POLYNOMIALLY COMPLETE FAULT DETECTION PROBLEMS

O. H. Ibarra and S. K. Sahni

PRINTED IN THE U.S.A. Annals No. 503CX003

Polynomially Complete Fault Detection Problems

OSCAR H. IBARRA AND SARTAJ K. SAHNI

Abstract—We look at several variations of the single fault detection problem for combinational logic circuits and show that deciding whether single faults are detectable by input–output (I/O) experiments is polynomially complete, i.e., there is a polynomial time algorithm to decide if these single faults are detectable if and only if there is a polynomial time algorithm for problems such as the traveling salesman problem, knapsack problem, etc.

Index Terms—Deterministic and nondeterministic computations, fault detection, irredundant circuit, polynomially complete, polynomial time algorithm, tautology problem, traveling salesman problem, Turing machines (TM's).

I. INTRODUCTION

MUCH ATTENTION (see, e.g., [3] and [5]) has been focused on obtaining fast algorithms to obtain minimal test sets to detect all single stuck-at-zero (s-a-0) and stuck-at-one (s-a-1) faults in combinational logic circuits. The best algorithms known have an asymptotic computing time that is exponential in the number of input lines and gates. Hence these algorithms are computationally feasible only for very small circuits. In fact, it would appear that only algorithms with a computing time linear or at most a square of the number of input lines and gates would be feasible for large combinational circuits (e.g., large-scale integrated circuits). In this paper we show that several fault-detection problems are computationally related to problems such as traveling salesman, knapsack, maximal clique of a graph, multiprocessor job scheduling, intricate network flow problems, etc. Thus, these fault-detection problems can be solved in polynomial time if and only if (iff) the traveling salesman, knapsack problem, etc., can also be solved in polynomial time. Specifically, then, we show that several single fault-detection problems are polynomially complete (see [1], [4], [7], [9], and [10] for further examples of complete problems). Hence, it would appear very unlikely that the fault detection problems have a polynomial algorithm. This would tend to suggest that circuits be designed in some canonical form for which test sets are easily obtainable. The proof technique used in Lemmas 3.2 and 3.3 indicates that by increasing the number of gates it is possible to convert any circuit into an equivalent circuit for which the test set is easily obtainable. See [6] and [8] for some results on designing easily testable circuits.

Manuscript received December 10, 1973; revised June 21, 1974. The research of O. H. Ibarra was supported by the National Science Foundation under Grant GJ-35614. The research of S. K. Sahni was supported by the University of Minnesota under Research Grant 481-0906-4125-02.

The authors are with the Department of Computer Science, University of Minnesota, Minneapolis, Minn. 55455.

In Section II we introduce our notation and then in Section III we show that the following problems are polynomially complete:

Problem 1 (P1): Is the combinational circuit C irredundant (i.e., can all single faults be detected)?

Problem 2 (P2): Can a fault in a particular input line x_i be detected by input–output (I/O) experiments?

Problem 3 (P3): Can all single input faults be detected by I/O experiments?

Problem 4 (P4): Can faults in the output line be detected by I/O experiments.

Problem 5 (P5): Does the circuit C realize the Boolean function B (i.e., is $f_C(x_1,x_2,\cdots,x_n) = B(x_1,x_2,\cdots,x_n)$ for all binary inputs $(x_1,x_2,\cdots,x_n)$, where f_C is the switching function realized by C)?

Problem 6 (P6): Find the minimal circuit (using AND, OR, and NOT gates) that realizes a Boolean function B.

II. NOTATION

Let C be a combinational logic circuit (or simply *circuit*) with n input lines $x_1,x_2,\cdots,x_n$ and 1 output line z (see Fig. 1). We shall only focus our attention to detecting *single* faults which cause any wire to be s-a-0 or s-a-1. Let $\mathcal{L}$ be the set of all possible single fault locations (i.e., all input lines, output line, and input wires to gates). An $(n+3)$-tuple $(x_1 = i_1,\ x_2 = i_2,\ \cdots,\ x_n = i_n,\ z = j,\ F(0),\ (F(1))$ is a *fault detection test* (or simply *test*) for C if it satisfies the following properties.

Property 1: $i_1,i_2,\cdots,i_n,\ j = 0$ or 1.

Property 2: $F(0) \subseteq \mathcal{L}$, $F(1) \subseteq \mathcal{L}$, $F(0) \cap F(1) = \emptyset$, and either $F(0)$ or $F(1)$ is nonempty.

Property 3: C with inputs $x_1 = i_1, \cdots, x_n = i_n$ and output $z = j$ implies a s-a-0 fault in one of the locations specified by $F(0)$ or a s-a-1 fault in one of the locations specified by $F(1)$.

Let $L \subseteq \mathcal{L}$, $L \neq \emptyset$. A set T of tests is called a *fault-detection test set* (or simply *test set*) *for* L if:

Property 4: The union of all the $F(0)$'s of the tests in T is equal to L.

Property 5: The union of all the $F(1)$'s of the tests in T is equal to L. If $L = \mathcal{L}$, then we simply say that T is a *test set* of C. The size of a test set is the number of tests in it.

A circuit C is said to be *irredundant with respect to* $L \subseteq \mathcal{L}$ if it has a test set for L. If $L = \mathcal{L}$, we simply say *irredundant*.

