Automatic Test Case Generation for General Form Boolean Expressions Based on Predicate-Driven Petri Nets

Hongfang Gong, Chuangxia Huang
College of Mathematics & Computing Science, Changsha University of Science & Technology, Changsha 410004, China
Email: gonghf@csust.edu.cn, cxiahuang@126.com

Abstract
A predicate that appears in specification or code can implement a condition. In this paper, we propose a Predicate-Driven Petri Net (PDPN), which can be applied to analyze and test the behavior for the interaction between predicate operators and predicate individuals or variables or expressions. With the help of the behavior function, the dynamic properties of a PDPN and an iterative partition approach for PDPNs were investigated. A generalized BOR-MI strategy called as Meaning Match Search approach (MeMS) is developed to generate a minimal MeMS-constraint set. Using the presented algorithm, one can choose a test suite that guarantees the detection of Boolean operator faults.

Keywords: Predicate Test, Boolean Operators, Automatic Test Case Generation, Predicate-Driven Petri Nets, Iterative Partition.

1. Introduction
Software testing is one of key technologies to guarantee quality and reliability of software [1]. A predicate that appears in specification or code can implement a condition that describes a requirement of the software system under development. Predicate testing, which requires certain types of tests for each predicate, can guarantee to detect certain faults in the coding of conditions, and ensure that there are no errors in the implementation of conditions [2, 3]. Predicate testing targets three classes of fault: Boolean operator fault, relational operator fault, and arithmetic expression fault [4]. In this paper, we focus on the detection of Boolean operator faults in a general form Boolean expression.

Test cases generation criteria or strategies for Boolean expression have been proposed by software researchers in the past two decade. Tai [5] devised BOR-, BRO-adequate tests strategies for generating test cases, but they restricted their works to the testing of singular Boolean expressions. Weyuker et al. [6] designed a MI (Meaning Impact) strategy for automatically generating test cases that can be applied to singular or nonsingular Boolean expressions. Paradkar et al. [7] proposed a BOR-MI strategy that can generate more effective and less test cases than the original MI strategy. Chen et al. [8, 9, 10] proposed a set of more efficient test case generation strategies called MUMCUT. Kaminski et al. [11] introduced an extension of MUMCUT, called Minimal-MUMCUT, which guarantees to detect the same types of faults with fewer test cases.

All these fault-based strategies only select a subset of an exhaustive test set and achieve a higher fault-detection capability [12,13,14]. However, these strategies, which are referred to as IDNF-oriented test strategies [15], are applicable only to some restricted forms such as Irredundant Disjunctive Normal Form (IDNF) of Boolean expression, and not to the original expression. This means that faults are introduced in the context of a general form Boolean expression that is implemented by a designer or programmer [16]. The corresponding equivalent IDNF may change some faults of the original expression, and may miss detection of some faults of the original Boolean specifications, too.

In this paper, we propose a Predicate-Driven Petri Nets (PDPN) that is a high-level formalism of Petri Nets [17], one of techniques can be applied to analyze and test the behavior for the interaction between predicate operators and predicate individuals or variables or expressions. By using PDPNs, we present an generalized BOR-MI strategy called as Meaning Match Search approach, or simply MeMS, for testing general form Boolean expressions. In our strategy, a Boolean expression can be partitioned into several sub-expressions by using the iteration partition of PDPNs. Each sub-expression will be abstracted into a place of the PDPN in next iteration. The MeMS strategy of test case generation can
guarantee the detection of Boolean operator faults.

This paper is organized as follows. Section 2 formally defines PDPNs model. Section 3 describes the dynamic properties of a PDPN by the behavior function. In section 4, an iterative partition approach for PDPNs and the algorithm that generates test cases are described. Section 5 discusses an example for generating test suite. Finally, a conclusion is given in Section 6.

In the following, similarly proposed by Lau and Yu [18,19], we will use “,” “·” and “” to denote the Boolean operators AND (or “∧”), OR (or “∨”) and NOT (or “¬”), respectively. Moreover, we will omit the “·” symbol whenever it is clear from the context. The truth values “TRUE” and “FALSE” are denoted by “t” and “⊥”, respectively. We use “β∗” to denote the set of all truth values, i.e. β = {t, ⊥}. The τ-dimensional Boolean space is denoted by β∗. In a Boolean expression, a variable may occur as a positive literal or a negative literal. For example, for the same variable b in the Boolean expression \( b\_c + \overline{b}\_d \), b and \( \overline{b} \) are positive literal and negative literal, respectively, and referred to as complementary pairs, too.

