Fault Propagation in Tabular Expression-Based Specifications

Xin Feng and David Lorge Parnas
Software Quality Research Laboratory
Faculty of Informatics and Electronics
Limerick University, Limerick, Ireland
{xin.feng, david.parnas}@ul.ie

T. H. Tse
Department of Computer Science
The University of Hong Kong
Pokfulam, Hong Kong
thtse@cs.hku.hk

Abstract

Tabular expressions have been used in industry for many years to precisely document software in a readable notation. In this paper, we propose a fault-based testing technique that traces the propagation of faults from the expression in each cell of a tabular expression to the output of the program under test. The technique has been formalized in the form of abstract test case constraints also represented by tabular expressions, so that it can be easily applied and automated.

1. Introduction

In the past decades, many formal specification methods have been developed in an effort to document software specifications precisely and unambiguously, mostly by means of mathematical expressions. However, conventional mathematical expressions are too complex and difficult to read. To alleviate the problem, functional documentation using tabular expressions was proposed [6, 11]. They were proven to be useful and acceptable in industrial experiences with the US Navy's A-7 aircraft [3], a Bell Laboratories Service Evaluation system [4], the Darlington Nuclear Power Station [10], a Dell keyboard test program [13], and so on.

Two testing strategies were developed for tabular expressions: the partition strategy [9] and the interesting point selection strategy [1]. Acting as equivalence class testing and boundary value testing, respectively, they fulfill basic testing requirements. Test oracles in [12] can help verify testing results. To ensure a high quality of the software, however, we need more advanced strategies such as fault-based testing.

For the ease of discussions, we shall write tabular expressions interchangeably with their equivalent conventional mathematical expressions in this paper. Consider the following conventional definition of a simple function. We shall refer to $x > 10$ and $y > 5$ by the Boolean variables A and B, respectively. They are listed in bold and in parentheses.

$$f(x,y) = \begin{cases} x+y & \underline{x>10} \wedge \underline{y>5} \vee \underline{x \leqslant 10} \wedge \underline{y \leqslant 5} \\ & (\ \boldsymbol{A}\ \wedge\ \boldsymbol{B}\ \vee\ \boldsymbol{\neg A}\ \wedge\ \boldsymbol{\neg B}\ ) \\ x-y & \underline{x \leqslant 10} \wedge \underline{y>5} \vee \underline{x>10} \wedge \underline{y \leqslant 5} \\ & (\ \boldsymbol{\neg A}\ \wedge\ \boldsymbol{B}\ \vee\ \boldsymbol{A}\ \wedge\ \boldsymbol{\neg B}\ ) \end{cases}$$

Many fault-based testing strategies (such as [7, 8, 16]) have been proposed to generate test cases for Boolean expressions. Although these strategies can propagate faults (such as missing A or the negation of A) to affect the result of the expression $A \wedge B \vee \neg A \wedge \neg B$, more questions regarding fault propagation need to be addressed: (i) Can a change of the operator ">" in $x > 10$ affect A? (ii) Can the result of $A \wedge B \vee \neg A \wedge \neg B$ affect $f(x, y)$? (iii) Can a fault in $x + y$ be propagated to $f(x, y)$? While there are studies (such as [5, 14]) addressing the first question, the other two have almost been ignored.

Fault-based testing in tabular expressions takes into account all the above-illustrated reflections as well as other features related to tabular expressions.

2. Tabular Expressions

A *tabular expression* is an indexed set of grids and each grid is an indexed set of expressions [6]. Fig. 1 is an example of a two-dimensional inverted table T consisting of three grids, such that the cardinality $Card(T) = 3$ and the index sets $IndexSet(T) = \{0, 1, 2\}$, $IndexSet(T[1]) = IndexSet(T[2]) = \{0, 1, 2\}$, and $IndexSet(T[0]) = IndexSet(T[1]) \times IndexSet(T[2])$. $T[1]$ and $T[0]$ are predicate grids while $T[2]$ is an evaluation grid. $T[1]$ and each row of $T[0]$ are *proper*, that is, one and only one predicate expression can be *true* with respect to an assignment. If $T[i][idx]$ ($i \in IndexSet(T)$ and $idx \in IndexSet(T[i])$) is taken as a Boolean variable, the expressions in bold are Boolean expressions.

