Mastering combinatorial explosion with the Tobias-2 test generator.

Yves Ledru, Frédéric Dadeau, Lydie du Bousquet, Sébastien Ville, Elodie Rose.

Abstract

This paper briefly describes the second version of the Tobias combinatorial test generator. This version improves the architecture of the tool to include filtering and test selection mechanisms. These mechanisms, associated with an efficient implementation, allow to generate and filter test suites of up to 1 million test cases.

Yves Ledru, Frédéric Dadeau, Lydie du Bousquet, Sébastien Ville, Elodie Rose.
Mastering combinatorial explosion with the Tobias-2 test generator.
22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA, :535-536, 2007. Note: Demonstration.

Mastering Combinatorial Explosion with the Tobias-2 Test Generator

Yves Ledru, Frédéric Dadeau, Lydie du Bousquet, Sébastien Ville, and Elodie Rose
Laboratoire d'Informatique de Grenoble
B.P. 72
38402 St-Martin d'Hères, France
Yves.Ledru@imag.fr, Frederic.Dadeau@imag.fr, Lydie.du-Bousquet@imag.fr

ABSTRACT

This paper briefly describes the second version of the Tobias combinatorial test generator. This version improves the architecture of the tool to include filtering and test selection mechanisms. These mechanisms, associated with an efficient implementation, allow to generate and filter test suites of up to 1 million test cases.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms

Reliability, Verification

Keywords

Combinatorial testing, test suite reduction, Tobias

1. INTRODUCTION

One of the difficulties in the automation of testing is the generation of test cases. Such a generation is often deduced from a model of the software under test. This model can be a specification, like in model-based testing approaches, or the data or control flow diagram of its implementation, like in structural test generation. Another possibility is to capture the knowledge and the know-how of the test engineer into a compact description of the set of test cases. This is actually the approach taken in combinatorial testing where a model captures sets of input parameters for given sequences of method calls. The associated tool then unfolds the model into a possibly large set of test cases.

Tobias is a test generator based on the combinatorial approach. It aims to provide the test engineer with a tool to express his/her test suite in a concise way, and then unfold it into a set of abstract test cases which are eventually translated into executable tests in a given target technology (e.g. Java/Junit). The first version of the tool was used successfully in several case studies, including a mini banking application [1] and a multi-modal fusion engine [2]. These experiments showed that the tool actually improves the productivity of the test engineer: a large test suite can be produced from a few lines of combinatorial description, which give a compact and structured description of the test suite. Other experiments showed that the test suites generated by Tobias are generally more complete and detect more errors than manually generated test suites [4]. In summary, Tobias is a tool that amplifies the work the test engineer by liberating him from the clerical tasks of test production and letting him concentrate on the insightful operations.

2. TOBIAS-2: A COMPLETE REDESIGN OF THE TOOL

In 2006, we undertook a complete redesign of Tobias, based on the experience acquired with the first version of the tool.

- The input and output languages of the tool were redesigned to make them more expressive while simplifying some of the existing concepts.
- The test generation engine is a distinct piece of code, which can interface with various front ends. An Eclipse front end has been developed recently for the tool.
- The architecture of the engine has been opened to connect the tool to several "plug-ins" which help select a subset of the test suite.
- Finally, its efficient implementation allows to produce very large test suites, counting more than 1 million test cases.

The last two improvements are directly linked to combinatorial explosion. While combinatorial techniques allow to easily produce a large set of test cases, they also suffer from combinatorial explosion. In order to master combinatorial explosion, we propose to exploit several mechanisms which help select a subset of the original test suite. These mechanisms also require the tool to be able to generate temporarily a large set of test cases, before the selection takes place. This is why we wanted the tool to be able to generate up to 1 million test cases.

3. A SMALL EXAMPLE

The following example gives the Tobias test suite for an electronic purse. The purse allows simple operations to credit or debit integer amounts of money. The test engineer starts by defining two groups. Group `Amounts` lists various amounts of money which can be debited or credited from the card. The specification of the purse mandates such amounts to be in the range 0..100. Here the test engineer has chosen six valid values, including boundary values 0 and 100, and two forbidden values (-17 and 123) which will test the robustness of the application. The test engineer has defined a second group, OpNames, which contains the names of both operations provided by the purse.

```
Amounts ::= {-17,0,7,12,56,99,100,123}
OpNames ::= {credit,debit}
```

From these two groups, and given an object p of type Purse, one can construct group Modify which combines the operation names with the listed amounts. Unfolding this group leads to 16 test cases, corresponding to all combinations of OpNames with Amounts. These test cases only define the input of the tests. They should be associated to an automated oracle to compute their associated verdicts. In [3] we proposed the use of JML specifications as an oracle for the tests generated by Tobias.

```
Modify ::= { begin p.OpNames(Amounts) end  }
```

