2,'ToRead2S.pas');
    Rewrite(infile2);
    N := N_Input ;
```

```
    WRITE(' SET THE ERROR REQUIRED TO .....  ');
    READ(ERROR_ASS);
    WRITELN(' ');
    WRITE(' SET THE INPUT SEQUENSE TO .....  ');
    READ(SAVE_ISL);
    WRITELN(' ');
    WRITE(' SET THE NUMBER OF FAULTS UNDER TEST .....  ');
    READ(Flt);
    Fault := Flt ;
    WRITELN(' ');
    writeln ('..... oh whawo i am fonctioning .....');
    WRITELN('  ');
    Constr_Tree;
    Close(infile2);
    writeln ('.... finally i finish the tree .....');


(.. This program will save the data in output file (called
    ToRead2s.Pas). However, this file will have the input data for
    the program of testing phase .....}

{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
{:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
END.
```

Appendix B

Appendix : The testing phase program

```
{.. This program shown here enables us to use a preconsrtucted tree
    from appendix A (tree informations are in "ToRead2s.pas" file)
    in this testing phase .

    For the application of this program we simulate a faulty circuit
    (at different lines, once at the time) . The input data for this
    program are :
            - The information of the tree ( ToRead2s.pas file ).
            - The number of the faulty circuit simulation .
                                                                    }

{$U+}

Program TestingTree(Input,Output);
Const
   N_Input = 7  ; { We having 7 input circuit }
   N_Fault = 26 ; { We having 26 possible faults + 2 whene the output
                    is always 0 or 1 }

Type
   Node     = ¬NodeRec ;
   NodeRec = Record
               Left : Node ;       { Pointer to left subtree }
               Right: Node ;       { Pointer to right subtree }
               TH   : Real ;       { Threshold level }
               NF   : Integer ;    { Number of fault of each node }
               L    : INTEGER ;    { Input sequence legth  }
               Fault: Integer ;    { Number of fault under study }
               ZRef : Array [1..N_Fault] Of Integer ;{Fault refference}
               Prob : Array [1..N_Input] Of Real ;{Input probabilities}
             End;


{.. The global varibles that we are going to use are :

   ISL       : Input Sequence Length .
   NFault    : Number of faults under consideration .
   CF        : Counter for fault F.
   RESULT    : The value of this result indicate the faulty line .
   Flt       : To indicate the faulty circuit .
   MEAN      : The mean Z of the output under test .
   THRLD     : Threshold level H .
   Z_Ref     : The references of the faults .
   Prob_Xin  : Probability of the input data .
   InFile    : To read the preconstructed tree data .
   Root      : 1st node of the tree .
   X         : Input data .
                                                                  ..}


Var
   ISL,NFault,CF,RESULT,Flt : Integer ;
```

```
   MEAN,THRLD : REAL ;
   Z_Ref      : ARRAY [1..N_FAULT] OF INTEGER ;
   Prob_Xin   : ARRAY [1..N_INPUT] OF REAL ;
   InFile     : Text ;
   Root       : Node ;
   X          : Array [1..N_Input] Of Integer ;



{  SEVERAL FUNCTIONS USED TO SIMULATE A LOGIC NETWORK          }
{-------------------------------------------------------------}
FUNCTION NORG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 0  THEN NORG := 1  ELSE NORG := 0 ;
  END;
FUNCTION ORG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 0  THEN  ORG := 0  ELSE  ORG := 1 ;
  END;
FUNCTION NANDG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 5  THEN  NANDG := 0  ELSE  NANDG := 1 ;
  END;
FUNCTION ANDG (U,W,X,Y,Z : INTEGER ) : INTEGER ;
  BEGIN
      IF U+W+X+Y+Z = 5  THEN  ANDG := 1  ELSE  ANDG := 0 ;
  END;
FUNCTION NOTG (X : INTEGER ) : INTEGER ;
  BEGIN
      IF X=1 THEN  NOTG := 0  ELSE  NOTG := 1 ;
  END;
FUNCTION EX_ORG (X,Y : INTEGER ) : INTEGER ;
  BEGIN
      IF X=Y THEN  EX_ORG := 0  ELSE EX_ORG := 1 ;
  END;
FUNCTION EX_NORG (X,Y : INTEGER ) : INTEGER ;
  BEGIN
      IF X=Y THEN EX_NORG := 1  ELSE EX_NORG := 0 ;
  END;


