Combinatorial Testing of Software with Binary Inputs: A State-of-the-Art Review

Sergiy Vilkomir
Department of Computer Science
East Carolina University
Greenville, NC 27858 USA
vilkomir@ecu.edu

Abstract—Combinatorial methods are useful and effective approaches for software testing. In this paper, we provide a brief review of the research results in one area of combinatorial methods applications: software with binary or Boolean inputs. This includes the testing of logical expressions that is crucial for many safety-critical applications such as avionics software and computer systems at nuclear power plants. We consider three different aspects of this research direction: (1) effectiveness of combinatorial testing with binary inputs, (2) combinatorial test generation, and (3) comparison combinatorial testing with other approaches for testing logical expressions.

Keywords—combinatorial testing; pairwise; t-way; binary inputs; effectiveness

I. INTRODUCTION

Combinatorial methods are useful and effective approaches for software testing. The most general combinatorial method is t-way testing, which requires every combination of any t input parameter (or configuration option) values to be covered by at least one test [1]. The higher value of t is more effective, but it requires more test cases, so in practice this approach is used for t from 1 to 6. When t = 1 and t = 2, t-way testing is equivalent to each-choice and pairwise [2] testing, respectively. Considerable information about combinatorial testing methods is provided in book [3], which also includes a comprehensive list of references to 232 publications in this area. Additional information can be found in surveys [4] (53 references) and [5] (101 references).

In this paper, we provide a brief review of the research results in one area of the combinatorial methods applications: software with binary or Boolean inputs. This includes the testing of logical expressions that is crucial for many safety-critical applications such as avionics software and computer systems at nuclear power plants. Logical expressions are used in the branch points in software code where two types of faults are possible [3, p. 57]: code block faults when the branching statement is correct, but one of the branches has the faulty code, and condition faults when the branching statement is faulty itself. Detecting faults of the second type can be demanding and therefore applying combinatorial testing to such situations requires special consideration. In this paper, we consider three different aspects of this research direction:

• Effectiveness of combinatorial testing with binary inputs.
• Combinatorial test generation.
• Comparison of combinatorial testing with other approaches for testing logical expressions.

Effective (considered in Section II) is the ability of the test cases to detect faults and can be measured by the expected number (or percentage) of detected faults. Combinatorial test generation for binary inputs (considered in Section III) can use special techniques—for example, binary decision diagrams, topological structure of logical expressions, cause-effect graphs, others—that are not applicable in general cases. To understand the benefits and drawbacks of combinatorial testing in contrast to the testing criteria created specifically for testing logical expressions, such as the Modified Condition/Decision Coverage (MCDC) criterion [6, 7], comparison with other approaches (considered in Section IV) is necessary.

II. EFFECTIVENESS OF TESTING

As far as we know, the first results on the effectiveness of binary combinatorial testing are by Kobayashi et al. [8]. The authors experimentally investigated the combinatorial testing of 20 logical expressions taken from the specifications of the TCAS II avionics system [9, 10]. From this set of expressions, the numerous faulty expressions were generated using five different types of faults suggested in [11, 12]: Variable Negation Fault (VNF), Operator Reference Fault (ORF), Variable Reference Fault (VRF), Expression Negation Fault (ENF), and Associative Shift Fault (ASF). Combinatorial test sets for 2-, 3-, and 4-way testing were applied to the faulty logical expressions and a mutation score (the percentage of the detected faults) was used as a measure of testing effectiveness.

The results reported by Kobayashi et al. are as follows:

- 33.8% effectiveness for 2-way combinatorial testing,
- 70.5% for 3-way, and
- 81.3% for 4-way. The highest level of effectiveness was for the ENF fault type and the lowest was for VRF. Combinatorial testing was also compared with random and antirandom [13] testing, and it was shown that combinatorial testing is superior to both of these approaches.

