# Automated testing on the analysis of variability-intensive artifacts: An exploratory study with SAT Solvers *

Ana B. Sánchez and Sergio Segura

Department of Computer Languages and Systems, University of Seville,
Av Reina Mercedes S/N Seville, Spain

**Abstract.** The automated detection of faults on variability analysis tools is a challenging task often infeasible due to the combinatorial complexity of the analyses. In previous works, we successfully automated the generation of test data for feature model analysis tools using metamorphic testing. The positive results obtained have encouraged us to explore the applicability of this technique for the efficient detection of faults in other variability-intensive domains. In this paper, we present an automated test data generator for SAT solvers that enables the generation of random propositional formulas (inputs) and their solutions (expected output). In order to show the feasibility of our approach, we introduced 100 artificial faults (i.e. mutants) in an open source SAT solver and compared the ability of our generator and three related benchmarks to detect them. Our results are promising and encourage us to generalize the technique, which could be potentially applicable to any tool dealing with variability such as Eclipse repositories or Maven dependencies analyzers.

## 1 Introduction

*Variability models* are a key asset to represent the common and variable features of a configurable software system. The automated analysis of variability models deals with the automated extraction of information from the models [1], e.g. determining the number of possible configurations of a software product. The analysis operations that can be performed on variability models are often very complex [1, 4]. This hinders the testing of these applications making it difficult, often infeasible, to determine the correctness of the outputs, i.e. oracle problem.

*Feature models* are de-facto standard to manage variability in software product lines [1]. In previous works [4], we presented an automated test data generator for the analysis of feature models overcoming the oracle problem using metamorphic testing [5]. Roughly speaking, we proposed a set metamorphic relations (so-called metamorphic relations) between feature models (inputs) and the set of products that they represent (expected output). Using these relations,

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT Project SETI (TIN2009-07366), and Projects ISABEL (TIC-2533) and THEOS (TIC-5906) funded by Andalusian Government.

A. Ruíz, L. Iribarne (Eds.): JISBD'2012, pp. 387-392, ISBN:978-84-15487-28-9.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

we enabled the automated generation of random feature models representing up to millions of products that were used as test data. Our results were promising showing the effectiveness of our generator to detect most faults in a few seconds without the need for a human oracle. We also detected two defects in FaMa and another two in SPLOT, two popular open source feature model analysis tools.

The positive results obtained in our previous works have encouraged us to explore the possibility of generalizing our approach to other variability-intensive domains. As a proof of concept, in this paper we propose using metamorphic testing for the automated generation of test data for the analysis of propositional formulas, i.e. SAT problems. In particular, we present a set of metamorphic relations between propositional formulas (input) and their set of solutions (output) and a test data generator based on them. The generator starts by creating an empty formula that is then extended by adding random variables to it. At each step, the set of solutions of the formula is computed by using our metamorphic relations. Complex formulas representing thousands of solutions can be efficiently generated by applying this process iteratively. Once generated, solutions are automatically inspected to get the expected output of the analyses over the formulas, e.g. number of solutions of the formula. In order to show the feasibility of our approach, we introduced 100 artificial faults (i.e. mutants) in a popular open source SAT solver and compared the ability of our generator and three related benchmarks to detect them. As a result, our generator found 98.7% of the faults while the rest of benchmarks only detected 76.3% of them as a maximum. We may remark that although a number of benchmarks for testing SAT solver exists, they focus on the evaluation of the performance and not on the functionality. To the best of our knowledge, this is the first work applying metamorphic testing addressing the automated detection of faults in SAT solvers.

We are aware of only one related work that also addresses the automated detection of faults [2]. This work [2] focused on generating random instances in order to find defects in current solvers. However, in this paper, we focus on generating not only random instances (test data inputs) for SAT solvers but we also obtain expected outputs for these data inputs using metamorphic testing.

