Symbolic Execution and Model Checking for Testing

Corina S. Păsăreanu[1] and Willem Visser[2]

[1] Perot Systems Government Services/NASA Ames Research Center
Moffett Field, CA 94035, USA
Corina.S.Pasareanu@nasa.gov
[2] SEVEN Networks
Redwood City, CA 94063, USA
willem@gmail.com

Techniques for checking complex software range from model checking and static analysis to testing. We aim to use the power of exhaustive techniques, such as model checking and symbolic execution, to enable thorough testing of complex software. In particular, we have extended the Java PathFinder model checking tool (JPF) [3] with a symbolic execution capability [4, 2] to enable test case generation for Java programs. Our techniques handle complex data structures, arrays, as well as multithreading, and generate optimized test suites that satisfy user-specified testing coverage criteria.

Programs are executed on symbolic, rather than concrete, inputs; the variable values are represented as expressions and constraints that reflect the code structure. JPF generates and analyzes different symbolic execution paths. The input constraints for one path are solved (using off-the-shelf constraint solvers) to generate tests that are guaranteed to execute that path. To bound the search space we put a limit on the model checking search depth, or on the number of constraints along one path. Alternatively, we use abstract state matching [1], which enables JPF to analyze an under-approximation of the program behavior.

Our techniques have been used in black box and white box fashion [5]. They have been applied to generate test sequences for object-oriented code [6] and test vectors for NASA software. Recently, we have also applied our techniques to (executable) models – using a JPF extension for UML Statecharts.

References

1. S. Anand, C. S. Păsăreanu, and W. Visser. Symbolic execution with abstract subsumption checking. In *Proc. SPIN*, 2006.
2. S. Anand, C. S. Păsăreanu, and W. Visser. JPF-SE: A symbolic execution extension to Java PathFinder. In *Proc. TACAS*, 2007.
3. Java PathFinder. http://javapathfinder.sourceforge.net.
4. S. Khurshid, C. S. Păsăreanu, and W. Visser. Generalized symbolic execution for model checking and testing. In *Proc. TACAS*, 2003.
5. W. Visser, C. Păsăreanu, and S. Khurshid. Test input generation with Java PathFinder. In *Proc. ISSTA*, 2004.
6. W. Visser, C. Păsăreanu, and R. Pelánek. Test input generation for Java containers using state matching. In *Proc. ISSTA*, 2006.