2016 IEEE International Conference on Software Quality, Reliability and Security Companion

Moving Switching Functions to Continuous Domain

Tolga Ayav
Department of Computer Engineering
İzmir Institute of Technology
35430 Urla-Izmir Turkey
Email: tolgaayav@iyte.edu.tr

Hasan Sözer
Department of Computer Science
Ozyegin University
34794 Cekmekoy-Istanbul Turkey
Email: hasan.sozer@ozyegin.edu.tr

***Abstract*—This paper proposes a method for moving switching functions to continuous domain. The benefits of this approach are twofold. First, the elementary calculus works with the transformed functions. Second, this transformation approach facilitates various analyses relying on Boolean algebra and other existing Boolean-based calculi like Boolean difference. We present one of the potential applications and show how MCDC test pairs can be computed solely by means of elementary calculus.**

1. Introduction

Switching functions (or Boolean functions) are essential abstractions that are used for the specification and analysis of digital systems. They are used for the implementation of basic control structures that steer the control flow during system operation. As such, switching functions constitute fundamental concepts for the design and implementation of systems and software. Boolean algebra is mainly used as the basic mathematical tool for analyzing these functions [1]. In this paper, we introduce a novel analysis approach in which, switching functions are transformed to their counterpart representations in continuous domain. These representations allow the application of elementary calculus for analysis. They support various Boolean-based calculi like Boolean difference. They can also be transformed back to their counterpart representations in discrete domain with an inverse transformation.

Our approach is subject to a broad range of applications since switching functions are fundamental building blocks for the specification of any harware/software system. In this paper, we present one of these potential applications in the context of test case generation. In particular, we show how a set of test cases can be generated such that these test cases satisfy a structural coverage criterion, namely the modified condition and decision coverage (MCDC). Concerning this specific application of our approach, there already exist other methods to compose a set of test cases that satisfy MCDC [2]. In a broader context, there also exist methods to analyze Boolean expressions in general [3]. However, unlike existing methods, computations in our approach are performed based solely on elementary calculus. To the best of our knowledge, the use of elementary calculus has not been considered for the analysis of switching functions before.

The remainder of this paper is organized as follows. In the following section, we provide background information regarding switching functions and introduce the notation we use for Boolean variables and operations. In Section 3, we introduce Boolean difference calculus. In Section 4, we explain our transformation approach for representing switching functions in continuous domain. In Section 5, we illustrate the application of the approach for computing test pairs that satisfy MCDC. Finally, in Section 6, we conclude the paper.

2. Switching Functions

In this section, we introduce the basic notation for Boolean variables and operations that take part in switching functions. This notation will be used throughout the rest of the paper.

In this study, Boolean variables and operations are defined as follows:

$$x ::= \mathtt{F} \mid \mathtt{T} \mid \neg x \mid x_1 \Theta x_2, \quad \text{where } \Theta = \{\wedge, \vee, \uparrow, \downarrow, \oplus\}.$$

where x, x_1 and x_2 are boolean variables. Hereby, F and T symbols denote false and true values, respectively. Negation of a variable is denoted as $\neg x$. In addition, 5 more operators (Θ) can be defined on two different variables. These operators are "$\wedge$", "$\vee$", "$\uparrow$", "$\downarrow$", "$\oplus$", and they represent AND, OR, NAND, NOR and XOR operations, respectively. We define switching functions as $g : \mathbb{B}^n \rightarrow \mathbb{B}$ where $\mathbb{B} = \{\mathtt{F}, \mathtt{T}\}$. For example,

$$g(\mathbf{x}) = \neg a \vee b$$

is a switching function where $\mathbf{x} = [a, b]^T$ is the input vector. The truth table for this function is as follows:

a	b	$g(a,b)$
F	F	T
F	T	T
T	F	F
T	T	T

978-1-5090-3713-1/16 $31.00 © 2016 IEEE
DOI 10.1109/QRS-C.2016.40

3. Boolean Difference Calculus

Boolean difference calculus was introduced almost five decades ago to formally derive test patterns for logic circuits or Boolean expressions [4] [5] [6]. There exist various definitions for Boolean difference in the literature. The following one is the most common:

$$\frac{dg}{dx_i} = g(x_i \leftarrow \mathtt{F}) \oplus g(x_i \leftarrow \mathtt{T}) \tag{1}$$

