Comparative Assessment of Testing and Model Checking Using Program Mutation

Jeremy S. Bradbury
Faculty of Science • University of Ontario Institute of Technology
Oshawa • Ontario • Canada
jeremy.bradbury@uoit.ca

James R. Cordy, Juergen Dingel
School of Computing • Queen’s University
Kingston • Ontario • Canada
{cordy, dingel}@cs.queensu.ca

Mutation 2007 • Sept. 10-11, 2007
...humans are quickly overwhelmed by concurrency and find it much more difficult to reason about concurrent than sequential code. Even careful people miss possible interleavings...

- Herb Sutter & James Larus, Microsoft [SL05]
In the future applications will need to be **concurrent** to fully exploit CPU throughput gains [Sut05]

How can we increase our confidence in the correctness of concurrent programs?
Research Goals

1. To compare the effectiveness and efficiency of testing and model checking tools using mutation

2. To better understand any complementary relationship that might exist between testing and model checking
Our Approach

- Conduct a **controlled experiment** to evaluate the ability of testing and model checking
- We use **mutation** to generate the faulty **concurrent** programs required for our experiments
- Mutation [DLS78] traditionally used within the **sequential testing** community
  - evaluate the effectiveness of test suites

Experimental Setup

Approach
Selection

Quality Artifacts

Original Example Program

Mutant Operators

M1 M1 ... Mn

Testing with ConTest

Mutant Example Program

Model Checking with Java PathFinder

Collection and Display of Results

Comparison Results Database

Mutation 2007 • September 10-11, 2007
© J.S. Bradbury, J.R. Cordy, J. Dingel • 2007 • 8
Concurrency Testing with IBM’s ConTest

Run Test
1. Rerun Test with heuristically generated interleaving
2. Record interleaving
3. Update Coverage

Check Results
Correct
Problem

Check Coverage Target
Reached
Not Reached

Fix Bug
Rerun test using replay

Finish


Mutation 2007 • September 10-11, 2007
© J.S. Bradbury, J.R. Cordy, J. Dingel • 2007 • 9
Model Checking with Java PathFinder (JPF)

- Model checking **exhaustively** searches the entire state space of a program (i.e., all interleavings)
- Allows for the analysis of **assertions** and **deadlock** detection

Model Checking with Java PathFinder (JPF)

- Detailed view of JPF architecture
Experimental Setup

Approach Selection

Example Program Selection
Example Programs

• Ticket Order Simulation
  – Simulates multiple agents selling tickets for a flight
• Linked List
  – Involves storing data in a concurrent linked list (data structure)
• Buffered Writer
  – Two different types of writer threads are updated a buffer that is being read by a reader thread
• Account Management System
  – Manages a series of transactions between a number of accounts
### Metrics for the Example Programs

<table>
<thead>
<tr>
<th>Example Program</th>
<th>Lines of Code</th>
<th>Statements</th>
<th>Critical Regions</th>
<th>Critical Region Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>TicketsOrderSim</td>
<td>75</td>
<td>21</td>
<td>1</td>
<td>6 (28.5%)</td>
</tr>
<tr>
<td>LinkedList</td>
<td>303</td>
<td>70</td>
<td>2</td>
<td>4 (5.7%)</td>
</tr>
<tr>
<td>BufWriter</td>
<td>213</td>
<td>55</td>
<td>3</td>
<td>20 (36.4%)</td>
</tr>
<tr>
<td>AccountProgram</td>
<td>145</td>
<td>40</td>
<td>5</td>
<td>8 (20%)</td>
</tr>
</tbody>
</table>
Experimental Setup

Approach
Selection
Example
Program
Selection

Mutation
Selection

ConMAN Operators
ASK ASTK SPCR

Quality Artifacts
Original Example Program
Quality Artifacts

Testing with ConTest
Mutant Example Program
Model Checking with Java PathFinder

Collection and Display of Results

Comparison Results Database
The ConMAn Operators [BCD06a]

- **ConMAn = Concurrency Mutation Analysis**
- What are the ConMAn operators?
  - “…a comprehensive set of 24 operators for Java that are representative of the kinds of bugs that often occur in concurrent programs.”
  - based on an existing fault model for Java concurrency [FNU03]
- Can be used as a comparative metric
- In this experiment we used a subset of the operators for Java 1.4.

Example ConMAn Mutation
SKCR – Shrink Critical Region

Object lock1  = new Object();
...

public void m1 () {
    <statement n1>
    synchronized (lock1) {
        //critical region
        <statement c1>
        <statement c2>
        <statement c3>
    }
    <statement n2>
    ...
}
Example ConMAn Mutation
SKCR – Shrink Critical Region

Object lock1 = new Object();
...
public void m1 () {
    <statement n1>
synchronized (lock1) {
        //critical region
        <statement c1>
        <statement c2>
        <statement c3>
    }
    <statement n2>
...
Example ConMAn Mutation
SKCR – Shrink Critical Region

No Lock Bug!
Example ConMAn Mutation
ESP – Exchange Synchronized Block Parameters

Object lock1 = new Object();
Object lock2 = new Object();
...
\textbf{synchronized} (lock1) {
    <statement c1>
    ...
\textbf{synchronized} (lock2) {
    <statement c2>
    ...
}
}
...

Example ConMAn Mutation
ESP – Exchange Synchronized Block Parameters

Object lock1 = new Object();
Object lock2 = new Object();
...
synchronized (lock1) {
  <statement c1>
  ...
  synchronized (lock2) {
    <statement c2>
    ...
  }
}
...