Reference [8] was a starting point for a further investigation by Vilkomir et al. [14, 15, 16]. The goal of [14] was to repeat experiments by Kobayashi et al. and compare the results. The same set of 20 logical expressions as in [8] and the same fault types were used. Three pairwise test sets...
were generated by using the Allpairs and TConfig tools. Fault Evaluator [17], a software tool developed at East Carolina University for experimental investigation of testing logical expressions in software, was used for generating faulty expressions, testing, and evaluating effectiveness. The effectiveness of pairwise testing was in the range 20% to 50%, depending on the type of fault: 33% for VNF, 46% for ORF, 19% for VRF, 48% for ENF, and 25% for ASF with 23% of the weighted average. The results for average effectiveness were quite close to the results by Kobayashi et al., although there were significant variations in the evaluation of some separate expressions.

Reference [15] continued the investigation of effectiveness of pairwise testing of logical expressions. To extend the scope of testing, 100 expressions (five sets of 20 expressions in each) were generated. The sizes and complexities of the expressions in each set were similar to 20 expressions from TCAS II specifications to retain consistency and the ability to compare the results. More than 20,000 faulty expressions were generated from these initial 100 expressions for further testing and effectiveness evaluation. For all faults, the weighted average value of effectiveness was 28% and the average values between the types of fault ranged from 21.8% for VRF to 73.4% for ENF. The effectiveness of pairwise testing was compared with the effectiveness of random test sets of the same size. It was shown that pairwise testing has an advantage over random testing for all fault types with the 3% excess in average.

Because the logical expressions in [15] had different sizes from 5 to 15 variables, another goal of this investigation was to evaluate how pairwise-testing effectiveness depended on expression size. Five new logical expression sets were generated so that all expressions in one set had the same size (5, 8, 10, 12, and 15 variables in each expression for each set). The results of testing showed that effectiveness decreased when the expression size increased. Fig.1 (adapted from [15]) presents the plotting of the numerical values of effectiveness.

This investigation of combinatorial testing effectiveness was extended in [16] from pairwise to t-way for t from 2 to 6. The scope of testing was increased to 2,000 logical expressions (10 sets of 200 expressions each) for which more than 250,000 faults were generated and tested. The ACTS tool [18] was used to generate 55 combinatorial test sets. The average effectiveness was 27.7% for pairwise, 42.6% for 3-way, 57.7% for 4-way, 71.1% for 5-way, and 81.5% for 6-way. The results of t-way testing effectiveness were very stable for different logical expressions as shown in Fig. 2 (adapted from [16]). Additional analysis of these results can be found in [3, p. 59].

**TABLE I. COMPARISON OF T-WAY AND RANDOM TESTING EFFECTIVENESS FOR DIFFERENT TYPES OF FAULTS [16]**

<table>
<thead>
<tr>
<th>Type of fault</th>
<th>Type of testing</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNF</td>
<td>t-way</td>
<td>37.3</td>
<td>52.8</td>
<td>67.3</td>
<td>79.1</td>
<td>87.7</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>34.9</td>
<td>48.9</td>
<td>63.1</td>
<td>76.7</td>
<td>85.4</td>
</tr>
<tr>
<td>ORF</td>
<td>t-way</td>
<td>41.7</td>
<td>57.0</td>
<td>70.8</td>
<td>82.0</td>
<td>89.6</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>39.4</td>
<td>54.1</td>
<td>67.4</td>
<td>80.1</td>
<td>87.9</td>
</tr>
<tr>
<td>VRF</td>
<td>t-way</td>
<td>23.2</td>
<td>38.1</td>
<td>53.6</td>
<td>67.8</td>
<td>79.1</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>23.4</td>
<td>35.8</td>
<td>50.1</td>
<td>65.5</td>
<td>76.8</td>
</tr>
<tr>
<td>ENF</td>
<td>t-way</td>
<td>55.5</td>
<td>69.9</td>
<td>80.5</td>
<td>88.8</td>
<td>93.9</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>52.7</td>
<td>65.8</td>
<td>77.0</td>
<td>86.8</td>
<td>92.6</td>
</tr>
<tr>
<td>ASF</td>
<td>t-way</td>
<td>46.2</td>
<td>61.6</td>
<td>75.1</td>
<td>85.4</td>
<td>91.3</td>
</tr>
<tr>
<td></td>
<td>Random</td>
<td>43.7</td>
<td>58.2</td>
<td>71.9</td>
<td>83.0</td>
<td>89.4</td>
</tr>
</tbody>
</table>

Figure 1. Pairwise effectiveness depending on expression size [15].