The equation above defines the derivative of function g with respect to input x_i. That is the response of g to the switching of x_i from F to T. This is quite similar to the definition of derivative in calculus. Boolean derivative can take the value of either F or T. For example, the derivative of $\neg a \vee b$ with respect to b can be computed as:

$$\frac{dg}{db} = (\neg a \vee \mathtt{F}) \oplus (\neg a \vee \mathtt{T}) = \neg a \oplus \mathtt{T} = a$$

This means that the output is responsive to the change in b only in case that $a = \mathtt{T}$. This calculation is particularly useful for generating and evaluating test cases. For instance, we saw from the analysis of the example function g that test cases with different values assigned to b are relevant only when $a = \mathtt{T}$. In general, detection of a fault related to the value of a Boolean variable requires that the derivative of the switching function with respect to that variable is true. Therefore, it is of interest to find the proper values for the test inputs that make the derivative true. Boolean difference calculus provides a formal method for this. Nevertheless, the analytical computation of the difference for large expressions may be cumbersome. The transformation we introduce in the following section enables the use of elementary calculus for the same purpose.

4. Transforming Switching Functions

This section defines the rules that transform any switching or Boolean function into its continuous counterpart. For this purpose, we define a transformation $\mathcal{T} : g \mapsto G$ having the following properties, where $x \in \mathbb{B}$, $\mathbf{x} \in \mathbb{B}^n$, $X \in \mathbb{R}$, $\mathbf{X} \in \mathbb{R}^n$, and $G : \mathbb{R}^n \to \mathbb{R}$:

$$(Property\ 1) \qquad x = \mathtt{T} \iff X = 1 \tag{2}$$

$$(Property\ 2) \qquad x = \mathtt{F} \iff X = 0 \tag{3}$$

$$(Property\ 3) \qquad g(\mathbf{x}) = \mathtt{T} \iff G(\mathbf{X}) = 1 \tag{4}$$

$$(Property\ 4) \qquad g(\mathbf{x}) = \mathtt{F} \iff G(\mathbf{X}) = 0 \tag{5}$$

$$(Property\ 5) \qquad \frac{dg}{dx_i}(\mathbf{x}) = \mathtt{T} \iff \frac{\partial G}{\partial X_i}(\mathbf{X}) = \pm 1 \tag{6}$$

$$(Property\ 6) \qquad \frac{dg}{dx_i}(\mathbf{x}) = \mathtt{F} \iff \frac{\partial G}{\partial X_i}(\mathbf{X}) = 0 \tag{7}$$

The linear function seen in Figure 1 satisfies these properties for the identity function.

Similarly, for negation operation $g(x) = \neg x$, $G(X) = 1 - X$ shown in Figure 2 satisfies the properties. For AND operation $g(x_1, x_2) = x_1 \wedge x_2$, a simple multiplication of the variables satisfies all the properties. The plot of

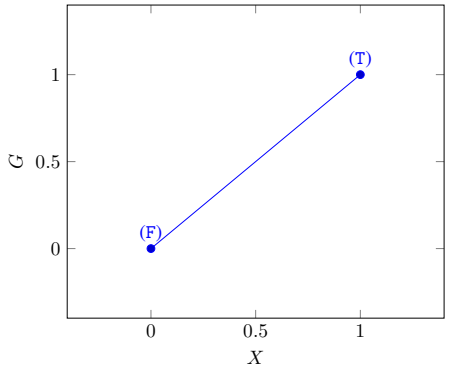

Figure 1. Plot for $G = X$.

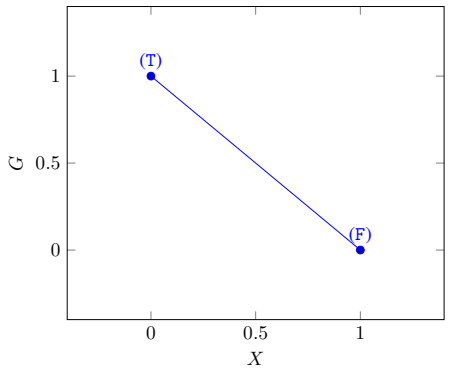

Figure 2. Plot for $G = 1 - X$.