Object lock1 = new Object();
Object lock2 = new Object();
...
synchronized (lock2) {
  <statement c1>
  ...
  synchronized (lock1) {
    <statement c2>
    ...
  }
}
...
Example ConMAn Mutation
ESP – Exchange Synchronized Block Parameters

Object lock1 = new Object();
Object lock2 = new Object();
...
**synchronized** (lock1) {
  <statement c1>
  ...
  **synchronized** (lock2) {
    <statement c2>
    ...
  }
}
...

Object lock1 = new Object();
Object lock2 = new Object();
...
**synchronized** (lock2) {
  <statement c1>
  ...
  **synchronized** (lock1) {
    <statement c2>
    ...
  }
}
...

**Deadlock bug!**
Experimental Setup

- Tests, Assertions
- Original Example Program
- ConMAn Operators (ASK, ASTK, SPCR)
- Testing with ConTest
- Mutant Example Program
- Model Checking with Java PathFinder
- Collection and Display of Results
- Comparison Results Database

Approach Selection
Example Program Selection
Mutation Selection
Program Artifact Selection
Experimental Procedure
Experimental Procedure

**ConMAn Operators**

- ASK
- ASTK
- SPCR

**Mutant Generation**

1. **Tests, Assertions**
2. **Original Example Program**
3. **Testing with ConTest**
4. **ConMAn Operators**
5. **Mutant Example Program**
6. **Model Checking with Java PathFinder**
7. **Collection and Display of Results**
8. **Comparison Results Database**
Experimental Procedure

- Tests, Assertions
- Original Example Program
- ConMAN Operators (ASK, ASTK, SPCR)
- Testing with ConTest
- Mutant Example Program
- Model Checking with Java PathFinder
- Collection and Display of Results
- Comparison Results Database

Mutant Generation

Testing
Experimental Procedure

- Tests, Assertions
- Original Example Program
- ConMAn Operators
  - ASK
  - ASTK
  - SPCR
- Testing with ConTest
- Mutant Example Program
- Model Checking with Java PathFinder
- Collection and Display of Results
- Comparison Results Database

Mutant Generation
Testing
Model Checking
Experimental Procedure

Mutant Generation
Testing
Model Checking
Collection and Display of Result
The ExMAn Framework [BCD06b]

- **ExMAn** = Experimental Mutation Analysis

- What is ExMAn?
  - “ExMAn is a reusable implementation for building different customized mutation analysis tools for comparing different quality assurance techniques.”
  - ExMAn automates the experimental procedure

ConTest vs. Java PathFinder

• How do we better understand the effectiveness of each technique?
  – We measure the mutant score for each technique (dependent variable)
  – We vary the analysis technique (factor)
  – We fix all other independent variables
    • quality artifacts (tests and properties), example programs …
Quantity of Mutants Killed

- Testing with ConTest
- Model Checking with Java PathFinder (JPF)

# of Mutants

- Assertion Violation
- OutputDifferent
- DeadlockDetected
- NoErrorDetected
- ToolFailure
Detection of Mutants

- 50% JPF & ConTest
- 38% Neither
- 6% JPF
- 6% ConTest
## Mutant Scores of JPF, ConTest and ConTest+JPF

<table>
<thead>
<tr>
<th>Example Program</th>
<th>ConTest Mutant Score</th>
<th>JPF Mutant Score</th>
<th>ConTest+JPF Mutant Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BufWriter</td>
<td>38.9%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>LinkedList</td>
<td>50%</td>
<td>50%</td>
<td>50%</td>
</tr>
<tr>
<td>TicketsOrderSim</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>AccountProgram</td>
<td>78%</td>
<td>56%</td>
<td>78%</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>56%</strong></td>
<td><strong>56%</strong></td>
<td><strong>62%</strong></td>
</tr>
</tbody>
</table>
ConTest and JPF are most likely *alternative* fault detection techniques with respect to the example programs.
ConTest vs. Java PathFinder

• How do we better understand the **efficiency** of each technique?
  – If ConTest and Java PathFinder are both capable of finding a fault in a program is either of them faster?
ConTest vs. Java PathFinder

• Experimental Setup
  – null hypothesis ($H_0$): Time to detect a fault for JPF > Time to detect a fault for ConTest
  – dependent variable(s): analysis time
  – independent variables:
    • factor: analysis technique
    • fixed: quality artifacts (tests and properties) software under evaluation
ConTest vs. Java PathFinder

• **Time for ConTest** (seconds)
  – Mean = 2.0314
  – Median = 1.2030

• **Time for Java PathFinder** (seconds)
  – Mean = 3.2835
  – Median = 2.3320

• Conducted a **paired t-test** for n=19
  – P-value = 0.0085 (reject $H_0$ at the 0.05 level)
  – JPF is not more efficient than ConTest for our example programs
Threats to Validity

• **internal validity**
• **external validity:**
  – Threats to external validity include:
    • the software being experimented on is not representative of concurrent Java programs in general
    • The configurations of Java PathFinder and ConTest limit our ability to generalize to each approach
  • **construct validity**
  • **conclusion validity**
Conclusions

• For our example programs…
  – **Effectiveness:** ConTest and Java PathFinder are most likely alternatives (potential to be used with other examples in a complementary way).
  – **Efficiency:** ConTest is more efficient and can kill a mutant in less time on average than Java PathFinder.

• Future work is **further empirical studies** in order to generalize our conclusions. 😊
Comparative Assessment of Testing and Model Checking Using Program Mutation

Research Talk

Jeremy S. Bradbury
Faculty of Science • University of Ontario Institute of Technology
Oshawa • Ontario • Canada
jeremy.bradbury@uoit.ca

James R. Cordy, Juergen Dingel
School of Computing • Queen’s University
Kingston • Ontario • Canada
{cordy, dingel}@cs.queensu.ca

Mutation 2007 • Sept. 10-11, 2007