Figure 2. Effectiveness of t-way testing for different sets of expressions [16].

Reference [16] also provided a comparison of effectiveness between t-way testing and randomly generated test cases of the same sizes. The results for different fault types are presented in Table I (adapted from [16]). For all types of faults and all sets of expressions, it was shown that t-way testing is more effective than random testing. However, the difference is not significant (maximum 3.6 in absolute values and 6.7% in relative values).

Another investigation of combinatorial testing effectiveness in terms of fault-detection probability was carried out by Wang and Qi [19]. They provided a theoretical mathematical analysis of the low bound of fault-detection probability of combinatorial test sets based on so-called Minimal Failure-Causing Schemas and the results of experimental testing for different fault types. The same set of 20 logical specifications from the TCAS II system was used.
but the list of different faults was expanded to 10 types based on the fault class hierarchy from [20]. It was reported that pairwise effectiveness is greater than 40% in most cases, 3-way effectiveness is greater than 75% for most faults, and the low bound of 4-way combinatorial test suites approaches 100%, which is slightly higher than in the research cited earlier.

TCAS II specifications were also used by Okun [21], who performed the reverse experimental testing: test cases were created according to the different mutation operators and pairwise coverage of the test sets was then measured. The results showed that the mutation operators get high pairwise coverage.

When combinatorial testing is used in practice, usually one test set is selected according to some combinatorial approach. For the same approach and the same software, however, there are many different combinatorial test sets and so there are many options to select one. The question is, how stable is the effectiveness among these different combinatorial sets or, in other words, what is the level of variations of effectiveness? This problem can be considered for any testing approach (not only for combinatorial testing) and was studied in [22] as a tolerance of testing criteria.

Stability of pairwise testing with binary inputs was studied by Vilkomir and Pencell [23] for one specific case: pairwise test sets with “don’t care” values. These are values of input variables that are unimportant for the coverage and can be selected randomly (i.e., they do not affect the already achieved 100% pairwise coverage). The portion of “don’t care” values in pairwise tests can be up to 20% of the total inputs. Sets of logical expressions of sizes from 4 to 8 that differed only in “don’t care” values were generated by the ACTS tool using three algorithms (IPOG, IPOG-F, and IPOG-F2).

The results showed that the level of effectiveness is very stable. The standard deviation ranged from 0.4 to 3.1 for all algorithms and all expressions with an average value of 1.9, and the coefficient of variation was an average 3.2%, proving the high stability of the effectiveness. The example of variations of effectiveness for 10 different pairwise test sets for one group of logical expressions is shown in Fig. 3 (adapted from [23]).

In this section, we consider research papers that suggested specific approaches to generate combinatorial testing with binary inputs and more general algorithms suitable for any input spaces but illustrated by examples with binary tests. We also mention some works that are close to this direction and consider general approaches of test generation based on logical models for input spaces and logical relationship (constraints) among input parameters.

Segall et al. [24] introduced an algorithm for combinatorial test design using binary decision diagrams (BDDs). The goal was to increase the scalability of the approach to find a set of tests that satisfies $t$-way coverage and makes it applicable to large complex systems. The IBM FoCuS tool was used, which supports restrictions for test cases, stated as logical expressions. The provided experimental results show superior results of the proposed approach over other algorithms on some real-life test spaces. Salecker et al. [25] also proposed using BDDs for generating combinatorial test cases with constraints among input parameters. The same problem was described by Lawrence et al. in [26], where some theoretical models based on a constraint programming approach were reviewed. They also described a small set of experiments with Boolean inputs to select the best strategy for covering array modeling.

This problem was also considered by Gargantini et al., who suggested several approaches to create combinatorial test cases with logical constraints. Thus, Calvagna and Gargantini [27] developed a logic-based approach to create pairwise test sets with constraints for the input variables. They formalized combinatorial testing as a logic problem and applied a model checker tool to solve it by using generic predicates to describe constraints. For the same purpose, Gargantini and Vavassori suggested using multivalued decision diagrams (MDDs) [28] and decision trees [29].