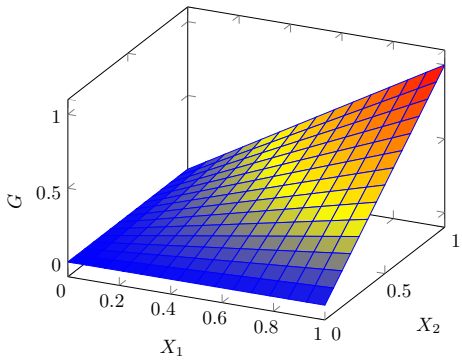

Figure 3. 3D plot for $G = X_1 X_2$.

$G(X_1, X_2) = X_1 X_2$ can be seen in Figure 3. Identity and the two essential operations NOT and AND are sufficient to describe the entire transformation since the other Boolean operations can be derived from these two. For example, $a \vee b$ can also be written as $\neg(\neg a \wedge \neg b)$. Before discussing the rest of the rules, we shall dwell upon some specific cases. In particular, we need to make a simplification regarding 4 exceptional functions: *i)* $g_1(x) = x \wedge x$, *ii)* $g_2(x) = \neg x \wedge x$, *iii)* $g_3(x) = \neg x \vee x$ and *iv)* $g_4(x) = \neg x \wedge \neg x$. The common feature of these functions is that they are degenerate functions, so by the idempotent and complementation

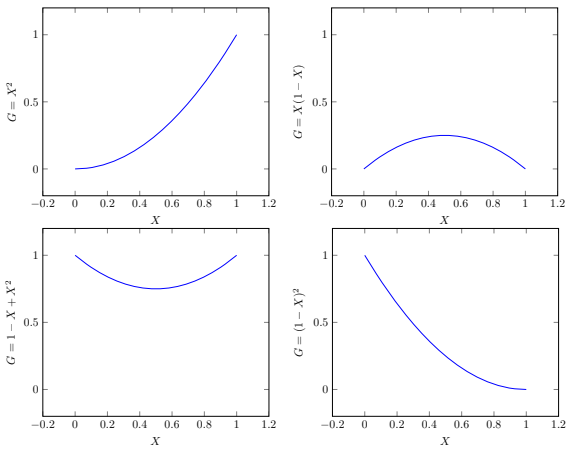

Figure 4. Plots for g_1, g_2, g_3 and g_4.

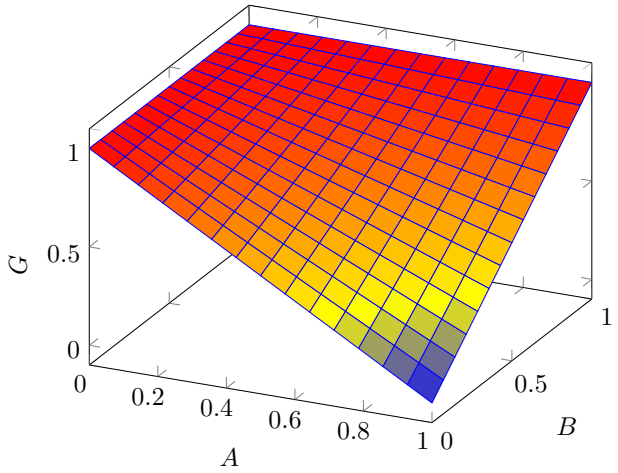

Figure 5. 3D plot for $G = 1 - A + AB$.

rules of Boolean algebra, we can simplify these functions to $g_1(x) = x$, $g_2(x) = 0$, $g_3(x) = 1$ and $g_4(x) = \neg x$. However, a direct transformation would yield

$$\begin{aligned} G_1(X) &= XX = X^2 && (8)\\ G_2(X) &= (1-X)X = X - X^2 && (9)\\ G_3(X) &= 1 - X(1-X) = 1 - X + X^2 && (10)\\ G_4(X) &= (1-X)(1-X) = 1 - 2X + X^2 && (11) \end{aligned}$$

The plots are seen in Figure 4. The problem with these degenerate functions is that Property 5 and 6 are violated. For g_1, we can find that $\frac{dg_1}{dx} = \mathtt{F} \oplus \mathtt{T} = \mathtt{T}$ whereas $\frac{dG_1}{dX} = 2X$ and for $X = 0$ and $X = 1$, the derivative will be 0 and 2, respectively. This is because of the nonlinearity of G_1 and can be overcome by simply making it a multilinear polynomial that is linear in each of its variables. This is achieved by replacing X^2 with X. In this case, those functions reduce to:

$$\begin{aligned} G_1(X) &= X^2 \rightsquigarrow X && (12)\\ G_2(X) &= X - X^2 \rightsquigarrow X - X = 0 && (13)\\ G_3(X) &= 1 - X + X^2 \rightsquigarrow 1 - X + X = 1 && (14)\\ G_4(X) &= 1 - 2X + X^2 \rightsquigarrow 1 - 2X + X = 1 - X && (15) \end{aligned}$$

The derivative of g with respect to a is $\mathtt{F}$ whereas derivative of G is $1-2A$ which implies that $G(0) = 1$ and $G(1) = -1$. These transformed functions now satisfy all the properties given in (2-7).

