ParTeG - Integrating Model-Based Testing and Model Transformations

Dehla Sokenou
GEBIT Solutions
dehla.sokenou@gebit.de

Stephan Weißleder
Fraunhofer-Institut FIRST
stephan.weissleder@first.fraunhofer.de

Abstract: In this paper, we present model-based testing for UML state machines with the test generator ParTeG. We sketch the impact of model transformations on model-based testing and also sketch the results of a corresponding case study with ParTeG.

1 Model-Based Testing

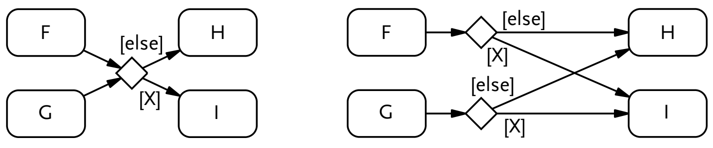
ParTeG [Wei] is a model-based testing tool that was initially developed to implement new algorithms into a prototype as a proof of concept. By now, ParTeG 1.3.1 is available as a free Eclipse plug-in, hosted by Sourceforge. It automatically generates test cases from UML state machines and class diagrams that are annotated with OCL expressions. ParTeG interprets a wide range of OCL expressions, including inequations. ParTeG supports test suite generation for JUnit 3.8 / 4.3, Java Mutation Analysis, and CppUnit 1.12. In the following, we present a short overview of ParTeGs features for model-based test generation.

Coverage criteria are a widely accepted means of test suite quality measurement. There are several kinds of coverage criteria. The most important feature of ParTeG is the ability to satisfy *combined coverage criteria*: For instance, control-flow-based coverage criteria like MC/DC or transition-based coverage criteria like All-Transitions can be combined with boundary-based coverage criteria like Multi-Dimensional: Transition paths are generated according to the control-flow-based respectively transition-based coverage criterion. The selected coverage criterion is converted into a set of model-specific test goals: For instance, All-States is converted into test goals, each of which is referencing one state of the state machine. ParTeG generates paths from the state machine's initial state to the states that are referenced by the test goals. All conditions on each path are transformed and used for concrete input parameter selection corresponding to the selected boundary-based coverage criterion. With these input parameters, the state machine paths are converted into executable test cases of one provided target language. ParTeG also supports test goal monitoring so that already covered test goals are excluded from further test case generations.

Mutation analysis is a wide-spread approach to measure the fault detection capability of a test suite. This approach is based on fault injection in a correct implementation. ParTeG supports mutation analysis by using a mutation factory provided by the tester that delivers a new mutant for each test execution and by generating JUnit code which can be used by the mutation tool Jumble [UTC+07].

2 Model Transformations

Model transformations are means to convert a model into any other model of any other modeling language. ParTeG supports several model transformations to improve the fault detection capability of the generated test suite. It changes the structure of a state machine while preserving its semantics. Several transformations can be found in a corresponding report on an industrial cooperation [Wei09]. We present just one example: State machines can contain choice pseudostates that reference more than one incoming and more than one outgoing transition (see Figure 1(a)). The outgoing transitions also contain guards. Now we compare the effects of existing coverage criteria: Transition-based coverage criteria are focussed on transition sequences. They can enforce that all transition sequences from the states *F* or *G* to *H* or *I* are traversed. They fail, however, if the guard condition is more complicated and we are interested in condition values. Control-flow-based coverage criteria are focussed on guard conditions, but they do not necessarily cover all transition paths. As a consequence, the guard values *[X]* and *[else]* may be tested, e.g. just including the state *G* – all faults that are related to state *F* may be undiscovered. An intuitive solution seems to consist of generating two test suites for both kinds of coverage criteria. As a result, transition sequences and guards are tested. However, the guards are not properly tested for each sequence. Instead, we propose to transform the state machine, e.g. by splitting choice pseudostates according to their number of incoming transitions. Figure 1(b) shows the transformed test model for this example. Since there is only one incoming transition for each choice pseudostate, any selected control-flow-based coverage criterion now has to be satisfied for both states *F* and *G*. In our industrial cooperation, all proposed model transformations resulted in an significant increase of the generated test suite's fault detection capability. All proposed transformations are implemented in ParTeG. They can be automatically executed for all used state machines.



(a) Part of the initial state machine. (b) Part of the transformed state machine.

Figure 1: Model transformation to split choice pseudostates.

References

[UTC+07] Mark Utting, Len Trigg, John G. Cleary, Archmage Irvine, and Tin Pavlinic. Jumble. http://jumble.sourceforge.net/, 2007.

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

[Wei09] Stephan Weißleder. Influencing Factors in Model-Based Testing with UML State Machines: Report on an Industrial Cooperation. In *Models 2009*, October 2009.