2. Predicate-Driven Petri Nets

**Definition 1** A Predicate-Driven Petri Net (PDPN) is a 15-tuple \( (P, T, F, V, B, L, ψ, Θ, S, M_0, E, In, Out, s_o, s_f) \), where

1. \( P \) is a finite set of predicate places modeling states of a system; \( T \) is a finite set of transitions involving operations of predicate expressions, and \( P \cap T = φ, P \cup T = F \subseteq (P \times T) \cup (T \times P) \) is a flow relation, or simply a set of arcs.
2. \( V \) is a set of predicate formulae including individuals (constants), variables and first order logic formulae built from individuals, variables and operations.
3. \( B \) represents the infinite set of all values for corresponding variables in \( V \), and \( β \subseteq B \).
4. The inscription in a PDPN \( Σ \) is defined by a 5-tuple \( (L, ψ, Θ, S, M_0) \), where

\[-L : F \rightarrow (V', V', \ldots, V')\]

is a labeling function on arcs. Given an arc \( f \in F \), the labeling of \( f \), \( L(f) \), is a set of labels, which are \( n \)-tuple of the element in \( V' \) that is a subset of \( V \) and only embraces individuals, variables in \( V \). The tuples in \( L(f) \) have the same length, representing the arity of the predicate connected to the arc \( f \).

\[-ψ : T \rightarrow V \]

is a mapping. The inscription formula on transition \( t \in T \), \( ψ(t) \), is a first order logical formula in \( V \). If \( (t, p) \in F \), then \( ψ(t) \) can be denoted as \( ψ\_t(t) \).

\[-Θ \]

is a set of all substitutions. An enabled transition \( t \in T, ∃ b \in B \) such that \( b/l \in M(p) \) or \( ∀ l \in L(p, t) \), and \( ψ(t) \) evaluates TRUE, where \( b/l \) yields a token by substituting all variables in label \( l \) with the corresponding bound values of \( b \). Let \( θ = \{b/l\} \), and referred to as a substitution.

\[-S : V \rightarrow β \]

is a constraint set. For \( ∀ p \in V \), any test case makes \( p \) either TRUE or FALSE. Thus, a set of constraint can be divided into true constraint set \( S'_p \) and false constraint set \( S'_p \) such that \( S_p = S'_p \cup S'_p \), where \( ∀ s \in S'_p, ψ\_s(p) = TRUE \) and \( ∀ s \in S'_p, ψ\_s(p) = FALSE \).

\[-M_0 \]

is the initial marking. \( M_0 = \bigcup_{p \in P} M_p(p) \), where \( M_p(p) \) is the set of tokens residing in predicate \( p \). Each token is described as \( b/l \), where \( b \in B, l \in L(p, t) \) or \( l \in L(t, p) \).

\[(5) E = [tθ] t \in T \land θ \in Θ \]

is a set of events, and represents the firing of enabled transition \( t \ w.r.t \ θ \).

\[(6) In \subseteq (P \cup E) \times T \]

is a set of input, and denotes input actions associated with the transition.

\[(7) Out \subseteq T \times (P \cup E) \]

is a set of output, and denotes output actions associated with the transition.

\[(8) \]

In a PDPN, there is initial node \( s_o \) and terminal node \( s_f \), and

\[s_o \notin P \cap E \cap T \land \forall p \in P \land d^\_o(p) = 0 \rightarrow (s\_o, p) \notin F, \]

\[s_f \notin P \cap E \cap T \land \forall p \in P \land d^\_f(p) = 0 \rightarrow (p, s\_f) \notin F. \]

Where \( d^\_o(p) \) and \( d^\_f(p) \) denote outdegree and indegree of the node \( p \), respectively.
For example, a general form Boolean expression \( a(bc + \overline{bd}) + \overline{de} \) can be represented as a PDPN as shown in Figure 1.

![Figure 1. A PDPN for A Boolean Expression](image)

In Figure 1, each single variable or its negation is directly described in the predicate node, and each inscription formula \( y(t) \) on transition \( t \in T \), for the corresponding arc \(( t, p) \in F \), is represented in angle brackets near predicate node \( p \). A sub-PDPN denoted in a dotted frame is supplied for iteration.