In the continuous form of a Boolean expression, if any variable occurs to a power of 2 or higher, this expression needs to be converted to multilinear form using the simplification rule that replaces X^n with X where $n \geq 2$. Consider the following degenerate function as an example:

$$g = a \vee a \wedge b$$

Using the Boolean algebra identities *Distribution* and *Dominance*, we can simplify it as follows:

$$g = a \wedge (1 \vee b) = a$$

The transformation converts it to the following continuous form:

$$\begin{aligned} G &= 1 - (1-A)(1-AB) && (16)\\ &= 1 - (1 - A - AB + A^2B) && (17) \end{aligned}$$

Simplification in continuous form is quite easy. Since $A^2B = AB$, the function can be rewritten as:

$$G = 1 - (1 - A) = A$$

The entire set of transformation rules are given below:

$$\begin{aligned} &\text{(Identity)} && \mathcal{T}\{x\} = X && (18)\\ &\text{(Not)} && \mathcal{T}\{\neg x\} = (1-X) && (19)\\ &\text{(And)} && \mathcal{T}\{x_1 \wedge x_2\} = X_1X_2 && (20)\\ &\text{(Or)} && \mathcal{T}\{x_1 \vee x_2\} = 1 - (1-X_1)(1-X_2) && (21)\\ &\text{(Xor)} && \mathcal{T}\{x_1 \oplus x_2\} = 1 - X_1X_2(1-X_1)(1-X_2) && (22)\\ &\text{(Nand)} && \mathcal{T}\{x_1 \uparrow x_2\} = 1 - X_1X_2 && (23)\\ &\text{(Nor)} && \mathcal{T}\{x_1 \downarrow x_2\} = (1-X_1)(1-X_2) && (24)\\ &\text{(Simplification)} && X^n \rightsquigarrow X \text{ where } n \geq 2 && (25) \end{aligned}$$

where $x \in \mathbb{B}$ is a Boolean variable and $X \in [0,1]$ is a real number. All the rules are reversible and they work in both directions. For our example function $g(\mathbf{x}) = \neg a \vee b$, the transformation results in

$$\begin{aligned} G &= 1 - (1 - (1-A))(1-B) && (26)\\ &= 1 - A + AB && (27) \end{aligned}$$

Figure 5 illustrates the 3D plot of G. When the variables are restricted in the range of $[0,1]$, G will always be in this range, as remarked in Lemma 1.

Lemma 1. *Let $\mathbf{X_0}$ be a vector of real numbers associated with the Boolean conditions. $\mathcal{T}\{g(\mathbf{x})\}\ |_{\mathbf{x}=\mathbf{x_0}}$ is a real number and $0 \leq G(\mathbf{X_0}) \leq 1$.*

Proof. Rules (18-24) define the entire set of transformation rules and for each individual transformation rule, it can be

shown that $0 \leq \mathcal{T}\{g(x)\} \leq 1$. We know that $0 \leq X \leq 1$. Substituting with -1 we get $-1 \leq X-1 \leq 0$. Multiplying the inequality with -1, we get $0 \leq 1-X \leq 1$. It is easy to see also that $X_1X_2 \in [0,1]$. Since AND and NOT are the essential operations from which the others are derived, the rest of the rules can be derived from Rule (19) and Rule (20), which completes the proof. □

Starting with Lemma 1, it can be shown that the transformations satisfy all the properties. Since the entire proof takes much space, we shall explain how to proceed to complete the proof. For Property 3 and 4, each transformation can be checked using the truth tables for 2-input logic operations. For Property 5 and 6, however, the following should be proved first:

$$\frac{\partial G}{\partial x_i}(\mathbf{X}) = \mathcal{T}\{\frac{dg}{da}(\mathbf{x})\}$$

Hence, by following Properties 3 and 4 again, 5 and 6 can also be proved.

