ESC /Java 2
Extended Static Checking / Java 2

Melissa J Fernandes
Charan Cherukuri
Srikanth Vadada
CSE 6323, Spring 2010, 29th April 2010

Agenda

- Introduction
- Tool Architecture
- Discovering Errors with ESC/Java 2
- Tool Demo – Stack Example
- ESC/Java 2 Features
- Conclusion
- Question & Answers
**History of Extended Static Checking**

- **1950 - 1960**
  - Focus on Modern Programming Languages (FORTRAN, LISP, COBOL)

- **1967 - 1978**
  - Focus on Establishing Fundamental Paradigms (System, OO, Logic)

- **1980 - 1984**
  - Focus on Re-Use, Performance (C++)

- **1990 - 1997 and **
  - Internet Age & Rapid Application Development (Java, PHP, Ruby,...)

- **1997 - Till Date**
  - Focus on Security and Reliability Verification to the Languages
  - Birth of Extended Static Checking
  - Pioneering effort in the use of Static Program Analysis & Verification Methods
  - ESC for Modula in 1995
  - ESC / Java in 1997 from DEC
  - Renaissance of ESC/Java 2 in 2002 as an Industrial Strength Tool

---

**Classes of Checkers**

- **Static Checking**
  - Type Checking
  - Extended Static Checking
  - Program Verification

- **Dynamic Checking**

- **Coverage vs Effort?**

---

*Fig. Source: Extended Static Checking: a Ten-Year Perspective by K. Rustan M. Leino*
Theoretical Foundation of Extended Static Checking

- Deciding which errors to Check
  - Unsoundness – Missing Errors
  - Checks 3 Types of Errors
    - Runtime Checks (null dereferences, array index bounds errors…)
    - Synchronization Errors (race conditions, deadlocks)
    - Violation of Program Annotations (meeting invariants, preconditions…)

- Defining Formal Semantics for Modern Languages
  - Guarded Command Languages

- Using a Theorem Prover
  - Should be Automated – Else Learning Curve High
  - Produce Counter Examples – Reason for Error
  - Should be fast – Checker used many times during Development

- Producing meaningful Warning Messages

- Program Annotations

User's View

Program (Java) with JML Specifications

```java
public class Bag {
    private int[] a;
    private int n;
    //@ invariant 0 <= n && n <= a.length;
    public Bag(int[] initialElements) {
        n = initialElements.length;
        a = new int[n];
        System.arraycopy(initialElements, 0, a, 0, n);
    }
    ..........
    ..........
}
```

Bag.java:18:
Array index possibly too large
**Tool Architecture**

Annotated Java Program (Java Code + JML Spec.)

- Translator
- Verification Condition
- Automatic Theorem Prover
  - Valid
  - Resource exhausted
- Counterexample Context
- Post Processor
- Warning Messages

University of Texas at Arlington

---

**Tool Architecture Detail**

Annotated Java Program (Java Code + JML Spec.)

- Translator
  - Sugared Command
  - Primitive Command
  - Passive Command
- Verification Condition
- Automatic Theorem Prover
- Counterexample Context
- Post Processor
- Warning Messages

University of Texas at Arlington
Annotation Language

- **Simple**
  - non_null

- **Method annotations**
  - requires E;
  - modifies w;
  - ensures P;
  - ....

- **Object invariants**
  - invariant E;
  - ........
  - ........
  - ............
**Sugared Commands**

\[ S, T ::= \]
- assert E
- assume E
- \( x = E \)
- raise
- \( S ; T \)
- \( S ! T \)
- \( S [] T \)
- loop \{inv E\} S \rightarrow T end
- call \( x = t.m(E) \)
- ...
**Primitive Commands**

- \( \mathbf{v} = \text{o.f} \); .......at Line 27
  - check Null,27,\( \text{o!}=\text{null} \);
  - \( \mathbf{v} = \text{select(f,o)} \);
  - assert (label Null@27:0!=null);
  - or
  - assume \( \text{o!=null} \);