The test engineer then defines another group CreditDebit, which credits an amount then debits the credited amount[1]. The unfolding of this group produces 8 test cases.

```
CreditDebit ::= { begin x := p.credit(Amounts);
                  p.debit(x) ;  end  }
```

The following group (T1) creates sequences of 3 calls to group CreditDebit. The unfolding of this group includes 8*8*8=512 test cases.

```
T1 ::= CreditDebit^3..3
```

Finally, group (T2) combines (T1) with (Modify), leading to 16*512=8192 test cases.

```
T2 ::= { begin  Modify ; T1 end  }
```

This example shows that thousands of test cases can be quickly generated from a few lines of Tobias input. Unfortunately, the number of test cases can rapidly become intractable as sequences get longer. Therefore, Tobias proposes several filtering/selection mechanisms to reduce the size of the resulting test suite.

4. FILTERING TOBIAS TEST SUITES

A Tobias filter is a property expressed by the test engineer that must be fulfilled by the test cases of the test suite.

For example, one may filter T1 and keep all (boundary) test cases which refer to amount 100. This results in 169 test cases (out of 512). Also, one may filter T2 to keep all test cases with valid amounts and where the balance of the purse remains between 0 and 100. This keeps 257 test cases out of 8192.

The initial idea of filters already existed in the first version of Tobias [3]. Tobias-2 makes it easier to define filters as boolean functions over the text of the test case, or over its syntax tree. For example, the first filter (keep test cases which refer to amount 100) is expressed as four lines of java:

[1]Actually, credit returns its argument if this argument is not negative, and if the resulting balance is less than 100.

```
public static boolean ConsthasVal100() {
  if (test.indexOf("<val>100</val>")>=0) {
    return true;}
  else {return false;}}
```

Filters provide an easy way to master the size of a test suite. Actually, the test engineer can arbitrarily reduce the number of test cases: if he chooses the true function, all test cases are kept, while false filters out every test case. It is thus the responsibility of the test engineer to design a boolean function which reduces the size of the test suite while still keeping interesting test cases.

5. SELECTING A SUBSET

Tobias-2 provides a second way to reduce the size of a test case: "selectors". While a filter takes a single test case as argument, a selector takes the whole test suite as argument, and returns a subset of the test suite.

Simple selectors use random selection techniques. Other selectors could ensure some coverage criterion, based on the input data (e.g. n-way coverage) or on application specific data (structural coverage of the code, or coverage of the specification). For example, we used a random selector to keep 10% of testsuite T2, resulting in 820 test cases.

This selector mechanism is new in Tobias-2. As for filters, selectors are Java functions. They return an array filled in with the numbers of the selected test cases.

6. CONCLUSION

The Tobias-2 engine is an improved version of the Tobias combinatorial test generator. It keeps the principles of the original tool, but benefits from a better architecture to interface with filters and selectors. Its implementation makes it possible to produce hundreds of thousands of test cases in several minutes. For example, the total time to unfold all the groups of this paper is about 30 seconds on a laptop.

Filters and selectors provide a flexible mechanism to master combinatorial explosion. The number of resulting test cases can be arbitrarily reduced by strong filtering predicates or weak coverage criteria. It is thus the responsibility of the test engineer to figure out the filters and selectors which reduce the size of the test suite while keeping its fault detection capabilities.

In the spirit of Tobias, we expect that Tobias input files, and their associated filters and selectors, will capture the know-how of the test engineer and amplify his/her activity by letting the tool perform the repetitive tasks.

Acknowledgements Part of this work was performed within the POSE project (ANR-05-RNTL-01001), sponsored by the french National Research Agency (ANR).

7. REFERENCES

[1] L. du Bousquet, Y. Ledru, O. Maury, C. Oriat, and J.-L. Lanet. Case study in JML-based software validation. In *ASE 2004*. IEEE CS Press, 2004.

[2] S. Dupuy-Chessa, L. du Bousquet, J. Bouchet, and Y. Ledru. Test of the ICARE platform fusion mechanism. In *DSVIS'05*, volume 3941 of *LNCS*. Springer, 2006.

[3] Y. Ledru, L. du Bousquet, O. Maury, and P. Bontron. Filtering Tobias combinatorial test suites. In *FASE 2004*, volume 2984 of *LNCS*. Springer, 2004.

[4] O. Maury, Y. Ledru, P. Bontron, and L. du Bousquet. Using Tobias for the automatic generation of VDM test cases. In *3rd VDM Workshop (in conjunction with FME'02)*, 2002.