Yu et al. [30] suggested a topological model (T-model) to select test cases using the structure of logical expressions. They used the pairwise design approach to reduce the number of test cases while keeping the capability of defect detection. Based on the results of the experimental testing, the authors claimed that a T-model-based approach detected more fault types than MC/DC and many other testing strategies.

Verification of formal specifications automatically generated by the Prospec tool [31] was considered by Salamah et al. [32]. They used pairwise testing to reduce the number of formulas that should be tested during verification. Generation of pairwise tests from a cause-effect graph was suggested by Chung [33] and illustrated by the example with binary inputs. The cause-effect graphs were transformed into Alloy models and then pairwise tests were produced via the Alloy analyzer [34].

Several algorithms specifically for constructing binary covering arrays were developed in [35, 36, 37]. Thus, Bracho-Rios, Torres-Jimenez, and Rodriguez-Tello [35] proposed a backtracking algorithm based on the Branch & Bound technique. Rodriguez-Tello and Torres-Jimenez [36] proposed an algorithm based on a Simulated Annealing (SA)
[38], a general-purpose stochastic optimization technique, and Torres-Jimenez and Rodriguez-Tello [37] considered an improved implementation of SA. All of these theoretical models are accompanied by the experimental results, which confirmed the practical advantages of suggested algorithms and showed that they improved on previously known solutions and were “near-optimal.”

Instead of traditional covering arrays, Yilmaz [39] suggested so-called “test case-aware” covering arrays, which were illustrated by examples with binary inputs. This new type of arrays eliminated masking effects caused by constraints in traditional covering arrays. Test case-aware covering arrays were used by Yilmaz et al. [40] in a new combinatorial testing process, which they called “feedback driven adaptive” (FDA-CIT) and evaluated by six empirical studies.

IV. COMPARISON WITH OTHER TESTING CRITERIA

Comparisons of combinatorial testing methods with other testing approaches are an important research topic but still insufficiently studied for a special case of binary inputs. Only a few research papers can be considered in this connection. Thus, Bryce et al. [41] investigated the combinatorial testing of a Flight Guidance System (FGS) [42] that contained 40 input variables, all Boolean. The test sets for combinatorial testing from 2- to 5-way were generated with 13, 33, 83, and 214 test cases in the set, respectively. For comparison, random test sets of the same sizes were created.

One of the goals of experimental testing was to evaluate the coverage achieved with combinatorial test sets and compare it to the random test coverage. The coverage metrics included the decision (branch) coverage and MC/DC. The reported levels of decision coverage were 74% for pairwise, 75.3% for 3-way, 77.6% for 4-way, and 79.1% for 5-way, and the MC/DC levels were 10.3%, 11.0%, 12.3%, and 12.3%, respectively. Regarding the poor coverage achieved by combinatorial testing, the authors explained it by the nature of the FGS system because achieving high coverage required reaching some specific states in the system. All combinatorial test sets showed the better results comparing with random testing both for the decision coverage and MC/DC. However, the differences were not significant, ranging from 0.4% to 2.9%.

The relationship between pairwise and MC/DC testing was experimentally analyzed by Vilkomir and Anderson [43]. Two research questions were investigated:

- What is the MC/DC coverage level for pairwise test cases?
- Is the MC/DC coverage level for pairwise test cases higher than the level for random test cases?

Several sets of logical expressions of sizes from 3 to 8 variables were generated for experimental testing. Both simple (without repetition of variables) and complex (with repetition of variables) expressions were generated for each size. Two situations were considered: when the numbers of inputs coincided with the sizes of expressions (mode 1) and when the numbers of inputs were fixed to 10 and 20, but not all of them used in each expression (mode 2). The ACTS tool was used for combinatorial test generation and the CodeCover [44] tool, developed at the University of Stuttgart, Germany, was used for evaluation of the MC/DC coverage. The results for mode 1 are presented in Table II (adapted from [43]).