**Passive Commands**

- \( S,T ::= \) assert \( E \)
  - assume \( E \)
  - \( \mathbf{x} = \mathbf{E} \)
  - raise
  - \( S \; ; \; T \)
  - \( S \; ! \; T \)
  - \( S \; || \; T \)
**Passive Commands**

```c
if (x < 0) { x = -x; }
/* @ assert x >= 0; */

( assume x_0 < 0; assume x_1 = -x_0;
  assume x_2 = x_1
[] assume !(x_0 < 0);
  assume x_2 = x_0
);
assert x_2 >= 0
```

**Verification Condition**

- Universal background predicate (UBP)
- Type-specific background predicate (TSBP)
- Verification Condition Generation
  Uses UBP & TSBP & previous stages
```java
class T {
    static int abs(int x) {
        if (x <= 0) {
            x = -x;
        }
        @ assert x >= 0;
    }
```

**Theorem Prover: “Simplify”**
Counter examples and Warnings

Counterexample:
labels: ([IndexTooBig@26.5] vc.Bag.add.20.2 |trace.Then^0,21.23])
context:
(AND
 (NEQ |tmp1!a:23.23| null)
 (NEQ this null)
 (EQ |alloc@pre| alloc)
 (EQ |tmp4!n:26.6| 0)
 (<= alloc (vAllocTime |tmp3!a:26.4|))
)

Bag: add(int) ...
-----------------------------------------------
Bag.java:26: Warning: Array index possibly too large (IndexTooBig)
 a[n] = x;
-----------------------------------------------
Execution trace information:
Executed then branch in "Bag.java", line 21, col 23.

Some Errors that ESC / Java 2 discovers

- Pre Condition
- Post Condition
- Invariant
- Initially
- Index Negative
- Index Too Big
- Null
- ....
- .....
Some Runtime Errors Detected by ESC/Java 2

✓ Index Negative
   Issued when an array index < 0

✓ Index Too Big
   Issued when an array index >= Array Length

✓ Null
   Issued when there is a possibility of NullPointerException

Some Annotation Violations Detected by ESC/Java 2

✓ Pre and Post
   Issued in response to user-written preconditions (requires), post-conditions (ensures….)

✓ Invariant
   Invariant clause generate additional post-conditions for every method. If they do not hold, appropriate warnings are generated

✓ Initially
   Initially clause is a post-condition for every constructor
**Modular Reasoning**

✓ ESC/Java2 reasons about every method individually

```java
public class ModularReasoning {
    int[] b;
    ModularReasoning()
    b = new int[20];
    public void m()
    b[0] = 2;
}
```

Warns that `b[0]` may be a null dereference here, even though you can see that it won’t be.

**DEMO**

( Stack Example )
Unsound and Incomplete (1 / 3)

- **Unsound**

✓ Misses errors that are actually present in the program

- **Incomplete**

✓ Warns of Potential Errors when it is impossible for these to occur

ESC / Java2 not Sound and Complete

✓ Affects Complexity of Annotation Language

✓ Tradeoff to make it Cost effective

Unsound and Incomplete (2 / 3)

**Example 1**

```java
int[] array = new int[10];
for(int i = 0; i < 20; i++)
    array[i] = i;
```

ArrayIndex out of Bound - Error occurs but will not be caught by Tool

Reason: Tool does not consider all Possible Iterations

**Example 2**

```java
int i = 32000;
i = i * i;
```

Arithmetic Overflow - Error occurs but will not be caught by Tool

Reason: Assumes that (i) is of unlimited magnitude
Unsound and Incomplete (3 / 3)

```java
public void n()
{
    char c;
    c = "ESC/Java2".charAt(2);
    // assert c=='C';
}
```

Semantics for String Operations are weak.

