Automatic Test Case Generation from UML Models and OCL Expressions

Stephan Weißleder[1], Dehla Sokenou[2]

[1]Humboldt-Universität zu Berlin, Institut für Informatik
Rudower Chaussee 25, 12489 Berlin, Germany
weissled@informatik.hu-berlin.de

[2]GEBIT Solutions
Koenigsallee 75 b, 14193 Berlin, Germany
dehla.sokenou@gebit.de

Abstract: In this paper, we discuss one approach of automated test case generation from UML models and OCL expressions. We show how to use UML and OCL to support several coverage criteria. We introduce our current prototype implementation, compare it to commercial tools, and sketch shortcomings and further development.

1 Introduction

Testing is the primary means to detect faults in a system under test (SUT), e.g. a software product. Modelling languages like the Unified Modeling Language (UML) [Gro05] are means to design tests. There is established tool support for model-based test generation based on UML. For instance, Rhapsody ATG [Tel] supports test generation for the target language C++. Leirios Test Designer [Lei] supports a subset of the Object Constraint Language (OCL). We argue that the tool support of model-based testing could be improved. The quality of test suites is often measured with coverage criteria (although there is no proof of their effectiveness). We argue that the test generation support for boundary-based coverage criteria [KLPU04] could also be improved. For instance, AETG [Tec] depends on user-defined partition boundaries instead of deriving them from the model.

Our approach is to automatically generate test cases from UML state machines, UML class diagrams, and OCL expressions to satisfy boundary-based coverage criteria. We correlate OCL pre-/postconditions of operations and guard conditions of state machines to derive test data input partitions automatically [WS07a]. This allows to generate test suites that satisfy, e.g., a combination of *Multi-Dimensional* [KLPU04] (each variable takes each boundary value of each partition at least once) and *MC/DC* [UL06] (every condition in a decision in the program has taken on each possible outcome at least once, and each condition has been shown to affect that decision outcome independently) (see [WS08]). To our knowledge, ParTeG is the first tool that generates equivalence classes for test input parameters from UML and OCL.

2 Test Case Generation from UML and OCL

UML is a widely accepted means to express models used in model-based test generation. Since OCL constraints can express conditions that can not be phrased in UML, OCL is often used together with UML models for test case generation. We shortly sketch previous approaches to use UML and OCL: One approach is to use OCL as an additional test oracle (compare e.g. [Sok06]). In this case, OCL expressions are simply transformed to the language of the SUT. Another approach is to relate pre-/postconditions of one operation and derive behavioural information from static models [BBH02]. A more advanced approach is presented by Leirios [Lei], that combines different OCL conditions by the control flow information of a state machine. Equations in postconditions are interpreted as assignments in order to allow a symbolic execution of the model. However, this approach supports only a subset of OCL. For instance, our approach additionally considers inequations and a wider set of logical expressions (e.g. ”or”, ”not”) in postconditions.

The basic idea of relating OCL conditions is that postconditions attached to a transition affect the satisfaction of subsequent guard conditions. Consequently, it is possible to identify dependencies between them in the control-flow of state machines. A classification of the elements comprised in postconditions allows to define the attributes of a postcondition, whose values are changing [WS07a]. This allows to correlate certain elements of guard conditions and preconditions to corresponding elements of postconditions. As a result of this approach, guard expressions can be transformed into conditions for input parameters. These resulting conditions are interpreted as partition boundaries.

2.1 Prototype Implementation

We implemented our approach in an Eclipse-based tool called ParTeG [Wei]. This tool checks all paths from the initial state of the UML state machine until a loop is reached. Every time the tool encounters, e.g., a guard condition, it searches the path backwards to find a postcondition that affects the outcome of this guard condition. The postconditions describe, e.g., the effect of input parameters on attribute values. This allows to create a connection between the values of test input parameters and the control flow within the state machine. The constraints on test input parameters are interpreted as equivalence classes and the concrete values are selected close to the corresponding boundaries (see Fig. 1).

2.2 Coverage Criteria

Coverage criteria are the primary means to measure the quality of a test suite. Nevertheless, there is no proof for a relationship between satisfied coverage criteria and the number of identified faults. This is fortified by the success of random testing compared to model-based testing, which brings up the question for cost efficiency of model-based testing [Pre06]. Consequently, some approaches combine random testing and the mere