In the rest of the paper, we use TCAS-II Specification 4 [7] to demonstrate the method and the example application:

$$g = a \wedge (\neg b \vee \neg c) \wedge d \vee e$$

By transformations (19-21), we get

$$\mathcal{T}\{g\} = G = E + AD - ADE - ABCD + ABCDE \quad (28)$$

G is a continuous function and it is differentiable. Below, we find the derivative of G with respect to A and simplify it in two steps:

$$\begin{aligned} \frac{\partial G}{\partial A} &= D - DE - BCD + BCDE && (29)\\ &= D(1-E) - BCD(1-E) && (30)\\ &= D(1-E)(1-BC) && (31) \end{aligned}$$

Partial derivatives with respect to each variable constitute the gradient vector:

$$\nabla G = \begin{bmatrix} \partial G/\partial A \\ \partial G/\partial B \\ \partial G/\partial C \\ \partial G/\partial D \\ \partial G/\partial E \end{bmatrix} = \begin{bmatrix} D(1-E)(1-BC) \\ -ACD(1-E) \\ -ABD(1-E) \\ A(1-E)(1-BC) \\ 1-AD(1-BC) \end{bmatrix} \quad (32)$$

It is straightforward to return to the switching function by applying the inverse transformation. For instance, the following inverse transformation rules may give some intuition:

$$\begin{aligned} \mathcal{T}^{-1}\{X\} &= x && (33)\\ \mathcal{T}^{-1}\{1-X\} &= \neg x && (34)\\ \mathcal{T}^{-1}\{X_1X_2\} &= x_1 \wedge x_2 && (35)\\ \mathcal{T}^{-1}\{1-X_1X_2\} &= \neg x_1 \vee \neg x_2 && (36) \end{aligned}$$

By applying Rules (33-36) to the derivative given in Equation (31), the function can be transformed to the discrete domain as follows:

$$\begin{aligned} \frac{dg}{da} &= \mathcal{T}^{-1}\{\frac{\partial G}{\partial A}\} && (37)\\ &= d \wedge \neg e \wedge (\neg b \vee \neg c) && (38) \end{aligned}$$

The same result would be achieved using the Boolean difference calculus:

$$\begin{aligned} \frac{dg}{da} &= (\mathtt{F} \wedge (\neg b \vee \neg c) \wedge d \vee e) \oplus (\mathtt{T} \wedge (\neg b \vee \neg c) \wedge d \vee e)\\ &= e \oplus ((\neg b \vee \neg c) \wedge d \vee e)\\ &= d \wedge (\neg b \vee \neg c) \wedge \neg e && (39) \end{aligned}$$

5. Example Application: MCDC Analysis

We demonstrate the proposed method on a potential application, Modified Condition and Decision Coverage (MCDC) testing of software. MCDC is a structural coverage criterion that is required to verify Level A software as per the DO-178C standard [8]. It requires that every point of entry and exit in the program has been invoked at least once, every condition in a decision in the program has taken all possible outcomes at least once, every decision in the program has taken all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome by varying just that condition while holding fixed all other possible conditions [9]. As such, a test suite should contain two test cases for each basic condition, *C* such that only the value of *C* is altered and the evaluation of the corresponding compund condition differs for the two cases. Consider the code snippet shown in Listing 1, which is generated by the Flex tool[1].

Listing 1. A code snippet [9] that takes place in C lexical analyzers generated by Flex.

```
1  for(n = 0;
2      n < max_size
3      && (c = getc( yyin )) != EOF
4      && c != '\n';
5      n++)
6              buf[n] = (char) c;
```

Hereby, there is a loop (compound) condition that contains three basic conditions. Let's refer to the basic conditions at lines 2, 3, and 4 as *a*, *b*, and *c*, respectively. For instance, *b* refers to the basic condition

$$(c = getc(yyin))! = EOF$$

In this case, the compound condition can be represented as

$$g = a \wedge b \wedge c$$

Table 1 lists a set of test cases that satisfies MCDC for this condition. For instance, test cases 2 and 4 are the same except the value of b. The overall evaluation of the compound condition is True when b is True and vice versa. Similarly, test cases 1 and 4 satisfy the coverage criteria for a, whereas test cases 3 and 4 cover the two cases for c.