## 2 Propositional formulas

A *propositional formula* consists of a set of literals or variables and a set of logical connectives constraining the values of the variables, e.g. $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$. A SAT solver is a software package that takes as input a propositional formula and determines if the formula is satisfiable, i.e. there is a variable assignment that makes the formula evaluate to true [1]. Input formulas are usually specified in conjunctive normal form (CNF), that is a standard form to represent propositional formulas where only three connectives are allowed: $\neg$, $\wedge$, $\vee$. Propositional formulas in conjunctive normal form consists of the conjunction of a number of *clauses*, where a clause is a disjunction of a number of *literals* or their negations. Consider the following CNF formula: $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3)$ where $x_i$ rep-

resents literals and $\neg x_i$ their negations, $\vee$ represents the *or* connective and $\wedge$ represents *and* connective, resulting in a formula with two disjunctions and one conjunction. A possible solution for this formula is $x_1$=1, $x_2$=0, $x_3$=1. Hence, this propositional formula is satisfiable. SAT solving is one of the well known NP-complete problems [3].

## 3 Automated metamorphic testing on the analyses of propositional formulas

### 3.1 Metamorphic relations on propositional formulas

In this section, we define a set of metamorphic relations between propositional formulas (i.e. input) and their corresponding set of solutions (i.e. output). Given a propositional formula $f$ in disjunctive format (i.e. a clause), we say that $f$' is a neighbouring formula if it can be derived from $f$ by adding or removing a literal. Next, we present the metamorphic relations among the solutions of a propositional formula $f$ and the one of their neighbours $f$':

*#R1 -New positive literal-*: Consider the formulas and associated set of solutions depicted in Figure 1. $f$' is created from $f$ by adding a new positive literal (C) to the clause. The relation between the solutions of f and f' can be informally described as follows: the solutions of f' include the solutions of f duplicated and extended with all the possible values of the new literal (i.e. C=1 and C=0). Also, f' adds a new solution with the new literal equal to one and the rest of literals negated.

| A | B | f = A v B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Neighbours

Metamorphic relation

| A | B | C | f' = A v B v C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Fig. 1.** Neighbour formula. New positive literal.

*#R2 -New negative literal-*: Consider the formulas and associated set of solutions depicted in Figure 2. $f$' is created from $f$ by adding a new negated literal ($\neg C$) to the clause. In this case, the solutions of f' include the solutions of f duplicated and extended with all the possible values of the new literal (i.e. C=1 and C=0). Also, f' adds a new solution with the new literal equal to zero and the rest of literals negated.

*#R3 -Existing variable with different state-*: Consider the formulas $f = A \vee B$ and $f' = A \vee B \vee \neg A$ . $f$' is created from $f$ by adding a literal ($\neg A$) that already existed in the clause with opposite state (A). This is known as tautology, i.e. the formula is true for any variable assignment. This is, the solutions of f' should include all possible literal assignments.

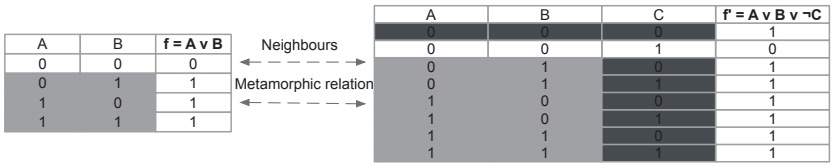


**Fig. 2.** Neighbour formula. New negative literal.

For completeness, we also considered the case in which an existing variable is added to a clause (e.g. $f = A \vee B$ and $f' = A \vee B \vee A$). In this case, the set of solutions of f' and f remains equal.

### 3.2 Automated test data generation

We propose a process to automatically generate test data for the analyses of propositional formulas using previous metamorphic relations. The figure 3 illustrates an example of our approach. The generation process is iterative. On each iteration, a clause (i.e. disjunction) and its set of solutions is computed. The generation of each clause starts from an empty formula in which random literals are added creating neighbouring formulas and its corresponding set of solutions according to the metamorphic relations. These iterations end when all the clauses of the formula together with their set of solutions have been generated. At the end of the process, all the clauses are joined using conjunctions and creating a valid CNF formula. The final set of solutions is calculated as the intersection of the set of solutions of all the generated clauses. In the example, a formula with two clauses is created ($f = (A \vee B \vee \neg C) \wedge (A \vee B)$). The formula has five different solutions, i.e. it is satisfiable.