A *specification* is a statement of all the properties required of a program; an *actual description* is a statement about actual attributes of a program. Accordingly, there are two kinds of table in this paper: a specification table

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.115

$Q(intx, inty) =$

$T[2]$

$x+y$	$x-y$	$x \times y$
$y<6$	$y=6$	$y>6$
$y<x$	$y>x$	$y=x$
$y<-x$	$y>-x$	$y=-x$

$T[0]$

$x>3$ $(x<3)$
$x=3$
$x<3$ $(x>3)$

$T[1]$

$$Q(x,y) = \begin{cases} x+y & (x>3 \wedge y<6) \vee (x=3 \wedge y<x) \vee (x<3 \wedge y<-x) \\ & (T[1][0] \wedge T[0][0,0] \vee T[1][1] \wedge T[0][1,0] \vee T[1][2] \wedge T[0][2,0]) \\ x-y & (x>3 \wedge y=6) \vee (x=3 \wedge y>x) \vee (x<3 \wedge y>-x) \\ & (T[1][0] \wedge T[0][0,1] \vee T[1][1] \wedge T[0][1,1] \vee T[1][2] \wedge T[0][2,1]) \\ x \times y & (x>3 \wedge y>6) \vee (x=3 \wedge y=x) \vee (x<3 \wedge y>-x) \\ & (T[1][0] \wedge T[0][0,2] \vee T[1][1] \wedge T[0][1,2] \vee T[1][2] \wedge T[0][2,2]) \end{cases}$$

Figure 1. A specification (description) table.

(denoted by T_S) and a description table (denoted by T_D). For the readers' convenience, in Fig. 1, the specification and a description share the same table with different expressions in $T[1][0]$ and $T[1][2]$. The expressions in parentheses are for the description table.

3. Fault Propagation

Fault propagation spreads the faulty result in a problem to the output and causes a failure of the program. It can be revealed by an execution of the program [15] or an analysis of the source code [2].

Boolean expression-based strategies are successful in detecting faults in single Boolean expressions. For tabular expressions, however, additional issues need to be addressed. We discuss these issues based on the example in Fig. 1.

(1) $T[i][idx]$ represents an expression. Information is needed about the faults that may affect the expression, such as a change of "$<$" in $x<3$ of $T[1][2]$, and a change of "x" in $x+y$ of $T[2][0]$.

(2) The change in $T[1][2]$ invokes a change in $T[1][0]$ because $T[1]$ is proper.

(3) A change of result of a Boolean expression does not always entail a change of the output. Consider the second Boolean expression. Suppose $x=1$ and $y=0$. Then, the expression is evaluated to *true* in T_S but *false* in T_D. Since the first Boolean expression is evaluated to *true* in T_D, the expressions in various evaluation grids are computed (namely, $x-y$ and $x+y$). However, since y is 0, $x+y=x-y$. Thus, $x=1, y=0$ fails to propagate the fault.

3.1. Notation and Assumptions

Various programs may be written to implement a specification, with different kinds of fault. Hence, this paper considers only a subset of T_D. Further, we have the following assumptions: (i) All the programs are deterministic. (ii) Faults can either be in a predicate grid or in a cell of an evaluation grid. (iii) If an input is undefined in T_D, the output is out of the range of the programs. (iv) Since many techniques (such as fault-based testing [14] and mutation testing [5]) can be used for testing fault propagation from an expression in a cell to the result of the expression, test cases for any propagation exist.

Test cases are expressed in terms of test case constraints. In the sequel, "test case constraint" will be abbreviated to "constraint". If no test data can be found, the constraint for the expression is *false*. Suppose *idx* is the index of an element in grid $T[i]$. Let $C[i][idx]$ indicate the set of the constraints for the test data of expression $T[i][idx]$ in assumption (iv). The following is the notation:

(a) $Card(C[i][idx])$ denotes the number of constraints in $C[i][idx]$.

(b) $C[i][idx][k]$ denotes the kth constraint in $C[i][idx]$.

(c) $T_S[i][idx][k]$ and $T_D[i][idx][k]$, respectively, denotes the expected result and actual result of the expression $T_S[i][idx]$ with respect to a test case that satisfies $C[i][idx][k]$.

(d) $P[i][idx]$ denotes the set of constraints that propagate the faults in expression $T_D[i][idx]$ to the output.

