IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 29, 2020, accepted September 14, 2020, date of publication September 22, 2020, date of current version October 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3025825

CTFTP: A Test Case Generation Strategy for General Boolean Expressions Based on Ordered Binary Label-Driven Petri Nets

HONGFANG GONG[1], JUNYI LI[2], AND RENFA LI[2], (Senior Member, IEEE)
[1]School of Mathematics and Statistics, Changsha University of Science and Technology, Changsha 410114, China
[2]College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China

Corresponding author: Hongfang Gong (ghongfang@126.com)

This work was supported in part by the National Natural Science Foundation of China through the Key Project under Grant 61932010, in part by the National Natural Science Foundation of China under Grant 61972055, and in part by the Science Research Foundation (SRF) of Hunan Provincial Education Department of China through the Key Project under Grant 18A145.

ABSTRACT Boolean expression testing requires certain types of tests for each Boolean expression in program specification or implementation. Fault-based testing essentially uses a subset of the exhaustive test set to detect certain special types of faults. A fault-based Boolean expression testing strategy called constraint true and false test point (CTFTP) is proposed. The test consists of two test case generation strategies, namely a unique constraint true point (UCTP) strategy and a near constraint false point (NCFP) strategy. An ordered binary label-driven Petri net model is presented to analyze the interaction between Boolean transitions and Boolean literals and yield the test paths of a singular term for the irredundant disjunctive normal forms (IDNFs). On the basis of the test paths, we develop a configuration-based IDNF test generation algorithm, which is employed to obtain the UCTP, NCFP, and CTFTP test sets for the IDNFs. The proposed test generation algorithm based on literal substitution is applied to extend the CTFTP strategy and generate a test suite for general Boolean expressions, which are evaluated using TCAS II specifications. Experimental results show that the CTFTP strategy can detect the same seven types of faults similar to the MUMCUT strategy when testing IDNFs, but only a subset of the MUMCUT test set is required. Five types of faults of general Boolean expressions can also be detected using CTFTP strategies.

INDEX TERMS Automatic test cases generation, fault-based testing, general Boolean expression test, ordered binary label-driven Petri net, path-oriented test criteria.

I. INTRODUCTION

The correctness and validity of Boolean expressions lay the foundation for the correctness and robustness of those software applications. Boolean expression testing requires certain types of tests for each Boolean expression in program specification or implementation. Given a Boolean expression with n variables, an exhaustive testing requires 2^n different test cases, and the test size will grow exponentially as the number of variables increases [1]. Selecting subsets from all possible test cases based on the test criteria can yield small and effective test suites, but their fault detection capabilities are reduced relative to exhaustive testing [2]. Therefore, new test strategies need to be introduced in view of ensuring that a given Boolean expression can be tested thoroughly, thus maximizing the fault detection capabilities while keeping the number of test cases as small as possible.

Test case generation strategies for Boolean expressions have attracted considerable attention over the past two decades. Tai and Su [3] proposed two test case generation algorithms to ensure that operator errors could be detected. Weyuker *et al.* [4] designed a family of meaningful impact (MI) strategies that can generate test cases automatically for a given Boolean specification, among which the MAX-B strategy is the most powerful because it subsumes all the other strategies in the family. MI strategies have been applied to the fault detection of the irredundant disjunctive normal form (IDNF) of Boolean expressions and exhibited extremely effective detection capabilities. A Boolean expression in disjunctive normal form is said to be *irredundant* if

The associate editor coordinating the review of this manuscript and approving it for publication was Shouguang Wang.

none of its terms can be omitted from the expression and none of its literals can be omitted from any term in the expression [5]. Chen and Lau [6],[7] proposed three test case selection strategies, namely, multiple unique true point (MUTP), multiple near false point (MNFP), and the corresponding unique true point (UTP) and near false point (NFP) pair (CUTPNFP) strategies; in this manner, two types of faults in the Boolean expressions, namely, literal insertion faults (LIFs) and literal reference faults (LRFs), can be detected. The above strategies and the MI strategies can detect seven types of faults in IDNFs, but the aforementioned three strategies are more cost-effective than the MAX-B strategy in detecting faults because the selected test cases typically form a subset of those cases selected by the MAX-B strategy [7]. Chen *et al.* [5] integrated the three strategies into the MUMCUT strategy, which requires fewer test cases than the MAX-A and MAX-B strategies when detecting the same seven types of faults. However, all the aforementioned strategies are suitable only for testing Boolean expressions in some restricted forms, such as IDNF, but not the original expressions. Testing a general form of the Boolean expression by using IDNF-oriented strategies always results in excessive test costs and misses the detection of certain faults.

Software practitioners are more likely to write conditions and logical decisions in general form rather than in IDNF, but their approach may introduce faults in the general form context [8]. A single fault in the general Boolean expression may cause more than one fault in the corresponding equivalent restricted form. Chen *et al.* [8] confirmed experimentally that the MUMCUT strategy has high efficiency in detecting the faults of general Boolean expressions and their mutation expressions and is more effective in detecting the faults of the original expressions than those of the mutants. Following the work of Chen *et al.* [8], Sun *et al.* [9] presented the characteristics of undetected faults of general Boolean expressions and their certainty of being undetected by the MUMCUT, then they analyzed why a MUMCUT test suite would fail to detect five undetected mutation patterns by means of forward/reverse MUMCUT experiments for fault-based testing of general Boolean expressions. However, mutation patterns may be undetected for each type of fault in a general Boolean expression. Sun *et al.* [10] evaluated and compared 18 fault-based testing strategies by means of a series of experiments, using more than 4000 randomly generated fault-based general Boolean expressions. Their experiments showed that the family of fault-based testing strategies, such as the MUMCUT, would usually provide the best performance.

Kapoor and Bowen [11] calculated the necessary and sufficient conditions to detect ten different types of fault categories of general Boolean expressions and proved the fault hierarchy relationship. However, their study overlooked the possibility of a mutant of the Boolean specifications as an equivalent in the testing. Hence, each of the fault relationships was either incorrect or presented an incorrect proof. Chen *et al.* [12] used counterexamples to reveal the incorrect fault relationships and provide new proof to validate fault relationships. A co-stronger fault relation was introduced to establish a new fault class hierarchy for general Boolean specifications. Gargantini and Fraser [13] proposed a test case generation approach for general Boolean expressions. The approach was a mutation-based method designed to detect all the ten types of faults studied in [11] and [12]. Wang *et al.* [14] proposed the minimal failure-causing schema (MFS) and probabilistic failure-causing schema (PFS) to describe the characteristics of the failure test cases for general Boolean specifications. The experiments results based on TACS II specifications [15] indicated that the PFS model is more competitive than the MFS model for the input-level fault localization scenario. In [16], the fault detection ability of the combinatorial test was verified to be better than that of the random test by means of the fault detection experiment of general Boolean expressions. However, the mutation analysis technology involved excessive test costs and yielded inaccurate test results [17].

Petri nets are a promising tool for describing and studying systems that are characterized as concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic [18]. Gong and Huang [19] proposed a generalized Boolean operator (BOR)-MI strategy based on predicate driven Petri nets to generate test cases for general Boolean expressions. However, the BOR-MI strategy can only detect BOR faults. In this study, we focus on detecting faults in Boolean expressions by developing an extended Petri net approach and generating test cases directly from a given general form of a Boolean expression. An ordered binary label-driven Petri net (OBLDPN) is developed to analyze the interaction between Boolean transitions and Boolean literals and yield test paths of a singular term for IDNFs. A Boolean transition is an intuitive representation of the logical relationship between an input and an output represented as a Boolean expression (Fig.1). By using OBLDPN, we propose a fault-based test strategy called the constraint true and false test point (CTFTP) strategy consisting of two test case generation strategies, namely, a unique constraint true point (UCTP) strategy and a near constraint false point (NCFP) strategy. A configuration-based IDNF test generation algorithm is further proposed to obtain the UCTP, NCFP, and CTFTP test sets for the IDNFs. The proposed CTFTP strategy is used to test the IDNF of the Boolean expressions in view of guaranteeing the detection of seven types of faults the same as those of the MUMCUT strategy, but the test set used by the CTFTP strategy is smaller.

Then, the CTFTP strategy is extended to directly test the general Boolean expressions. A general Boolean expression can be expressed as a Boolean function in the form of ''sum of products of sums,'' which is composed of multiple IDNF subexpressions [20]. When the CTFTP strategy is applied to test a general Boolean expression, a test set of each sub-expression of the Boolean expression is generated. Subsequently, an algorithm called general Boolean expression test generation based on literal substitution is developed to generate the required test cases for the general Boolean expressions. We use the TCAS II specifications to evaluate

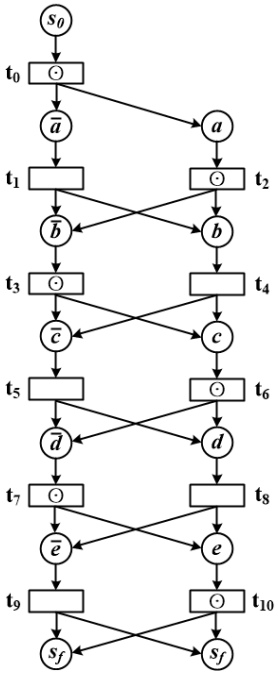

FIGURE 1. The OBLDPN of Boolean expressions involving five variables.

the CTFTP test strategy. The empirical result shows that the CTFTP strategy can detect five types of faults for a general form of a Boolean expression.

The rest of this article is organized as follows. In Section II, the UCTP, NCFP, and CTFTP strategies for generating the IDNF test cases are proposed. In Section III, we define an OBLDPN model and describe its dynamic properties by means of the behavior function. In Section IV, an algorithm is proposed for generating automatically a CTFTP test set for the IDNF, and a set of test adequacy criteria is provided for the path coverage criterion based on the OBLDPN. In Section V, we extend the CTFTP test strategy to generate test cases for the general Boolean expression and present a general Boolean expression test generation algorithm based on literal substitution. Section VI reports the experimental results on the basis of the TCAS II specifications. The related works are described in Section VII. The conclusions are given in Section VIII.

II. CTFTP TEST STRATEGY

A. NOTATION AND TYPES OF FAULTS

We follow the notation and terminology proposed in [1], [5], and [21]. In this study, the BORs of AND (or "$\wedge$"), OR ("$\vee$"), and NOT (or "$\neg$") are denoted as "$\cdot$", "$+$", and "$-$", respectively. If the context is clear, then the operator "$\cdot$" can be omitted. The set of all truth values is represented by B, that is, $B = \{0, 1\}$. The n-dimensional Boolean space is denoted by B^n. In Boolean expressions, a *positive* literal or a *negative* literal may denote an occurrence of a Boolean variable. For example, b and $\overline{b}$ represent the positive and negative literals in the Boolean expression $bc + \overline{b}d$, respectively.

TABLE 1. Test Point and Test Set of (2) in B^n.

A test point $\vec{t}$	Conditions	Test set	Express
True point of f	$f(\vec{t}) = 1$	All true points of f	$TP(f)$
False point of f	$f(\vec{t}) = 0$	All false points of f	$FP(f)$
True point of p_i	$p_i(\vec{t}) = 1$	All true points of p_i	$TP_i(f)$
UTP of p_i	$p_i(\vec{t}) = 1$, and $p_j(\vec{t}) = 0, j \neq i$	All UTPs of p_i	$UTP_i(f)$
False point of p_i	$p_i(\vec{t}) = 0$	All false points of p_i	$FP_{i,\bar{j}}(f)$
NFP of p_i	$p_{i,\bar{j}}(\vec{t}) = 1$ and $f(\vec{t}) = 0$	All NFPs of p_i	$NFP_{i,\bar{j}}(f)$

A Boolean expression with n variables uniquely defines a Boolean function $f : B^n \to B$ but not vice versa. We will not distinguish a Boolean function from a Boolean expression in this article. Here, suppose a general Boolean function $G(\vec{x})$ can be expressed in the form of "sum of products," that is,

$$G(\vec{x}) = f_1(\vec{x}) + f_2(\vec{x}) + \cdots + f_k(\vec{x}), \tag{1}$$

where the Boolean vector $\vec{x} = (x_1, x_2, \cdots, x_n) \in B^n$, k is the total number of terms, and $f(\vec{x})$ is the i-th term. In $G(\vec{x})$, each term $f_i(\vec{x})$, $1 \leq i \leq k$ may be expressed in the form of "product of sums," which contains some sub-expressions in the IDNF.

