VTG : Vulnerability Test cases Generator, a Plug-In for Rodin

Aymerick Savary, Jean-Louis Lanet,
Marc Frappier and Tiana Razafindralambo

January 16, 2012

This project aims to develop a plugin within the Rodin platform in order to generate vulnerability tests using Event B and ProB. Vulnerability testing consists in testing behaviors which should not be accepted by a system. In the context of this work, a vulnerability test is an event trace which contains at least one invalid event. In this first version of the tool, we want to generate traces containing exactly one invalid event. Our approach is to build an abstract model M of the system which describes only the valid behaviors. Then we pick one event to test, called the *event under test* (EUT), and add it to model M by negating its guard. Thus, the execution of this altered event will generate an invalid trace. We use ProB to generate traces, using temporal formulas to guide the search of traces. If one wants a trace that reaches a state s satisfying a conditions $c(s)$, we simply ask ProB to check whether the LTL formula $G(\neg c(s))$ holds; if it finds a counter example, then this counter example is a test case.

The crux of the tool is to generate negation of guards. The tool takes in input a set of rewrite rules for this purpose. There are several possibilities to generate the negation of a guard: for instance, $a \wedge b$ can be rewritten simply as $\neg(a \wedge b)$, but this does not necessarily result into "useful" test cases with a good coverage when ProB is used to find a trace. The rewrite rules allow the specifier to guide the generation of test cases by describing several variants of a negation, in order to obtain specific test cases; in essence, rewrite rules describe test criteria in a generic manner. For instance, $a \wedge b$ could be rewritten into three test cases using three rules: i) $a \wedge \neg b$, ii) $\neg a \wedge b$, iii) $\neg a \wedge \neg b$. Rewrite rules can be defined for each operator of the Event B language, including logical connectives and predicates (*e.g.*, $=$, $<$, $\in$, ...). Directives can also be added to indicate which part of a guard should be rewritten and which part should not, in order to avoid meaningless or unfeasible test cases.

The tool can take as input a model M and it iterates over each event of the model, applying all possible rewrite rules to the event's guard and calling ProB to generate traces for each possible rewriting of the event's guard.

We are using this tool to generate vulnerability tests for Java Card Byte Code Verifiers (JBCV). A JBCV is responsible for checking that JBC programs satisfy the security constraints of the Java specification. Thus, a vulnerability

test case for a JBCV is a JBC program that contains at least one JBC instruction violating the JVM specification. It can be generated using our tool by creating a machine M that contains one event for each instruction of the JBC language. The security constraints of the JVM specification are represented as event guards for JBC instructions. We have defined rewrite rules for this application domain, but most of these rules are general enough to be re-used in other domains as well. We use a single temporal formula to generate test cases: $G(\ not(\ \{eut = TRUE\ \&\ halt = TRUE\}\ )\ )$. Variable eut is set to $FALSE$ in the initialization of the machine. The guard of the EUT checks that $eut = FALSE$ and sets it to $TRUE$ in its actions, ensuring that the EUT is executed only once. Variable $halt$ indicates that the JVM has reached a terminal state, thus the trace represents a complete JBC program.

This plugin enables the use of Event B and ProB to rapidly generate complex test cases using an abstract model of the system to test. This combination provides a simple and very flexible mechanism for generating vulnerability test cases for various application domains.