3. Behavior Description of PDPNs

In a PDPN \( \Sigma \), Let \( \cdot = \{ p \} | ( p, t) \in F \land p \in P \} \) be the precondition predicates and the post-condition predicates of transition \( t \), respectively. Let \( \cdot = \{ t \} | ( ( p, t) \in F \land t \in T ) \lor ( p, s) \} \) be the sets of input transitions and output transitions of predicate \( p \), respectively. An event \( e \in E \), which represents firing of an enabled transition \( t \) w.r.t. \( \theta \in \Theta \), removes all tokens in \( \{ b/l \} b \in B \land l \in L(p, t) \) from each input predicate \( p \in t \), and adds all tokens in \( \{ b/l \} b \in B \land l \in L(p, t) \) to each output predicate \( p \in t \). Let \( \cdot = \{ b/l \} b \in B \land l \in L(p, t) \land ( \forall p \in t ) \) and \( \cdot = \{ b/l \} b \in B \land l \in L(t, p) \land ( \forall p \in t ) \) be the input (precondition) tokens and output (post-condition) tokens of firing \( t \theta \), respectively [20].

A behavior function \( \rho \) of a PDPN \( \Sigma \) maps a transition to a set of all possible scenarios developed, and is formal defined as follows:

**Definition 2** Let \( \Pi = \{ \pi \} | \pi : V \to B \} \) be a set of scenarios, then the behavior function \( \rho \) of a PDPN \( \Sigma \) is \( \rho : T \to 2^\Pi \), and represented as follows:

\[
\rho(t) = (y(\cdot (t)) \in S^\prime \cap (M^\cdot (t)) \cup (t'))
\]

Where \( M \) denotes a marking of the PDPN \( \Sigma \) before the firing \( t \theta \), and \( M^\cdot (t) \cup (t') \) denotes the new marking after the firing \( t \theta \).

**Definition 3** Given a set of substitution sequence \( \Theta = \Theta_1, \Theta_2, \cdots, \Theta_k > > \), such that enable transition sequence \( T = t_1, t_2, \cdots, t_k \), where \( k \) is the number of all transitions in PDPN \( \Sigma \) and \( k > 0 \), can in turn be fired under the initial marking \( M_0 \) w.r.t. \( \Theta \), respectively. The sequence \( TP = s_0, t_1, t_2, \cdots, t_k, s_j > > \) is referred to as a test path of \( \Sigma \). A test case under \( TP \) for testing the Boolean expression is a truth value assignment to every Boolean variable in the formula, and
represented as $TC = \omega_1, \omega_2, \ldots, \omega_r$, where $r$ is the number of variables in the Boolean expression, $\omega_i \in V$ denotes a variable of $V$, and $\omega_i \in \beta(i=1,2,\ldots,r)$, which satisfies $(\omega_1, \omega_2, \ldots, \omega_r) \in S_p \land (p^* = s_j)$. A test suite is a subset of the $\tau$-dimensional Boolean space $\beta'$, and represented as $TS = \{TC \subset \beta'\}$.

4. Test Cases Generation Based on PDPNs

4.1. Iterative partition for PDPNs

As we all know, the weighted EDPN is a high-level formalism of Petri Nets [21]. In [22], using iteration of the weighted EDPN, an approach that partitions a system of OOP into several subsystems was proposed. Similarly, a Boolean expression can be partitioned into several sub-expressions by using the iteration of PDPNs, too. A sub-expression $p_i$ will be abstracted into a place of the PDPN in next iteration. The new place is denoted using a literal that is the complementary of any literal in another sub-expression $p_j$. However, we generate a test suite $TS$ from a given Boolean expression $p$ such that $TS$ is minimal. The complementary literal replacing $p_i$ should firstly be chosen from literals simultaneously occurring in both $p_i$ and $p_j$.

For example, we consider the general form Boolean expression $a(b+c+d) + \overline{e}$, As shown in a dotted frame in Figure 1, the sub-PDPN of sub-expression $bc + \overline{d}$ is replaced by the complementary literal $d$ of literal $d$ in $e$, and then we can obtain a new Boolean expression $ad + \overline{e}$. After iterating the PDPN, the new PDPN for $ad + \overline{e}$ must be processed as follows:

- A predicate place $p$ for $d^*(p) > 1$ is divided into $d^*(p)$ duplicate places $p_1, p_2, \ldots, p_{d^*(p)}$ of $p$, and $d^*(p) = 1, i=1,2,\ldots,d^*(p)$. All predicate places satisfy a first order logic formula $\forall p \exists t ((p,t) \in F \land (p, t') \in F \rightarrow (t = t'))$.
- If some duplicate place $p_i$ of $p$ is the same as the place represented in the complementary literal using iteration, we remove $p_i$ and its associated transition or initial node $s_0$.
- If $d^*(p_i) = 0$ for any place $p_i$, we add an initial node $s_0$ such that $s_0 \in p_i \land s_0 \notin T$.