Example 1: The circuit of Fig. 2 is irredundant and has the following test set.

$$\begin{aligned}T = \{ &(y_1 = 1, y_2 = 1, y_3 = 1, z = 0, \{z,y_1,y_2,y_3,y_4\},\emptyset),\\ &(y_1 = 0, y_2 = 1, y_3 = 1, z = 1, \emptyset, \{z,y_1,y_4\}),\\ &(y_1 = 1, y_2 = 0, y_3 = 1, z = 1, \emptyset, \{z,y_2,y_4\}),\\ &(y_1 = 1, y_2 = 1, y_3 = 0, z = 1, \emptyset, \{z,y_3,y_4\})\}.\end{aligned}$$

Thus, if input $y_1 = 1$, $y_2 = 1$, $y_3 = 1$ is applied at the input terminals and the output is $z = 0$, then a single s-a-0 fault must have occurred in one of the locations z,y_1,y_2,y_3,y_4.

Let P be the class of languages accepted by deterministic

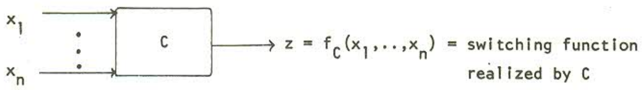

Fig. 1.

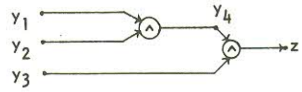

Fig. 2.

Turing machines (DTM's) operating in polynomial time and NP the class of languages accepted by nondeterministic Turing machines (NDTM's) operating in polynomial time (see Hopcroft and Ullman [2] for a detailed discussion of these machines). The problem "Is $P = NP$?" is a longstanding open problem in complexity theory.

Definition 2.1: A problem P_1 is said to be *P-reducible* to the problem P_2 (written $P_1 \alpha P_2$) iff the existence of a polynomial algorithm for P_2 implies the existence of a polynomial algorithm for P_1.

Definition 2.2: Two problems, P_1 and P_2, are *P-equivalent* iff $P_1 \alpha P_2$ and $P_2 \alpha P_1$.

Definition 2.3: P-Complete (PC) is the equivalence class of P-equivalent problems having a deterministic polynomial time algorithm iff $P = NP$.

In [1], Cook showed that $P = NP$ iff there is a polynomial algorithm to determine if a disjunctive normal form (DNF) formula B having at most three literals per clause was a tautology (i.e., deciding if B has the truth value "True" or 1 for all possible assignments of truth values to its variables). In [7], it was shown that it is sufficient to consider only those DNF formulas with exactly three literals per clause (3-DNF).

Thus, 3-DNF is P-Complete. For further examples of P-Complete problems see [1], [4], [7], [9], and [10].

In the next section we show that the problems $P1$–$P6$ of Section I are P-Complete. In most cases the proof proceeds by showing 1) if $P = NP$ then problem P_i has a polynomial algorithm and 2) 3-DNF tautology α problem P_i. 2) together with the knowledge that 3-DNF tautology is P-complete and that α is transitive implies that if P_i has a polynomial algorithm then $P = NP$.

III. POLYNOMIALLY COMPLETE PROBLEMS

In this section, we show that problems $P1$–$P6$ together with other related problems are P-Complete.

We begin with the following lemma.

Lemma 3.1: Consider the circuit of Fig. 3. If C_1 is irredundant with test set T_1 of size m, then C_2 is irredundant with test set T_2 of size m.

Proof: We construct the test set T_2 of C_2 from T_1 using the following rule.

If $(x_1 = i_1, \cdots, x_k = i_k, \cdots, x_n = i_n, z = j, F(0), F(1))$ is in T_1, let $(x_1 = i_1, \cdots, y_k = \bar{i}_k, \cdots, x_n = i_n, z = j, F'(0), F'(1))$ be in T_2, where $\bar{i}_k$ is the complement of i_k and $F'(0)$, $F'(1)$ are defined as follows:

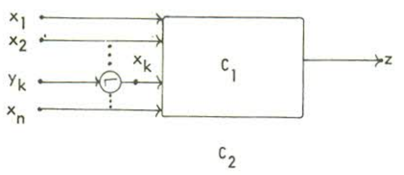

Fig. 3.

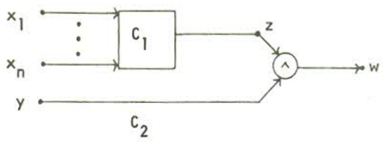

Fig. 4.

1) $F'(0) = F(0)$ and $F'(1) = F(1) \cup \{y_k\}$ if x_k is in $F(0)$.

2) $F'(1) = F(1)$ and $F'(0) = F(0) \cup \{y_k\}$ if x_k is in $F(1)$.

3) $F'(0) = F(0)$ and $F'(1) = F(1)$ if x_k is not in $F(0) \cup F(1)$.

It is obvious that T_2 as constructed is a test set of C_2 and that T_2 is of size m = size of T_1.

Lemma 3.2: Consider the circuit of Fig. 4. If C_1 is irredundant with test set T_1 of size m, then C_2 is irredundant with test set T_2 of size $m + 1$.

Proof: Let T_1 be the test set of C_1. The tests in T_2 are obtained by noting that by setting $y = 1$, the test set for C_2 can be used to test faults in $C_1 \cup \{w\}$ and a s-a-0 fault at y. An extra test for a s-a-1 fault at y is constructed separately. Formally, construct the test set T_2 of C_2 as follows:

1) Let $(x_1 = i_1, \cdots, x_n = i_n, z = 0, F(0), F(1))$ be in T_1. We consider two cases.

Case 1: z is not in $F(0)$. Then let $(x_1 = i_1, \cdots, x_n = i_n, y = 1, w = 0, F(0), F(1))$ be in T_2.

