

Tools for Symbolic Execution and Dynamic Analysis

<http://d3s.mff.cuni.cz>

Department of
Distributed and
Dependable
Systems



Pavel Parízek



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

JPF extensions

- JPF-core
- **JPF-symbc**
 - Symbolic Pathfinder
- Other
 - JPF-abstraction
 - JPF-statechart
 - JPF-awt
 - JPF-inspector
 - JPF-trace-server
 - ... and much more

Symbolic PathFinder (SPF)



- Performs symbolic execution of Java bytecode
 - Symbolic values stored in attributes associated with program variables (tracked during state space traversal)
- Supported data types
 - int, long, boolean, float, double, arrays, strings (limited)
- Uses the JPF-core to handle multi-threading
- Third-party decision procedures (SMT) are used to check satisfiability of path conditions (PaC)

- Web site
 - <https://github.com/SymbolicPathFinder/jpf-symbc>
- Documentation
 - <https://github.com/SymbolicPathFinder/jpf-symbc/wiki/Documentation>

Using Symbolic PathFinder

- Download and unpack
 - <http://d3s.mff.cuni.cz/files/teaching/nswi132/files/JPF-SE.zip>
- Example 1
 - `run-spf.bat jpf-symbc\src\examples\summerschool\SwapSimple.jpf`
 - Output: analyzed symbolic tree, several derived concrete input values (-99, -100)
- Example 2
 - `run-spf.bat jpf-symbc\src\examples\summerschool\Loop.jpf`
 - Output: path conditions over integer constants (“CONST_xx”) and symbolic values (“n_1_SYMINT”)

SPF: Mixed concrete and symbolic execution

- Symbolic execution can start at any point
 - program state, code location (method boundary)
- Mixed concrete and symbolic values
 - every local variable in a given procedure has either symbolic value or concrete value

Symbolic PathFinder: more examples

- Example 3: floating point numbers
 - `src/examples/NumberExample.jpf`
- Example 4: large path conditions
 - `src/examples/rjc/RJCSymbConfig.jpf`
- Example 5: heap and threads
 - `src/examples/symbolicheap/HeapAndThreads.jpf`

RoadRunner

- Dynamic analysis framework for concurrent Java programs
- Important characteristics
 - Written purely in Java, lightweight, modular, easy composition of dynamic analyses (tool chains)
- Web site
 - <http://dept.cs.williams.edu/~freund/rr/>
 - <https://github.com/stephenfreund/RoadRunner>

RoadRunner: Features

- Adds instrumentation code at the bytecode load time using a special class loader
- API for implementing custom dynamic analyses
 - Filters over the stream of events generated by a target program → composition
- Events: field access, lock acquire, lock release, thread start, method call, return, ...
- Shadow state (analysis data)
 - memory locations (fields, variables), threads, locks

RoadRunner: Usage

- Download
 - <http://d3s.mff.cuni.cz/files/teaching/nswi132/files/RoadRunner.zip>
- Basic test
 - Command: `build\bin\rrrun.bat test.Test`
- Lock-set analysis
 - `build\bin\rrrun.bat "-tool=TL:RS:LS" test.Test`
 - reports many data races on the field `Test.y`
- Shortcuts (TL, RS, LS, ...)
 - Look into the files `classes/**/rrtools.properties`
- Problem: works maybe just for Java 8 (compilation, agent)

RoadRunner: Designing custom analyses

- Abstract superclass: `Tool`
- Define handlers for interesting events
- Manage shadow state properly
- Be careful about thread synchronization
- Examples
 - `src/rr/simple/CountTool.java`
 - `src/rr/simple/ThreadLocal.java`

SharpDetect

- Dynamic analysis framework for C#/.NET programs
- Web site
 - <https://github.com/acizmarik/sharpdetect>
- Implements few dynamic analyses
 - C. Flanagan and S.N. Freund. **FastTrack: Efficient and Precise Dynamic Race Detection**. PLDI 2009
- Live demo: building, sample analysis, plugins