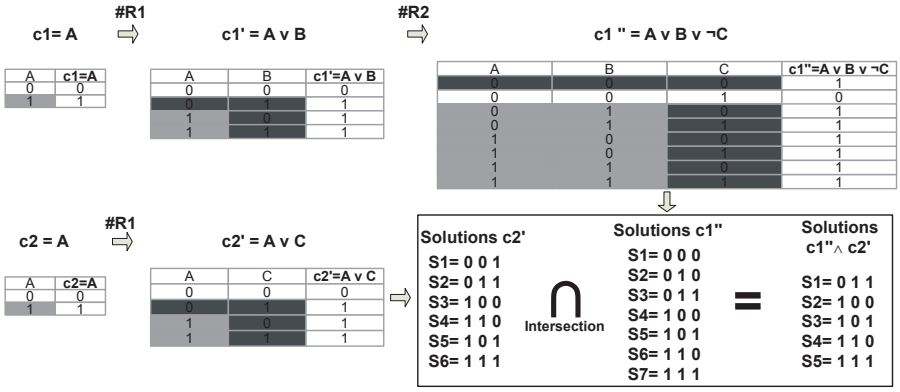


**Fig. 3.** Example random propositional formula generation using metamorphic relations

## 4 Preliminary evaluation

As a part of our proposal, we implemented a prototype test data generator. Our tool generates random CNF formulas and its expected solutions. The parameters for the generation are received as input and include, among others, the number of literals, the number of clauses and the number of literals per clause. In order to measure the effectiveness of our proposal, we evaluated the ability of our test data generator to detect faults in the software under test. We applied mutation testing on an open source solver for the analysis of propositional formulas. *Mutation testing* is a common fault-based testing technique that measures the effectiveness of test cases. First, simple faults are introduced into a program creating a collection of faulty versions, called *mutants*. These mutants are created from the original program by applying syntactic changes to its source code. Then, test cases are used to check if the mutants and the original program return different responses. If a test case distinguishes the original program from a mutant we say the mutant has been *killed* and the test case has proved to be effective at finding faults in the program. Otherwise, the mutant remains *alive*. Mutants that keep the program's semantics unchanged and thus cannot be detected are referred to as *equivalents*. *Mutation score* is the percentage of killed mutants with respect to the total number of them (discarding equivalent mutants) and it provides an adequacy measurement of the test suite.

We selected Sat4j[1], a popular open source Java SAT solver as good candidate to be mutated. In particular, we mutated the class *Solver.java* (53 methods, 2,223 lines of code), main class of the solver. To automate the mutation process, we used the mutation tool PIT[2] which showed to be suitable for our approach. Test cases were generated randomly using our prototype tool as described in section 3.1. For the evaluation of our approach, we followed three steps: First, we checked whether the original Sat4j solver passed all the test before its analyses. All the test cases were passed successfully. Second, we generated the mutants by applying all the mutation operators available in PIT, a total of 12. Finally, for each mutant, we ran our test data generator and tried to find a test case that kills it (a maximum of 250 test cases were generated and ran for each mutant).

Table 1 shows the evaluation results after comparing our approach with other 3 benchmarks from the library SATLIB[3]. Out of the 100 generated mutants, 22 were manually discarded because they modified code not covered by any of the benchmarks under evaluation nor our tool. We found that most of them affected secondary functionality of the program not addressed by the tests (e.g. computation of statistics). Also, we discarded two more mutants not exercised by any of the benchmarks and whose semantic equivalence was not clear. This resulted in a total of 76 mutants being evaluated. Results revealed that our generator found 98.7% of the faults (i.e. 75 out of 76) while the rest of benchmarks only detected 76.3% of them as a maximum. We may mention that our generator provides

[1] `http://www.sat4j.org/`
[2] `http://pitest.org/`
[3] `http://www.satlib.org/`

information about the satisfiability of the formula and its exact set of solutions, while the benchmarks only report about the satisfiability of the problem. Hence, with our generator it is possible a more rigorous comparison of the results. For this reason, our generator got a higher score even with simpler formulas. Finally, we generated 250 test cases for efficiency but results suggest that the mutation score could reach 100% by increasing the number of test cases generated.