We first consider the test case generation of IDNFs in testing the general Boolean expressions. Suppose the IDNF $f(\vec{x})$ with n variables is represented as

$$f(\vec{x}) = p_1(\vec{x}) + p_2(\vec{x}) + \cdots + p_m(\vec{x}), \tag{2}$$

where m is the number of terms, and the i-th term $p_i(\vec{x})$, $1 \leq i \leq m$ is a singular term that does not contain the operator "$+$" and occurs only once for each variable. Let

$$p_i(\vec{x}) = x_1^i x_2^i \cdots x_j^i \cdots x_{k_i}^i, 1 \leq k_i \leq n, \tag{3}$$

where k_i is the number of literals and expressed as $|p_i| = k_i$, x_j^i in which the j-th literal occurs, and all literals appear in lexicographical order. By negating the literal x_j^i of $p_i(\vec{x})$, we obtain

$$p_{i,\bar{j}}(\vec{x}) = x_1^i x_2^i \cdots \bar{x}_j^i \cdots x_{k_i}^i, 1 \leq j \leq k_i. \tag{4}$$

For simplicity, we abbreviate the IDNF $f(\vec{x})$ as $f = p_1 + p_2 + \cdots + p_m$ without ambiguity. A test case (or test point) $\vec{t}$ is a value of the Boolean vector $\vec{x}$. Let $\vec{t} = (t_1, t_2, \cdots, t_n)$ be expressed directly as $\vec{t} = t_1 t_2 \cdots t_n$, where $t_j \in B$ and $1 \leq j \leq n$. Table 1 shows the test point and test set of (2). In this study, we consider the seven types of faults detection of IDNFs as reported in [4], [5], and [21]. For Boolean expressions in program specification or implementation, typical programming errors involve missing or extra literals/variables and the use of incorrect operators and operands [8]. These faults include expression negation fault (ENF), literal negation fault (LNF), term omission fault (TOF), operator reference fault (ORF), literal omission fault (LOF), LIF, and LRF.

B. CTFTP TEST STRATEGY FOR IDNFS

In this section, we describe the CTFTP test strategy of (2) containing two test case generation strategies, namely, UCTP and NCFP strategies.

Definition 1 (UCTP Strategy): The points in test set T_i of (3) are obtained from the UTP set $UTP_i(f)$. For any point in T_i, if for all variables not occurring in (3), only one variable has a value of 1 (or 0), and the other variables have a value of 0 (or 1) in the corresponding position, then T_i is said to satisfy the UCTP strategy. The test set T_i is called the UCTP test set and is represented by $UCTP_i(f)$. If $UCTP_i(f)$ is an empty set, then whenever possible, a point $\vec{t}_i$ is selected from $UTP_i(f)$, while a corresponding NFP $\vec{t}_{i,\bar{j}}$ is selected from each $NFP_{i,\bar{j}}(f)$, $j = 1, 2, \cdots, k_i$, in that $\vec{t}_i$ and $\vec{t}_{i,\bar{j}}$ differ only in the truth value of the j-th literal appearing in (3). Then, $\vec{t}_i$ and each $\vec{t}_{i,\bar{j}}$ are added to $UCTP_i(f)$ and the corresponding $NCFP_{i,\bar{j}}(f)$, respectively. The set of all UCTPs of (2) is denoted as $UCTP(f) = \bigcup_{i=1}^{m} UCTP_i(f)$. The UCTP test strategy can ensure that faults, such as ENF, LNF, TOF, ORF, and LIF, are detectable.

In this study, the IDNF example considered for generating test cases is written as

$$f_0 = abc + a\bar{b}d + e. \tag{5}$$

The UTP set of abc in (5) is $UTP_1(f_0) = \{11100, 11110\}$, but the UCTP set of abc is $UCTP_1(f_0) = \{11110\}$.

Definition 2 (NCFP Strategy): The points in test set T_{ij} of (3) are obtained from the NFP set $NFP_{i,\bar{j}}(f), j = 1, 2, \cdots, k_i$. For any point in T_{ij}, if for all variables not occurring in (3), only one variable has a value of 1 (or 0), and the other variables have a value of 0 (or 1) in the corresponding position, then T_{ij} is said to satisfy the NCFP strategy. The test set T_{ij} is called the NCFP test set and represented by $NCFP_{i,\bar{j}}(f)$. If $NCFP_{i,\bar{j}}(f)$ is an empty set, then whenever possible, a point $\vec{t}_{i,\bar{j}}$ is selected from $NFP_{i,\bar{j}}(f)$, while a corresponding UTP $\vec{t}_i$ is selected from $UTP_i(f)$, in that $\vec{t}_{i,\bar{j}}$ and $\vec{t}_i$ differ only in the truth value of the j-th literal appearing in (3). Each $\vec{t}_{i,\bar{j}}$ and $\vec{t}_i$ are added to the corresponding $NCFP_{i,\bar{j}}(f)$ and $UCTP_i(f)$, respectively. The set of all NCFPs of (3) is given by $NCFP_i(f) = \bigcup_{j=1}^{k_i} NCFP_{i,\bar{j}}(f)$. Similarly, the set of all NCFPs of (2) is expressed by $NCFP(f) = \bigcup_{i=1}^{m} NCFP_i(f)$. The NCFP test strategy can ensure that faults, such as ENF, LNF, ORF, and LOF, are detectable.

For example, for the first, second, and third literals of abc in (5), the points 01100, 10100, and 11000 are NFPs, but they are not NCFPs. Points 01110 and 11010 are the NCFPs of the first literal a and the third literal c of abc, respectively. However, the NCFP set of the second literal b is an empty set. Therefore, we select the NFP 10100 from $NFP_{1,\bar{2}}(f_0)$ as the NCFP, that is $NCFP_{1,\bar{2}}(f_0) = \{10100\}$. A unique truth point 11100 with a different truth value only at the corresponding position of the second literal is selected from $UTP_1(f_0)$ as the UCTP, in that $UCTP_1(f_0) = \{11100, 11110\}$.

Neither the UCTP strategy nor the NCFP strategy can detect separately the LRF, but they can detect the LRF when used in combination. The CTFTP strategy is defined as follows.

Definition 3 (CTFTP Strategy): Each element in the test set T_i of (3) is composed of a UCTP $\vec{t}_i$ and multiple NCFPs $\vec{t}_{i,\bar{i}_1}, \vec{t}_{i,\bar{i}_2}, \cdots, \vec{t}_{i,\bar{i}_r}$ of (3), where $\vec{t}_i \in UCTP_i(f)$ and a certain j exists, and thus, $\vec{t}_{i,\bar{i}_h} \in NCFP_{i,\bar{j}}(f)$, that is, $i_h = j$. $\vec{t}_i$ and each $\vec{t}_{i,\bar{i}_h}, h = 1, 2, \cdots, r$ differ only in the truth value of the j-th literal appearing in (3). In this case, T_i is said to meet the CTFTP strategy. The CTFTP test set of (3) is written as

$$CTFTP_i(f) = \{(\vec{t}_i, [\vec{t}_{i,\bar{i}_1}, \vec{t}_{i,\bar{i}_2}, \cdots, \vec{t}_{i,\bar{i}_r}]) | \vec{t}_i \in UCTP_i, \vec{t}_{i,\bar{i}_h} \in NCFP_{i,\bar{j}}, 1 \le j, r \le k_i, h = 1, 2, \cdots, r\}.$$

The CTFTP test set of (2) is given by $CTFTP(f) = \bigcup_{i=1}^{m} CTFTP_i(f)$, which guarantees the detection of the LRF.

For abc in (5), we have $UTP_1(f_0) = \{11100, 11110\}$, $NFP_{1,\bar{1}}(f_0) = \{01100, 01110\}$,$NFP_{1,\bar{2}}(f_0) = \{10100\}$, and $NFP_{1,\bar{3}}(f_0) = \{11000, 11010\}$. Moreover, $UCTP_1(f_0) = \{11110\}$, $NCFP_{1,\bar{1}}(f_0) = \{01110\}$, $NCFP_{1,\bar{2}}(f_0) = \emptyset$, and $NCFP_{1,\bar{3}}(f_0) = \{11010\}$ are not difficult to obtain, in which $\emptyset$ denotes the space set. Hence, we select test point 10100 from $NFP_{1,\bar{2}}(f_0)$ and add it to $NCFP_{1,\bar{2}}(f_0)$, that is, $NCFP_{1,\bar{2}}(f_0) = \{10100\}$. According to Definition 2, the point 11100 in $UTP_1(f_0)$ also needs to be added to $UCTP_1(f_0)$. We obtain $UCTP_1(f_0) = \{11100, 11110\}$. The CTFTP test set of abc is given by $CTFTP_1(f_0) = \{(11100, [10100]), (11110, [01110, 11010])\}$.

The MI strategy can detect five types of faults, such as ENF, LNF, TOF, ORF, and LOF, in IDNFs, but it may not be able to detect LIF and LRF [5], [21]. In this study, the proposed CTFTP strategy not only can detect the same five types of faults, but also LIF in the IDNFs; however, the test case set used is smaller than that of the MUMCUT strategy. Consider using the CTFTP strategy to detect LRF. Suppose the j-th literal x_j^i of (3) is replaced by the literal x_l^i that does not occur in (3), that is, $x_l^i, \bar{x}_l^i \notin \{x_{k_1}^i, x_{k_2}^i, \cdots, x_{k_s}^i\}$. In $CTFTP_i(f)$, an element whose UCTP and corresponding NCFP differ only in the truth value of x_j^i exists, while the truth value of x_l^i is 0. Then, LRF can be detected. The following theorem proves that the CTFTP test set is a non-empty set.

Theorem 1: For any UCTP $\vec{t}$ in $UCTP_i(f)$ of (3), an NCFP $\vec{t}'$ always exists in $NCFP_i(f)$, in that $\vec{t}$ and $\vec{t}'$ differ only in the corresponding truth value of a certain literal of (3) and vice versa.

Proof: For any UCTP $\vec{t}$ of (3), that is, $\vec{t} \in UCTP_i(f)$, let $\vec{t} = \alpha_1\alpha_2\cdots\alpha_j\cdots\alpha_n$, where $\alpha_r \in B$ and $r = 1, 2, \cdots, n$. Let $\vec{t}' = \alpha_1\alpha_2\cdots\bar{\alpha}_j\cdots\alpha_n$, where $\bar{\alpha}_j$ is the opposite of α_j. We have $p_i(\vec{t}) = 1$ and $p_i(\vec{t}') = 0$. For any $h \neq i$, $p_h(\vec{t}) = 0$ holds. Consider the following cases:

(1) If the literal x_j^i (or $\bar{x}_j^i$) does not appear in p_h, then $p_h(\vec{t}') = p_h(\vec{t}) = 0$. Thus, $f(\vec{t}') = 0$. From (4), we have $p_{i,\bar{j}}(\vec{t}') = 1$. Thus, $\vec{t}' \in NCFP_i(f)$.

(2) Suppose the literal x_j^i occurs in p_h. From $p_i(\vec{t}) = 1$, $p_i(\vec{t}') = 0$, the truth value α_j of x_j^i is 1, and the truth value $\bar{\alpha}_j$ of $\bar{x}_j^i$ is 0. Therefore, we obtain $p_h(\vec{t}') = 0$, that is $f(\vec{t}') = 0$. From (4), we have $p_{i,\bar{j}}(\vec{t}') = 1$. Thus, $\vec{t}' \in NCFP_i(f)$.