Case 2: z is in $F(0)$, i.e., the test $(x_1 = i_1, \cdots, x_n = i_n, z = 0, F(0), F(1))$ detects a s-a-0 fault at z. Then let $(x_1 = i_1, \cdots, x_n = i_n, y = 1, w = 0, F(0) \cup \{y,w\}, F(1))$ be in T_2. Note that the case z in $F(1)$ is impossible.

2) Now suppose $(x_1 = i_1, \cdots, x_n = i_n, z = 1, F(0), F(1))$ is in T_1. Again consider two cases.

Case 1: z is not in $F(1)$. Then let $(x_1 = i_1, \cdots, x_n = i_n, y = 1, w = 1, F(0), F(1))$ be in T_2.

Case 2: z is in $F(1)$. Then let $(x_1 = i_1, \cdots, x_n = i_n, y = 1, w = 1, F(0), F(1) \cup \{w\})$ be in T_2. We note again that z in $F(0)$ cannot happen.

3) We need a test to detect a s-a-1 fault in y. So let $(x_1 = i_1, \cdots, x_n = i_n, z = 0, F(0), F(1))$ be a test in T_1 such that z is in $F(0)$. Then let $(x_1 = i_1, \cdots, x_n = i_n, y = 0, w = 1, \emptyset, \{y\})$ be in T_2. Clearly, such a test detects a s-a-1 fault in y.

Lemma 3.3: Let $C_1, C_2, C_3, \cdots, C_k$ be irredundant circuits with test sets of size m. Then the circuit C shown in Fig. 5 is irredundant and has a test set T of size $k(m + 1)$.

Proof: Since $C_1, C_2, C_3, \cdots, C_k$ are irredundant with test sets of size m, the circuits $C_1', C_2', C_3', \cdots, C_k'$ are also irredundant and have test sets $T_1, T_2, T_3, \cdots, T_k$ (each of size $m + 1$) by Lemma 3.2. Note that by setting $y_2 = 0, \cdots, y_k = 0$ and using the test set for C_1' we can detect faults in $C_1' \cup \{z_1, \cdots, z_{k-1}, r\}$. Similarly, one may obtain the tests for detecting faults at other points. Thus:

1) If $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l, w_1 = 0, F(0), F(1))$ is a test in T_1, let $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l, y_2 = 0, y_3 = 0, \cdots, y_k = 0, r = 0, F'(0), F(1))$ be a test in T, where $F'(0) = F(0)$ if w_1 is not in $F(0)$ and $F'(0) = F(0) \cup \{z_1, z_2, z_3, \cdots, z_{k-1}, r\}$ if w_1 is in $F(0)$.

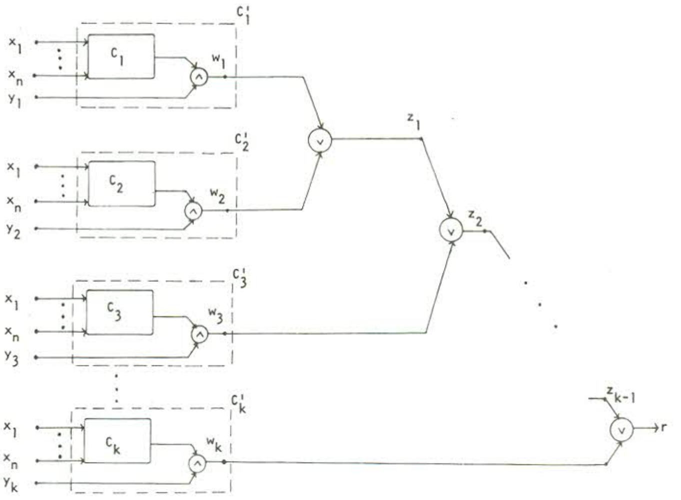

Fig. 5. Circuit C.

2) If $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l, w_1 = 1, F(0), F(1))$ is a test in T_1 let $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l, y_2 = 0, y_3 = 0, \cdots, y_k = 0, r = 1, F(0), F'(1))$ be in T, where $F'(1) = F(1)$ if w_1 is not in $F(1)$ and $F'(1) = F(1) \cup \{z_1, z_2, z_3, \cdots, z_{k-1}, r\}$ if w_1 is in $F(1)$.

Repeat procedures (1) and (2) for $T_2, T_3, \cdots, T_k$. It is straightforward to verify that the test set T constructed as described above is a test set for C. Moreover, the size of T is $k(m + 1)$ if each test set C_i is of size m.

The construction below is used in the proof of Lemma 3.4 which leads to the main result of this paper.

Construction: Consider the circuit Q of Fig. 6. It is composed of circuits C and C', where C is of the form shown in Fig 5. We note the following.

1) The number of AND gates in C exclusive of those in $C_1, C_2, \cdots, C_k$ is k.

2) The number of OR gates in C exclusive of those in $C_1, C_2, \cdots, C_k$ is $k - 1$.

3) The number of AND gates in C' is $k - 1$.

4) The number of OR gates in C' is $(k - 1) + (k - 2) = 2k - 3$.

5) The number of inverters in C' is 1.

6) There is one other OR gate (leading to α) outside C and C'.

