Beyond Hoare Logic Assertions

Advanced Specification and Verification with JML and ESC/Java2

Joseph Kiniry
Acknowledgements

- original DEC/Compaq SRC team

- ESC/Java2 collaborators
  - David Cok (Kodak R&D), Cesare Tinelli (Univ. of Iowa), and Aleksey Schubert (Univ. of Warsaw)

- JML community
  - led by Gary Leavens and major participants Yoonsik Cheon, Curtis Clifton, Todd Millstein, and Patrice Chalin

- verification community
  - Peter Müller, Marieke Huisman, Joachim van den Berg

- SoS Group at Radboud University Nijmegen
  - Erik Poll, Bart Jacobs, Cees-Bart Breunesse, Martijn Oostdijk, Martijn Warnier, Wolter Pieters, Ichiro Hasuo
JML: The Java Modeling Language

- a behavioral interface specification language
- syntax and semantics are very close to Java
- annotations written as comments in code
- JML is a rich language (some say too rich)
  - core constructs include preconditions, postconditions, invariants, etc.
- one language used for documentation, runtime checking, and formal verification
Deconstructing JML

- JML is a research specification language
- if you need a new keyword, just ask for it
- a very rich specification language, but...
- JML’s size is overwhelming to new users and interested researchers
- thus, JML is being “deconstructed” to document which parts are core with a fixed semantics across all tools
- JML level 0 (aka JML₀), JML∥, etc.
JML is an Open Cooperative Project

<table>
<thead>
<tr>
<th>Leavens's group at Iowa State</th>
<th>Chalin's group at Concordia, Montreal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cheon's group at UTEP</td>
<td>David Cok</td>
</tr>
<tr>
<td>SpEx and SAnToS group at Kansas State (Hatcliff, Robby)</td>
<td>KeY Project at Univ. Karlsruhe</td>
</tr>
<tr>
<td>Dwyer at U. Nebraska</td>
<td>Edwards's group at Virginia Tech.</td>
</tr>
<tr>
<td>Flanagan at U. California Santa Cruz</td>
<td>Müller's group at ETH Zürich</td>
</tr>
<tr>
<td>Jacob's SoS group at Nijmegen (Poll,...)</td>
<td>Poetzsch-Heffter's group at Kaiserslautern</td>
</tr>
<tr>
<td>Kiniry at U. Dublin</td>
<td>Logical group at INRIA Futurs &amp; Université Paris-Sud</td>
</tr>
<tr>
<td>Ernst's group, and others, at MIT</td>
<td>Huisman’s group at INRIA Sophia-Antipolis</td>
</tr>
</tbody>
</table>
ESC/Java2 is an extended static checker

- based upon DEC/Compaq SRC ESC/Java
- operates on JML-annotated Java code
- behaves like a compiler
  - error messages similar to javac & gcc
  - completely automated
  - hides enormous complexity from user
We Write more than just Hoare Assertions

- four different method “outcomes” of a method invocation
  - normal termination
  - exceptional termination
  - non-termination
  - JVM Error
- each of these outcomes has a specific specification construct
public class Point {
    private /*@ spec_public @*/ int x, y;
    //@ requires x >= 0;
    @ assignable y;
    @ ensures delta > 0 && y == \old(y + delta) &&
    @       \result == \old(y);
    @ signals_only IllegalArgumentException;
    @ signals (IllegalArgumentException e)
    @       delta < 0 && e != null && y == \old(y);
    @*/
    public int moveUp(int delta);
}
Case Analysis with “also”

```java
/*@ public normal_behavior
@   requires x >= 0 && delta > 0;
@   assignable y;
@   ensures y == \old(y + delta) &&
@       \result == \old(y);
@ also
@ public exceptional_behavior
@   requires x >= 0 && delta < 0;
@   assignable \nothing;
@   signals_only IllegalArgumentException;
@*/
public int moveUp(int delta);
```
Desugaring

/*@ public behavior
@   requires (x >= 0 && delta > 0) ||
@     (x >= 0 && delta < 0);
@   assignable y;
@   ensures (\old(x >= 0 && delta > 0) ==> 
@     (y == \old(y + delta) &&
@      \result == \old(y))) &&
@     (\old(x >= 0 && delta < 0) ==> false);
@   signals_only IllegalArgumentException;
@   signals (Exception e)
@     (\old(x >= 0 && delta > 0) ==> false) &&
@     (\old(x >= 0 && delta < 0) ==> true &&
@      \not_assigned(y));
@*/

public int moveUp(int delta);
public class Super {
    /*@ public behavior B1 @*/
    public T m();
}

public class Sub extends Super {
    /*@ also public behavior B2 @*/
    public T m();
}

Completed Specification of Subclass

public class Sub extends Super {
    /*@ public behavior B1
     @ also public behavior B2 @*/
    public T m();
}
What Gets Inherited?

- specifications of instance methods (with “also”)
- fields and associated specifications
- instance invariants and constraints
- forces behavioral subtyping
  - subtypes are behavioral subtypes
  - if try to avoid it, you get an unimplementable specification
  - thus, need to underspecify supertypes
“Advanced” JML
Frame Axioms

- necessary for modular reasoning
- default is too loose to be useful
- originally only focused on outbound references (assignable/modifies clauses)
- now must specify inbound references for reasoning about concurrent programs

- rely-guarantee style
Data Groups

- data groups are used in frame axioms
- they are used to define
  - sets of related fields
  - fields with interdependencies
- fields are concrete or specification-only
- built-in datagroup is called objectState
Ghost Variables

* specification-only variables from which one can read and to which one can write
* typically used to represent
  * simplified ownership
  * object state
  * element and key types
  * specification-only constants
  * explicit specification variables
Model Variables

- used to model
  - abstractions of classes and objects
  - generic mathematical abstractions
  - implicit specification variables
- can be refined to concrete representations
  - functionally
  - relationally
Other Model Constructs

- specification-only classes, methods, and programs with optional semantics
- implementation
- sometimes used for fixed base specification for reference and refinement
- helper methods to shorten or clarity complex specifications
Current Research

- integration of multiple theorem provers
- support multiple logics concurrently
- new (sub)logics for new (sub)problems
- identifying and specifying security properties
- new constructs like functional, immutable, several different kinds of purity, atomicity