{.. A PROCEDURE TO SIMULATE SOME FAULTS IN ALU CIRCUIT  .. }

PROCEDURE SIMULAT_CIRCUIT ;
VAR A,B,D,F : INTEGER;

BEGIN

              CASE  flt OF

{.................. for line (#1,0) or (#10,0) .....................}
1:BEGIN
```

```
A := NORG(0,0,0,0,ANDG(1,1,X[6],X[2],NOTG(X[5])));
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]),ANDG(1,1,1,X[4],X[7]));
F := NORG( 0,0,0,B, ANDG(1,1,1,A,D) );
IF F = 1 THEN CF := CF + 1 ;
END;


{..................... for line (#3,1) ......................}
6:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),1), ANDG(1,1,1,X[4],X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),1), ANDG(1,1,1,X[4],X[7]));
F := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F = 1 THEN CF := CF + 1 ;
END;


{..................... for line (#5,0) ......................}
9:BEGIN
A := NORG(0,0,0,0, ANDG(1,1,X[6],X[2],1) );
B := NORG(0,0,X[3],0,X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[4],X[7]));
F := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F = 1 THEN CF := CF + 1 ;
END;


{..................... for line (#8,0) ......................}
15:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]),0);
B := NORG(0,0,0, ANDG(1,1,1,X[4],X[5]),X[6]);
D := NORG(0,0,0, ANDG(1,1,1,NOTG(X[7]),X[3]), ANDG(1,1,1,X[7],X[4]));
F := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F = 1 THEN CF := CF + 1 ;
END;


{..................... for line (#9,1) ......................}
18:BEGIN
A := NORG(0,0,0, ANDG(1,1,X[5],X[1],X[6]), ANDG(1,1,X[6],X[2],
     NOTG(X[5])) );
B := NORG(0,0, ANDG(1,1,1,NOTG(X[5]),X[3]), ANDG(1,1,1,X[4],X[5]),
     X[6]);
D := NORG(0,0,0,X[3], ANDG(1,1,1,X[4],X[7]));
F := NORG( 0,0,0,B, ANDG(1,1,1,A,D));
IF F = 1 THEN CF := CF + 1 ;
END;
                END;( .. END CASE ...)

END;
```

```
(.. PROCEDURE TO GENERATE THE INPUT SEQUENCE ..)

PROCEDURE INPUT_OUTPUT_DATA ;
VAR  I,P : INTEGER ;
BEGIN
   CF := 0 ;
   FOR P := 1 TO ISL DO
       BEGIN
          FOR I := 1 TO N_Input DO
              BEGIN
                 IF RANDOM(ISL)+1 <= ABS(Prob_Xin[I]*ISL)
                    THEN X[I] := 1
                    ELSE X[I] := 0 ;
              END;
           SIMULAT_CIRCUIT;
       END;
    MEAN := CF/ISL ;
END;



(..PROCEDURE TO VERIFY THE CONTENTS OF A NODE  ..)

PROCEDURE WRITE_NODE( q : NODE );
VAR I : INTEGER ;
BEGIN
   WRITELN ( '   THE NUMBER OF FAULT = ',Q^.NF);
   IF Q^.NF > 1
      THEN  WRITELN('   THE THRLD LEVEL     = ',Q^.TH:5:4)
      ELSE  WRITELN(' THE CIRCUIT UNDER TEST HAS FAULT# ',Q^.FAULT);

END;



(.. PROCEDURE TO INSERT INFORMATION IN THE TREE NONTERMINAL NODE ..)

PROCEDURE INSERT (Var q : Node);
Var
   S : Node ;
   I : Integer ;
BEGIN
   New(S) ;
   S^.Left := NIL ;
   S^.Right:= NIL ;
   S^.NF   := NFault ;
   READLN(INFILE,S^.L);
   Readln(InFile,S^.TH);
   FOR I := 1 TO N_Input DO Readln(InFile,S^.Prob[I]);
```

```
   q := S ;
END;



(.. PROCEDURE TO INSERT INFORMATION IN THE TREE TERMINAL NODE ..)