The iterated PDPN $\Sigma'$ can be expressed as the following:

![Figure 2. A new PDPN after being iterated](image-url)
4.2. Test Cases Generation Algorithms

In order to generate minimal test suites, the set product must be considered firstly. The onto set product operator, which is written as “⊗”, can be defined as follows: for finite sets \( P \) and \( Q \). \( P \otimes Q \) is a minimal set of pairs \((p, q)\) such that \( p \in P, q \in Q \), and each element of \( P \) appears at least once as \( p \) and each element of \( Q \) appears at least once as \( q \) [2]. Boolean operator such as “∧”, “∨” and “¬” in a transition node \( t \) of a PDPN \( Σ \) is denoted as \( t_{op} \) by a joint name, i.e. \( t_{op} = ∧, t_{op} = ∨ \) or \( t_{op} = ¬ \).

We have presented an extended BOR-MI strategy named Meaning Match Search approach (MeMS) for testing general form Boolean expressions based on PDPNs. The strategy involves two algorithms named MeM-procedure and MeMS-procedure, which generate MeM-constraint set, simply MeM-CSET, and MeMS-constraint set, simply MeMS-CSET, respectively. The MeM-procedure is as follows:

Procedure for generating a minimal MeM-CSET from a new PDPN \( Σ \) of a Boolean formula \( ψ_p(t) \) associated with predicate place \( p \).

Input: A PDPN \( Σ \) for \( p \).
Output: Generating MeM-CSET for \( p \).
Procedure: MeM-Procedure.
Step 1 Search a test path \( TP = s, t, t, \ldots, t, s, > \).
Step 2 For \( ∃ p \in P, s_0 t p, s_0 t p \notin T \), we initialize each predicate place \( p \) with its constraint set \( S_p = \{t, f\} \) and \( S_p = S_p' \cup S_p'^1, S_p^1 = \{t\}, S_p^1 = \{f\} \).
Step 3 Under \( TP \), in accordance with the order of firing \( t, \theta, \ldots, t, \theta, > \), we have \( S_p \) of predicate place \( p \in t^* \), and \( t, \theta, \ldots, t, \theta, > \). \( i = 1, 2, \ldots, k \). For each transition \( t_i \), consider its associated predicate places \( p, p_1, \) and \( p_2 \).

3.1 If \( t_{op} = ∧ \), and \( p_1, p_2 \in t^* \), then \( S_p^i = S_p^i \otimes S_p'^i, S_p'^i = (S_p^i \otimes S_p'^i) \cup (S_p'^i \otimes S_p'^i) \).

3.2 If \( t_{op} = ∨ \), and \( p_1, p_2 \in t^* \), then

3.2.1 Assume that the inscription formula \( ψ_{p_1}(t_{p_1}), ψ_{p_2}(t_{p_2}) \) and \( ψ_p(t_{p}) \) have m, n and \( τ \) variables, respectively, where \( t_{p_1} \in p_1 \wedge t_{p_1} \neq s_0 \) and \( t_{p_2} \in p_2 \wedge t_{p_2} \neq s_0 \). For each constraint \( s_{p_1} \) of \( S_p^i \), add \( τ - m \) Boolean components in \( β \) by dictionary sort such that \( s_{p_1} \in β^i \). In this way, \( 2^{τ-n} \) constraints are obtained from each \( s_{p_1} \) in \( S_p^i \). Similarly, the number of constraints in \( S_p^i \) can be increased to \( 2^{τ-n} \) times as many.

3.2.2 Noting that \( CS_p^i \cap CS_p'^i = \emptyset \), where \( \emptyset \) represents an empty set, we denote \( CS_p^i = S_p^i \ominus S_p'^i \), and \( CS_p'^i = S_p^i \ominus S_p'^i \).

3.2.3 Construct \( S_p'^i \) by including one constraint from \( CS_p^i \) and \( CS_p'^i \), respectively, note that for each constraint \( s \in S_p'^i \), we obtain \( ψ_{p_{s_0}}(t_{s}) = TRUE \).

3.2.4 Let \( CS_p'^i = S_p^i \ominus S_p'^i \cup S_p'^i \), and \( CS_p'^i = S_p^i \ominus S_p'^i \cup S_p'^i \), construct \( S_p'^i = CS_p'^i \cap CS_p'^i \), for any constraint \( s \in S_p'^i \), we obtain \( ψ_{p_{s_0}}(t_{s}) = FALSE \).