(e) For any predicate grid $T[i]$, $C_T[i][idx][k]$ and $C_F[i][idx][k]$ denote $C[i][idx][k] \wedge T_S[i][idx]$ and $C[i][idx][k] \wedge \neg T_S[i][idx]$, respectively.

(f) $P_T[i][idx][k]$ and $P_F[i][idx][k]$ denote the set of constraints to propagate faults in expression $T_D[i][idx]$ when the test cases satisfy $C_T[i][idx][k]$ or $C_F[i][idx][k]$, respectively. They denote $\emptyset$ when $C_T[i][idx][k]$ or $C_F[i][idx][k]$ is *false*.

(g) $P[i][idx][k]$ denotes the set of constraints to propagate faults in expression $T_D[i][idx]$ when the test cases satisfy $C[i][idx][k]$.

(h) ∇ is defined by

$$S_1 \nabla S_2 = \begin{cases} S_1 \cup S_2 & S_1 = \emptyset \vee S_2 = \emptyset \\ S_1 \ or \ S_2 & S_1 \neq \emptyset \wedge S_2 \neq \emptyset \end{cases}$$

3.2. Test Case Generation

To make the expressions more compact, we use $\bigvee_{j=1}^{h} v[j]$ to denote $v[1] \vee v[2] \vee \cdots \vee v[h]$, $\bigwedge_{j=1}^{h} v[j]$ to denote $v[1] \wedge v[2] \wedge \cdots \wedge v[h]$, and d_i to denote $Card(T[i])$ $(1 \leqslant i \leqslant Card(T)-1)$. The output of an inverted table can be written in the following form:

$$O = T[n][j_n] \qquad \bigvee_{j_1=0}^{d_1-1} \cdots \bigvee_{j_{n-1}=0}^{d_{n-1}-1} \Big(T[0][j_1, \ldots, j_n] \wedge \big(\bigwedge_{i=1}^{n-1} T[i][j_i] \big) \Big) \quad (1)$$

for $0 \leqslant j_n \leqslant d_n - 1$, where $n = Card(T) - 1$. It means that the output is $T[n][j_n]$ if the conditional expression subsequent to this output is satisfied. Moreover, an inverted table has the following characteristics:

- For $1 \leqslant i \leqslant n-1$, $\bigvee_{j=0}^{d_i-1} T[i][j] = true$.
- For $1 \leqslant i \leqslant n-1$ and $0 \leqslant j, j' \leqslant d_i - 1$ such that $j \neq j'$, $T[i][j] \wedge T[i][j'] = false$.
- Given any input, if $\bigwedge_{i=1}^{n-1} T[i][j_i] = true$, $\bigvee_{j_n=0}^{d_n-1} T[0][j_1, \ldots, j_{n-1}, j_n] = true$ and, for $0 \leqslant j_n, j'_n \leqslant d_n - 1$ such that $j_n \neq j'_n$, $T[0][j_1, \ldots, j_{n-1}, j_n] \wedge T[0][j_1, \ldots, j_{n-1}, j'_n] = false$.

Faults may be in $T[i]$ $(1 \leqslant i \leqslant n-1)$, $T[n]$, or $T[0]$. Thus, we have three cases:

(1) The faults are in evaluation expressions in $T[n]$.

Suppose that there are faults in $T[n][j]$ $(0 \leqslant j \leqslant d_n - 1)$. To propagate the faults, the evaluation expression must be evaluated with respect to an input, that is, $\bigvee_{j_1=0}^{d_1-1} \cdots \bigvee_{j_{n-1}=0}^{d_{n-1}-1} \Big(T[0][j_1, \ldots, j_{n-1}, j] \wedge (\bigwedge_{i=1}^{n-1} T[i][j_i])\Big)$ must be *true*. Hence, for $0 \leqslant j \leqslant d_n - 1$, $P[n][j] = \Big\{C[n][j][k] \wedge \Big(\bigvee_{j_1=0}^{d_1-1} \cdots \bigvee_{j_{n-1}=0}^{d_{n-1}-1} \Big(T[0][j_1, \ldots, j_{n-1}, j] \wedge (\bigwedge_{i=1}^{n-1} T[i][j_i])\Big)\Big) \;\Big|\; 1 \leqslant k \leqslant Card(C[n][j])\Big\}$.