### TABLE II. MC/DC COVERAGE FOR MODE 1 [43]

<table>
<thead>
<tr>
<th>Size of expressions</th>
<th>MC/DC coverage, %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple expressions</td>
</tr>
<tr>
<td></td>
<td>PW</td>
</tr>
<tr>
<td>3-var</td>
<td>77.8</td>
</tr>
<tr>
<td>4-var</td>
<td>76</td>
</tr>
<tr>
<td>5-var</td>
<td>70</td>
</tr>
<tr>
<td>6-var</td>
<td>64.2</td>
</tr>
<tr>
<td>7-var</td>
<td>59.3</td>
</tr>
<tr>
<td>8-var</td>
<td>57.5</td>
</tr>
<tr>
<td>Average</td>
<td>67.1</td>
</tr>
</tbody>
</table>

As in the previously mentioned research by Bryce et al., pairwise tests provided better coverage for mode 1, but the difference for all expressions is not very significant (3.4%). For mode 2, the differences in MC/DC coverage for pairwise and random tests were higher: 10% for 10 inputs and 7.2% for 20 inputs.

Some researchers did not consider testing software directly with binary inputs, but they still compared combinatorial approaches with coverage criteria for testing logical expressions. Bartholomew [45, 46] considered combinatorial testing of a radio’s control software containing 196,000 executable source lines of C++. The goal was to evaluate cost-effectiveness, maturity, scalability, usability, and other quality features of the practical application of combinatorial testing and checked its ability to achieve the required coverage level. The initial results were 75% statement coverage, 71% branch coverage, and 68% MC/DC coverage. However, after redefining the input space and iterative test case generation, full branch coverage was achieved. The author claimed that the results of this industry proof-of-concept demonstration were positive and suggested that combinatorial testing would be more cost-effective than other approaches.

Czerwonka [47] studied the stability of size and coverage of t-way (t from 1 to 5) test sets. As software under test, four utilities included in Windows 7 OS were used. Combinatorial test sets were generated by the PICT tool [2]. The results for branch coverage are presented in Table III (information derived from Tables I-IV in [47]).

The experimental data obtained show that coverage levels increase and coverage variability levels decrease with

### TABLE III. BRANCH COVERAGE OF T-WAY TEST SETS (DATA FROM [47])

<table>
<thead>
<tr>
<th>Software</th>
<th>t=1</th>
<th>t=2</th>
<th>t=3</th>
<th>t=4</th>
<th>t=5</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrib.exe</td>
<td>52.05</td>
<td>58.15</td>
<td>60.14</td>
<td>61.05</td>
<td>63.29</td>
</tr>
<tr>
<td>fc.exe</td>
<td>61.42</td>
<td>67.83</td>
<td>68.00</td>
<td>68.09</td>
<td>68.09</td>
</tr>
<tr>
<td>find.exe</td>
<td>54.47</td>
<td>54.47</td>
<td>54.47</td>
<td>54.47</td>
<td>54.47</td>
</tr>
<tr>
<td>findstr.exe</td>
<td>54.88</td>
<td>60.56</td>
<td>60.84</td>
<td>60.86</td>
<td>60.86</td>
</tr>
</tbody>
</table>
increasing values of $t$. General recommendations were included for researchers and practitioners based on the results of the investigation.

V. CONCLUSIONS

Combinatorial testing software with binary inputs is an important task, particularly for safety-critical software with the complex logical structure. Although this area has not been thoroughly studied, some interesting results on the effectiveness and coverage levels of combinatorial testing with binary inputs have been obtained, which sometimes differ significantly from results for more general input spaces. The goal of this review paper is to give an overview of the current research in this area.

To summarize the results of several investigations, the effectiveness of combinatorial testing with binary inputs can be approximately evaluated from 30% for pairwise and up to 80% for 6-way testing. The level of MC/DC and branch coverage of $t$-way test sets can be approximately evaluated as 60%–70%. Comparing combinatorial binary test sets with random test sets of the same sizes shows better effectiveness and coverage for combinatorial sets. However, the differences were not significant and were in the range of 3% to 10%.

In our opinion, the application of combinatorial testing approaches for software with binary inputs is an interesting and promising research direction and new results are expected in the near future.

ACKNOWLEDGMENT

This work was performed under the following financial assistance award 70NANB15H217 from the U.S. Department of Commerce, National Institute of Standards and Technology.

REFERENCES


