Formal Methods for Java
Lecture 29: Java Pathfinder and Design By Contract

Jochen Hoenicke
Software Engineering
Albert-Ludwigs-University Freiburg

Feb 15, 2012
Testing Programs with Java Pathfinder
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

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

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

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

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

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

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

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

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

\[\text{Var} ::= \text{ID} \]
Example

@Invariant({"numElems\(\triangleright=0\)\n  ,
    "elems\(\triangleright\)satisfies\(\triangleright\)Heap$\text{IsSorted}(numElems)"})
public class  Heap implements  PriorityQueue {
  private  @NonNull Comparable[]  elems;
  private  int  numElems;

  @SandBox
  static class  IsSorted implements  Predicate {
    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\(\triangleright\)sorted";
        if  (2*I+2  <  numElems
            &&  elems[i].compareTo(elems[2*I+2])  >  0)
          return  "not\(\triangleright\)sorted";
      }
      return  null;
    }
  }
}
Limitations of jpf-aprop

- The syntax for predicates very restricted.
- The syntax feels adhoc, e.g. \textit{a within b +- 2}.
- Syntax check is done at run-time.
- Cannot express \textit{numElems <= elems.length} (yet).
- No check for typos in identifiers.
- Surprising results: \textit{true == false} holds.
- Many things not implemented, e.g. functions (but no warning).
Demo
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?
Compiling:

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

Running:

- jmlrac runs Java with a special classpath.
- If one sets classpath by hand one can run with java.
- Change Java Pathfinder scripts to include JML runtime and model classes.
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!