(3) Suppose the literal $\bar{x}_j^i$ occurs in p_h. If we can select another literal x_l^h in p_h that differs from $\bar{x}_j^i$, and the literal x_l^h appears in (3), then the true value α_l of x_l^h is 1. For the UCTP $\vec{t}$ of (3), let $\vec{t}' = \alpha_1\alpha_2\cdots\alpha_j\cdots\bar{\alpha}_l\cdots\alpha_n$ (or $\vec{t}' = \alpha_1\alpha_2\cdots\bar{\alpha}_l\cdots\alpha_j\cdots\alpha_n$), where $\alpha_l = 1$. We obtain $p_i(\vec{t}') = p_h(\vec{t}') = 0$ and $p_{i,\bar{l}}(\vec{t}') = 1$, and thus $f(\vec{t}') = 0$. Consequently, $\vec{t}' \in NCFP_i(f)$. Similarly, if we select another literal $\bar{x}_l^h$ in p_h that differs from $\bar{x}_j^i$, and the literal $\bar{x}_l^h$ appears in (3), and we let $\vec{t}' = \alpha_1\alpha_2\cdots\alpha_j\cdots\bar{\alpha}_l\cdots\alpha_n$ (or $\vec{t}' = \alpha_1\alpha_2\cdots\bar{\alpha}_l\cdots\alpha_j\cdots\alpha_n$), where $\alpha_l = 0$. We can also obtain $\vec{t}' \in NCFP_i(f)$. If we cannot find such a literal x_l^h in p_h, then any literal x_τ^i in (3) that differs from x_j^i is selected. The NCFP $\vec{t}'$ corresponding to x_τ^i satisfies $p_i(\vec{t}') = p_h(\vec{t}') = 0$, $p_{i,\bar{\tau}}(\vec{t}') = 1$. Thus $f(\vec{t}') = 0$, that is, $\vec{t}' \in NCFP_i(f)$.

Similarly, we can prove that the theorem holds when the literal $\bar{x}_j^i$ occurs in (3). □

III. ORDERED BINARY LABEL-DRIVEN PETRI NET

A. DEFINITION OF OBLDPN

A Petri net consists of places, transitions, and arcs that connect places and transitions [18]. In this study, the basic Petri net is extended to an OBLDPN model, which is used to analyze the interaction between Boolean transitions and Boolean literals, construct test paths, and generate test cases for Boolean expressions.

Definition 4 (OBLDPN): An OBLDPN of a Boolean expression in the IDNF is an ordered binary digraph in which the number of nodes is related only to the number of variables. The OBLDPN is a 12-tuple $\Sigma = (P, T, F, s_0, s_f, V, B, L, \delta, E, In, Out)$.

(1) P is a finite set of literal places used to model the system state.

(2) Let $T = \{(lchild, rchild, ltag, rtag, sign) | \ lchild, rchild \in P, ltag, rtag, sign \in B\}$ be a set of all transitions, where *ltag*, *rtag*, and *sign* are the local labels in the transition t, and (i) *lchild* and *rchild* are the left and right children of t, respectively, indicating the state at which t arrives; (ii) *ltag* and *rtag* are the left and right labels of t, respectively, and $ltag, rtag \in L$. For the i-th term of a Boolean expression, if $t.lchild \in \bar{V}$, or $t.lchild = s_f$, then $t.ltag = 1$, and $t.ltag = 0$ otherwise; if $t.rchild \in V$, or $t.rchild = s_f$, then $t.rtag = 1$, and $t.rtag = 0$ otherwise; and (iii) *sign* indicates whether an interaction occurs between the L-subnet and the R-subnet. If $t.sign = 1$, then an interaction has occurred; otherwise, no interaction has occurred.

(3) F is a collection of arcs representing a flow relationship, and $F \subseteq (P \times T) \cup (T \times P)$ and $P \cap T = \emptyset$, $P \cup T \neq \emptyset$.

(4) The initial state s_0 represents the starting point of the search path in Σ, where $s_0 \in P$.

(5) The final state s_f represents the ending point of the search path in Σ, where $s_f \in P$.

(6) V represents the positive literal set of a Boolean expression, while $\bar{V}$ represents the corresponding negative literal set, and $P = V \cup \bar{V} \cup \{s_0, s_f\}$.

(7) $B = \{0, 1\}$ represents a finite set of truth values for the variables in V.

(8) $L = \{ltag, rtag, sign, intertag\}$ is the label set, where *intertag* is the global label in Σ.

(9) $\delta : L \to B$ is a labeling function.

(10) E is an event set, in which each event denotes a firing of an enabled transition.

(11) *In* and *Out* are the input and output relationship sets representing the input and output actions related to the transitions, respectively, and $In \subseteq (P \cup E) \times T$ and $Out \subseteq T \times (P \cup E)$.

An OBLDPN Σ is an ordered binary digraph divided into a left sub-OBLDPN (L-subnet for short) denoted by Σ_l and a right sub-OBLDPN (R-subnet for short) denoted by Σ_r. Two transitions from the left and right subnets may have the same left and right children. In Σ_l, except for s_0 and s_f, the literal in each place is a negative literal. In Σ_r, except for s_f, the literal in each place is a positive literal. All literal places in Σ_l and Σ_r are arranged in lexicographical order from the initial state s_0 to the final state s_f. The OBLDPN Σ of the Boolean expressions involving five variables is shown in Fig. 1. The dot circle in transition indicates the *sign* of the transition, that is, $t.sign = 1$.

B. DYNAMIC BEHAVIOR OF OBLDPN

Let ${}^\bullet t = \{p | (p, t) \in F \wedge p \in P\}$ and $t^\bullet = \{p | (t, p) \in F \wedge p \in P \wedge (p = t.lchild \vee p = t.rchild)\}$ in an OBLDPN Σ be the precursor literal set and the successor literal set of transition t, respectively, where the successor literals of t are $t.lchild$ and $t.rchild$. Let ${}^\bullet p = \{t | (t, p) \in F \wedge t \in T \wedge p \neq s_0 \wedge (t.lchild = p \vee t.rchild = p)\}$ and $p^\bullet = \{t | (p, t) \in F \wedge t \in T \wedge p \neq s_f\}$ be the input transition set and the output transition set of literal p, respectively. The pre- and post- conditions that fire transition t are represented by $t.pre$ and $t.post$, respectively, where $t.pre = (t.ltag, t.rtag, t.sign, intertag) \in L$, but $t.post = (intertag = 0, t.rchild = 0) \notin L$. The firing condition of transition t is determined by its pre-conditions and post-conditions. The behavior of an OBLDPN Σ is achieved by firing all transitions. The behavior of transition t is described as follows:

(1) If $t.ltag = 0$ and $t.rtag = 1$, then transition t is fired along its right-child place. At this point, $t.rchild$ is assigned to 1. If t is in Σ_l, that is, $intertag = 1$, then *intertag* is assigned to 0. The pre- and post-conditions of transition t are represented as $t.pre = (t.ltag = 0, t.rtag = 1, t.sign = 0, intertag = 1 or 0)$ and $t.post = (intertag = 0, t.rchild = 1)$, respectively.

(2) If $t.ltag = 1$ and $t.rtag = 0$, then transition t is fired along its left-child place. The pre- and post- conditions of t are written by $t.pre = (t.ltag = 1, t.rtag = 0, t.sign = 0, intertag = 1 or 0)$ and $t.post = (intertag = 1, t.rchild = 0)$, respectively.

(3) If $t.ltag = 0$ and $t.rtag = 0$, then transition t is fired, and (i) if $t.sign = 1$ and $intertag = 1$, then t is fired along its right-child place. The right child $t.rchild$ is assigned to 1 and $intertag$ to 0. Then, we have $t.pre = (t.ltag = 0, t.rtag = 0, t.sign = 1, intertag = 1)$ and $t.post = (intertag = 0, t.rchild = 1)$; (ii) If $t.sign = 1$ and $intertag = 0$, then t is fired along its left-child place. The left child $t.lchild$ is assigned to 0 and $intertag$ to 1. Then, $t.pre = (t.ltag = 0, t.rtag = 0, t.sign = 1, intertag = 0)$ and $t.post = (intertag = 1, t.lchild = 0)$; (iii) If $t.sign = 0$ and $intertag = 1$, then t is fired along its left-child place and $t.lchild$ is assigned to 0. However, the value of $intertag$ does not change. Therefore, $t.pre = (t.ltag = 0, t.rtag = 0, t.sign = 0, intertag = 1)$ and $t.post = (intertag = 1, t.lchild = 0)$; and (iv) If $t.sign = 0$ and $intertag = 0$, then t is fired along its right-child place and $t.rchild$ is assigned to 1. However, the value of $intertag$ does not change. Consequently, $t.pre = (t.ltag = 0, t.rtag = 0, t.sign = 0, intertag = 0)$ and $t.post = (intertag = 0, t.rchild = 1)$ are obtained.

(4) If $t.ltag = 1$ and $t.rtag = 1$, then transition t is no longer fired, the test path scan ends, and the test path is obtained. If $intertag = 1$ and $(t.rchild)^{\bullet}.sign = 1$ or $intertag = 0$ and $(t.lchild)^{\bullet}.sign = 1$, then the OBLDPN Σ ends. The signs in all transitions of Σ_l and Σ_r are alternately presented, as shown by the dot circles in Fig. 1.

The behavior function ρ of an OBLDPN Σ maps a transition to all possible sets of the developed scenarios. Let $\Pi = \{\pi | \pi : V \to B\}$ be a scenario set and the behavior function ρ be $\rho : T \to 2^{\Pi}$, which is expressed as

$$\rho(t_i) = (t_i.pre \wedge t_i.post) \wedge (\bigcup_j \rho(t_j))|_{\bullet t_j = t_i^{\bullet}},$$

where ${}^{\bullet}t_j = t_i^{\bullet}$ represents ${}^{\bullet}t_j = t_i.lchild \vee {}^{\bullet}t_j = t_i.rchild$, and $(\bigcup_j \rho(t_j))|_{\bullet t_j = t_i^{\bullet}}$ indicates that all Boolean variables and labels (including all local labels and global labels) have values of 0 or 1 in the scenario set $\bigcup_j \rho(t_j)$ of the left- and right-child places of transition t_i.

For example, in Fig. 1, the behavior function ρ of the OBLDPN Σ of the second term $a\bar{b}d$ in (5) can be given by

$$\begin{aligned}
\rho(t_0) &= (t_0.ltag = 0, t_0.rtag = 1, t_0.sign = 0, intertag \\
&\quad = 1) \wedge (intertag = 0, a = 1) \wedge \rho(t_2)|_{\bullet t_2 = t_0.rchild}, \\
\rho(t_2) &= (t_2.ltag = 1, t_2.rtag = 0, t_2.sign = 0, intertag \\
&\quad = 0) \wedge (intertag = 1, b = 0) \wedge \rho(t_3)|_{\bullet t_3 = t_2.lchild}, \\
\rho(t_3) &= (t_3.ltag = 0, t_3.rtag = 0, t_3.sign = 0, intertag \\
&\quad = 1) \wedge (intertag = 1, c = 0) \wedge \rho(t_5)|_{\bullet t_5 = t_3.lchild} \\
&\quad \cup (t_3.ltag = 0, t_3.rtag = 0, t_3.sign = 1, intertag \\
&\quad = 1) \wedge (intertag = 0, c = 1) \wedge \rho(t_6)|_{\bullet t_6 = t_3.rchild}, \\
\rho(t_5) &= (t_5.ltag = 0, t_5.rtag = 1, t_5.sign = 0, intertag \\
&\quad = 1) \wedge (intertag = 0, d = 1) \wedge \rho(t_8)|_{\bullet t_8 = t_5.rchild}, \\
\rho(t_6) &= (t_6.ltag = 0, t_6.rtag = 1, t_6.sign = 0, intertag \\
&\quad = 0) \wedge (intertag = 0, d = 1) \wedge \rho(t_8)|_{\bullet t_8 = t_6.rchild}, \\
\rho(t_8) &= (t_8.ltag = 0, t_8.rtag = 0, t_8.sign = 1, intertag \\
&\quad = 0) \wedge (intertag = 1, e = 0) \wedge \rho(t_9)|_{\bullet t_9 = t_8.lchild} \\
&\quad \cup (t_8.ltag = 0, t_8.rtag = 0, t_8.sign = 0, intertag \\
&\quad = 0) \wedge (intertag = 0, e = 1) \wedge \rho(t_{10})|_{\bullet t_{10} = t_8.rchild}.
\end{aligned}$$

In an OBLDPN, if the precursor literal ${}^{\bullet}t$ of a transition t obtains a token, then transition t is called *enabled*. If a transition is enabled, then its precursor literal contains a token; otherwise, it does not contain any token. An enabled transition that meets its pre- and post-conditions is called a fired transition, which is also called a label-driven transition. When a transition t fires, it removes the token from its precursor literal ${}^{\bullet}t$ and adds a token to the successor literal $t^{\bullet}$ it outputs, thereby changing the system state. The behavior function ρ of the above OBLDPN Σ provides the label of each enable transition. When s_0 obtains the token, these transitions will be fired one by one in accordance with the label sequence starting from transition t_0, thereby yielding the test paths of (3).