ESC/Java 2 and Spec# Systems

<table>
<thead>
<tr>
<th></th>
<th>ESC/ Java2 Tool</th>
<th>Spec # Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language</td>
<td>Java</td>
<td>C#</td>
</tr>
<tr>
<td>Annotation Language</td>
<td>JML</td>
<td>Spec #</td>
</tr>
<tr>
<td>Automatic Theorem Prover</td>
<td>Simplify</td>
<td>Z3</td>
</tr>
<tr>
<td>Verifier</td>
<td>ESC/Java2</td>
<td>Boogie</td>
</tr>
</tbody>
</table>
**Competing Technologies & Tools (1/2)**

- **FindBugs**
  - Finds Bugs in Java
  - Static Checker
  - Detects Synchronization Problems
  - Plug-ins for Eclipse, NetBeans

- **JLint**
  - Static Checker
  - C, C++, Java

---

**Competing Technologies & Tools (2/2)**

<table>
<thead>
<tr>
<th>Bug Category</th>
<th>Examples</th>
<th>ESC/Java2</th>
<th>FindBugs</th>
<th>JLint</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Null dereference</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Possible deadlock, race</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Possible unexpected exception</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Array</td>
<td>Length may be less than zero</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Mathematics</td>
<td>Division by zero</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Conditional, loop</td>
<td>Unreachable code</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I/O stream</td>
<td>Stream not closed on all paths</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Unused or duplicate statement</td>
<td>Unused local variable</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

✓ Bug Category

*Example only*

**Source:** A Comparison of Bug Finding Tools for Java by Nick Rutar, Christian B. Almazan, Jeffrey S. Foster
Limitations & Future Challenges

- **Limitations**
  - Iterates through Loops only once
  - Limitations on checking Arithmetic Overflow
  - Does not check for Non Functional Properties
  - Does not check Functional Properties not specified by User
  - Feasible only on Small Programs
  - Writing Annotations is labor Intensive

- **Future Challenges**
  - Reduce Annotation Burden
    - Perform Non-Modular Checking
    - Develop Annotation Assistants (Houdini is for ESC/Java2)
  - Teaching JML & ESC/Java2 with Programming Languages

How ESC/Java2 is Useful

- Possible run-time errors can be identified at compile time.
- Assumptions made by the programmer are made explicit.
- JML annotations provide documentation.
Our Opinion on the Tool

- Likes
  - Uses JML which is easy to understand
  - Integrated into Eclipse

- Dislikes
  - Counter example difficult to decode
  - Manuals for Installing & Configuring Tool is not comprehensive

Things learnt from the Tool

- Thinking in terms of Specifications while programming
- Improving Quality of Code
- Thinking from both perspectives
  - Client
  - Supplier
Summary

- Purpose of Extended Static Checking
- ESC/Java2 Tool Architecture
- Errors detected by ESC/Java2
- Features of ESC/Java2

References (1/2)

3. Extended Static Checking for Java by Cormac Flanagan, K. Rustan M. Leino, Mark Lillibridge, Greg Nelson, James B. Saxe, Raymie Stata
4. ESC/Java2: Uniting ESC/Java and JML by David R. Cok, Joseph R. Kiniry
5. An overview of JML tools and applications by Lilian Burdy, Yoonsik Cheon, David R. Cok, Michael D. Ernst, Joseph R. Kiniry, Gary T. Leavens, K. Rustan M. Leino, Erik Poll
8. Soundness and completeness warnings in ESC/Java2 by Joseph R. Kiniry, Alan E. Morkan, Barry Denby
9. Improving the Quality of Web-based Enterprise Applications with Extended Static Checking: A Case Study by Fredric Rioux, Patrice Chalin
10. ESC/Java2 as a Tool to Ensure Security in the Source Code of Java Applications by Aleksy Schubert and Jacek Chrzaszczyz
References (2 / 2)
13. ESC/Java2 Implementation Notes by David R. Cok, Joseph R. Kiniry, Dermot Cochran

Questions?