If each circuit C_i has exactly two AND gates, no OR gate, at most three inverters (i.e., NOT gates), then Q will have a total of $k + k - 1 + 2k = 4k - 1$ AND gates, a total of $(k - 1) + (2k - 3) + 1 = 3k - 3$ OR gates, and at

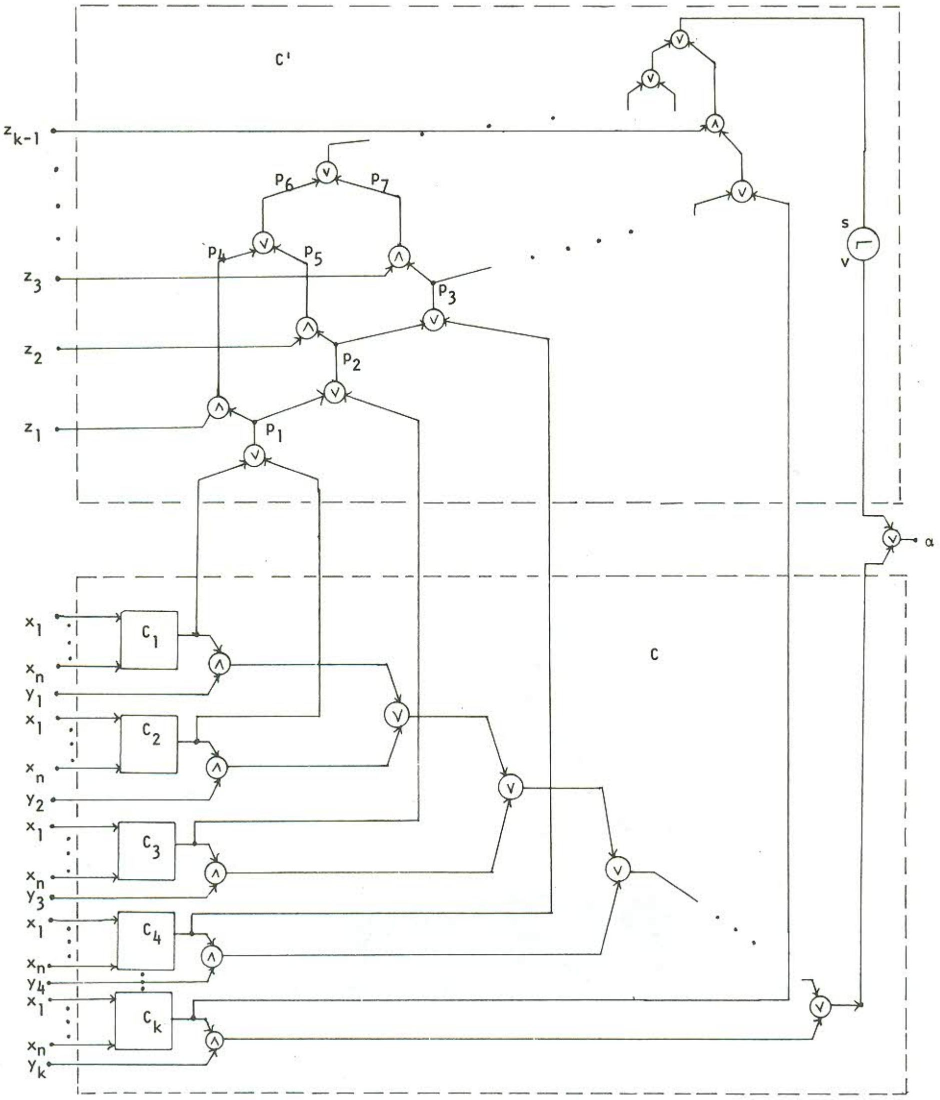

Fig. 6. Circuit Q.

most $3k + 1$ inverters. Thus Q will have at most $(4k - 1) + (3k - 3) + (3k + 1) = 10k - 3$ gates.

7) The number of input lines of Q is $n + k + k - 1 = n + 2k - 1$.

Definition: Consider the circuit C of Fig. 5. Let T be its test set. T is called *valid* if for each test $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l_1, y_2 = l_2, y_3 = l_3, \cdots, y_k = l_k, r = j, F(0), F(1))$ in T, we have $f_1(i_1,\cdots,i_n) + f_2(i_1,\cdots,i_n) + f_3(i_1,\cdots,i_n) + \cdots + f_k(i_1,\cdots,i_n) \geq 1$, where $f_1,f_2,\cdots,f_k$ are the switching functions realized by circuits $C_1,C_2,\cdots,C_k$.

We now prove the following lemma.

Lemma 3.4: Consider the circuit Q of Fig. 6. Let m be the size of the test sets of the C_i's. Let T be a test set of C. If T is valid, then Q is irredundant (with test set TQ of size $k(m + 1) + 2(k - 1) + 1$) if and only if $f_1(x_1,\cdots,x_n) + f_2(x_1,\cdots,x_n) + f_3(x_1,\cdots,x_n) + \cdots + f_k(x_1,\cdots,x_n) = 0$ for some $x_1,\cdots,x_n$, where $f_1,f_2,f_3,\cdots,f_k$ are the switching functions realized by $C_1,C_2,C_3,\cdots,C_k$.

Proof: For ease in exposition, we shall give the proof for the case $k = 4$. The technique is easily extended for any k. For this purpose, we use Fig. 7. Suppose $f_1(x_1,\cdots,x_n) + \cdots + f_4(x_1,\cdots,x_n) \geq 1$ for all $x_1,\cdots,x_n$. Then a s-a-1 fault at p_3 (see Fig. 7) is not detectable. Thus, circuit Q is not irredundant.

Now suppose $f_1(x_1,\cdots,x_n) + \cdots + f_4(x_1,\cdots,x_n) = 0$ for some $x_1,\cdots,x_n$. We show that Q is irredundant by constructing its test set T_Q. (Note that the input lines of Q are $x_1,\cdots,x_n, y_1,\cdots,y_4,z_1,z_2,z_3$.)

We first derive tests for single faults in circuit C.

