Formal Methods for Java

Lecture 28: Conclusion

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Feb 12, 2012

KeY and Procedures

In KeY, the default rule is to inline the procedures.

Advantages:

- No function contract needed.
- No separate proof for correctness of function needed.

But it has several disadvantages:

- Proof gets larger (especially important if proof is interactive).
- Proof has to be repeated for every function call.
- No recursive procedures possible.

The rule Use Operation Contract

The rule "Use Operation Contract" allows compositional proofs.
It opens three subgoals:

- Pre: Show that pre-condition holds (this includes class invariants).
- Post: Show that with the post-condition, the remaining program is correct.
- Exceptional Post: Show that if called method throws an exception, the remaining program is correct.

Note: Use Operation Contract cannot be used for the method you are just proving.

Proving Pure Recursive Functions

Unfortunately, KeY has no direct support for recursive functions.

An induction proof can work. Ingredients:

- A precondition *pre*,
- A postcondition *post*,
- A ranking function *rank*.

Show by induction over r:

```
\forall int x. (pre & rank < r) -> \< result = methodcall(x); \> post
```

Lecture Summary

Topics

Lecture	Topics
1	Introduction to JML and JLS
2–3	Operational Semantics
4–5	JML
6–10	Java Pathfinder
11	ESC/Java
12–14	Ownership/Friendship and Invariants
15–19	Jahob
20–22	Sequent Calculus
23–28	Dynamic Logic and Proving with KeY

Motivations

Quality

- Leads to better understood code.
- Different view point reveals bugs.
- Formal proof can rule out bugs entirely.

Productivity

- Error detection in early stages of development.
- Modular specifications allow reuse of components.
- Documentation, maintenance.
- Automatic test case generation.
- Clearer specification leads to better software.

Operational Semantics for Java

Idea: define transition system for Java

Definition (Transition System)

A transition system (TS) is a structure $TS = (Q, Act, \rightarrow)$, where

- Q is a set of states,
- Act a set of actions,
- $\rightarrow \subseteq Q \times Act \times Q$ the transition relation.

- Q reflects the current dynamic state (heap and local variables).
- Act is the executed code.
- Idea from: D. v. Oheimb, T. Nipkow, Machine-checking the Java specification: Proving type-safety, 1999

State of a Java Program

The state of a Java program consists of a flow component and valuations for local and global (heap) variables.

- $Q = Flow \times Heap \times Local$
- $Flow ::= Norm | Ret | Exc\langle\langle Address \rangle\rangle$
- $Heap = Address \to Class \times \mathrm{seq}\ Value$
- $Local = Identifier \to Value$
- $Value = \mathbb{Z}, Address \subseteq \mathbb{Z}$

A state is denoted as $q = (flow, heap, lcl)$, where $flow : Flow$, $heap : Heap$ and $lcl : Local$.

Rules of Operational Semantics

$$\frac{(Norm, heap, lcl) \xrightarrow{e_1 \triangleright v_1} q \quad q \xrightarrow{e_2 \triangleright v_2} q'}{(Norm, heap, lcl) \xrightarrow{e_1 * e_2 \triangleright (v_1 \cdot v_2) \bmod 2^{32}} q'}$$

$$\frac{(Norm, heap, lcl) \xrightarrow{st_1} q \quad q \xrightarrow{st_2} q'}{(Norm, heap, lcl) \xrightarrow{st_1 ; st_2} q'}$$

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} q \quad q \xrightarrow{bl_1} q'}{(Norm, heap, lcl) \xrightarrow{\textbf{if}(e)\ bl_1 \textbf{else} bl_2} q'}, \text{ where } v \neq 0$$

... and many more.

Rules for Exceptions

$$\frac{(Norm, heap, lcl) \xrightarrow{e \triangleright v} (Norm, heap', lcl')}{(Norm, heap, lcl) \xrightarrow{\textbf{throw } e} (Exc(v), heap', lcl')}$$