PROCEDURE  INSERT_END_NODE (Var q : Node);

Var S : Node ;
BEGIN
   New(S) ;
   S^.Left := NIL ;
   S^.Right:= NIL ;
   S^.NF   := NFault ;
   Readln(InFile,S^.Fault);
   q := S ;
END;



(.. PROCEDURE TO RECONSTRUCT THE TREE OF CONSTRUCION PHASE ..)

PROCEDURE  INSERT_PREORDER (Var T : Node);
Var I : Integer ;
BEGIN
   Readln(InFile,NFault);
   If NFault > 1
      Then BEGIN
              INSERT(T^.Left);
              INSERT_PREORDER(T^.Left);
           END
      Else INSERT_END_NODE(T^.Left);
   Readln(InFile,NFault);
   If NFault > 1
      Then BEGIN
              INSERT(T^.Right);
              INSERT_PREORDER(T^.Right);
           END
      Else INSERT_END_NODE(T^.Right);
END;



(.. PROCEDURE TO INSERT INFORMATION IN THE ROOT NODE ..)

PROCEDURE  INSERT_TREE_INFO ;
Var I : Integer ;
BEGIN
   New(Root) ;
```

```
   Root^.Left := NIL;
   Root^.Right:= NIL ;
   Readln(InFile,Root^.NF);
   FOR I := 1 TO Root^.NF DO Readln(InFile,Root^.ZRef[I]);
   READLN(INFILE,Root^.L);
   Readln(InFile,Root^.TH);
   FOR I := 1 TO N_Input DO Readln(InFile,Root^.Prob[I]);
   INSERT_PREORDER(Root);
END;



{.. PROCEDURE FOR THE RESULT OF TESTING ..}

PROCEDURE GET_TEST_RESULT ;
BEGIN
  WRITELN(' ');
  WRITELN(
  '::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::');
  WRITELN(
  '::::                                                              ::::');
  WRITELN(
  '::::     THE RESULT FROM THE BINARY DECISION TREE IS :            ::::');
  WRITELN(
  '::::                                                              ::::');
                CASE  RESULT OF

   1:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line #1 OR #10.... S_A_0 ');
   2:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 1 ......... S_A_1 ');
   3:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line #2 OR #11.... S_A_0 ');
   4:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 2 ......... S_A_1 ');
   5:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 3 ......... S_A_0 ');
   6:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 3 ......... S_A_1 ');
   7:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 4 ......... S_A_0 ');
   8:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 4 ......... S_A_1 ');
   9:   WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 5 ......... S_A_0 ');
   10:  WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 5 ......... S_A_1 ');
   11:  WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 6 ..  ..... S_A_0 ');
   12:  WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 6 ......... S_A_1 ');
   13:  WRITELN(
        '       THE FAULT LOCATION IS THAT Line # 7 ......... S_A_0 ');
```

```
14:   WRITELN(
      '       THE FAULT LOCATION IS THAT Line # 7 ..... S_A_1 ');
15:   WRITELN(
      '       THE FAULT LOCATION IS THAT Line # 8 ..... S_A_0 ');
16:   WRITELN(
      '       THE FAULT LOCATION IS THAT Line # 8 ..... S_A_1 ');
17:   WRITELN(
      '       THE FAULT LOCATION IS THAT Line #9 or #15.... S_A_0 ');
18:   WRITELN(
      '       THE FAULT LOCATION IS THAT Line # 9 ..... S_A_1 ');
19:BEGIN
      WRITELN(
      '       THE FAULT LOCATION IS THAT : ');
      WRITELN(
      '                  Line #10 or #11 or #15 or #16.......S_A_1 ');
      WRITELN(
      '       OR THAT Line #17 or #19 or #21 .............S_A_0 ');
   END;
20:   WRITELN(
      '       THE FAULT LOCATION IS THAT Line # 12..... S_A_0 ');
21:BEGIN
      WRITELN(
      '       THE FAULT LOCATION IS THAT : ');
      WRITELN(
      '                  Line #12 or #13 or #14..............S_A_1 ');
      WRITELN(
      '       OR THAT Line #18 or #20.....................S_A_0 ');
   END;
22:   WRITELN(
      '      THE FAULT LOCATION IS THAT Line # 13 ..........S_A_0 ');
23:   WRITELN(
      '      THE FAULT LOCATION IS THAT Line # 14 ..........S_A_0 ');
24:   WRITELN(
      '      THE FAULT LOCATION IS THAT Line # 16 ..........S_A_0 ');
25:   WRITELN(
      '      THE FAULT LOCATION IS THAT Line # 17 ..........S_A_1 ');
26:   WRITELN(
      '      THE FAULT LOCATION IS THAT Line # 19 ..........S_A_1 ');
27:BEGIN
      WRITELN(
      '      THE FAULT LOCATION IS THAT : ');
      WRITELN(
      '                  Line #18 or #20 or #21..............S_A_1 ');
      WRITELN(
      '      OR THAT Line #22 ..........................S_A_0 ');
   END;
28:   WRITELN(
      '      THE FAULT LOCATION IS THAT Line # 22..........S_A_1 ');

