

Teaching program specification and verification using **JML** and **ESC/Java2**

Erik Poll

Digital Security

Radboud University Nijmegen

or...

Formal Methods for the Masses

Erik Poll

Digital Security

Radboud University Nijmegen

Teaching Formal Methods in 1900s

$$\begin{aligned} U_x &: P.m \{x \geq 0 \wedge (x=0 \vee b=0)\} \\ &\quad ; x := x+1 \\ &\quad ; \{x \geq 0 \wedge (x=1 \vee b=0)\} \\ &\quad ; \text{if } b > 0 \rightarrow \{b > 0 \wedge x=1\} V.s \\ &\quad \quad \square b=0 \rightarrow \{x \geq 0 \wedge (x=0 \vee b=0)\} V.m \\ &\quad \underline{f} \end{aligned}$$

$$\begin{aligned} P_x &: P.m \{x \geq 0 \wedge (x=0 \vee b=0)\} \\ &\quad \text{if } x > 0 \rightarrow \{b=0\} \text{ skip} \\ &\quad \square x=0 \rightarrow b := b+1 ; V.m ; P.s ; \{x=1 \wedge b > 0\} \\ &\quad \quad b := b-1 \{x=1 \wedge b \geq 0\} \\ &\quad \underline{f} \\ &\quad \{x > 0 \wedge (b=0 \vee x=1)\} \\ &\quad ; x := x-1 \\ &\quad \{x \geq 0 \wedge (b=0 \vee x=0)\} \\ &\quad ; V.m \end{aligned}$$

invariant versterkt tot $s=0 \vee (x=1 \wedge b > 0)$
zie EWD 734

Teaching Formal Methods in 1900s

Characteristics:

- toy programming language
 - toy programs
 - pencil & paper exercises
 - relegated to some Master course on Formal Methods
-
- NB "toy" not necessarily negative - toys can be fun to play with !

Teaching Formal Methods in 2000s



Characteristics:

- ~~toy programming language~~
- toy programs
- pencil & paper exercises
- relegated to some Master course on Formal Methods

a real one:
Java (and JML)

Teaching Formal Methods in 2000s

Characteristics:

- ~~toy programming language~~  a real one:
Java (and JML)
- toy programs
- ~~pencil & paper exercises~~  tool support:
ESC/Java2
- relegated to some Master course on formal methods

Teaching Formal Methods in 2000s

Characteristics:

- ~~toy programming language~~
- toy programs
- ~~pencil & paper exercises~~
- ~~relegated to some Master course on formal methods~~

a real one:
Java (and JML)

tool support:
ESC/Java2

in any
Bachelor course

JML



- Specification language for (sequential) Java
annotating code with **pre/postconditions, invariants,...**
- Initiative start by Gary Leavens, since joined by many others
- Various tools: **runtime assertion checking, program verification, ...**

```
public class Taxpayer{
    Taxpayer spouse;
    boolean isMarried;
    //@ invariant isMarried ==> spouse.spouse == this;

    //@ requires spouse != null;
    //@ ensures isMarried;
    void marry(Taxpayer spouse) {...}
```


ESC/Java(2)

- Extended Static Checker for Java

automated program checker/verifier by Rustan Leino & co at DEC

[Cormac Flanagan, Rustan Leino, Mark Lillibridge, Greg Nelson, James Saxe, and Raymie Stata, Extended static checking for Java, PLDI'2002]

- ESC/Java2

ongoing development initiative to keep tool alive & improved

[Joe Kiniry and David Cok, Uniting ESC/Java and JML, CASSIS'2004]

Course module in *applied FM*

- Two hour lecture on JML (incl. demo)
just pre- & postconditions and (class) invariants
- Afternoon exercise session
students are given sample code to annotate with JML and
check/debug with ESC/Java2
- Taught to variety of audiences
Undergraduate courses (Comp.Science and Information Science),
post-MSc course, tutorials, ..

Aims

- Making any BSc student aware of (the dangers of) implicit assumptions & design decisions
- Letting them experience that (FM-based) tools can help
- hopefully, piquing their interest in FM....
- We avoid the use of detailed functional specifications
hardly any postconditions, just preconditions and invariants are interesting enough, and easier to apply in the real world

Example implicit assumptions

```
class Taxpayer{  
  
    Taxpayer spouse;  
    boolean isMarried;  
    //@ invariant isMarried ==> spouse.spouse == this;  
  
    //@ requires spouse != null;  
    void marry(Taxpayer spouse) {...}
```

Example implicit assumptions

```
class Taxpayer{  
  
    Taxpayer spouse;  
    boolean isMarried;  
    //@ invariant isMarried ==> spouse.spouse == this;  
  
    /*@ requires spouse != null  
           && !isMarried  
           && spouse != this;   @*/  
    void marry(Taxpayer spouse) {...}
```

Experiences

- Students enjoy playing with the tools
- Suitable for a wide variety of audiences
- Little marking effort, but help & hints during practical session needed for some
- Letting students work in pairs, or discuss, prevents “mindless” experiments
- Caveat: letting students use ESC/Java2 **on code they write themselves** is **not** an option, except in carefully constrained setting

Experiences

For most students, it is also a first experience in using

- **propositional logic** (outside a logic course)
- a (behind the scenes) **automated theorem prover**

Eg in

```
class Amount{  
    int euros, cents;
```

```
is //@ invariant cents < 0 ==> ! (euros > 0);  
    //@ invariant cents > 0 ==> ! (euros < 0);
```

equivalent with

```
    //@ invariant ! (cents < 0 && euros > 0);  
    //@ invariant ! (cents < 0 && euros > 0); ?
```

Some related possibilities

- **Spec# tool** for **C#**

[Rustan Leino and Rosemary Monahan, Automatic verification of textbook programs that use comprehensions, FTfJP'07]

- **KeY tool** for **Java or simple while language**

Talk in session today after lunch

- **Krakatoa tool** for **Java**
- **SparkAda** for **Ada**

Conclusions

- Time for Formal Methods to come out of the ghetto...
 - make a guest appearance in general Bachelor courses, not only be subject in specialised Master courses
 - tools are crucial for doing this

Conclusions

- Time for Formal Methods to come out of the ghetto...
 - make a guest appearance in general Bachelor courses, not only be subject in specialised Master courses
 - tools are crucial for doing this
- JML and ESC/Java2 provide one way to do this
 - requires minimal Java knowledge
 - requires no knowledge of formal methods
 - beyond very basic propositional logic

Questions?

Feel free to download & reuse

<http://www.cs.ru.nl/~erikpoll/Teaching/JML>