Definition 5. (*test path*) Given a label sequence $L' =< l_0, l_{i_1}, l_{i_2} \cdots, l_{i_n} >$, the transition sequence $T' =< t_0, t_{i_1}, t_{i_2}, \cdots, t_{i_n} >$ can be fired successively with respect to L', where n is the number of all variables in a Boolean expression. A test path (TPH) of (3) based on the OBLDPN Σ is defined as $TPH =< s_0, l_{i_1}/t_{i_1}/x_{i_1}, l_{i_2}/t_{i_2}/x_{i_2}, \cdots, l_{i_n}/t_{i_n}/x_{i_n}, l_{i_{n+1}}/t_{i_{n+1}}/s_f >$, where $l_j/t_j/x_j, j = i_1, i_2, \cdots, i_n$ indicating that the transition t_j satisfying its pre-condition $t_j.pre$ and post-condition $t_j.post$ under the label l_j can be fired. Thus, the value of the left child (or right child) x_j is equal to 0 (or 1), where $x_j = t_j.lchild$ (or $t_j.rchild$).

A test case or test point of (2) under a TPH is described as $TC =< x_{i_1} = \alpha_{i_1}, x_{i_2} = \alpha_{i_2}, \cdots, x_{i_n} = \alpha_{i_n} >$, which is abbreviated as $TC = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n}$, where $\alpha_{i_j} \in B$ and $\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n} \in UCTP(f) \cup NCFP(f) \subset B^n$. A test suite (TS) is a subset of n-dimensional Boolean space B^n and $TS = \{TC | TC \in UCTP(f) \cup NCFP(f)\} \subset B^n$.

Definition 6 (Configuration): In the TPH of (3), the variable sequence $\theta =< x_{i_1}x_{i_2}\cdots x_{i_{n-k_i}} >$ composed of variables not occurring in (3) is called the complement of (3), in which the variables in θ are sorted in lexicographic order. An assignment $\theta(\eta) =< x_{i_1} = \alpha_{i_1}, x_{i_2} = \alpha_{i_2}, \cdots, x_{i_{n-k_i}} = \alpha_{i_{n-k_i}} >$ of a variable sequence θ of (3) is called a complementary value or a configuration of (3), where $\eta = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_{n-k_i}}$ and $\alpha_{i_j} \in B$, $1 \le j \le n - k_i$. The configuration length $|\eta|$ is represented as $|\eta| = \sum_{j=1}^{n-k_i} \alpha_{i_j}$.

Definition 7 (Constraint True (or False) Point): In the OBLDPN Σ of (2), a true point test path (TPTPH) of (3) is a TPH satisfying $p_i(TC) = 1$. A false point test path (FPTPH) for (3) is generated by negating the j-th literal in the TPTPH of (3) in that $p_i(TC) = 0$, but $p_{i_{\bar{j}}}(TC) = 1$. A constraint TPTPH (or constraint FPTPH) of (3) is a TPTPH (or FPTPH) that satisfies $|\eta| = 1$ or $|\eta| = n - k_i - 1$, where $\eta \subset TC$. We call $TC = \alpha_1\alpha_2\cdots\alpha_n$ a constraint true point (CTP) (or

constraint false point (CFP)). The CTP set and CFP set of (3) are expressed as follows, respectively.

$$\begin{aligned} CTP_i(f) &= \{TC | p_i(TC) = 1, \eta \subset TC, \\ &\quad |\eta| = 1 or n - k_i - 1, |p_i| = k_i\}, \\ CFP_{i,\bar{j}}(f) &= \{TC | p_i(TC) = 0, p_{i,\bar{j}}(TC) = 1, \eta \subset TC, \\ &\quad |\eta| = 1 or n - k_i - 1, |p_i| = k_i, 1 \le j \le k_i\}. \end{aligned}$$

The UCTP set $UCTP_i(f)$ (or NCFP set $NCFP_{i,\bar{j}}(f)$) of (3) is the set of test points in $CTP_i(f)$ (or $CFP_{i,\bar{j}}(f)$) that satisfy Definition 1 (or Definition 2).

For example, the variable sequence $\theta =< ce >$ is a complement of the second term $a\bar{b}d$ in (5). $a\bar{b}d$ has four configurations, namely, $\eta = 00, 01, 10$, and 11, and the configuration lengths are $\eta = 0$, 1, 1, and 2, respectively. According to the OBLDPN Σ, $a\bar{b}d$ has four TPTPHs, namely, $tp1 =< s_0, l_0/t_0/a, l_2/t_2/\bar{b}, l_3/t_3/\bar{c}, l_5/t_5/d, l_8/t_8/\bar{e}, l_9/t_9/s_f >$, $tp2 =< s_0, l_0/t_0/a, l_2/t_2/\bar{b}, l_3/t_3/\bar{c}, l_5/t_5/d, l_8/t_8/e, l_{10}/t_{10}/s_f >$, $tp3 =< s_0, l_0/t_0/a, l_2/t_2/\bar{b}, l_3/t_3/c, l_6/t_6/d, l_8/t_8/\bar{e}, l_9/t_9/s_f >$, and $tp4 =< s_0, l_0/t_0/a, l_2/t_2/\bar{b}, l_3/t_3/c, l_6/t_6/d, l_8/t_8/e, l_{10}/t_{10}/s_f >$. However, only $tp2$ and $tp3$ are two constraint TPTPHs of $a\bar{b}d$. We can obtain $CTP_2(f_0) = \{10011, 10110\}$. Similarly, $CTP_{2,\bar{j}}(f_0)$ can be obtained, in which $1 \le j \le 3$.

IV. GENERATING AUTOMATICALLY THE CTFTP TEST SET

A. CTFTP TEST SET GENERATION ALGORITHM

According to OBLDPN, developing a test suite auto-generation algorithm for Boolean expressions is essential. We consider testing (3) and develop an algorithm called configuration-based IDNF test generation that generates automatically the test set $UCTP_i(f)$ and $NCFP_{i,\bar{j}}(f)$ of (3). The algorithm first generates the TPTPH of (3) to obtain the sets $TP_i(f)$ and $CTP_i(f)$. By negating the j-th literal in the TPTPH of (3), k_i FPTPHs are derived to obtain sets $FP_{i,\bar{j}}(f)$ and $CFP_{i,\bar{j}}(f)$, where $1 \le j \le k_i$. Finally, the set $CTFTP_i(f)$ is constructed. The configuration-based IDNF test generation algorithm that generates the sets $UCTP_i(f)$ and $NCFP_{i,\bar{j}}(f)$ is proposed as Algorithm 1.

In Algorithm 1, the transitions in the OBLDPN Σ satisfying the pre- and post-conditions are fired to obtain the TPTPH set and corresponding truth point test case set of (3). The 2^{n-k_i} configurations are obtained for each TPTPH. A CTP is obtained when the corresponding configuration length is 1 or $n-k_i-1$. The corresponding k_i CFPs are obtained by negating the j-th literal occurring in (3). Repeat Steps 1 to 4 to generate test sets $TP_i(f)$, $CTP_i(f)$, $FP_{i,\bar{j}}(f)$, and $CFP_{i,\bar{j}}(f)$, and then generate test sets $UCTP_i(f)$, $NCFP_{i,\bar{j}}(f)$, and $CTFTP_i(f)$ of (3), where $j = 1, 2, \cdots, k_i$. Algorithm 1 can be terminated. Its time complexity is $O(2^{n-k_i})$.

Example 1: Consider (2) that can be written as a general Boolean expression, that is,

$$G_0 = a(bc + \bar{b}d) + e. \tag{6}$$

Algorithm 1 Configuration-Based IDNF Test Generation Algorithm

Input: OBLDPN Σ of (2) with n variables.

Output: $UCTP_i(f)$, $NCFP_{i,\bar{j}}(f)$, and $CTFTP_i(f)$.

1: Initialize the sets $TP_i(f)$, $UTP_i(f)$, $FP_{i,\bar{j}}(f)$, $NFP_{i,\bar{j}}(f)$, $CTP_i(f)$, $CFP_{i,\bar{j}}(f)$, $UCTP_i(f)$, and $UCFP_{i,\bar{j}}(f)$ to the empty set $\emptyset$.

2: In Σ, any transition t_{i_j} that satisfies its pre-condition $t_{i_j}.pre$ and post-condition $t_{i_j}.post$ will be fired. If all transitions satisfy their pre- and post-conditions, then these transitions will be fired in turn from the initial state s_0 to the final state s_f. A TPTPH path of (3) is generated. Let $TPTP_i =< s_0, l_{i_1}/t_{i_1}/x_{i_1}, l_{i_2}/t_{i_2}/x_{i_2}, \cdots, l_{i_n}/t_{i_n}/x_{i_n}, l_{i_{n+1}}/t_{i_{n+1}}/s_f >$. The corresponding test case $TC_i = \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_n}$ can also be obtained, where $\alpha_{i_j} \in B$ and $1 \le j \le n$.

3: Suppose θ_j is the complement of (3). Let $\theta_j = \emptyset$. In Σ, if $t_{i_j}.pre = (t_{i_j}.ltag = 0, t_{i_j}.rtag = 0, t_{i_j}.sign = 1 or 0, intertag = 1 or 0)$ and $t_{i_j}.post = (intertag = 0, t_{i_j}.rchild = 1)$, then $\theta_j = \theta_j \cup \{x_{i_j}\}$; if $t_{i_j}.pre = (t_{i_j}.ltag = 0, t_{i_j}.rtag = 0, t_{i_j}.sign = 0 or 1, intertag = 1 or 0)$ and $t_{i_j}.post = (intertag = 1, t.lchild = 0)$, then $\theta_j = \theta_j \cup \{\bar{x}_{i_j}\}$. The complement $\theta_j =< x_{j_1}x_{j_2}\cdots x_{j_{n-k_i}} >$ and 2^{n-k_i} configuration η_j of (3) can be obtained, that is, $\eta_j = \alpha_{j_1}\alpha_{j_2}\cdots\alpha_{j_{n-k_i}}$, where $\alpha_{j_k} \in B$, $1 \le j \le 2^{n-k_i}$, and $1 \le k \le n - k_i$.

4: A set $\{TC_i\}$ involving 2^{n-k_i} test cases is generated by assigning the truth values of x_{j_k} in θ_j contained in $TPTP_i$ by using all configurations of (3). For each $tc_i \in \{TC_i\}$, if $tc_i \notin TP_i(f)$, then $TP_i(f) = TP_i(f) \cup \{tc_i\}$; if the corresponding configuration length $|\eta_j|$ is 1 or $n - k_i - 1$ and $tc_i \notin CTP_i(f)$, then $CTP_i(f) = CTP_i(f) \cup \{tc_i\}$. Similarly, we can derive the k_i FPTPHs of (3) by negating the j-th literal appearing in (3) in the above TPTPH, thereby obtaining sets $FP_{i,\bar{j}}(f)$ and $CFP_{i,\bar{j}}(f)$, where $1 \le j \le k_i$.