        END; {... End Cases .....}

WRITELN(
':::::                                                        :::::');
WRITELN(
':::::                                                        :::::');
```

```
  WRITELN(
  ':::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::');

END;



{.. PROCEDURE TO OBSERVE THE MEAN OF THE FAULTY CIRCUIT OUTPUT ..}

PROCEDURE GET_MEAN ;
VAR
   COUNT : INTEGER ;
   SAVE_MEAN,SAVE_DIFF,DIFF : REAL ;
BEGIN
   Count := 1 ;
   SAVE_MEAN := 100 ;
   SAVE_DIFF  := 0 ;
   REPEAT
      INPUT_OUTPUT_DATA ;
      DIFF := ABS(MEAN - THRLD) ;
      IF DIFF > SAVE_DIFF
      THEN BEGIN
              SAVE_MEAN := MEAN ;
              SAVE_DIFF := DIFF ;
           END;
      IF DIFF < 1/100
         THEN  BEGIN
                  Count := count + 1 ;
                  IF Count = 4 THEN COUNT := 0 ;
                  WRITELN(' I AM REGENERATING INPUT BECAUSE DIFF = ',
                          DIFF);
               END
      ELSE Count := 0 ;
  UNTIL Count = 0 ;
  MEAN := SAVE_MEAN ;
END;



{.. PROCEDURE FOR THE DECISION TO BE MADE ..}

PROCEDURE DECISION_MAKING ;
VAR
   S : Node ;
   I,LEVEL : Integer ;
BEGIN
   FOR I := 1 TO N_Fault DO Z_Ref[I] := Root^.ZRef[I] ;
   FOR I := 1 TO N_INPUT DO Prob_Xin[I] := Root^.Prob[I] ;
   ISL := Root^.L ;
   S := Root ;
   WRITELN(' ');
```

```
   WRITELN('..... I AM DOING THE Root NODE .....................');
   INPUT_OUTPUT_DATA;
   IF MEAN = 0 THEN RESULT := 27
      ELSE IF MEAN = 1 THEN RESULT := 28
      ELSE BEGIN
             LEVEL := 0 ;
             WHILE S^.NF > 1 DO
                BEGIN
                    IF MEAN < S^.TH
                       THEN S := S^.Left
                       ELSE S := S^.Right ;
WRITELN(' ');
WRITELN('..I FINISH FROM LEVEL <',LEVEL,'> AND I AM IN LEVEL <',
         LEVEL+1,'>...');
                   IF S^.NF > 1
                      THEN BEGIN
                             ISL   := S^.L ;
                             THRLD := S^.TH ;
                             FOR I := 1 TO N_INPUT DO
                                   Prob_Xin[I] := S^.Prob[I] ;
                             GET_MEAN ;
                             LEVEL := LEVEL + 1 ;
                           END;
                END;
WRITELN('.....BUT LEVEL <',LEVEL+1,
         '> HAS NO CALCULATION AND THE ANSWER IS ');
              RESULT := S^.Fault ;
           END;
    GET_TEST_RESULT ;
END;



(.. THE MAIN PROGRAM ..)

BEGIN
   Assign (InFile,'ToRead2S.Pas');
   Reset(Infile);
   INSERT_TREE_INFO ;
   WRITE(' GIVE ME THE NUMBER OF THE CIRCUIT UNDER TEST ..... ');
   READ(Flt);
   WRITELN(' ');
   WRITELN('..... THANK YOU , WELL I AM TESTING IT AND , THEREFORE....')
   DECISION_MAKING ;
   Close(InFile) ;
END.
```