1) If $(x = i_1, \cdots, x_n = i_n, y_1 = l_1, y_2 = l_2, y_3 = l_3, y_4 = l_4, r = j, F(0), F(1))$ is in T, then let $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l_1, y_2 = l_2, y_3 = l_3, y_4 = l_4, z_1 = 1, z_2 = 1, z_3 = 1, \alpha = j, F'(0), F'(1))$ be in T_Q, where

$$F'(0) = \begin{cases} F(0) & \text{if } r \text{ is not in } F(0) \\ F(0) \cup \{\alpha\} & \text{if } r \text{ is in } F(0) \end{cases}$$

$$F'(1) = \begin{cases} F(1) & \text{if } r \text{ is not in } F(1) \\ F(1) \cup \{\alpha\} & \text{if } r \text{ is in } F(1). \end{cases}$$

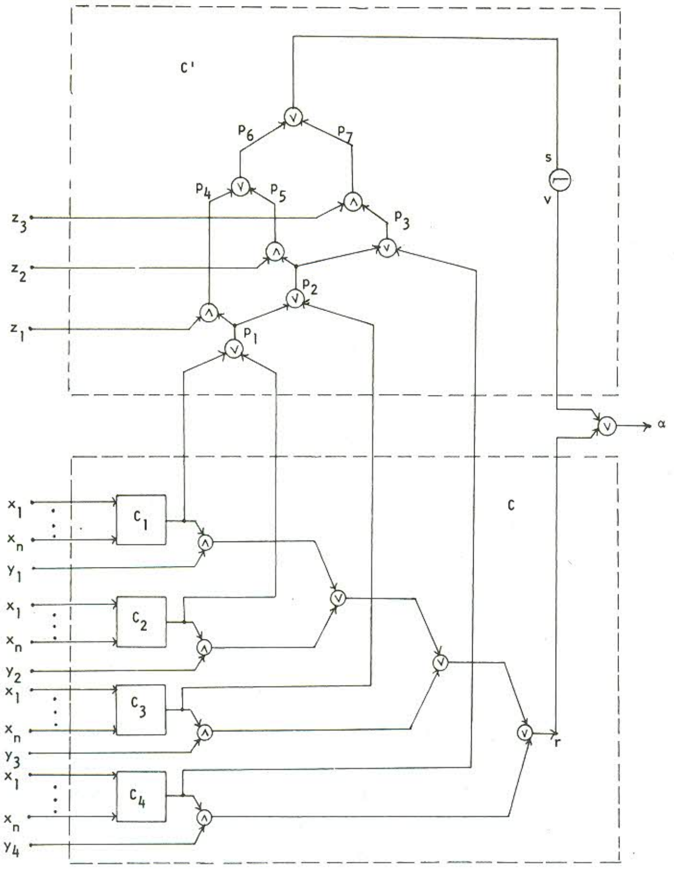

Fig. 7. Circuit Q (with $k = 4$).

The test works since by assumption, T is valid and test $(x_1 = i_1, \cdots, x_n = i_n, y_1 = l_1, y_2 = l_2, y_3 = l_3, y_4 = l_4, z_1 = 1, z_2 = 1, z_3 = 1, r = j, F'(0), F'(1))$ gives a "1" output at s, and hence a "0" output at v.

There remains to show that we can find tests for single faults at $z_1,z_2,z_3,s,v,p_1,p_2,p_3,p_4,p_5,p_6$ and p_7.

2) Since $f_1(x_1,\cdots,x_n) + \cdots + f_4(x_1,\cdots,x_n) = 0$ for some $x_1 = i_1, \cdots, x_n = i_n$, the following test detects a single s-a-0 fault at v or a s-a-1 fault at $s,p_1,p_2,\cdots,p_7$: $(x_1 = i_1, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 1, z_2 = 1, z_3 = 1, \alpha = 0, F(0), F(1))$, where $F(0) = \{v\}$, $F(1) = \{s,p_1,\cdots,p_7\}$.

We construct the s-a-0 fault tests by setting all but one of the z_i's to zero while the s-a-1 tests are obtained by setting them all to zero. Thus,

3) To detect a s-a-0 fault at s,p_1,p_4,p_6,z_1 or s-a-1 fault at v, choose $i_1,\cdots,i_n$ such that $f_1(i_1,\cdots,i_n) = 1$. Then the test is $(x_1 = i_1, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 1, z_2 = 0, z_3 = 0, \alpha = 1, \{s,p_1,p_4,p_6,z_1\}, \{v\})$. Include also the test $(x_1 = i_1, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 0, z_2 = 0, z_3 = 0, \alpha = 0, \emptyset, \{z_1\})$ which detects a s-a-1 fault at z_1.

4) To detect a s-a-0 fault at s,p_3,p_7,z_3 or s-a-1 fault at v, choose $i_1,\cdots,i_n$ such that $f_4(i_1,\cdots,i_n) = 1$. Then the test is $(x_1 = i_1, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 0, z_2 = 0, z_3 = 1, \alpha = 1, \{s,p_3,p_7,z_3\}, \{v\})$. Include also the test $(x_1 = i_n, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 0, z_2 = 0, z_3 = 0, \alpha = 0, \emptyset, \{z_3\})$ which detects a s-a-1 fault at z_3.

5) To detect a s-a-0 fault at s,p_2,p_5,p_6,z_2 or s-a-1 fault at v, choose $i_1,\cdots,i_n$ such that $f_3(i_1,\cdots,i_n) = 1$. Then the test is $(x_1 = i_1, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 0, z_2 = 1, z_3 = 0, \alpha = 1, \{s,p_2,p_5,p_6,z_2\}, \{v\})$. Include also the test $(x_1 = i_1, \cdots, x_n = i_n, y_1 = 0, y_2 = 0, y_3 = 0, y_4 = 0, z_1 = 0, z_2 = 0, z_3 = 0, \alpha = 0, \emptyset, \{z_2\})$ which detects a s-a-1 fault at z_2.