5: For each term p_h of (2), where $h = 1, 2, \cdots, m$, repeat Steps 1 to 4 to generate $TP_h(f)$, $CTP_h(f)$, $FP_{h,\bar{j}}(f)$, and $CFP_{h,\bar{j}}(f)$.

6: Let $UCTP_i(f) = CTP_i(f) - \bigcup_{j \ne i} TP_j(f)$ and $UTP_i(f) = TP_i(f) - \bigcup_{j \ne i} TP_j(f)$. For each j, $NCFP_{i,\bar{j}}(f) = CFP_{i,\bar{j}}(f) - \bigcup_{k=1}^{m} TP_k(f)$ and $NFP_{i,\bar{j}}(f) = FP_{i,\bar{j}}(f) - \bigcup_{k=1}^{m} TP_k(f)$.

7: If $UCTP_i(f) = \emptyset$ or $NCFP_{i,\bar{j}}(f) = \emptyset$, then select a point $\vec{t}$ from $UTP_i(f)$ and a point $\vec{t}'$ from each $NFP_{i,\bar{j}}(f)$, $j = 1, 2, \cdots, k_i$, in that $\vec{t}$ and $\vec{t}'$ differ only from the truth value of the j-th literal appearing in (3).

8: Select the test cases in $NCFP_{i,\bar{j}}(f)$ corresponding to the elements in $UCTP_i(f)$ to construct $CTFTP_i(f)$.

9: **return** $UCTP_i(f)$, $NCFP_{i,\bar{j}}(f)$, and $CTFTP_i(f)$.

By calling Algorithm 1, the test cases of the subexpression (7) are generated.

$$g = bc + \bar{b}d. \tag{7}$$

For the first term bc of (7), the sets $CTP_1(g)$, $CFP_{1,\bar{1}}(g)$, and $CFP_{1,\bar{2}}(g)$ are calculated as

$$CTP_1(g) = \{01101, 01110, 01111, 11100, 11101, 11110\},$$
$$CFP_{1,\bar{1}}(g) = \{00101, 00110, 00111, 10100, 10101, 10110\},$$
$$CFP_{1,\bar{2}}(g) = \{01001, 01010, 01011, 11000, 11001, 11010\}.$$

For the second term $\bar{b}d$ of (7), the sets $CTP_2(g)$, $CFP_{2,\bar{1}}(g)$, and $CFP_{2,\bar{2}}(g)$ are computed as

$$CTP_2(g) = \{00011, 00110, 00111, 10010, 10011, 10110\},$$
$$CFP_{2,\bar{1}}(g) = \{01011, 01110, 01111, 11010, 11011, 11110\},$$
$$CFP_{2,\bar{2}}(g) = \{00001, 00100, 00101, 10000, 10001, 10100\}.$$

We can obtain the sets $UCTP_i(g)$ and $NCFP_{i,\bar{j}}(g)$, where $i, j = 1, 2$. Here,

$$UCTP_1(g) = CTP_1(g),\ UCTP_2(g) = CTP_2(g),$$
$$NCFP_{1,\bar{2}}(g) = CFP_{1,\bar{2}}(g),\ NCFP_{2,\bar{2}}(g) = CFP_{2,\bar{2}}(g),$$
$$NCFP_{1,\bar{1}}(g) = \{00101, 10100, 10101\},$$
$$NCFP_{2,\bar{1}}(g) = \{01011, 11010, 11011\}.$$

From Algorithm 1, we have

$$CTFTP_1(g) = \{(01101, [00101, 01001])\},$$
$$CTFTP_2(g) = \{(00110, [00100])\}.$$

Algorithm 1 can generate automatically a test suite for a given IDNF. This test suite can detect LRF effectively. Therefore, by using fewer test cases, the CTFTP strategy can detect the same seven types of faults similar to those of the MUMCUT strategy.

B. TEST-ADEQUACY CRITERIA

The adequacy of a test set is measured against a finite set of elements. Depending on the adequacy criteria of interest, the elements can be derived from the OBLDPN Σ of the tested Boolean expression. If the corresponding coverage domain depends only on the internal structure of the source code, then the criterion is a white-box test adequacy criterion, in which the execution of the program statements can be detected through branches, loops, and paths [22]. If a path is a sub-path of at least one of the transition paths of the test case TC from the initial position to the end position, then the path is covered by the test case TC. For each path, at least one test case TC can be used to cover the path called the path coverage criterion [23]. Here, we consider a path coverage criterion based on the OBLDPN.

Path Coverage Criterion: The CTFTP test set T is considered adequate in relation to the OBLDPN Σ if, for each element of T, each constraint TPTPH and the corresponding constraint FPTPHs from the initial state s_0 to the final state s_f are executed at least once.

The set $TS(f) = \{UCTP(f), NCFP(f), CTFTP(f)\}$ of (2) is considered adequate if

(1) each term p_h of (2) is covered,

(2) each literal appearing in p_h is given a truth value of 1 and 0, and

(3) each test point in $TS(f)$ affects independently the result of (2) [24].

Example 2: Consider the first term abc of (5). The variable sequence $\theta_1 =< de >$ is a complement of abc. Then, four TPTPHs for abc are generated as $tp1 =< s_0, l_0/t_0/a, l_2/t_2/b, l_4/t_4/c, l_6/t_6/\bar{d}, l_7/t_7/\bar{e}, l_9/t_9/s_f >$, $tp2 =< s_0, l_0/t_0/a, l_2/t_2/b, l_4/t_4/c, l_6/t_6/\bar{d}, l_7/t_7/e, l_{10}/t_{10}/s_f >$, $tp3 =< s_0, l_0/t_0/a, l_2/t_2/b, l_4/t_4/c, l_6/t_6/d, l_8/t_8/\bar{e}, l_9/t_9/s_f >$, and $tp4 =< s_0, l_0/t_0/a, l_2/t_2/b, l_4/t_4/c, l_6/t_6/d, l_8/t_8/e, l_{10}/t_{10}/s_f >$. However, only $tp1$ and $tp3$ are two unique TPTPHs, and $tp3$ is a unique constraint TPTPH. The sequence of transitions fired by $tp3$ is $T_{tp3} =< t_0, t_2, t_4, t_6, t_8, t_9 >$.

On the basis of Definition 7, we can obtain the corresponding constraint FPTPHs $fp31$, $fp32$, and $fp33$ of $tp3$; the corresponding transition sequences fired by the constraint FPTPHs are denoted as $T_{fp31} =< t_0, t_1, t_4, t_6, t_8, t_9 >$, $T_{fp32} =< t_0, t_2, t_3, t_6, t_8, t_9 >$, and $T_{fp33} =< t_0, t_2, t_4, t_5, t_8, t_9 >$, respectively. The transition t_{10} is not covered. We select the TPTPH $tp2$ as the constraint TPTPH, and its transition sequence is $T_{tp2} =< t_0, t_2, t_4, t_6, t_7, t_{10} >$. The transition sequences of the corresponding constraint FPTPHs $fp21$, $fp22$, and $fp23$ are denoted as $T_{fp21} =< t_0, t_1, t_4, t_6, t_7, t_{10} >$, $T_{fp22} =< t_0, t_2, t_3, t_6, t_7, t_{10} >$, and $T_{fp23} =< t_0, t_2, t_4, t_5, t_7, t_{10} >$, respectively. Therefore, t_{10} is covered. Then, the test suite of abc is given by $TS_1(f_0) = \{UCTP_1(f_0), NCFP_{1,\bar{1}}(f_0), NCFP_{1,\bar{2}}(f_0), NCFP_{1,\bar{3}}(f_0), CTFTP_1(f_0)\}$, which meets the path coverage criteria.

V. GENERATING AUTOMATICALLY TEST SETS FOR GENERAL BOOLEAN EXPRESSIONS

We extend the proposed CTFTP strategy to generate a test suite of general Boolean expressions, such as (1). Suppose the i-th term $f_i(\vec{x})$ of (1) is written as

$$f_i(\vec{x}) = g_1^i g_2^i \cdots g_{\tau_i}^i,\ 1 \le i \le k, \tag{8}$$

where τ_i is the number of factors consisting of literals and subexpressions and g_j^i, $1 \le j \le \tau_i$ is a literal or a sub-expression in the IDNF. For example, in (6), the first term $a(bc + \bar{b}d)$ contains a single literal a and a subexpression $bc + \bar{b}d$, but the second term consists only of a single literal e.

We consider the test generation of (8), which contains the subexpressions $g_{j_1}^i, g_{j_2}^i, \cdots, g_{j_v}^i$, where $1 \le j_1, j_2, \cdots, j_v \le \tau_i$, in the form of IDNF. After replacing the subexpressions $g_{j_1}^i, g_{j_2}^i, \cdots, g_{j_v}^i$ with the literals $x_{j_1}^i, x_{j_2}^i, \cdots, x_{j_v}^i$ occurring in $g_{j_1}^i, g_{j_2}^i, \cdots, g_{j_v}^i$, respectively, Eq. (8) is transformed into a new simple conjunctive form as

$$q_i(\vec{x}) = g_1^i g_2^i \cdots g_{j_1-1}^i x_{j_1}^i g_{j_1+1}^i \cdots g_{j_2-1}^i x_{j_2}^i g_{j_2+1}^i \cdots g_{j_v-1}^i x_{j_v}^i g_{j_v+1}^i \cdots g_{\tau_i}^i, \tag{9}$$

Algorithm 2 General Boolean Expression Test Generation Algorithm Based on Literal Substitution

Input: OBLDPN Σ of (1).

Output: $UCTP_i(G)$, $NCFP_{i,\bar{j}}(G)$, and $CTFTP_i(G)$.

1: Replace the subexpressions in (8) with the corresponding literals to obtain (9).

2: In (8), suppose the subexpression $g^i_{j_h}$ contains k_u items, where the u-th item $y^{j_h}_u$ involves k_w literals. For $y^{j_h}_u$, call Algorithm 1 to generate the sets $UCTP_u(g^i_{j_h})$ and $NCFP_{u,\bar{w}}(g^i_{j_h})$, where w represents the sequence number of the literals appearing in $y^{j_h}_u$ and $1 \le u \le k_u$ and $1 \le w \le k_w$. On the basis of Definitions 1 and 2, the UCTP set $UCTP_{j_h}(g^i_{j_h})$ and NCFP set $NCFP_{i,\bar{j}_h}(g^i_{j_h})$ for $g^i_{j_h}$ are given by $UCTP_{j_h}(g^i_{j_h}) = \bigcup_{u=1}^{k_u} UCTP_u(g^i_{j_h})$ and $NCFP_{i,\bar{j}_h}(g^i_{j_h}) = \bigcup_{u=1}^{k_u}\bigcup_{w=1}^{k_w} NCFP_{u,\bar{w}}(g^i_{j_h})$, respectively.

3: For (9), use Algorithm 1 to generate the sets $UCTP_i(q_i)$ and $NCFP_{i,\bar{j}}(q_i)$, where $j = 1, 2, \cdots, j_1, j_2, \cdots, j_v, \cdots, \tau_i$.

4: The set $UCTP_i(G)$ for (8) is given by

$$UCTP_i(G) = UCTP_i(q_i) \cap (\bigcup_{h=1}^{v} UCTP_{j_h}(g^i_{j_h})).$$

5: If g^i_j represents the literal occurring in (8), then the set $NCFP_{i,\bar{j}}(G)$ is written by

$$NCFP_{i,\bar{j}}(G) = NCFP_{i,\bar{j}}(q_i) \cap (\bigcup_{h=1}^{v} UCTP_{j_h}(g^i_{j_h})),$$

where $j \neq j_1, j_2, \cdots, j_v$. If g^i_j represents the subexpression $g^i_{j_h}$ appearing in (8), then the set $NCFP_{i,\bar{j}}(G)$ is written by

$$NCFP_{i,\bar{j}}(G) = NCFP_{i,\bar{j}_h}(q_i) \cap (NCFP_{i,\bar{j}_h}(g^i_{j_h}) \cup (\bigcup_{z=1}^{h-1} UCTP_{j_z}(g^i_{j_z})) \cup (\bigcup_{z=h+1}^{v} UCTP_{j_z}(g^i_{j_z}))),$$

where $UCTP_{j_z}(g^i_{j_z}) = \bigcup_{\sigma=1}^{k_\sigma} UCTP_\sigma(g^i_{j_z})$, k_σ is the number of terms in $g^i_{j_z}$, and $j = j_1, j_2, \cdots, j_v$.

