Soundness and Completeness Warnings in ESC/Java2

Joe Kiniry, Alan Morkan, and Barry Denby

presented by David Cok

Systems Research Group
School of Computer Science and Informatics
University College Dublin

Mobius

ESC/Java2

- by design, neither sound nor complete
- popularity of similar tools growing as (lightweight) static analysis tools become more widely used (e.g, Eclipse & FindBugs)
- developer comprehension and confidence are paramount (*program* safety via *programmer* safety)
- complaints from “soundationalists” drives a desire for “tool honesty” and disclosure

Checking Limitations

- a fast, automatic tool must “cheat”
  - many scientific and engineering trade-offs
- several sources of soundness and completeness problems
  - Java and JML semantic incompleteness
  - unsound verification methodology
  - limitations of dependent tools (provers)
  - *problems with user specifications*

Requirements on New Warning Subsystem

- contextually warn the user (in detail) about potential soundness and incompleteness
  - e.g., must take into account the program code, annotations, execution path in tool, and theorem prover in use
- provide “tunable” feedback so as to not overwhelm the user with warnings
- be itself sound and complete
  - have no false positives or negatives

Detection Methodology

- manually analyze and classify all soundness and completeness issues
- define a type- and annotation-aware AST pattern match for each issue
- each issue implemented as a single “smart” visitor pattern (separation of concerns)
- customized warning levels, messages, and criticality per issue

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

Incompleteness Warning:
Simplify cannot deal with
large integer values.

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

Soundness Warning:
Exposed field may be used
in other class invariants.

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

Soundness Warning:
Heuristics for class invariant
analysis are not sound.

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Incompleteness Warning:
Semantics for floating
point numbers.

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Example Warnings

```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Incompleteness Warning:
Semantics for bitwise OR
are not complete.

Example Warnings

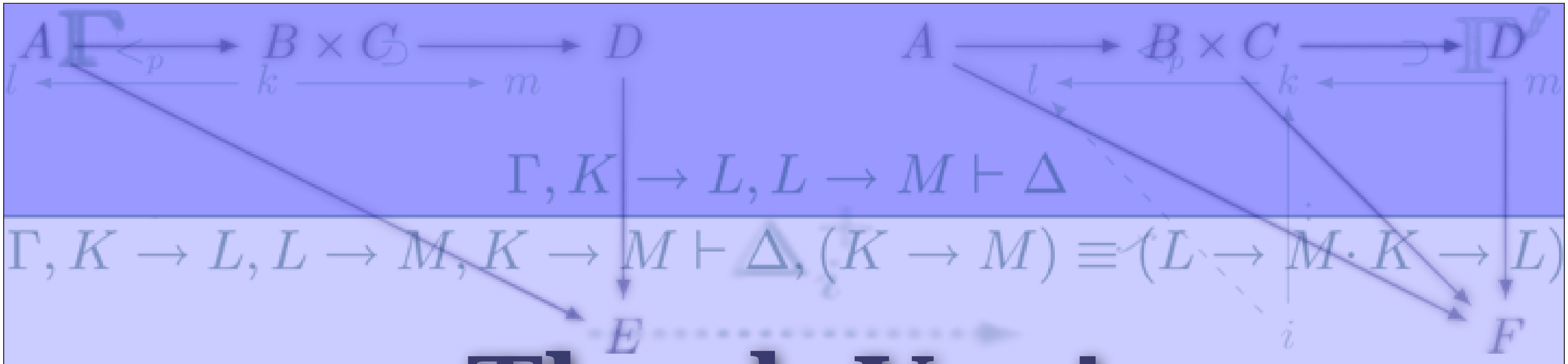
```
public class CreditCard {
  //@ invariant balance <= maxCredit;
  public double balance = 0, maxCredit = 100000;
  public static int STANDARD = 1, SILVER = 2, GOLD = 4;
  private int accountType = 1;

  //@ ensures accountType == 4;
  public void goldCard()
    { accountType = 4; }

  //@ requires cost < (maxCredit - balance);
  //@ ensures \result == \old(balance + cost);
  public double purchase(double cost)
    { return balance + cost; }

  //@ ensures (accountType == GOLD ? 1 : 0);
  public /*@ pure @*/ boolean isGoldCard()
    { return accountType | GOLD; }
}
```

Benefits and Drawbacks

- increase user awareness of tool limitations
  - no more “creeping toward functional verification”
- increase in user confidence
- possible excess of user feedback
  - leads to user confusion and frustration
- text-based warnings need refinement
  - prioritization, graphical feedback, etc.

Future Work

- finish implementation
  - only for default code paths
    - strongest postcondition calculus, loop unrolling and safe loops, simplify
- integration with the ESC/Java2 Eclipse plugin and Mobius Tool
- use theorem proving during analysis
- automatic visitor generation



Thank You!

Questions and Comments?

Extra Slides for Questions

Warning Levels

- three options for warnings
    - standard warning mode
    - verbose warning mode
    - no warnings mode

Examining the AST: The Precondition

```
   MethodDecl
      |
   ModifierPragmaVector
   /
BinaryExpr
/     \
  .....     BinaryExpr (<)
      /        \
   FieldAccess    ParenExpr
      (cost)         |
               BinaryExpr(-)
               /     \
            FieldAccess FieldAccess
            (maxCredit)    (balance)
```

The Postcondition

```
   MethodDecl
      |
   ModifierPragmaVector
   /
   ExprModifierPragma
      |
   BinaryExpr (==)
   /     \
ResExpr     BinaryExpr(+)
         /        \
      NaryExpr    FieldAcccess
      (\old)            (cost)
         |
      FieldAccess
      (balance)
```

The Invariant

```
   .......
      |
   BinaryExpr(<=)
   /        \
FieldAccess    FieldAccess
(balance)         (maxCredit)
```