There are $4(m + 1)$ tests in (1), one test in 2), two tests in 3), two tests in 4), and two tests in 5). It follows that T_Q is of size $4(m + 1) + 2(4 - 1) + 1$.

In general, if there are k circuits $C_1,\cdots,C_k$ each with test set of size m, then T_Q is of size $k(m + 1) + 2(k - 1) + 1$.

We are now ready to prove the main result of this section.

Theorem 3.1:

$$P1 \in PC.$$

Proof: First we show that $P1 \in NP$, i.e., if languages accepted by NDTM's operating in polynomial time are also accepted by DTM's operating in polynomial time then there is polynomial time algorithm for $P1$. (Here, polynomial algorithm means polynomial in the number of gates and input lines.) Construct a NDTM that systematically generates (one at a time) all circuits C_i arising from a single fault in circuit C, and checks each one for non-

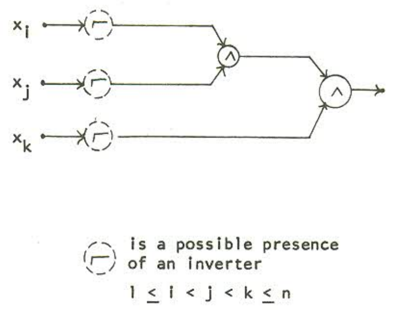

Fig. 8.

equivalence with C. (Note that there are at most $6m$ such circuits.) To check whether C_i is not equivalent to C, the NDTM nondeterministically chooses an input combination and determines whether or not C and C_i give the same output. If C and C_i give the same output, the NDTM halts. Then NDTM accepts if every C_i is not equivalent to C. Clearly, if $NP = P$, then a polynomial algorithm for $P1$ exists.

To complete the proof it is sufficient to show that a polynomial algorithm for $P1$ implies a polynomial algorithm for the tautology problem, i.e., tautology α $P1$. So assume that we have a polynomial algorithm for $P1$. We describe a polynomial algorithm for tautology. (Here polynomial algorithm means polynomial in the number of clauses and variables.)

Given any formula B in 3-DNF with k clauses $B_1, B_2, \cdots, B_k$ and variables $x_1, \cdots, x_n$, perform the following steps:

Step 1: Construct circuits $C_1, C_2, \cdots, C_k$ from $B_1, B_2, \cdots, B_k$. Since each B_i contains exactly three literals, each C_i will be of the form shown in Fig. 8.

From Example 1 and Lemma 3.1, each C_i is irredundant and has test set of size $m = 4$.

Step 2: Using Lemmas 3.2 and 3.3, construct the circuit of Fig. 5. The circuit will have $k(m + 1) = 5k$ tests in its test set T.

Step 3: Determine if T is valid. (There are only $5k$ tests in T. Thus this step can be done easily.) If T is not valid then B is not a tautology, in this case stop. Otherwise, perform Step 4.

Step 4: Construct the circuit Q of Fig. 6.

Step 5: Use the polynomial algorithm for $P1$ to decide whether or not Q is irredundant. B is not a tautology if and only if Q is irredundant (by Lemma 3.4).

Now, Q has at most $10k - 3$ gates and $(n + 2k - 1)$ input terminals. If the algorithm for $P1$ is polynomial in $N = (10k - 3) + (n + 2k - 1) = 12k + n - 4$, it follows that the algorithm we have just described is polynomial in $(k + n)$.

Lemma 3.5: If $B(x_1, \cdots, x_n)$ is a formula from the propositional calculus whose truth value is independent of the truth value of its variables taken one at a time (i.e., $B(x_1, \cdots, x_i, \cdots, x_n) = B(x_1, \cdots, \bar{x}_i, \cdots, x_n)$ for all $x_1, \cdots, x_{i-1}, x_{i+1}, \cdot, x_n$ for $1 \leq i \leq n$), then either 1) $B(x_1, \cdots, x_n) = 0$ for all $(x_1, \cdots x_n)$ or 2) $B(x_1, \cdots, x_n) = 1$ for all $(x_1, \cdots, x_n)$.

Proof: Obvious.

Theorem 3.2:

$$P2 \in PC.$$

Proof: 1) $P2 \in NP$:

A fault in line x_i is detectable iff there is an input $(x_1, x_2, \cdots, x_i, \cdots x_n)$ such that $f(x_1, x_2, \cdots, x_i, \cdots, x_n) \neq f(x_1, x_2, \cdots, \bar{x}_i, \cdots x_n)$. We construct a NDTM, T, that guesses an input combination $(x_1, \cdots, x_n)$ and verifies that $f(x_1, \cdots, x_i, \cdots, x_n) \neq f(x_1, \cdots, \bar{x}_i, \cdots, x_n)$ for the given circuit. The verification can be done in polynomial time. T, then, accepts the circuit description iff line x_i faults are detectable.

2) Tautology α *P2:*

Given any formula B in 3-DNF and with variables $x_1, \cdots, x_n$ and clauses $B_1, \cdots, B_k$ we construct the circuit C with inputs $x_1, \cdots, x_n$ such that $B(x_1, \cdots, x_n) = f_C(x_1, \cdots, x_n)$. This construction can be carried out in an obvious manner in an amount of time linear in n and k. A fault in line x_i is detectable iff $f_C(x_1, \cdots, x_i, \cdots, x_n) \neq f_C(x_1, \cdots, \bar{x}_i, \cdots, x_n)$ for some $(x_1, \cdots, x_n)$.