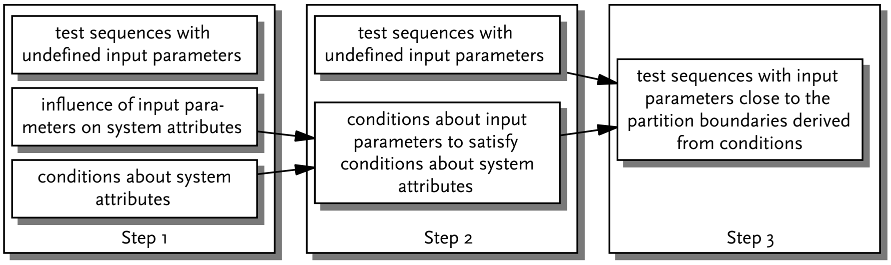
test sequences with undefined input parameters
influence of input parameters on system attributes
conditions about system attributes
Step 1
test sequences with undefined input parameters
conditions about input parameters to satisfy conditions about system attributes
Step 2
test sequences with input parameters close to the partition boundaries derived from conditions
Step 3

Figure 1: Basic idea of the test generation process.

observation of coverage criteria. For instance, Rhapsody ATG [Tel] follows the approach of generating test cases and measuring the satisfied coverage. Although our current approach is not random, we also observe the percentage of satisfied coverage criteria instead of enforcing it. However, we plan to change this strategy to enforce the satisfaction of each part of a coverage criterion (test goal).

In addition to satisfying just one coverage criterion, the satisfaction of a combination of different coverage criteria seems to be promising [WS08]. Since ParTeG is already focussed on boundary coverage, it allows to combine, e.g., boundary-based coverage criteria and control-flow-based coverage criteria.

2.3 Comparison to Commercial Tools

We compared our prototype implementation to already existing commercial tools. In this context, we used mutation analysis with a subset of the “sufficient mutation operators” defined in [OLR+96]. Since our approach is focused on the deduction of test input data partitions, the test suites generated by ParTeG reached a higher number of killed mutants than Rhapsody ATG and Leirios Test Designer [WS07b].

Most tools for automatic test generation are fixed on satisfying certain coverage criteria. Since each coverage criterion has its eligibility for a certain application field and a certain required safety level, it would desirable to let the user choose the coverage criterion to satisfy. ParTeG allows the user to choose from a set of coverage criteria, which makes it applicable for a broader set of possible application environments.

2.4 Shortcomings and Further Development

The results obtained from the comparisons to commercial tools only hold for examples with few loops and a few conditions based on the repetition of executing these loops. This is caused by the internal structure used for deriving test cases: a tree that describes possible test execution paths. In order to avoid infinite tree size, the length of the paths has to be limited. This results in problems for covering conditions whose satisfaction depends on certain executed loops in a path (e.g., a cash card that is only retained after entering a false PIN three times). Since this leads to suboptimal coverage, we aim at a different approach: Instead of creating a tree, we stick to the graph structure of the state machine. Based on the coverage criteria to satisfy, we define certain test goals like a set of transitions for transition coverage or a set of assignments of a condition's attribut values for MC/DC. Following that, we iterate through all test goals and apply our approach of deducing test input values to a test case generation algorithm like presented in [OA99] or [Sok06].

Acknowledgements. This work was supported by grants from the DFG (German Research Foundation, research training group METRIK).

References

[BBH02] M. Benattou, J.-M. Bruel, and N. Hameurlain. Generating Test Data from OCL Specification. In *WITUML'02*, 2002.

[Gro05] Object Management Group. Unified Modeling Language (UML), version 2.0, 2005.

[KLPU04] N. Kosmatov, B. Legeard, F. Peureux, and M. Utting. Boundary Coverage Criteria for Test Generation from Formal Models. In *ISSRE '04*, pages 139–150. IEEE, 2004.

[Lei] Leirios. LTG/UML. http://www.leirios.com.

[OA99] J. Offutt and A. Abdurazik. Generating Tests from UML Specifications. In *UML'99*, pages 416–429, 1999.

[OLR$^+$96] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, pages 99–118, 1996.

[Pre06] A. Pretschner. Zur Kosteneffektivität des modellbasierten Testens. In *MBEES'06: Modellbasierte Entwicklung eingebetteter Systeme*, pages 85–94, 2006.

[Sok06] D. Sokenou. Generating Test Sequences from UML Sequence Diagrams and State Diagrams. In *INFORMATIK 2006*, pages 236–240, 2006.

[Tec] Telcordia Technologies. AETG. http://aetgweb.argreenhouse.com.

[Tel] Telelogic. Rhapsody Automated Test Generation. http://www.telelogic.com.

[UL06] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., 2006.

[Wei] S. Weißleder. ParTeG (Partition Test Generator). http://parteg.sourceforge.net.

[WS07a] S. Weißleder and B.-H. Schlingloff. Automatic Test Generation from Coupled UML Models using Input Partitions. In *MoDeVVa*, 2007.

[WS07b] S. Weißleder and B.-H. Schlingloff. Deriving Input Partitions from UML Models for Automatic Test Generation. In *LNCS Volume on Models in Software Engineering*, 2007.

[WS08] S. Weißleder and B.-H. Schlingloff. Quality of Automatically Generated Test Cases based on OCL Expressions. In *ICST*, April 2008.