There are several forms of MCDC discussed in the literature [9], [10], [11], [12]. In this work, we employ

1. Flex is a tool that is distributed with the Linux operating system. It is used for automated generation of lexical analyzers.

TABLE 1. A TEST SUITE THAT SATISFIES MCDC CRITERION FOR THE COMPOUND CONDITION SHOWN IN LISTING 1.

Test Case	a	b	c	g
1	**F**	T	T	F
2	T	**F**	T	F
3	T	T	**F**	F
4	**T**	**T**	**T**	T

the so-called *unique-cause MCDC* [13], which is defined as follows:

$$T = \{(m,n) : \forall i,j \in \{1,\ldots,n\} : i \neq j, \\ m_j = n_j, m_i = \neg n_i, g(m) = \neg g(n), \\ \frac{\partial g}{\partial_{x_i}}(m) = \mathtt{T}, \frac{\partial g}{\partial_{x_i}}(n) = \mathtt{T}\} \quad (40)$$

The above formal definition should be read as:

- m and n are two test vectors that form Unique-cause MCDC independence pair for the i$^{\text{th}}$ condition.
- Condition i must toggle between two tests ($a_i = b'_i$)
- Expression must return different results for the two tests ($g(m) = g'(n)$)
- Condition i must have an influence on the outcome of the expression when the first test is applied ($\frac{\partial g}{\partial_{x_i}}(m) = \mathtt{T}$)
- Condition i must have an influence on the outcome of the expression when the second test is applied ($\frac{\partial g}{\partial_{x_i}}(n) = \mathtt{T}$)

We are interested in the values satisfying condition $\frac{\partial G}{\partial A} = 1$. G is a nonlinear function. Hence, gradient-based techniques can be used to find a solution. A solution can be found by applying the following formula iteratively:

$$\mathbf{X}^+ = \mathbf{X} + \eta \nabla(\frac{\partial G}{\partial A}) \quad (41)$$

where η is known as the learning rate that must be tuned properly to achieve a satisfactory result in the gradient based optimization.

Lemma 2. *Let $G : \mathbb{R}^n \rightarrow \mathbb{R}$ be a transformed switching function and $\mathbf{X_0} = [\overbrace{0.5, 0.5, \cdots, 0.5}^{n}]^T$. There exist $2^n G(\mathbf{X_0})$ combinations that make function g True.*

We shall not give the proof, yet we will show it on our example. For $\frac{\partial G}{\partial A}$, taking that $\mathbf{X_0} = [0.5, 0.5, 0.5, 0.5, 0.5]^T$, we can find that $2^5 \times \frac{\partial G}{\partial A}(\mathbf{X_0}) = 32 \times 0.1875 = 6$ combinations exist that satisfy $\frac{dg}{da}(x) = \mathtt{T}$. This can be verified from Equation (31) easily. The condition to be satisfied can be explicitly specified as follows:

$$\frac{dg}{da} = d \wedge e' \wedge (b' \vee c') = \mathtt{T}$$

We can intuitively find the following solution set:

$$(a,b,c,d,e) = \{(\mathtt{X},\mathtt{F},\mathtt{F},\mathtt{T},\mathtt{F}), (\mathtt{X},\mathtt{F},\mathtt{T},\mathtt{T},\mathtt{F}), (\mathtt{X},\mathtt{T},\mathtt{F},\mathtt{T},\mathtt{F})\}$$

TABLE 2. VALUES COMPUTED IN 10 ITERATIONS OF $\mathbf{x_0^+} = \mathbf{x_0} + \eta\nabla(\frac{\partial G}{\partial A})$ ($\eta = 0.1$)

iteration	B	C	D	E	$F(A,B,C,D,E)$
1	0.49	0.49	0.54	0.46	0.22
2	0.47	0.47	0.58	0.41	0.26
3	0.46	0.46	0.63	0.36	0.31
4	0.44	0.44	0.68	0.31	0.38
5	0.42	0.42	0.73	0.25	0.45
6	0.40	0.40	0.80	0.18	0.55
7	0.37	0.38	0.87	0.11	0.67
8	0.34	0.35	0.95	0.02	0.81
9	0.31	0.32	1.00	0.00	0.90
10	0.28	0.29	1.00	0.00	0.92
23	0.01	0.13	1.00	0.00	1.00