| Test data | Literals | Clauses | Formulas | Mutants | Alive | Score |
|---|---|---|---|---|---|---|
| Our generator | 12 | 24 | 250 | 76 | 1 | 98.7 |
| SATLIB uf20-91 | 20 | 91 | 250 | 76 | 38 | 50.0 |
| SATLIB uf50-218 | 50 | 218 | 250 | 76 | 27 | 64.5 |
| SATLIB uf20-91-2 | 20 | 91 | 250 | 76 | 18 | 76.3 |

**Table 1.** Evaluation results

## 5 Conclusions and future work

In this paper, we presented a set of metamorphic relations between propositional formulas and a test data generator based on them. Our results show that the application of metamorphic testing in the domain of SAT solvers is effective in detecting most faults without need for a human oracle. To the best of our knowledge, this is the first work applying metamorphic testing addressing the automated detection of faults in SAT solvers.

We identify several challenges for our future work. First, we intend to work in the formal definition of the metamorphic relations and in a more exhaustive evaluation of our test data generator. Secondly, our results are promising and encourage us to generalize this technique. SAT problems are the base of many variability analysis tools such as web configurators, Eclipse repositories or Debian packages repositories analyzers. In fact, this work has allowed us to automate testing on CUDF documents where we have already been able to test p2CUDF automatically, a tool based on Sat4j. Thus, we envision that the concepts shown here could be applicable to many other variability-intensive domains.

## References

1. David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analyses of feature models 20 years later: A literature review. *Information Systems*, 2010.
2. R. Brummayer, F. Lonsing, and A. Biere. Automated testing and debugging of sat and qbf solvers. In *Conf. on Theory and Applications of Satisfiability Testing*, 2010.
3. S. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.
4. S.Segura, R.Hierons, D.Benavides, and A.Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Information Software Technology*, 2011.
5. Z.Q.Zhou, DH.Huang, TH.Tse, Z.Yang, H.Huang, and TY.Chen. Metamorphic testing and its applications. In *Symposium on Future Software Technology*, 2004.

# PrMO: An Ontology of Process-reference Models

César Pardo[1,2], Félix García[2], Francisco J. Pino[1,2], Mario Piattini[2] and María Teresa Baldassarre[2]

[1] IDIS Research Group
Electronic and Telecommunications Engineering Faculty
University of Cauca, Street 5 # 4 - 70.
Kybele Consulting Colombia (Spinoff)
Popayán, Cauca, Colombia.
{cpardo, fjpino}@unicauca.edu.co

[2] ALARCOS Research Group
ITSI, Information Systems and Technologies Department
University of Castilla–La Mancha, Paseo de la Universidad 4, Ciudad Real, Spain
{Felix.Garcia, Mario.Piattini}@uclm.es

[3] Department of Informatics, University of Bari.
SER&Practices, SPINOFF, Via E. Orabona 4, 70126, Bari, Italy
baldassarre@di.uniba.it

**Resumen.** For a couple of decades, process quality has been considered as one of main factors in the delivery of high quality products. Multiple models and standards have emerged as a solution to this issue, but the harmonization of several models in a company for the fulfillment of its quality requirements is no easy task. The difficulty lies in the lack of specific guidelines and in there not being any homogeneous representation which makes this labor less intense. To address that situation, this paper presents an Ontology of Process-reference Models, called PrMO. It defines a Common Structure of Process Elements (CSPE) as a means to support the harmonization of structural differences of multiple reference models, through homogenization of their process structures. PrMO has been validated through instantiation of the information contained in different models, such as CMMI-(ACQ, DEV), ISO (9001, 27001, 27002, 20000-2), ITIL, Cobit, Risk IT, Val IT, BASEL II, amongst others. Both the common structure and the homogenization method are presented, along with an application example. A WEB tool to support the homogenization of models is also described, along with other uses which illustrate the advantages of PrMO. The proposed ontology could be extremely useful for organizations and other consultants that plan to carry out the harmonization of multiple models.

**Keywords:** Harmonization of multiple models and standards; homogenization; mapping; integration; ontology, processes.

A. Ruíz, L. Iribarne (Eds.): JISBD'2012, pp. 393-406, ISBN:978-84-15487-28-9.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.