Distributed, Multi-threaded Verification of Java Programs

Perry R. James, Patrice Chalin,
Leveda Giannas, George Karabotsos

Dependable Software Research Group
Department of Computer Science and Software Engineering
Concordia University, Montreal, Canada
{perry, chalin, leveda, george}@dsrg.org

10 November 2008 / SAVCBS’08
Extended Static Checking

- More than type checking
- Neither sound nor complete
- Able to detect many errors
  - contract violations
  - uncaught runtime exception
- fully automatic
- compiler-like interface
Example: A violated contract

```java
//@ requires b != 0;
//@ ensures \result == (a == b);
public static boolean areEqual(int a, int b) {
    return (a - b) == 0;
}

public static boolean test(int c, int d) {
    return areEqual(c, d);
}
```

Problems

14 errors, 0 warnings, 0 others

- Possible assertion failure - 1 - (Precondition).
- Possible assertion failure - 1 - (Precondition). (Contract_test_1)
Example: An invalid loop invariant

```java
package cube;

import java.util.concurrent.TimeUnit;

public class Cube {
    public static void main(String[] args) {
        int n = 100000000;
        long startTime = System.currentTimeMillis();
        long result = length(n);
        long endTime = System.currentTimeMillis();
        System.out.println("Length of "+n+" is "+result);
        System.out.println("Time spent:");
        System.out.println((endTime - startTime)/1000.0 +"ms");
    }

    public static long length(int len) {
        int n = len;
        int result = 0;
        int w = 0;
        while (w > 0) {
            int z = w;
            n = n + z;
            w = w + z;
        }
        return n;
    }
}
```
JML4: An IVE for JML

- Integrate support for JML into Eclipse
  - scanning, parsing, type checking, flow analysis, code generation
- Process inline & out-of-band JML
- Enhance non-null type system
- JML errors look like others
- Ensure hooks for RAC, ESC, FPV

Compiler phases:
- JML+Java (*.java)
- Diet Parsing
  - diet AST
- (Full) Parsing
  - AST
- Type Checking
- Flow Analysis
- External Spec Merge
- Static Verification (incl. ESC)
- AST (+ proof status)
- Code Generation (incl. RAC code)
- * .class

References:
- Eclipse
- JML4: An IVE for JML
- Extended Static Checking
- Some examples

Incorporate support for JML into Eclipse by integrating scanning, parsing, type checking, flow analysis, and code generation. Process both inline and out-of-band JML. Enhance the non-null type system and ensure that JML errors appear similar to others. Ensure hooks for RAC, ESC, and FPV.
ESC4: ESC in JML4

- Static analysis just before code generation
- Input is a “good” AST
- Multiple forms can be performed

Compiler phases
Overview of ESC4

- Converts AST to CFG (similar to GCs)
- Converts CFG to VC
- Uses a variety of strategies to discharge
- Reports unprovable assertions as problems
- Adds proof status to AST

Dataflow in ESC4
Generating VCs

AST converted to VC using techniques by Barnett & Leino

- Translate AST to an acyclic CFG
- Replace method calls with specs
- Remove side effects with DSA
- Produce single VC per method with wp
Discharging VCs

Configurable Proof Coordinator to discharge VCs

- Different prover strategies easily implemented
- First try to prove entire VC with Simplify
- If fails,
  - break into sub-VCs
  - try with Simplify, CVC3, and Isabelle/HOL

\(^a\)as an ATP!
Prover back-end

- Prover Coordinator used to discharge VCs
  - gets strategy from a factory
  - factory governed by compiler options
- Adapters hide communication with provers
- Visitors to pretty print the VCs for ATPs
- VC proof-status cache persisted

ESC4’s prover back-end
Benefits of multiple provers

Some enhancements in ESC4 include

- Non-linear arithmetic
- Numeric quantifiers
- First-class quantifiers
- Full power of Isabelle/HOL

Using multiple provers comes at a price... ESC is very useful, but CPU intensive
Benefits of multiple provers

Some enhancements in ESC4 include
- Non-linear arithmetic
- Numeric quantifiers
- First-class quantifiers
- Full power of Isabelle/HOL

Using multiple provers comes at a price... ESC is very useful, but CPU intensive
ESC is very useful, but CPU intensive