6: The test cases in $NCFP_{i,\bar{j}}(G)$ corresponding to the elements in $UCTP_i(G)$ are selected to construct $CTFTP_i(G)$.

7: **return** $UCTP_i(G)$, $NCFP_{i,\bar{j}}(G)$, and $CTFTP_i(G)$.

where the literal $x^i_{j_h}$, $h = 1, 2, \cdots, v$, is the last literal that appears in the subexpression $g^i_{j_h}$ in lexicographic order, but its negated literal $\bar{x}^i_{j_h}$ does not occur in $g^i_{j_h}$. If $x^i_{j_h}$ and $\bar{x}^i_{j_h}$ occur in $g^i_{j_h}$, then the previous literal $x^i_{j_h-1}$ occurring in $g^i_{j_h}$ is selected in lexicographic order. However, $x^i_{j_h-1}$ cannot appear in $g^i_{j_h}$, and so on. Subsequently, we present an algorithm called the general Boolean expression test generation algorithm based on literal substitution to generate automatically the test suite for general Boolean expressions. The proposed algorithm is described as Algorithm 2.

In Algorithm 2, for each subexpression in (8), We replace the subexpression with the last literal appearing in (8) to obtain (9). For (9), we call Algorithm 1 to determine the UCTP set $UCTP_i(q_i)$ and the NCFP set $NCFP_{i,\bar{j}}(q_i)$. Algorithm 1 is used for each term of each subexpression $g^i_{j_h}$ in (8) to generate the sets $UCTP_{j_h}(g^i_{j_h})$ and $NCFP_{i,\bar{j}_h}(g^i_{j_h})$. The intersection of the set $UCTP_i(q_i)$ and the sets $UCTP_{j_h}(g^i_{j_h})$ of the v subexpressions $g^i_{j_h}$ yields $UCTP_i(G)$ for (8). If (8) contains no subexpressions, then the intersection of the set $NCFP_{i,\bar{j}}(q_i)$ and the set $UCTP_{j_h}(g^i_{j_h})$ of the v subexpressions $g^i_{j_h}$ is generated. If (8) contains subexpressions, then the intersection of the sets $NCFP_{i,\bar{j}}(q_i)$ and the union of all sets $NCFP_{i,\bar{j}_h}(g^i_{j_h})$ and $UCTP_{j_z}(g^i_{j_z})$ of the $v-1$ sub-expressions $g^i_{j_z}$ other than the j_h-th sub-expression $g^i_{j_h}$, which denotes the set $NCFP_{i,\bar{j}}(G)$ of the j-th literal or subexpression appearing in (8), is obtained.

Algorithm 2 can be terminated and its time complexity is $O((k-\mu)\cdot 2^{n-k_i} + \mu\cdot\nu\cdot k_u \cdot 2^{2n-\tau_i-k_w})$, where μ is the number of terms containing at least one subexpression in (1). For the μ-th terms, the test cases for v subexpressions need to be generated, and thus, the time complexity is $O(\mu\cdot\nu\cdot 2^{n-\tau_i})$. For the subexpression $g^i_{j_h}$ involving the k_u term, the test cases are generated for the term containing k_w literals, and thus, the time complexity is $O(k_u \cdot 2^{n-k_w})$. Therefore, the total complexity of the test case generation of μ items in (1) is $O(\mu\cdot\nu\cdot k_u\cdot 2^{2n-\tau_i-k_w})$.

Example 3: Consider the test generation of (6). We replace the subexpression with the last literal d in the subexpression $g = bc + \bar{b}d$ and obtain

$$\tilde{G}_0 = ad + e. \tag{10}$$

Algorithm 2 is used to generate the UCTP set $UCTP_1(\tilde{G}_0)$, and the corresponding NCFP sets $NCFP_{1,\bar{1}}(\tilde{G}_0)$ and $NCFP_{1,\bar{2}}(\tilde{G}_0)$ for the first term ad of (10). We have

$$\begin{aligned} UCTP_1(\tilde{G}_0) &= \{10110, 11010, 10010, 11110\},\\ NCFP_{1,\bar{1}}(\tilde{G}_0) &= \{00110, 01010, 00010, 01110\},\\ NCFP_{1,\bar{2}}(\tilde{G}_0) &= \{10100, 11000, 10000, 11100\}. \end{aligned}$$

In Example 1, the set $UCTP_i(g)$ and $NCFP_{i,\bar{j}}(g)$, where $i, j = 1, 2$, are yielded. Then, the sets $UCTP_1(G_0)$, $NCFP_{1,\bar{1}}(G_0)$, and $NCFP_{1,\bar{2}}(G_0)$ of (6) are computed as

$$\begin{aligned} UCTP_1(G_0) &= UCTP_1(\tilde{G}_0) \cap (UCTP_1(g) \cup UCTP_2(g)) = \{10110, 10010, 11110\},\\ NCFP_{1,\bar{1}}(G_0) &= NCFP_{1,\bar{1}}(\tilde{G}_0) \cap (UCTP_1(g) \cup UCTP_2(g)) = \{00110, 01110\},\\ NCFP_{1,\bar{2}}(G_0) &= NCFP_{1,\bar{2}}(\tilde{G}_0) \cap (NCFP_{1,\bar{1}}(g) \cup NCTP_{1,\bar{2}}(g) \cup NCTP_{2,\bar{1}}(g) \cup NCTP_{2,\bar{2}}(g))\\ &= \{10100, 10000, 11000\}. \end{aligned}$$

The sets $UCTP_2(G_0)$ and $NCFP_{2,\bar{1}}(G_0)$ for the second term e of (6) are given by

$$UCTP_2(G_0) = \{00011, 00101, 01001, 01111, 10001\},$$

TABLE 2. Comparison of Different Test Strategies for the TCAS II Expressions.

Spec. ID	Var. Num.	Exp. Length	Weighted Average	Size of Exhaustive Test Set	Size of test set as percentage of exhaustive test set						
					MUMCUT[a]		CTFTP	MAX-A(%)		MAX-B(%)	
					G-CUN(%)	G-UCN(%)	Test Set[b](%)	Original[a]	New[b]	Original[a]	New[b]
T01	7	33	1.2	128	30.1	30.5	31.25	39.1	37.5	45.3	43.75
T02	9	117	0.92	512	22.7	22.7	22.66	22.7	22.66	25.6	24.8
T03	12	170	6.08	4096	6	5.8	5.68	62	62.74	62.5	63.23
T04	5	9	2.67	32	36.5	36.7	34.38	87.5	87.5	96.9	93.75
T05	9	36	5.89	512	8.4	8.4	8.01	69.7	63.28	87.3	64.84
T06	11	63	1.33	2048	4.3	4.1	3.94	6.3	6.25	6.8	6.88
T07	10	67	2.5	1024	9.9	10.4	8.08	20.3	20.3	22.1	21.68
T08	8	35	0	256	14.1	14.1	14.06	43.8	14.06	49.2	17.19
T09	7	15	0	128	12.5	12.5	12.5	12.5	12.5	20.3	17.97
T10	13	65	3	8192	1	1	1	2.9	2.9	3.2	3.14
T11	13	71	6	8192	1.5	1.5	1.5	22.9	24.49	25.3	24.74
T12	14	59	7.5	16384	0.7	0.7	0.7	24.7	23.72	25.6	23.86
T13	12	19	9.67	4096	0.9	0.9	0.84	41.5	41.19	41.8	41.48
T14	7	21	4.33	128	25.7	26.6	25.06	71.1	67.19	89.1	74.22
T15	9	42	6.09	512	11.8	11.9	11.72	58	58	93	60.55
T16	12	109	8.22	4096	3.9	3.7	3.46	47.8	46.97	48.5	47.39
T17	11	37	5.67	2048	3.7	3.7	3.28	46.2	46.2	49.6	47.02
T18	10	45	5.25	1024	7.4	7.7	6.86	48.2	48.24	56.6	49.8
T19	8	23	3	256	17.3	17.4	18.36	67.2	43.75	89.1	47.66
T20	7	13	1	128	18.8	18.8	18.75	18.8	18.75	26.6	24.22
Average percentage					11.86	11.96	11.6	40.66	37.41	48.22	39.91

[a]These values are from Table II in [21]. [b]These values are generated by using Algorithm 1.

$$NCFP_{2,\bar{1}}(G_0) = \{00010, 00100, 01000, 01110, 10000\}.$$

Then, we obtain

$$CTFTP_1(G_0) = \{(10110, [00110, 10100]), (11110, [01110, 11100])\},$$
$$CTFTP_2(G_0) = \{(00011, [00010]), (01111, [01110])\}.$$

Algorithm 2 can generate automatically test cases for a given general Boolean expression. The CTFTP strategy can detect five types of faults, that is, ENF, LNF, TOF, ORF, and LOF, of general Boolean expressions. However, LIF and LRF cannot always be detected.

VI. EMPIRICAL RESULTS

We report an empirical study on the CTFTP testing strategy in this section. We use the same 20 Boolean expressions as in [4], which originated from the TCAS II specification of the aircraft collision avoidance system [15], to facilitate the comparison of results as those in [2] and [21]. We transform each TCAS II expression into an equivalent expression in the form of IDNF. The CTFTP test set of the IDNFs can be generated using Algorithm 1. Table 2 presents a comparison of the sizes of the four test sets, namely, CTFTP, MUMCUT, MAX-A, and MAX-B test sets, of the IDNF expressions. In Table 2, the CTFTP test set and the MAX-A and MAX-B test sets are generated by Algorithm 1 and the percentage of the size of the test sets relative to the size of the exhaustive test set is exhibited. For comparison, we list the MUMCUT test set generated by the G-CUN and G-UCN methods, which are two greedy incremental expansion methods, and reproduce the MAX-A and MAX-B test sets from Table 2 presented in [21].

In Table 2, Columns 2-4 list the number of variables in the expression, the length of the expression, and the weighted average of the number of literals that do not appear in all terms. The percentage of the CTFTP test set for the 20 Boolean expressions ranges from 0.70% to 35.57%, with an average value of 11.60%. The average percentage of the CTFTP test set is smaller than the average percentage of the MUMCUT test set generated by the G-CUN and G-UCN methods (11.86% and 11.96%, respectively). The size of the CTFTP test set is only slightly larger for the MUMCUT test set generated by the G-CUN and G-UCN methods in the two Boolean expressions of Specs. T01 and T19; however, the two test sets have the same size for Specs. T09, T10, T11, and T12. For the remaining Boolean expressions, the size of the CTFTP test set is relatively small.

As shown in Table 2, by using Algorithm 1, the percentage of the MAX-A test set of the 20 Boolean expressions ranges from 2.9% to 87.5%, with an average of 37.41%, while the percentage of the MAX-B test set ranges from 3.14% to 93.75%, with an average of 39.91%. For the MAX-A and MAX-B test strategies, the average percentages of the two test sets generated by Algorithm 1 are smaller than the average percentages of the two test sets generated in [21] (40.66% and 48.22%, respectively). For (3), we only select test points in which the configuration length $|\eta|$ of (3) satisfies $|\eta| = 1$ or $|\eta| = n - k_i - 1$. Theoretically, the maximum numbers of test points in the $UCTP_i(f)$ test set and the $NCFP_{i,\bar{j}}(f)$ test set of (3) are $2(n - k_i)$ and $2k_i(n - k_i)$, respectively. However, for the MAX-A test strategy, the maximum numbers of test points in the $UTP_i(f)$ test set and the $NFP_{i,\bar{j}}(f)$ test set of (3) are theoretically 2^{n-k_i} and $k_i \cdot 2^{n-k_i}$, respectively.