(2) The faults are in the predicate expressions in $T[i]$ $(1 \leqslant i \leqslant n-1)$.

The output function in (1) can be rewritten as

$$O = T[n][j_n] \qquad \bigvee_{j_i=0}^{d_i-1} \Big(T[i][j_i] \wedge Q(i, j_i, j_n)\Big) \tag{2}$$

where $Q(i, j_i, j_n) = \bigvee_{j_1=0}^{d_1-1} \cdots \bigvee_{j_{i-1}=0}^{d_{i-1}-1} \bigvee_{j_{i+1}=0}^{d_{i+1}-1} \cdots \bigvee_{j_{n-1}=0}^{d_{n-1}-1} \Big(T[0][j_1, \ldots, j_i, \ldots, j_{n-1}, j_n] \wedge (\bigwedge_{l=1}^{i-1} T[l][j_l]) \wedge (\bigwedge_{l=i+1}^{n-1} T[l][j_l])\Big)$. If $n = 2$, we have $i = 1$ (since $1 \leqslant i \leqslant n-1$) and $Q(i, j_i, j_n) = T[0][j_1, j_2]$. If $n > 2$, i can be any value that satisfies $1 \leqslant i \leqslant n-1$.

Suppose that a test case satisfies $C[i][j][k]$. There are two possibilities:

(*a*) *The test case satisfies* $C_T[i][j][k]$.
Obviously, it evaluates $T[i][j]$ to *true*, that is, $T_S[i][j][k] = true$ and $T_D[i][j][k] = false$. When $T_S[i][j][k]$ is *true*, $T_S[i][j'][k] = false$ for $0 \leqslant j' \leqslant d_i - 1$ such that $j' \neq j$. The output in (2) can be simplified to

$$O = T[n][j_n] \qquad Q(i, j, j_n)$$

There are two sub-cases:

- *For all* $j' \neq j$ *such that* $0 \leqslant j' \leqslant d_i - 1$, $T_D[i][j'][k] = false$.
In this case, for any j_i such that $0 \leqslant j_i \leqslant d_i - 1$, we have $T_D[i][j_i] = false$. Thus, no matter what j_n is, $\bigvee_{j_i=0}^{d_i-1} \Big(T_D[i][j_i] \wedge Q(i, j_i, j_n)\Big)$ is always *false*. Such a test case is undefined in the implementation. According to assumption (iii) in Section 3.1, the actual output is out of range and, therefore, different from any expected output under the predicate $T_S[i][j]$. Hence, for $1 \leqslant i \leqslant n-1$ and $0 \leqslant j \leqslant d_i - 1$, $P_T^1[i][j][k] = \Big\{C_T[i][j][k] \wedge (\bigvee_{j_n=0}^{d_n-1} Q(i, j, j_n))\Big\}$. Since $\bigvee_{j_n=0}^{d_n-1} Q(i, j, j_n) = true$, we have $P_T^1[i][j][k] = \{C_T[i][j][k]\}$.
- $T_D[i][j'][k] = true$ *for some* $j' \neq j$ *such that* $0 \leqslant j' \leqslant d_i - 1$.
Since the test case evaluates $T_D[i][j']$ to *true*, the output function for T_D is written as

$$O = T[n][j_n] \qquad Q(i, j', j_n)$$

Therefore, $P_T^2[i][j][k] = \Big\{C_T[i][j][k] \wedge \Big(\bigvee_{j_n=0}^{d_n-1} \Big(Q(i, j, j_n) \wedge \neg Q(i, j', j_n) \wedge \Big(\bigvee_{j'_n=0, j'_n \neq j_n}^{d_n-1} (Q(i, j', j'_n) \rightarrow T[n][j'_n] \neq T[n][j_n])\Big)\Big)\Big) \;\Big|\; 0 \leqslant j' \leqslant d_i - 1 \wedge j \neq j'\Big\}$.

Thus, we can determine $P_T[i][j][k] = P_T^1[i][j][k] \cup P_T^2[i][j][k]$.