**Multi-threaded VC Generation**
- Analyze methods in parallel
- Analysis for a method is independent of that for any others

**Distributed Discharging of VCs**
- Use non-local resources to reduce time
- Easy to include new proof strategies in ESC4
Multi-threaded VC Generation

Process methods in parallel
- Package processing of method as work item
- Add it to a thread pool’s task list
- Join point to wait until all finished

Process .java files in parallel
- Eclipse 3.4 JDT has concurrent compilation of source files
- ESC4 built atop JML4, which is built atop JDT
- Just have to make sure JML4 & ESC4 are thread safe
Distributed VC Processing

- Proof Coordinator made it easy
- Distributed strategy
  - sends pieces off to be verified
  - makes use of existing code to call provers
Deployment Scenarios

1. Prove whole VC remotely
   - Offloads work of Prover Coordinator for entire method
   - Sends VC to remote server processing

2. Prove sub-VCs remotely
   - Splits VC into sub-VCs & sends ‘em off for remote discharging
   - Uses remote services to discharge the sub-VCs in parallel

3. Doubly Remote Prover Coordinator
   - Combines the two above
   - Remote Prover Coordinator
   - Sub-VCs discharged on remote services
Deployment Scenarios

1. Prove whole VC remotely
   - Offloads work of Prover Coordinator for entire method
   - Sends VC to remote server processing

2. Prove sub-VCs remotely
   - Splits VC into sub-VCs & sends ’em off for remote discharging
   - Uses remote services to discharge the sub-VCs in parallel

3. Doubly Remote Prover Coordinator
   - Combines the two above
   - Remote Prover Coordinator
   - Sub-VCs discharged on remote services
Deployment Scenarios

1. Prove whole VC remotely
   - Offloads work of Prover Coordinator for entire method
   - Sends VC to remote server processing

2. Prove sub-VCs remotely
   - Splits VC into sub-VCs & sends ’em off for remote discharging
   - Uses remote services to discharge the sub-VCs in parallel

3. Doubly Remote Prover Coordinator
   - Combines the two above
   - Remote Prover Coordinator
   - Sub-VCs discharged on remote services
Validation - Setup

- 1 Java class with 51 methods \(\leadsto\) 235 sub-VCs
- Doubly Remote Prover Coordinator local & remote
- ESC4 run on 2.4 GHz Pentium 4
- Remote Prover Coordinator on a 3.0 GHz Pentium 4
- Provers hosted on 2.4 GHz Quad-core Xeon processors.
Validation - Prover Invocations

<table>
<thead>
<tr>
<th>Prover</th>
<th>No. VCs</th>
<th>No. Proved</th>
<th>(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplify</td>
<td>235</td>
<td>193</td>
<td>82</td>
</tr>
<tr>
<td>CVC3</td>
<td>42</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Negation(^a)</td>
<td>42</td>
<td>23</td>
<td>55(^b)</td>
</tr>
<tr>
<td>Isabelle</td>
<td>19</td>
<td>13</td>
<td>68</td>
</tr>
<tr>
<td>failed</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) Simplify used to prove the negation of the VC  
\(^b\) 80% of all false
Validation - Timing Results

<table>
<thead>
<tr>
<th>No. servers</th>
<th>No. cores</th>
<th>Time (s) with Prover Coordinator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>local</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>26.6</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>16.9</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>12.8</td>
</tr>
</tbody>
</table>

Running all locally took 72 s
Little difference between remote or local Prover Coordinator

\[ t = 7.4 + \frac{76.0}{n} \]
Next Steps for ESC4

ESC4 is a quickly evolving research platform.

- It can do a few things that ESC/Java2 cannot.
- ESC/Java2 can do much more than it.

To close this gap

we continue to flesh out JML4 and ESC4
to more fully support Java and JML
Next Steps for ESC4

Easy performance gains

- improve interface to the theorem provers
- add load balancing
- caching of distributed proof results
Next Steps for ESC4

Further validation

- more case studies
- gather more timings
Conclusions

- ESC4 exploits several levels of parallelism
  - compilation unit & methods
  - sub-VCs

- using
  - local multi-threading
  - remote processing resources

- over 90% can be parallelized
Thank You!