Note that the condition is independent of A, so there are 6 combinations for A's both F and T values. Using Formula (41), it is possible to walk on the curve towards the gradient direction and to achieve one of these six maxima. The gradient vector of $\frac{\partial G}{\partial A}$ can be computed as follows:

$$\nabla(\frac{\partial G}{\partial A}) = \begin{bmatrix} \partial^2 G/\partial A^2 \\ \partial^2 G/\partial B \partial A \\ \partial^2 G/\partial C \partial A \\ \partial^2 G/\partial D \partial A \\ \partial^2 G/\partial E \partial A \end{bmatrix} = \begin{bmatrix} 0 \\ -CD(1-E) \\ -BD(1-E) \\ (1-E)(1-BC) \\ -D(1-BC) \end{bmatrix} \quad (42)$$

To find a maximum relying on the gradient vector with Formula (41), we do the following explicit calculations iteratively:

$$B^+ = B - \eta\, CD(1-E) \quad (43)$$
$$C^+ = C - \eta\, BD(1-E) \quad (44)$$
$$D^+ = D + \eta\,(1-E)(1-BC) \quad (45)$$
$$E^+ = E - \eta\, D(1-BC) \quad (46)$$

Table 2 shows the results of the first 10 iterations and 23^{rd} iteration. As can be seen from the table, variables B, C, D and E converge to $0, 0, 1$ and 0 respectively. In conclusion, the MCDC test pair for condition A can be constructed as

$$(\mathtt{F},\mathtt{F},\mathtt{F},\mathtt{T},\mathtt{F}), (\mathtt{T},\mathtt{F},\mathtt{F},\mathtt{T},\mathtt{F})$$

We can present one gradient ascent formula to obtain the whole test suite at once:

$$\mathbf{M}^+ = \mathbf{M} + \eta\, \mathbf{H}(G) \quad (47)$$

Matrix $\mathbf{M}$ is $n \times n$ matrix. The columns are associated to MCDC test pairs, as one test pair for each condition. Each column contains the condition values assigned to the associating test pair. $\mathbf{M}$ is initially as follows:

$$\mathbf{M} = \begin{bmatrix} 0.5 & \ldots & 0.5 \\ \vdots & \ddots & \vdots \\ 0.5 & \ldots & 0.5 \end{bmatrix}$$

$\mathbf{H}(G)$ is Hessian matrix of second order partial derivatives of G, which has the following component-wise definition:

$$\mathbf{H}_{i,j} = \frac{\partial^2 G}{\partial x_i \partial x_j} \tag{48}$$

For our example, the Hessian matrix can be found as follow:

$$\begin{bmatrix} -CD(1-E) & -BD(1-E) & (1-E)(1-BC) & -D(1-BC) \\ 0 & -AD(1-E) & -AC(1-E) & ACD \\ -AD(1-E) & 0 & -AB(1-E) & ABD \\ -AC(1-E) & -AB(1-E) & 0 & -A(1-BC) \\ ACD & ABD & -A(1-BC) & 0 \end{bmatrix}$$

Note that the first column of the matrix, which has been given in Equation (42), is omitted due to the space limitations. After 20 iterations, $\mathbf{M}$ becomes:

$$\mathbf{M} = \begin{bmatrix} 0.50 & 1.00 & 1.00 & 1.00 & 0.09 \\ 0.06 & 0.50 & 1.00 & 0.06 & 0.59 \\ 0.15 & 1.00 & 0.50 & 0.15 & 0.59 \\ 1.00 & 1.00 & 1.00 & 0.50 & 0.17 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.50 \end{bmatrix}$$

At this point, it is straightforward to obtain the unique-cause MCDC test pairs, using the rule given below:

$$x_{i,j} = \begin{cases} \mathtt{F} & M_{i,j} < 0.5 \\ \mathtt{X} & M_{i,j} = 0.5 \\ \mathtt{T} & M_{i,j} > 0.5 \end{cases} \tag{49}$$

$$\mathbf{X} = \begin{bmatrix} \mathtt{X} & \mathtt{T} & \mathtt{T} & \mathtt{T} & \mathtt{F} \\ \mathtt{F} & \mathtt{X} & \mathtt{T} & \mathtt{F} & \mathtt{T} \\ \mathtt{F} & \mathtt{T} & \mathtt{X} & \mathtt{F} & \mathtt{T} \\ \mathtt{T} & \mathtt{T} & \mathtt{T} & \mathtt{X} & \mathtt{F} \\ \mathtt{F} & \mathtt{F} & \mathtt{F} & \mathtt{F} & \mathtt{X} \end{bmatrix}$$