(*b*) *The test case satisfies* $C_F[i][j][k]$.
It evaluates $T_S[i][j]$ to *false* and $T_D[i][j]$ to *true*. $T_S[i][j'][k] = true$ for some $j' \neq j$ such that $0 \leqslant j' \leqslant d_i - 1$. The analysis is similar to the second case in (a). The set of constraints is given by $P_F[i][j][k] = \Big\{C_F[i][j][k] \wedge T_S[i][j'] \wedge \Big(\bigvee_{j_n=0}^{d_n-1} \Big(Q(i, j', j_n) \wedge \neg Q(i, j, j_n) \wedge \Big(\bigvee_{j'_n=0, j'_n \neq j_n}^{d_n-1} (Q(i, j, j'_n) \rightarrow T[n][j'_n] \neq T[n][j_n])\Big)\Big)\Big) \;\Big|\; 0 \leqslant j' \leqslant d_i - 1 \wedge j \neq j'\Big\}$.

Thus, we can determine $P[i][j][k] = P_T[i][j][k] \nabla P_F[i][j][k]$ and hence $P[i][j] = \{p \in P[i][j][k] \mid 0 \leqslant k \leqslant Card(C[i][j]) - 1\}$.

(3) The faults are in the predicate expressions in $T[0]$.

Suppose that a test case satisfies $C[0][j_1, \cdots, j_{n-1}, j][k]$. Similarly to the above, there are two possibilities.

(*a*) *The test case satisfies* $C_T[0][j_1, \cdots, j_{n-1}, j][k]$.
Since $T_S[0][j_1, \ldots, j_{n-1}, j][k] = true$, we have $T_D[0][j_1, \ldots, j_{n-1}, j][k] = false$. There are two sub-cases:

- *For all* $j' \neq j$ *such that* $0 \leqslant j' \leqslant d_n - 1$, $T_D[0][j_1, \ldots, j_{n-1}, j'][k] = false$.
For any j_n such that $0 \leqslant j_n \leqslant d_n - 1$, we have $T_D[0][j_1, \ldots, j_{n-1}, j_n][k] = false$. Such a

test case is undefined in the implementation. To propagate faults, $T_D[0][j_1, \ldots, j_{n-1}, j]$ must be evaluated. Hence, $P_T^1[0][j_1, \ldots, j_{n-1}, j][k] = \left\{C_T[0][j_1, \ldots, j_{n-1}, j][k] \wedge \left(\bigwedge_{i=1}^{n-1} T[i][j_i]\right)\right\}$.

- $T_D[0][j_1, \ldots, j_{n-1}, j'][k]$ = *true for some* $j' \neq j$ *such that* $0 \leqslant j' \leqslant d_n - 1$.
  Since $T_D[0][j_1, \ldots, j_{n-1}, j']$ corresponds to the output $T[n][j']$, we must have $T[n][j'] \neq T[n][j]$ to make the output different. Therefore, $P_T^2[0][j_1, \ldots, j_{n-1}, j][k] = \left\{C_T[0][j_1, \ldots, j_{n-1}, j][k] \wedge \left(\bigwedge_{i=1}^{n-1} T[i][j_i]\right) \wedge T[n][j] \neq T[n][j'] \;\middle|\; 0 \leqslant j' \leqslant d_n - 1 \wedge j' \neq j\right\}$.

Hence, we can determine $P_T[0][j_1, \ldots, j_{n-1}, j][k] = P_T^1[0][j_1, \ldots, j_{n-1}, j][k] \cup P_T^2[0][j_1, \ldots, j_{n-1}, j][k]$.

(*b*) *The test case satisfies* $C_F[0][j_1, \ldots, j_{n-1}, j][k]$.
$T_S[0][j_1, \ldots, j_{n-1}, j'][k]$ is evaluated to *true* for some j' such that $0 \leqslant j' \leqslant d_n - 1$. Since $T_D[0][j_1, \ldots, j_{n-1}, j][k]$ is *true*, they correspond to $T[n][j']$ and $T[n][j]$, respectively. Hence, $P_F[0][j_1, \ldots, j_{n-1}, j][k] = \left\{C_F[0][j_1, \ldots, j_{n-1}, j][k] \wedge T_S[0][j_1, \ldots, j_{n-1}, j'] \wedge \left(\bigwedge_{i=1}^{n-1} T[i][j_i]\right) \wedge T[n][j] \neq T[n][j'] \;\middle|\; 0 \leqslant j' \leqslant d_n - 1 \wedge j' \neq j\right\}$.