3.3 If \( t_{op} = ¬ \), and \( p_1 \in^* t, p \in t^* \), then \( S_p = S_p^i \), \( S_p'^i = S_p^i \).

Step 4 Finally, we construct the desire constraint set for \( p \) as \( S_p = S_p^i \cup S_p'^i \).

End of MeM-Procedure.

Now, let us introduce the MeMS-procedure for generating a minimal constraint set. The aim of the procedure is to update the derived MeM-CSET of \( p \). If \( p \) involves a sub-expression \( λ \), then it can be changed into a simpler Boolean expression \( p' \) after \( λ \) is replaced by a literal \( v \). In addition, by
dictionary sort, it is easy to see that \( \nu \) is the \( j \)th variable of the original expression \( p_r \). The procedure is described as follows:

Procedure for generating a minimal MeMS-CSET from a PDPN \( \Sigma \) of a general form Boolean formula \( \psi_{p_r}(t) \) associated with predicate place \( p_r \).

**Input:** A PDPN \( \Sigma \) for \( p_r \), and \( s_j \in p_r^* \).

**Output:** Generating MeMS-CSET for \( p_r \).

**Procedure:** MeMS-Procedure.

Step 1 Generate the MeM-CSET \( S_{p_r}' \), \( S'_p \) and \( S'_J \) for inscription formula \( \psi_{p_r}(t) \) and \( \psi_j(t) \) in the new PDPN \( \Sigma' \), respectively, using the MeM-procedure.

Step 2 Using Step 3.2.1 from the MeM-procedure, one can extend each constraint set of \( S_{p_r}' \), \( S'_p \), \( S'_J \) and \( S'_J \), such that the component number of each constraint equals the variable number of \( p_r \).

Step 3 Let \( S_{p_r}' = S_{p_r}' \cap S'_J \), note that for each constraint \( s \in S_{p_r}' \), we have \( \psi_{p_r,0}(t) = \text{TRUE} \).

Step 4 The \( j \)th component Boolean value of each constraint \( s \) of constraint sets can be denoted as \( s.v_j \). Let \( S_{p_r}' = jS_{p_r}' \cup jS_{p_r}' \), \( S_{p_r}' = (jS_{p_r}' \cap S'_J) \cup (jS_{p_r}' \cap S'_J) \), where \( jS_{p_r}' \cap jS_{p_r}' = \emptyset \), \( jS_{p_r}' \) and \( jS_{p_r}' \) including all constraints satisfying \( s.v_j = t \) and \( s.v_j = f \), we have \( \psi_{p_r,0}(t) = \text{FALSE} \).

Step 5 Finally, we construct the desire constraint set for \( p_r \) as \( S_p = S_{p_r}' \cup S'_J \).

End of MeMS-Procedure.

### 5. An Example

We apply the obtained algorithm to generate the MeMS-CSET for the general form Boolean expression \( p_s : a(bc + bd) + de \). The PDPN \( \Sigma \) for \( p_s \) is shown in Figure 1. Let \( d \) replace \( bc + bd \), then the iterated PDPN for \( p_s : ad + de \) can be shown as Figure 2.

We compute MeM-CSET for \( p_s \) in Figure 2 by using MeM-procedure.

Firstly, we search a test path as \( TP = (s_j, t, \theta_j, t, \theta_j, t, \theta_j, t, \theta_j, s_j) \). The constraint sets for variables \( a, d, e \) are initialized as follows:

\[
S'_p = S'_J = S'_J = \{t\}, S'_p = S'_J = \{f\}.
\]

Secondly, we compute the constraint sets for \( p_s, \overrightarrow{d} \) and \( p_s \) in turn. For \( t, \neg op \), we have

\[
S'_{p_s} = S'_p \otimes S'_J = \{(t, t)\}, S'_J = (S'_p \otimes S'_J) \cup (S'_J \otimes S'_J) = \{(t, f), (f, t)\}.
\]

For \( t, \neg op \), we have \( S'_{p_s} = S'_J = \{f\}, S'_J = S'_J = \{t\} \).