TABLE 3. CTFTP Test Suites of Several Boolean Expressions.

Boolean Expressions	Test Suites			TC Num
	UCTP	NCFP	CTFTP	
$S_1 = ab + ac + d$	1100, 1010, 0011, 0111, 1001	0100, 0110, 1000, 0010	(1100, [0100, 1000]), (1010, [0010, 1000]), (0011, [0010]),(0111, [0110]),(1001, [1000])	9
$S_2 = a(b + c) + d$	1010, 1100, 0011, 0101,1001, 0111	1000, 0010, 0100, 0110	(1010, [1000]), (1100, [1000]), (0011, [0010]), (0101, [0100]), (1001, [1000]), (0111, [0110])	10
$S_3 = a\bar{b}d + a\bar{c}d + e$	10110, 11010, 00011, 11101	00010, 00110, 11110, 10000, 10100, 01010, 11000	(10110, [00110,11110, 10100]), (11010, [01010, 11110, 11000]), (00011, [00010])	11
$S_4 = abc + a\bar{b}d + e$	11100, 11110, 10010, 10110, 00011, 00101, 11011	01100, 01110, 10100, 11000, 11010, 00010, 00110, 10000	(11110, [01110, 11010]), (11100, [10100]), (10010, [11010]), (10110, [00110, 10100]), (00011, [00010]), (11011, [11010])	15

Weyuker *et al.* [4] reported that the MAX-A test set contains all UTPs and all NFPs associated with each term. When constructing the MAX-B test set, in addition to selecting all the elements in the MAX-A set test, one needs to select $\lceil n - k_i \rceil$ points from the set of overlapping true points of size 2^{n-k_i} and $\lceil n - k_i \rceil$ points from the set of remaining false points of size 2^{n-k_i}. Theoretically, the percentage is smaller when the Boolean expression contains more variables [21]. However, the size of a Boolean expression test set is related to the following factors: (1) the number of variables, (2) the length of the expression, and (3) the maximum and minimum number of literals that do not occur in all items. The maximum and minimum numbers are represented by the weighted average (i.e., mathematical expectation) of the number of literals that do not appear in all items. The size of a Boolean expression is determined by the expression length and the number of variables, in which the expression length is determined by the number of literals appearing in all terms and the number of terms. Table 2 shows the effect of the abovementioned three factors on the size of the four test sets.

(1) In general, when the number of variables is the same, the longer is the length, the more instances in which the four test cases are required, but there are exceptions. For Specs. T03, T13, and T16 with 12 variables, T03 has the largest test set. However, for Specs. T08 and T19 with eight variables, the length of T19 is smaller than that of T08, but the average number of non-appearing literals in T19 is larger than that in T08. Thus, the T19 test set is larger than the T08 test set.

(2) For the CTFTP and MUMCUT strategies, when the length of the Boolean expression is nearly the same, the higher is the number of variables, the smaller is the percentage of the test cases. This trend is consistent with the conclusion in [21]. For Specs. T01, T05, T08, and T17, Spec. T17 contains the most variables. Thus, the percentage of the CTFTP and MUMCUT test cases for T17 is the smallest.

(3) In the case of the same number of variables, if the length does not change excessively, then the larger is the average number of literals that do not appear in all terms, the larger are the four test sets. For Specs. T09 and T20 with seven variables, the length of T20 is slightly smaller than that of T09, but more literals that do not appear on the average in T20 than in T09. Thus, the four test sets of T20 are larger than those of T09. Specs. T08 and T19 are similar.

In this experiment, we present the CTFTP test suites of several Boolean expressions, as shown in Table 3. The expression S_3 represents Spec. T04, while S_4 corresponds to (5). The expression S_2 is a general form of S_1. The following results can be deduced from Table 3:

(1) The size of the CTFTP test set of a Boolean expression is related to the maximum and minimum numbers of literals that do not appear in all sub-expressions. The larger is the average number of literals that do not occur in the subexpressions of a Boolean expression, the smaller is the CTFTP test set of the expression. S_3 and S_4 have the same three factors abovementioned, but the average number of literals that do not occur in their subexpressions is different, which leads to different numbers of test cases. In S_3 and S_4, the average number of non-appearing literals of the respective sub-expressions $\bar{b} + \bar{c}$ and $bc + \bar{b}d$ are 4 and 3, respectively. Therefore, the CTFTP test set of S_4 is smaller than S_3.

(2) The CTFTP test set of a general Boolean expression is larger than its corresponding IDNF test set. For example, the CTFTP test set of S_2 is larger than that of S_1. In Example 3, the CTFTP test set of (6) contains 16 test cases, which is larger than the test set of S_4.

VII. RELATED WORK

Several strategies are used for the fault detection and analysis of Boolean expressions. Tai [25] extended the work in [3] to detect faults including BORs, Boolean and relational operators (BRO), and Boolean and relational expressions (BRE), and devised BOR-, BRO-, and BRE-adequacy testing strategies to generate the corresponding test cases. Yu *et al.* [21] extended and complemented the theoretical work in [7] to evaluate the cost-effectiveness of testing experimentally the IDNF by means of the MUMCUT test strategy. Their experimental results showed that the greedy CUN and UCN methods are better than other methods in generating smaller test sets, and the test set sizes are linearly correlated with the length of IDNFs. Chen *et al.* [26] developed a web-based Boolean expression fault-based test case generation tool called BEAT by using the MUMCUT test strategy. Kaminski *et al.* [27] extended the work in [7] to cover term insertion fault, term negation fault, semantic test criteria, and MUTP/NFP test criteria. Kaminski and Ammann [28] presented a new logic criterion called the Minimal-MUMCUT

criterion to reduce the MUMCUT test set size in minimum DNF. The experimental study that used the TCAS II specifications determined that if feasibility is not considered, then Minimal-MUMCUT will reduce the test set size to a few percentages of the required test set size without sacrificing fault detection. However, those efforts focused only on improving the existing test adequacy criteria to reduce test set size and ignored the randomness and repeatability of selecting MUMCUT test cases. By contrast, the proposed CTFTP strategy in the present work is used to test the IDNF of Boolean expressions and ensure that the same seven types of faults similar to those of the MUMCUT strategy can be detected, but the CTFTP test set is smaller. Our experimental results show that the average size of the CTFTP test set is 1.7 and 2.5 percentage points smaller than those of G-CUN and G-UCN, respectively.

Researchers have explored in recent years some test generation techniques for Boolean expressions. Feng *et al.* [29] explored the application of four test strategies (partition strategy, decision table-based test, basic MI strategy, and fault-based test) in table-based specifications. They compared the four test strategies on a mathematical basis by means of a formal and precise definition of the subsumption relationship and showed in most cases that the basic MI strategy was the strongest whereas the partition strategy was the weakest. Kalaee and Rafe [30] proposed a new method to test Boolean specifications based on cause-effect graphs to generate pairwise tests. Their method used a reduced ordered binary decision diagram to reduce the Boolean specification and utilized a particle swarm optimization algorithm to select the optimal test suite, which lowered the test cost and test time. Li *et al.* [31] proposed an SMT-based MI strategy with high spatiotemporal performance to improve the automatic testing technology of large Boolean expressions in the field of interlocking systems. Brida and Scilingo [32] proposed a Boolean expression expander to evaluate the adequacy of a test suite. Their proposed expander was a novel mutation operator focusing on Boolean expressions, and it operated by strengthening and weakening such expressions. An overview of the research efforts regarding software combination testing with binary inputs is presented in [33]. However, those studies were used mainly for the fault detection of special Boolean expressions but could not generate directly the test suites for general Boolean expressions. On the basis of the CTFTP strategy, we propose in the present work an algorithm called general Boolean expression test generation algorithm based on literal substitution that generates automatically a test suite of general Boolean expressions.

Paul *et al.* [34] proposed a singular true position coverage strategy to detect LIFs in general Boolean expressions by using approximately half of the test cases with the modified condition/decision coverage (MC/DC) strategy. The MC/DC strategy, a structural code coverage metric, was originally defined in the DO-178B standard and intended to be an efficient coverage metric for evaluating software testing processes that incorporate decisions with complex Boolean expressions [35]. By considering the MC/DC criterion, Jones and Harrold [36] proposed two new algorithms for reducing test suites, break-down reduction and build-up reduction algorithms, and a new algorithm for test-suite prioritization. The experimental studies that used TCAS II specifications indicated that the proposed test suite reduction techniques could reduce effectively the test suites while providing acceptable performance. Gay *et al.* [37] used four real-world avionics systems to explore the effects of an implementation structure on the efficiency of test suites meeting the MC/DC criterion. Their experimental results demonstrated that the test suites that achieved MC/DC over implementations with structurally complex Boolean expressions were generally larger and more effective than the test suites that achieved MC/DC over functionally equivalent but structurally simpler implementations. Ayav [38] proposed a Fourier analysis-based prioritization method for MC/DC test suites. Under certain fault hypotheses, the strict negative correlation between the fault exposure potentials of the test cases and the influence values of their associated input conditions allowed for easy the ordering of the test cases without extensive mutation analysis. Kitamura *et al.* [39] developed an algorithm based on SAT solving to generate a minimum MC/DC test suite of Boolean expressions within a reasonable time. However, the MC/DC strategy has weak fault detection capabilities. Finding fault types in complex Boolean expressions is not easy, and it requires an important amount of time. Yu and Lau [24] compared MC/DC, MUMCUT, and several other related coverage criteria for logical decisions by means of formal and empirical analysis. Their results showed the MC/DC test sets were effective, but some faults might still be missed even if they can almost always be detected by a test set that meets the MUMCUT criterion. In the present study, the CTFTP test suite has the same fault detection capabilities similar to the MUMCUT strategy. The CTFTP strategy can be used to generate automatically the IDNF and general Boolean expression test cases, and it is suitable for the test case generation of complex Boolean expressions. The time complexities of the two proposed test generation algorithms are $O(2^{n-k_i})$ and $O((k-\mu)\cdot 2^{n-k_i}+\mu\cdot\nu\cdot k_u\cdot 2^{2n-\tau_i-k_w})$.

Petri net and its extended models, such as stochastic timed Petri net, generalized connected self-loop free Petri net (S^4PR), logic and threshold Petri net, Petri net with data operations, etc., are widely used in the modeling and analysis of discrete event scenarios. These models have been applied to the development of simulation engines for resource optimal provisioning analysis in emergency healthcare systems [18], the calculation of emptiable minimal siphon [40], the design of web service discovery strategy based on user requirements [41], and data-flow error detection in business processes [42]. Ding *et al.* [43] proposed a systematic strategy with specific algorithms to construct an interactive control model based on Petri net for human-computer interaction systems to analyze and control mobile robots. However, those Petri net models did not involve the expression of Boolean logic operation characteristics and thus could not be directly

applied to Boolean expression test case generation. In the present study, the OBLDPN model is proposed to analyze the interaction between Boolean transitions and Boolean literals and generate the test paths for Boolean expressions in view of obtaining the CTFTP test suite.

VIII. CONCLUSION

In this study, we propose a novel Boolean expression test generation strategy called the CTFTP strategy, which is based on a new OBLDPN model, to generate automatically certain test cases. By using a smaller test set, the CTFTP strategy can detect the same seven types of faults similar to those of the MUMCUT strategy. The number of places in the proposed OBLDPN model is only related to the number of variables in the Boolean expression and does not require the number of terms in the Boolean expression and the number of literals in each term. We specify the structure and characteristics of OBLDPN and describe its dynamic behavior. The CTFTP strategy is further extended to generate a test suite for general Boolean expressions, and a general Boolean expression test case generation algorithm based on literal substitution is proposed. The algorithm can detect five types of faults of general Boolean expressions, but the LIF and LRF cannot always be detected.