Hence 1) if a fault in line x_i is detectable then B is not a tautology. 2) If it is not detectable then f_C and hence the value of B does not depend on x_i.

The following algorithm shows how one can solve the tautology problem in polynomial time if $P2$ is polynomial solvable.

Algorithm T2

Step 1: For B a 3-DNF formula construct a AND/OR/NOT circuit C realizing it.

Step 2: (Check if input faults of C are detectable.) Using the polynomial algorithm for $P2$ determine if input faults in line x_i are detectable by I/O experiments. Do this step for $1 \leq i \leq n$. If faults in any of the input lines are detectable then B is not a tautology, stop. Otherwise, from the lemma it follows that B is a tautology iff $B(1, \cdots, 1) = 1$. So compute $B(1, \cdots, 1)$ and decide the status of B.

Clearly if $P2$ can be solved in polynomial time, then

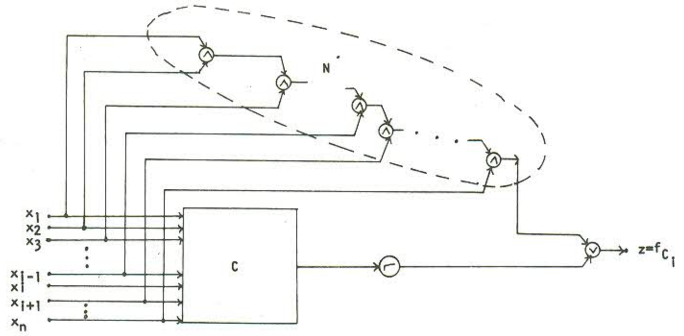

Fig. 9. Circuit C_i.

using Algorithm $T2$ would result in a polynomial solution to the tautology problem. Hence tautology α $P2$.

Theorem 3.3:

$$P3 \in PC.$$

Proof: 1) $P3 \in NP$:

If all single input faults are detectable by I/O experiments then there is a test set of size $\leq 2n$ where n is the number of input lines. We can construct a NDTM, T, that guesses a test set of size $\leq 2n$ and then using the circuit diagram for C verifies that this test set does indeed detect all single input faults. Clearly T can be constructed so as to operate in nondeterministic polynomial time.

2) Tautology α $P3$:

Given a 3-DNF formula $B(x_i,\cdots,x_n)$ construct in polynomial time an AND/OR/NOT circuit, $C(x_i,\cdots,x_n)$, realizing B. From C obtain the circuit C_i as in Fig. 9.

Note that in Fig. 9 the set of AND gates N does not connect with the input line x_i (for some i) of C but connects with every other input line.

Algorithm $T3$, below, shows how to decide in polynomial time if B is a tautology or not, using the construction outlined above, the tests $t_0,\cdots,t_n$ below, and a polynomial algorithm for $P3$.

1) $t_0 = (x_1 = 1,\ x_2 = 1,\ \cdots,\ x_n = 1,\ z = 0,\ \{x_1,\cdots, x_{i-1},x_{i+1},\cdots x_n\},\ \emptyset)$.

2) $t_k = (x_1 = 1,\ \cdots,\ x_{k-1} = 1,\ x_k = 0,\ x_{k+1} = 1,\ \cdots,\ x_n = 1,\ z = 1,\ \emptyset,\ \{x_k\})$ for $1 \leq k \leq n$.

Algorithm T3

Step 1: Verify that for the inputs corresponding to the $n+1$ tests $t_m (0 \leq m \leq n)$ the formula B is true. If this is not the case, stop as B is not a tautology. Obtain the realization C, of B.

Step 2: For $i = 1,2,\cdots,n$ construct C_i from C as described above. Using the polynomial algorithm for $P3$ decide if all single input faults of C_i are detectable by I/O experiments. If for some i, they are then B is not a tautology. Otherwise, by Lemma 3.5, B computes a constant which must be 1 (because of Step 1). Hence B is a tautology. [Now, for C_i, it should be clear that the tests t_m $(0 \leq m \leq n)$ and $m \neq i$ detect all single input faults except those in line i. For inputs other than those corresponding to t_0 the output at the end of the AND set N is 0 (assuming that the single fault is in the input line x_i). Hence f_{C_i} is essentially $\bar{B}$ for $(x_1,\cdots,x_n) \neq (1,\cdots,1)$. Hence a fault in line i is detectable iff there is an input $(x_1,\cdots,x_i,\cdots,x_n)$ such that $\bar{B}(x_1,\cdots,\bar{x}_i,\cdots,x_n) \neq \bar{B}(x_1,\cdots,x_i,\cdots,x_n)$. So, if all input faults in C_i are detectable, then B is not a tautology. If all are not detectable then it is the line i fault that is not detectable and so B is independent of x_i.]

Clearly, then, if $P3$ is polynomial solvable so then also is the tautology problem, i.e., tauthology α $P3$.

Theorem 3.4:

1) $P4 \in PC$

2) $P5 \in PC$

3) $P6 \in PC$.

Proof: Statement 1) follows from the fact that an output fault is detectable iff there are two inputs $x_1,\cdots,x_n$ and $y_1,\cdots,y_n$ such that $f_C(x_1,\cdots,x_n) \neq f_C(y_1,\cdots,y_n)$, i.e., iff B is not a tautology.

The proof for 2) proceeds by showing $P5 \in NP$ and tautology α $P5$ (i.e., given a formula B obtain its circuit realization C and then determine if C realizes the switching function 1).