For \( t, \neg op \), similar to \( S'_{p_s} \) and \( S'_J \), one can compute the constraint sets for \( p_s \) as follows:

\[
S'_{p_s} = \{(f, t)\}, S'_J = \{(f, f), (f, t)\}.
\]

Finally, we construct the MeM-CSET for \( p_s \). For \( t, \neg op \), we extend each constraint of \( S_{p_s} \) and \( S_{p_s} \) by the dictionary sort for variables \( a, d, e \).

\[
S'_{p_s} = \{(t, t), (t, f, t)\}, S'_J = \{(t, f, t), (t, f, f), (f, t, t), (f, f, t)\},
\]

\[
S'_{p_s} = \{(t, f, t), (f, f, f)\}, S'_J = \{(t, f, f), (f, f, f), (f, f, f)\}.
\]

Now, we construct \( S'_{p_s} \) and \( S'_{p_s} \), respectively, by applying Step 3.2 from MeM-procedure. We have

\[
CS'_{p_s} = \{(t, t), (t, t, f)\}, CS'_{p_s} = \{(t, f, t), (f, f, f)\}.
\]
Selecting one constraint from \( CS_{p_i} \) and \( CS'_{p_i} \), respectively, we obtain a minimal constraint set \( S'_{p_i} \) that make \( p_s \) TRUE, i.e. \( S'_{p_i} = \{ (t,t,t,t), (t,f,t,t) \} \).

We can derive the false constraint set \( S^f_{p_i} = CS^f_{p_i} \cup CS^f'_{p_i} = \{ (t,f,f,f), (f,t,t,t) \} \). Where

\[
CS^f_{p_i} = S^f_{p_i} - \bigcup S^f_{p_i} = \{ (t,f,f,t), (t,t,f,t), (f,f,t,t) \}, \quad CS^f'_{p_i} = S^f_{p_i} - \bigcup S^f_{p_i} = \{ (t,f,f,t), (f,f,f,t) \}.
\]

The set of constraint \( S_{p_i} \) contains a total of four constraints, i.e. \( S_{p_i} = \{ (t,t,f,f), (t,f,t,t), (f,f,t,t), (f,t,f,t) \} \).

Similarly, by using the MeM-procedure, we can obtain the MeM-CSET for \( p_k : bc + \bar{d}d \) which is represented as a sub-PDPN shown in a dotted frame in Figure 1. The sets \( S'_{p_i}, S^f_{p_i} \) and \( S_{p_i} \) are listed as below:

\[
S'_{p_k} = \{ (t,t,f,f), (f,f,t,t) \}, \quad S^f_{p_k} = \{ (t,f,f,f), (f,t,t,t) \}, \quad S_{p_k} = \{ (t,t,f,f), (f,f,t,t), (f,t,f,t) \}.
\]

We update MeMS-CSET for \( p_k \) in Figure 1. Firstly, we extend each constraint \( s \) of \( S'_{p_i}, S^f_{p_i}, S'_{p_i} \) and \( S_{p_i} \) such that each \( s \) includes five components. It is easy to obtain constraint sets of a general form

Boolean expression \( p_k : a(bc + \bar{d}d) + de \) using MeMS-procedure, and represented as follows:

\[
S'_{p_k} = \{ (t,f,t,t,f), (t,t,t,f,t) \}, \quad S^f_{p_k} = \{ (t,f,t,f,f), (f,f,t,t,t) \}, \quad S_{p_k} = \{ (t,t,f,t,f), (t,t,t,f,t), (t,f,f,t,t) \}.
\]

We generate a test suite \( TS \) involving four test cases for Boolean expression described above. \( TS \) can guarantee the detection of boolean operator faults, and is minimal. For example, A test case with the corresponding constraint \( s = (t,f,t,f,f) \) is \( t_e = a = t, b = f, c = t, d = t, e = f \) such that \( y_{p_k(s)} = \text{TRUE} \), et al.

### 6. Conclusion

In this paper, we proposed a novel test strategy for testing Boolean expressions. We specify a Boolean expression by PDPNs, and describe the dynamic behaviors of PDPNs. Furthermore, by using iteration partition for PDPNs, a MeMS approach for testing general form Boolean expressions is also presented to generate a minimal MeMS-CSET. Our strategy can guarantee the detection of Boolean operator faults.

### 7. Acknowledgement

The authors are extremely grateful to the anonymous reviewers for their constructive and valuable comments, which have contributed a lot to the improved presentation of this paper. The work was partially supported by National Natural Science Foundation of China (No. 11101053, 1087103, 11072041), the Science and Technology Project of Hunan of China (No. 2010FK3025), the Foundation of Chinese Society for Electrical Engineering (2008), the Key Project of Chinese Ministry of Education (No. 211118), the Excellent Youth Foundation of Educational Committee of Hunan Provincial (No. 10B002).

### 8. References