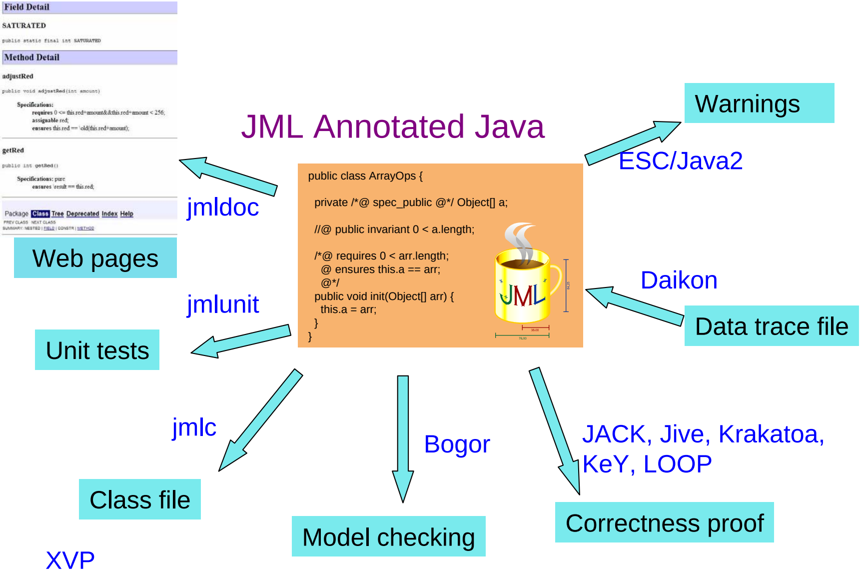
A null-pointer dereference works like a throw statement:

$$\frac{\begin{array}{c}(Norm, heap, lcl) \xrightarrow{e \triangleright 0} q' \\ q' \xrightarrow{\textbf{throw new } NullPointerException()} q''\end{array}}{(Norm, heap, lcl) \xrightarrow{e.fld \triangleright v} q''}, \text{where } v \text{ is some arbitrary value}$$

Propagating exceptions:

$$(flow, heap, lcl) \xrightarrow{\alpha} (flow, heap, lcl), \text{ where } flow \neq Norm$$

JML



Field Detail
SATURATED
public static final int SATURATED
Method Detail
adjustRed
public void adjustRed(int amount)
Specifications:
requires 0 <= this.red+amount && this.red+amount < 256;
assignable red;
ensures this.red == \old(this.red+amount);
getRed
public int getRed()
Specifications: pure
ensures \result == this.red;
Package Class Tree Deprecated Index Help
Web pages
jmldoc
JML Annotated Java
public class ArrayOps {
private /*@ spec_public @*/ Object[] a;
//@ public invariant 0 < a.length;
/*@ requires 0 < arr.length;
@ ensures this.a == arr;
@*/
public void init(Object[] arr) {
this.a = arr;
}
}
JML
Warnings
ESC/Java2
Daikon
Data trace file
jmlunit
Unit tests
jmlc
Class file
Bogor
Model checking
JACK, Jive, Krakatoa,
KeY, LOOP
Correctness proof
XVP

The Java Modelling Language (JML)

JML is a behavioral interface specification language (BISL) for Java

- Proposed by G. Leavens, A. Baker, C. Ruby:
  JML: A Notation for Detailed Design, 1999
- It combines ideas from two approaches:
  - Eiffel with it's built-in language for Design by Contract (DBC)
  - Larch/C++ a BISL for C++

Tools for JML

- `http://www.jmlspecs.org/`
- Release can be downloaded from
  `http://sourceforge.net/projects/jmlspecs/files`
- JML compiler (`jmlc`)
- JML runtime assertion checker (`jmlrac`)

External Tools:

- ESC/Java
- KeY
- and many more ...

Run-time Checking with jmlrac

Advantages of run-time checking:

- Easy to use.
- Supports a large sub-language of JML.
- No false warnings.

Disadvantages of run-time checking:

- Coverage only as good as test cases that are used.
- Does not prove absence of errors.

Model-checking with Java Pathfinder

Advantages of model-checking:

- Almost as easy as testing.
- More exhaustive than simple testing.

Disadvantages of model-checking:

- State explosion problem.
- Runtime vs. coverage.

Static Checking with ESC/Java

Advantages of static checking:

- Easy to use.
- No test cases needed.
- Better coverage than runtime checking.
- Can detect missing specification.

Disadvantages of static checking:

- Only a small subset of JML supported.
- Many spurious warnings (not complete).

Theorem Proving with KeY

Advantages of static checking:

- Prove of correctness.
- Both sound and complete (modulo Peano Axioms).

Disadvantages of static checking:

- Very difficult to use.
- Can require interactive proving.

Suggested order

1. Run-time checking, e.g. jmlrac and jmlunit.
2. Static checking, e.g. ESC/Java.
3. Model-checking, e.g. Java Pathfinder
4. Theorem proving, e.g. KeY.

Ensures that most bugs are already found before starting with theorem proving. Some prefer doing static checking before run-time checking (no test cases needed).