The 5 test pairs can be found as follows:

$$a : (\mathtt{F},\mathtt{F},\mathtt{F},\mathtt{T},\mathtt{F}), (\mathtt{T},\mathtt{F},\mathtt{F},\mathtt{T},\mathtt{F})$$

$$b : (\mathtt{T},\mathtt{F},\mathtt{T},\mathtt{T},\mathtt{F}), (\mathtt{T},\mathtt{T},\mathtt{T},\mathtt{T},\mathtt{F})$$

$$c : (\mathtt{T},\mathtt{T},\mathtt{F},\mathtt{T},\mathtt{F}), (\mathtt{T},\mathtt{T},\mathtt{T},\mathtt{T},\mathtt{F})$$

$$d : (\mathtt{T},\mathtt{F},\mathtt{F},\mathtt{F},\mathtt{F}), (\mathtt{T},\mathtt{F},\mathtt{F},\mathtt{T},\mathtt{F})$$

$$e : (\mathtt{F},\mathtt{T},\mathtt{T},\mathtt{F},\mathtt{F}), (\mathtt{F},\mathtt{T},\mathtt{T},\mathtt{F},\mathtt{T})$$

Remark 1. *In Formula (47), only one iteration is enough to obtain the MCDC test cases.*

This can be seen from Table 2. When we apply the rule given in (49) to the first and last rows of the table, both yield to the same result. This is, in fact, due to the multilinear nature of the transformed functions. Since the function is linear in each of its variables, the gradient direction does not change along the way. Therefore, the resulting matrix $\mathbf{M}$ can be computed directly from the Hessian matrix by assigning 0.5 to each variable. Hereby, a negative term in Hessian matrix yields to F whereas a positive one yields to T.

6. Conclusion

We proposed a transformation method for converting a Boolean expression into continuous form. This transformation allows to use of elementary calculus for analyzing Boolean expressions. As such, our method makes it possible to exploit existing methods and tools of calculus for the analysis of switching functions. We demonstrated one of the applications of the method for finding test pairs that satisfy unique-cause MCDC criteria. We showed that our method made it possible to solve this particular problem by only using elementary calculus. Future work will include the investigation of other potential applications and also distinguish between this method and Fourier transformation of switching functions.

References

[1] G. Langholz, A. Kandel, and J. Mott, *Foundations of Digital Logic Design*. London, UK: World Scientific, 2000.

[2] J. Jones and M. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," *IEEE Transactions on Software Engineering*, vol. 29, no. 3, pp. 195–209, 2003.

[3] R. O'Donnell, "Some topics in analysis of boolean functions," *Proceedings of the fourtieth annual ACM symposium on Theory of computing - STOC 08*, p. 569, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1374376.1374458

[4] I. S. Reed, "Boolean Difference Calculus and Fault Finding," pp. 134–143, 1973.

[5] A. Thayse and M. Davio, "Boolean Differential Calculus and Its Application to Switching Theory," *IEEE Trans. Comput.*, vol. 22, no. 4, pp. 409–420, 1973. [Online]. Available: http://dx.doi.org/10.1109/T-C.1973.223729

[6] J. George W., Smith, "Fault Detection Test Derivation Using Boolean Difference Techniques," in *ACM '72 Proceedings of the ACM annual conference*, 1972, pp. 369–378.

[7] E. Weyuker, T. Goradia, and A. Singh, "Automatically Generating Test Data from a Boolean Specification," pp. 353—-363, 1994.

[8] RTCA, "DO-178C: Software considerations in airborne systems and equipment certification," RTCA Inc., Tech. Rep., December 2011.

[9] M. Pezze and M. Young, *Software Testing and Analysis: Process, Principles and Techniques*. Hoboken, NJ, USA: John Wiley and Sons, 2008.

[10] P. Amman and J. Offut, *Introduction to Software Testing*. New York, NY, USA: Cambridge University Press, 2008.

[11] C. A. S. T. (Cast), "Rationale for Accepting Masking MC/DC in Certification Projects," 2001.

[12] T. K. Paul and M. F. Lau, "A systematic literature review on modified condition and decision coverage," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, 2014, pp. 1301–1308. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2554850.2555004

[13] J. J. Chilenski and S. P. Miller, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, vol. 9, no. 5, p. 193, 1994.