Formal Methods for Java
Lecture 10: Java Pathfinder and Design By Contract

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Nov 23, 2012
Testing Programs with Java Pathfinder
Java Pathfinder on an example

Java Pathfinder is extendable

There are several extensions already:

http://babelfish.arc.nasa.gov/trac/jpf/wiki/projects/start

We take a closer look into jpf-aprop.
What is jpf-aprop?

The jpf-aprop project contains Java annotation based program property specifications, together with corresponding listeners to check them.

- Uses Java annotations, see JDK 1.5.
- Property Specification similar to JML
- JSR-305 and JSR-308 proposals
- To check them, listeners need to be added to jpf config.
Annotations in Java use prefix @
They can be added as modifier to class, field, and method definitions.

- @NonNull – check for null values
- @Const – check for object modifications
- @SandBox – check for modifications
- @GuardedBy – lock policy specifications
- @NonShared – check for concurrent use
- @Requires, @Ensures and @Invariant – Design by Contract
- @Sequence, @SequenceEvent, @SequenceMethod, @SequenceObject – automatic UML sequence diagram creation
- @Test – in-source method test specifications
- @Confined, @Region – check that references do not leave regions.
Design By Contract

\[
\text{Contract} ::= \text{Contract LogicOp Contract} | \text{Term RelOp Term} \\
| \text{Term instanceof ID} | \text{Term matches String} \\
| \text{Term isEmpty} | \text{Term notEmpty} \\
| \text{Term within Term +- Term} | \text{Term within Term , Term} \\
| \text{Term satisfies Property}
\]

\[
\text{Term} ::= \text{Term BinOp Term} | \text{Function}(\text{Term}^*) | \text{old(Term)} \\
| \text{String} | \text{Number} | \text{Var} | \text{null} | \text{EPS} | \text{return}
\]

\[
\text{LogicOp} ::= \&\& | \mid\mid
\]

\[
\text{RelOp} ::= == | != | < | <= | > | >=
\]

\[
\text{BinOp} ::= + | - | * | / | ^
\]

\[
\text{Predicate} ::= \text{ID} | \text{ID(Term}^*)
\]

\[
\text{Function} ::= \text{ID} | \log | \log10
\]

\[
\text{Var} ::= \text{ID}
\]
@Invariant({"numElems\textsubscript{\textlt}\textgreater\textsubscript{\textleq}0",
   "elems\textsubscript{\textlt}satisfies\textsubscript{\textleq}Heap$\text{IsSorted}(numElems)"})

public class Heap implements PriorityQueue {
    private @Nonnull Comparable[][] elems;
    private int numElems;

    static class IsSorted implements Predicate {
        @SandBox
        public String evaluate (Object testObj, Object[] args) {
            Comparable[] elems = (Comparable[]) testObj;
            int numElems = (Integer) args[0];
            for (int i = 0; i < numElems; i++) {
                if (2*i+1 < numElems
                    && elems[i].compareTo(elems[2*i+1]) > 0)
                    return "not\textsubscript{\textlt}sorted";
                if (2*i+2 < numElems
                    && elems[i].compareTo(elems[2*i+2]) > 0)
                    return "not\textsubscript{\textlt}sorted";
            }
            return null;
        }
    }
}
Limitations of jpf-aprop

- The syntax for predicates is very restricted.
- The syntax feels adhoc, e.g. $a$ within $b \pm 2$.
- Syntax check is done at run-time.
- Cannot express $numElems <= elems.length$ (yet).
- No check for typos in identifiers.
- Surprising results: $true == false$ holds.
- Many things not implemented, e.g. functions (but no warning).
Combining JML and Java Pathfinder

Pathfinder:
+ Exhaustive model-checking.
+ Exact simulation of VM.
+ Can run any Java code.
  - No good Design By Contract specifications.

JML Runtime Assertion Checker:
+ Good Design by Contract Syntax.
+ Many features checkable at run time.
  - Can only find bugs at runtime.
  - Test cases have to be explicitly written.

Can we combine both programs?
Can we combine both programs? Yes!

Compiling:
- Set classpath to include Java Pathfinder runtime.
- Compile classes with jmlc.
- One can change compiler in ant script.

Running:
- Set classpath to include JML runtime and JML model classes.
- Classpath can be changed in Java Pathfinder script.
Demo
Conclusion

- Design by Contract with jpf-aprop is a good idea ... but it does not work.
- JML can be run inside of Java Pathfinder ... and it works!