The technologies proposed in this study are valid, and the samples provided are representative; thus, the threats to internal validity will not likely exist. We provide logical and mathematical representations of the extended Petri net method and Boolean expression testing technology and apply the proposed OBLDPN model to Boolean expression testing by means of detailed explanations and illustrations. For example, the study has investigated the relationship between the logical faults and topological changes of Boolean expressions, the relationship between path coverage and Boolean expression topology, the relationship between test path constraints and logic fault detection, and the relationship between label and behavior of then OBLDPN. All of these techniques have been tested and verified by many experiments. The results have been manually evaluated to ensure the correctness of the experiments. However, the problem of test generation of general Boolean expressions involving nested subexpressions is not considered in this study. Further work is necessary to extend the CTFTP strategy onto more particular or wider Boolean specifications taken from real programs and evaluate the adequacy of testing for general Boolean expressions. The Boolean expressions may involve nested subexpressions, implication operators, equivalent operators, and so on. Due to the complexity of the algorithm, we also need to simplify the OBLDPN model. Some alternative approaches to generate efficiently a CTFTP test suite are being investigated to reduce test costs.

REFERENCES

[1] L. Yu and W.-T. Tsai, "Test case generation for Boolean expressions by cell covering," *IEEE Trans. Softw. Eng.*, vol. 44, no. 1, pp. 70–99, Jan. 2018.
[2] G. Fraser and A. Gargantini, "Generating minimal fault detecting test suites for Boolean expressions," in *Proc. 3rd Int. Conf. Softw. Test., Verification, Validation Workshops*, Paris, France, Apr. 2010, pp. 37–45.
[3] K. C. Tai and H. K. Su, "Test generation for Boolean expressions," in *Proc. 11th Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, Tokyo, Japan, Oct. 1987, pp. 278–284.
[4] E. Weyuker, T. Goradia, and A. Singh, "Automatically generating test data from a Boolean specification," *IEEE Trans. Softw. Eng.*, vol. 20, no. 5, pp. 353–363, May 1994.
[5] T. Y. Chen, M. F. Lau, and Y. T. Yu, "MUMCUT: A fault-based strategy for testing Boolean specifications," in *Proc. 6th Asia–Pacific Softw. Eng. Conf. (ASPEC)*, Takamatsu, Japan, Dec. 1999, pp. 606–613.
[6] T. Y. Chen and M. F. Lau, "Two test data selection strategies towards testing of Boolean specifications," in *Proc. 21st Annu. Int. Comput. Softw. Appl. Conf. (COMPSAC)*, Washington, DC, USA, Aug. 1997, pp. 608–611.
[7] T. Y. Chen and M. F. Lau, "Test case selection strategies based on Boolean specifications," *Softw. Test., Verification Rel.*, vol. 11, no. 3, pp. 165–180, Sep. 2001.
[8] T. Y. Chen, M. F. Lau, K. Y. Sim, and C. A. Sun, "On detecting faults for Boolean expressions," *Softw. Qual. J.*, vol. 17, no. 3, pp. 245–261, Sep. 2009.
[9] C.-A. Sun, Y. Dong, R. Lai, K. Y. Sim, and T. Y. Chen, "Analyzing and extending MUMCUT for fault-based testing of general Boolean expressions," in *Proc. 6th IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Seoul, South Korea, Sep. 2006, pp. 184–189.
[10] C.-A. Sun, Y. Zai, and H. Liu, "Evaluating and comparing fault-based testing strategies for general Boolean specifications: A series of experiments," *Comput. J.*, vol. 58, no. 5, pp. 1199–1213, May 2015.
[11] K. Kapoor and J. P. Bowen, "Test conditions for fault classes in Boolean specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 3, Jul. 2007, Art. no. 10.
[12] Z. Chen, T. Y. Chen, and B. Xu, "A revisit of fault class hierarchies in general Boolean specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, Aug. 2011, Art. no. 13.
[13] A. Gargantini and G. Fraser, "Generating minimal fault detecting test suites for general Boolean specifications," *Inf. Softw. Technol.*, vol. 53, no. 11, pp. 1263–1273, Nov. 2011.
[14] Z. Wang, X. Li, Y. Li, and Y. Dai, "Comparing minimal failure-causing schema and probabilistic failure-causing schema on Boolean specifications," *Int. J. Performability Eng.*, vol. 15, no. 10, pp. 2709–2717, Oct. 2019.
[15] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, "Requirements specification for process-control systems," *IEEE Trans. Softw. Eng.*, vol. 20, no. 9, pp. 684–707, Sep. 1994.
[16] Z. Wang, Y. Li, X. Gu, X. Zheng, and M. Yu, "Fault detection capabilities of combinatorial testing and random testing for Boolean-specifications," *Int. J. Performability Eng.*, vol. 15, no. 11, pp. 2952–2961, Nov. 2019.
[17] R. A. Silva, S. D. R. S. De Souza, and P. S. L. de Souza, "A systematic review on search based mutation testing," *Inf. Softw. Technol.*, vol. 81, pp. 19–35, Jan. 2017.
[18] J. Zhou, J. Wang, and J. Wang, "A simulation engine for stochastic timed Petri nets and application to emergency healthcare systems," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 4, pp. 969–980, Jul. 2019.
[19] H. Gong and C. Huang, "Automatic test case generation for general form Boolean expressions based on predicate-driven Petri nets," *Int. J. Adv. Comput. Technol.*, vol. 3, no. 8, pp. 146–153, Sep. 2011.
[20] A. Paradkar and K. C. Tai, "Test generation for Boolean expressions," in *Proc. 6th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Toulouse, France, Oct. 1995, pp. 106–115.
[21] Y. T. Yu, M. F. Lau, and T. Y. Chen, "Automatic generation of test cases from Boolean specifications using the MUMCUT strategy," *J. Syst. Softw.*, vol. 79, no. 6, pp. 820–840, Jun. 2006.
[22] J. Sziray, "Evaluation of Boolean graphs in software testing," in *Proc. IEEE 9th Int. Conf. Comput. Cybern. (ICCC)*, Tihany, Hungary, Jul. 2013, pp. 225–230.
[23] A. Moraes, W. L. Andrade, and P. D. L. Machado, "A family of test selection criteria for timed input-output symbolic transition system models," *Sci. Comput. Program.*, vol. 126, pp. 52–72, Sep. 2016.
[24] Y. T. Yu and M. F. Lau, "A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions," *J. Syst. Softw.*, vol. 79, no. 5, pp. 577–590, May 2006.
[25] K.-C. Tai, "Theory of fault-based predicate testing for computer programs," *IEEE Trans. Softw. Eng.*, vol. 22, no. 8, pp. 552–562, Aug. 1996.

[26] T. Y. Chen, D. D. Grant, M. F. Lau, S. P. Ng, and V. R. Vasa, "BEAT: A Web-based Boolean expression fault-based test case generation tool," *Int. J. Distance Educ. Technol.*, vol. 4, no. 2, pp. 44–56, Apr. 2006.
[27] G. Kaminski, G. Williams, and P. Ammann, "Reconciling perspectives of logic testing for software," *Softw. Test. Verification Rel.*, vol. 18, no. 3, pp. 149–188, Sep. 2008.
[28] G. K. Kaminski and P. Ammann, "Using logic criterion feasibility to reduce test set size while guaranteeing fault detection," in *Proc. Int. Conf. Softw. Test. Verification Validation*, Denver, CO, USA, Apr. 2009, pp. 356–365.
[29] X. Feng, D. L. Parnas, T. H. Tse, and T. O'Callaghan, "A comparison of tabular expression-based testing strategies," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 616–634, Sep. 2011.
[30] A. Kalaee and V. Rafe, "An optimal solution for test case generation using ROBDD graph and PSO algorithm," *Qual. Rel. Eng. Int.*, vol. 32, no. 7, pp. 2263–2279, Nov. 2016.
[31] Z. Li, J. Liu, H. Sun, T. Zhou, and J. Sun, "Automatic test generation of large Boolean expressions in computer based interlocking system," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Nanjing, China, Dec. 2017, pp. 513–520.
[32] S. G. Brida and G. Scilingo, "Boolean expression extender—A mutation operator for strengthening and weakening Boolean expression," in *Proc. 43rd Latin Amer. Comput. Conf. (CLEI)*, Cordoba, Argentina, Sep. 2017, pp. 1–10.
[33] S. Vilkomir, "Combinatorial testing of software with binary inputs: A state-of-the-art review," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion (QRS-C)*, Vienna, Austria, Aug. 2016, pp. 55–60.
[34] T. K. Paul, M. F. Lau, and S. Ng, "On a new detecting technique for conjunctive literal insertion fault in Boolean expressions," in *Proc. 14th Int. Conf. Qual. Softw.*, Dallas, TX, USA, Oct. 2014, pp. 266–275.
[35] S. Kandl and S. Chandrashekar, "Reasonability of MC/DC for safety-relevant software implemented in programming languages with short-circuit evaluation," *Computing*, vol. 97, no. 3, pp. 261–279, Mar. 2015.
[36] J. A. Jones and M. J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Trans. Softw. Eng.*, vol. 29, no. 3, pp. 195–209, Mar. 2003.
[37] G. Gay, A. Rajan, M. Staats, M. Whalen, and M. P. E. Heimdahl, "The effect of program and model structure on the effectiveness of MC/DC test adequacy coverage," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, Aug. 2016, Art. no. 25.
[38] T. Ayav, "Prioritizing MCDC test cases by spectral analysis of Boolean functions," *Softw. Test. Verification Rel.*, vol. 27, pp. 1–18, Aug. 2017.
[39] T. Kitamura, Q. Maissonneuve, E.-H. Choi, C. Artho, and A. Gargantini, "Optimal test suite generation for modified condition decision coverage using SAT solving," in *Proc. 37th Int. Conf. Comput. Saf., Rel., Secur. (SAFECOMP)*, Vasteras, Sweden, Sep. 2018, pp. 123–138.
[40] S. Wang, W. Duo, X. Guo, X. Jiang, D. You, K. Barkaoui, and M. Zhou, "Computation of an emptiable minimal siphon in a subclass of Petri nets using mixed-integer programming," *IEEE/CAA J. Autom. Sinica*, early access, Jun. 2, 2020, doi: 10.1109/JAS.2020.1003210.
[41] J. Sha, Y. Du, and L. Qi, "A user requirement oriented Web service discovery approach based on logic and threshold Petri net," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 6, pp. 1528–1542, Nov. 2019.
[42] D. Xiang, G. Liu, C. Yan, and C. Jiang, "Detecting data-flow errors based on Petri nets with data operations," *IEEE/CAA J. Autom. Sinica*, vol. 5, no. 1, pp. 251–260, Jan. 2018.
[43] Z. Ding, H. Qiu, R. Yang, C. Jiang, and M. Zhou, "Interactive-control-model for human–computer interactive system based on Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 4, pp. 1800–1813, Oct. 2019.

HONGFANG GONG received the B.S. degree in mathematics from the Changsha University of Science and Technology, Changsha, China, in 1991, and the M.E. degree in computer application and the Ph.D. degree in computer science and technology from Hunan University, China, in 2004 and 2018, respectively. He is currently an Associate Professor of information science with the Changsha University of Science and Technology. His research interests include software test, cyber-physical systems (CPSs), and embedded computing systems.

JUNYI LI received the B.S., M.S., and Ph.D. degrees in computer application from Hunan University, China, in 1993, 2001, and 2008, respectively. He has been an Associate Professor with Hunan University, since 2005. From 2009 to 2010, he was a Visiting Researcher with Lakehead University, Canada. His research interests include big data analysis, software engineering, and autonomous driving.

RENFA LI (Senior Member, IEEE) is currently a Full Professor with the College of Computer Science and Electronic Engineering. He is also the Director of the Key Laboratory for Embedded and Network Computing of Hunan Province, Changsha, China. He is also an Expert Committee Member of the National Supercomputing Center, Changsha. His research interests include computer architecture, embedded computing systems, cyber-physical systems (CPSs), and the Internet of Things. He is a Senior Member of ACM and a member of the Council of China Computer Federation.