Thus, $P[0][j_1, \ldots, j_{n-1}, j] = \left\{p \in P[0][j_1, \ldots, j_{n-1}, j][k] \;\middle|\; 0 \leqslant k \leqslant Card(C[0][j_1, \ldots, j_{n-1}, j])\right\}$, where $P[0][j_1, \ldots, j_{n-1}, j][k] = P_T[0][j_1, \ldots, j_{n-1}, j][k] \nabla P_F[0][j_1, \ldots, j_{n-1}, j][k]$.

Application of the method is simple because all the formulas have been given. Testers need only copy the actual expressions from the tables to the formulas. The constraints for the first step of the propagation can be obtained by applying the *MIST* technique in [2]. The subsequent process can be easily automated.

4. Conclusion and Future Work

We have shown that fault propagation can be taken into account in tabular expression-based specifications. The testing method proposed in this paper propagates not only faults in predicate expressions but also faults in evaluation expressions. The formulae in our method make test case generation easier in two ways: (a) they facilitate the implementation of the tool that automates the method; and (b) if testers generate test data manually, they only need to replace the notation in the formulae with actual expressions in the specification table.

Although we have illustrated our method through inverted tables, we can also use it in other types of table. Our SQRL laboratory has been developing the tools to support this method. We have also designed the experiments to further compare this method with other selected testing strategies. On the other hand, we have noted that some generated constraints are equivalent because of the completeness of the grids in the table specification. Hence, we are studying the method to generate test case constraints for only part of the table but covering the whole.

References

[1] M. Clermont and D. L. Parnas. Using information about functions in selecting test cases. *ACM SIGSOFT Software Engineering Notes*, 30 (4): 1–7, 2005.

[2] X. Feng. *MIST: Towards a MInimum Set of Test Cases*. PhD Thesis. The University of Hong Kong, Pokfulam, Hong Kong, 2002.

[3] K. L. Heninger. Specifying software requirements for complex systems: new techniques and their application. *IEEE Transactions on Software Engineering*, SE-6 (1): 2–13, 1980.

[4] S. D. Hester, D. L. Parnas, and D. F. Utter. Using documentation as a software design medium. *The Bell System Technical Journal*, 60 (8): 1941–1977, 1981.

[5] W. E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, SE-8 (4): 371–379, 1982.

[6] R. Janicki, D. L. Parnas, and J. Zucker. Tabular representations in relational documents. In *Software Fundamentals: Collected Papers by David L. Parnas*, D. M. Hoffman and D. M. Weiss, editors, pages 71–87. Addison Wesley, 2001.

[7] D. R. Kuhn. Fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 8 (4): 411–424, 1999.

[8] M. F. Lau and Y. T. Yu. An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 14 (3): 247–276, 2005.

[9] S. Liu. *Generating Test Cases from Software Documentation*. Master's thesis. School of Graduate Studies, McMaster University, Hamilton, Ontario, Canada, 2001.

[10] D. L. Parnas, G. J. K. Asmis, and J. Madey. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32 (2): 189–198, 1991.

[11] D. L. Parnas, J. Madey, and M. Iglewski. Precise documentation of well-structured programs. *IEEE Transactions on Software Engineering*, 20 (12): 948–976, 1994.

[12] D. K. Peters and D. L. Parnas. Using test oracles generated from program documentation. *IEEE Transactions on Software Engineering*, 24 (3): 161–173, 1998.

[13] C. Quinn, S. Vilkomir, D. L. Parnas, and S. Kostic. Specification of software component requirements using the trace function method. In *Proceedings of the International Conference on Software Engineering Advances* (*ICSEA 2006*). IEEE Computer Society Press, Los Alamitos, CA, 2006.

[14] K.-C. Tai. Theory of fault-based predicate testing for computer programs. *IEEE Transactions on Software Engineering*, 22 (8): 552–562, 1996.

[15] J. M. Voas. PIE: a dynamic failure-based technique. *IEEE Transactions on Software Engineering*, 18 (8): 717–727, 1992.

[16] E. J. Weyuker, T. Goradia, and A. Singh. Automatically generating test data from a Boolean specification. *IEEE Transactions on Software Engineering*, 20 (5): 353–363, 1994.