Error Locating Driven Array[*]

Xintao Niu
State Key Laboratory for Novel
Software Technology
Nanjing University
China, 210023
niuxintao@gmail.com

Changhai Nie
State Key Laboratory for Novel
Software Technology
Nanjing University
China, 210023
changhainie@nju.edu.cn

JiaXi Xu
State Key Laboratory for Novel
Software Technology
Nanjing University
China, 210023
changhainie@nju.edu.cn

ABSTRACT

Combinatorial testing(CT) is proven to be effective to reveal the potential failures caused by the interaction of the inputs or options of the System Under Test(SUT). A key problem in CT is to isolate the failure-inducing combinations of the related failure as it can facilitate the debugging efforts by reducing the scope of code that needs to be inspected. Many algorithms has been proposed to identify such combinations, however, most of these studies either just consider the condition of one fault or ignore masking effects among multiple faults which can bias their identified results. In this paper, we analysed how the masking effect of multiple faults affect on the isolation of failure-inducing combinations. We further give a strategy of selecting test cases to alleviate this impact, which works by pruning these test cases that may trigger masking effect and replacing them with no-masking-effect ones. The test case selecting process repeated until we get enough information to isolate the failure-inducing combinations. We conducted some empirical studies on several open-source software. The result of the studies shows that multiple faults as well as the masking effects do exist in real software and our approach can assist combinatorial-based failure-inducing identifying methods to get a better result when handling multiple faults in SUT.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and debugging—*Debugging aids,testing tools*

General Terms

Reliability, Verification

Keywords

Software Testing, Combinatorial Testing, Covering Array, Failure-inducing combinations

1. INTRODUCTION

With the increasing complexity and size of modern software, many factors, such as input parameters and configuration options, can influence the behaviour of the SUT. The unexpected faults caused by the interaction among these factors can make testing such software a big challenge if the interaction space is too large. One remedy for this problem is combinatorial testing, which systematically sample the interaction space and select a relatively small set of test cases that cover all the valid iterations with the number of factors involved in the interaction no more than a prior fixed integer, i.e., the *strength* of the interaction.

Once failures are detected, it is desired to isolate the failure-inducing combinations in these failing test cases. This task is important in CT as it can facilitate the debugging efforts by reducing the code scope that needed to inspected.

2. MOTIVATING EXAMPLE

3. BACKGROUND

This section presents some definitions and propositions to give a formal model for the FCI problem.

3.1 Failure-inducing combinations in CT

Assume that the SUT is influenced by n parameters, and each parameter p_i has a_i discrete values from the finite set V_i, i.e., $a_i = |V_i|$ $(i = 1,2,..n)$. Some of the definitions below are originally defined in .

Definition 1. A *test case* of the SUT is an array of n values, one for each parameter of the SUT, which is denoted as a n-tuple $(v_1, v_2,...,v_n)$, where $v_1 \in V_1$, $v_2 \in V_2$ ... $v_n \in V_n$.

In practice, these parameters in the test case can represent many factors, such as input variables, run-time options, building options or various combination of them. We need to execute the SUT with these test cases to ensure the correctness of the behaviour of the software.

We consider the fact that the abnormally executing test cases as a *fault*. It can be a thrown exception, compilation error, assertion failure or constraint violation. When faults are triggered by some test cases, what is desired is to figure out the cause of these faults, and hence some subsets of this test case should be analysed.

Definition 2. For the SUT, the n-tuple $(-,v_{n_1},...,v_{n_k},...)$ is called a *k-value combination* $(0 < k \leq n)$ when some k parameters have fixed values and the others can take on their respective allowable values, represented as "-".

In effect a test case itself is a k-value *combination*, when k = n. Furthermore, if a test case contain a *combination*, i.e., every fixed value in the combination is in this test case, we say this test case *hits* the *combination*.

Definition 3. let c_l be a l-value combination, c_m be an m-value combination in SUT and $l < m$. If all the fixed parameter values in c_l are also in c_m, then c_m *subsumes* c_l. In this case we can also say that c_l is a *sub-combination* of c_m and c_m is a *parent-combination* of c_l, which can be denoted as $c_l \prec c_m$.

For example, in the motivation example section, the 2-value combination (-, 4, 4, -) is a sub-combination of the 3-value combination (-, 4, 4, 5), that is, $(-,4,4,-) \prec (-,4,4,5)$.

Definition 4. If all test cases contain a combination, say c, trigger a particular fault, say F, then we call this combination c the *faulty combination* for F. Additionally, if none sub-combination of c is the *faulty combination* for F, we then call the combination c the *minimal faulty combination* for F (It is also called Minimal failure-causing schema(MFS) in).

In fact, MFS and *minimal faulty combinations* are identical to the failure-inducing combinations we discussed previously. Figuring it out can eliminate all details that are irrelevant for causing the failure and hence facilitate the debugging efforts.

4. ALGORITHMS

4.1 Description

4.2 A case study

5. EMPIRICAL STUDIES

5.1 The existence of

5.1.1 Study setup

5.1.2 Result and discusssion

5.2 Performance of the traditional algorithms

5.2.1 Study setup

5.2.2 Result and discussion

5.3 Performance of our approach

5.3.1 Study setup

5.3.2 Result and discussion

5.4 Threats to validity

6. RELATED WORKS

7. CONCLUSIONS