The proof for 3) is very similar to the proof of Theorem $B1$ of [7] which shows that obtaining the minimal equivalent Boolean form of a formula B is P-complete.

Finally, we consider the following problem. Let L be a set of fault locations. Suppose we know that a circuit C is irredundant with respect to L and we wish to know how long it will take to find a test set for L. The following theorem shows that this problem is complete for several cases of L.

Theorem 3.5: The problem of finding for an arbitrary circuit C, irredundant with respect to L, a test set for L is complete for each of the following cases.

Case 1: L = set of all fault locations.

Case 2: $L = \{x\}$, x is an arbitrary input line.

Case 3: $L = \{x_1,\cdots,x_n\}$, x_i's are input lines.

Case 4: $L = \{z\}$, z is the output line.

Proof: It is easy to show (using a technique similar to that of Theorem 3.1) that $NP = P$ implies a polynomial time algorithm for all the cases above. We now show that a polynomial algorithm for finding a test set for L assuming C is irreducible with respect to L can be used to develop a polynomial time algorithm for deciding for an arbitrary circuit C' whether or not C' is irredundant with respect to L. The polynomial completeness of Cases 1–4 will then follow from Theorems 3.1–3.4.

Assume that we have a polynomial time Algorithm A that finds for an arbitrary circuit C, irredundant with respect to L, a test set for L. The following polynomial time algorithm decides for an arbitrary circuit C' whether or not C' is irredundant with respect to L. [$p(\cdot)$ is the polynomial time bound of A.]

Algorithm B

Step 1: Apply Algorithm A to C'.

Step 2: If A does not halt on C' after $p(\cdot)$ steps, C' cannot be irredundant with respect to L.

Step 3: If A halts on C' in less than or equal to $p(\cdot)$ steps, consider two cases.

Case 1: A halts with no output or halts with outputs that do not satisfy the definition of tests. In this case, C' must not be irredundant with respect to L.

Case 2: A halts with tests $t_1,t_2,\cdots,t_k$. For each such test $(x_1 = i_1, \cdots, x_n = i_n, z = j, F(0), F(1))$, check that if C with input $x = i_1, \cdots, x_n = i_n$ gives output $z = j$ then the locations in $F(0)$ and $F(1)$ are really s-a-0 and s-a-1 detectable faults. If every test checks out, the union of $F(0)$'s $= L$ and the union of the $F(1)$'s $= L$, then C' must be irredundant with respect to L; otherwise C' is not irredundant with respect to L.

Clearly, Algorithm B works in polynomial time if A does and B decides whether or not circuit C is irredundant with respect to L.

Hence it follows that Cases 1–4 are P-Complete.

IV. CONCLUSIONS

We have linked the single fault detection problem to the $P = NP$ problem of complexity theory. As a result of this it is clear that finding a polynomial algorithm for the fault-detection problem is as hard as finding similar algorithms for several combinatorial problems (traveling salesman, knapsack, etc.). In view of this, it would appear that no polynomial algorithm for single fault detection exists.

REFERENCES

[1] S. A. Cook, "The complexity of theorem proving procedures," in *Conf. Rec. 3rd ACM Symp. Theory of Computing*, 1971, pp. 151–158.

[2] J. E. Hopcroft and J. D. Ullman, *Formal Languages and their Relation to Automata*. New York: Addison-Wesley, 1969.

[3] H. E. Jauch, "A new algorithm for detecting faults," Ph.D. dissertation, Dep. Elec. Eng., Univ. Minnesota, Minneapolis, Dec. 1973.

[4] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York: Plenum, 1972, pp. 85–104.

[5] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1970.

[6] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Trans. Comput.*, vol. C-21, pp. 1183–1188, Nov. 1972.

[7] S. Sahni, "Some related problems from network flows, game theory and integer programming," in *Conf. Rec. 13th Annu. IEEE Symp. Switching and Automata Theory*, Oct. 1972.

[8] K. K. Saluja and S. M. Reddy, "Multiple faults in Reed-Muller canonic networks," in *Conf. Rec. 13th Annu. IEEE Symp. Switching and Automata Theory*, Oct. 1972.

[9] R. Sethi, "Complete register allocation problems," in *Conf. Rec. 5th ACM Symp. Theory of Computing*, 1973.

[10] J. D. Ullman, "Polynomial complete scheduling problems," in *Proc. IBM Conf. Operating Systems*, Yorktown Heights, N. Y., Oct. 1973, pp. 96–101.

Oscar H. Ibarra was born in Negros Occidental, Philippines, on September 29, 1941. He received the B.S. degree in electrical engineering from the University of the Philippines, Quezon City, in 1962. He received the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, in 1965 and 1967, respectively.

From 1967–1969 he was an Acting Assistant Professor in the Department of Electrical Engineering and Computer Sciences at the University of California, Berkeley. He joined the Department of Computer, Information, and Control Sciences at the University of Minnesota, Minneapolis, in 1969 where he is currently an Associate Professor. His research interests include automata theory, formal languages, and computational complexity.

Dr. Ibarra is a member of the Association for Computing Machinery and the Society for Industrial and Applied Mathematics.

Sartaj K. Sahni was born in Poona, India on July 22, 1949. He received the B.Tech (elec. eng.) degree from the Indian Institute of Technology, Kanpur, India in 1970 and the M.S. and Ph.D. degrees in computer science from Cornell University, Ithaca, N. Y., in 1972 and 1973, respectively.

He is currently an Assistant Professor of Computer Science at the University of Minnesota, Minneapolis. His research interests include the design and analysis of computer algorithms.