Finding Bugs in Concurrent Java Programs

A Comparison of Bug Detection Tools Using Mutation

Jeremy S. Bradbury, James R. Cordy, Juergen Dingel ● Queen's University ● Kingston, Ontario, Canada
{bradbury, cordy, dingel}@cs.queensu.ca

1. Motivation

- An increase in the need for concurrent software development
- Concurrent software offers a new set of challenges not present in sequential code

  For example: deadlock and race conditions
- A concurrency bug may only occur in a very small number of execution interleavings making it extremely difficult to detect prior to deployment
- Reasoning about all possible interleavings in a program and ensuring interleavings do not contain bugs is non-trivial

Research Goal: to empirically assess different bug detection tools using seeded faults created via experimental mutation analysis.

2. Concurrent Testing vs. Model Checking

- Concurrent testing of Java with the IBM tool ConTest
  - Inserts random delays at synchronization points
  - Generates different interleavings each time a program is run
- Model checking of Java with Java PathFinder or Bandera/Bogor
  - Exhaustively searches the entire state space of a model (i.e., all interleavings)
  - Allows for the analysis of assertions and deadlock detection

3. Experimental Mutation Analysis

- Use mutation to empirically assess testing, static analysis, model checking, and dynamic analysis [5,6]

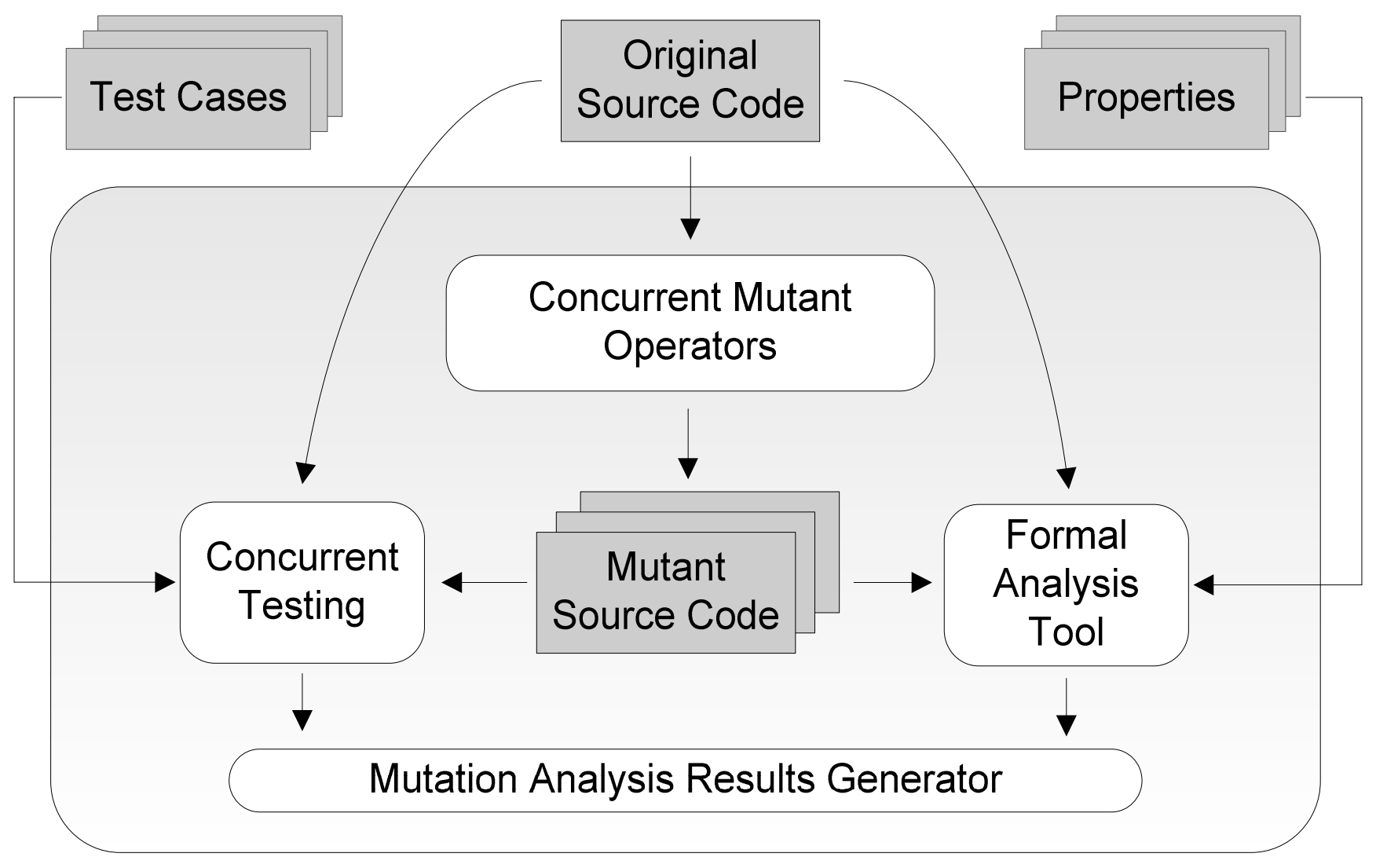

Figure 1: Experimental mutation analysis for comparing testing and model checking

4. Mutation Operators for Concurrent Java

- **ConMAn** (**Con**currency **M**utation **An**alysis) operators for Java (J2SE 5.0) [2,3]
- 24 ConMAn operators based on real concurrency bug patterns
- Implemented in TXL – a source transformation language
- The classes of operators include: modifying critical regions, keywords, concurrency method calls, parameters of concurrency method calls, and switching concurrency objects
- **An example of a Shrink Critical Region (SKCR) mutation:**

```
Original Code:
  <statement n1>
  synchronized (this) {
    / / critical region
    <statement c1>
    <statement c2>
    <statement c3>
  }
  <statement n2>
```

```
SKCR Code:
  <statement n1>
  //critical region
  <statement c1>
  synchronized (this) {
    <statement c2>
  }
  <statement c3>
  <statement n2>
```

- **An example of a Remove Static Keyword (RSTK) mutation:**

```
Original Code:
  public static synchronized void bMethod() { … }

RSTK Mutant:
  public synchronized void bMethod() { … }
```

5. ExMAn Framework

- **ExMAn** (**Ex**perimental **M**utation **An**alysis) Framework [1,4]
- A realization of our experimental mutation analysis approach
- A reusable implementation for building different customized mutation analysis tools for comparing different quality assurance techniques

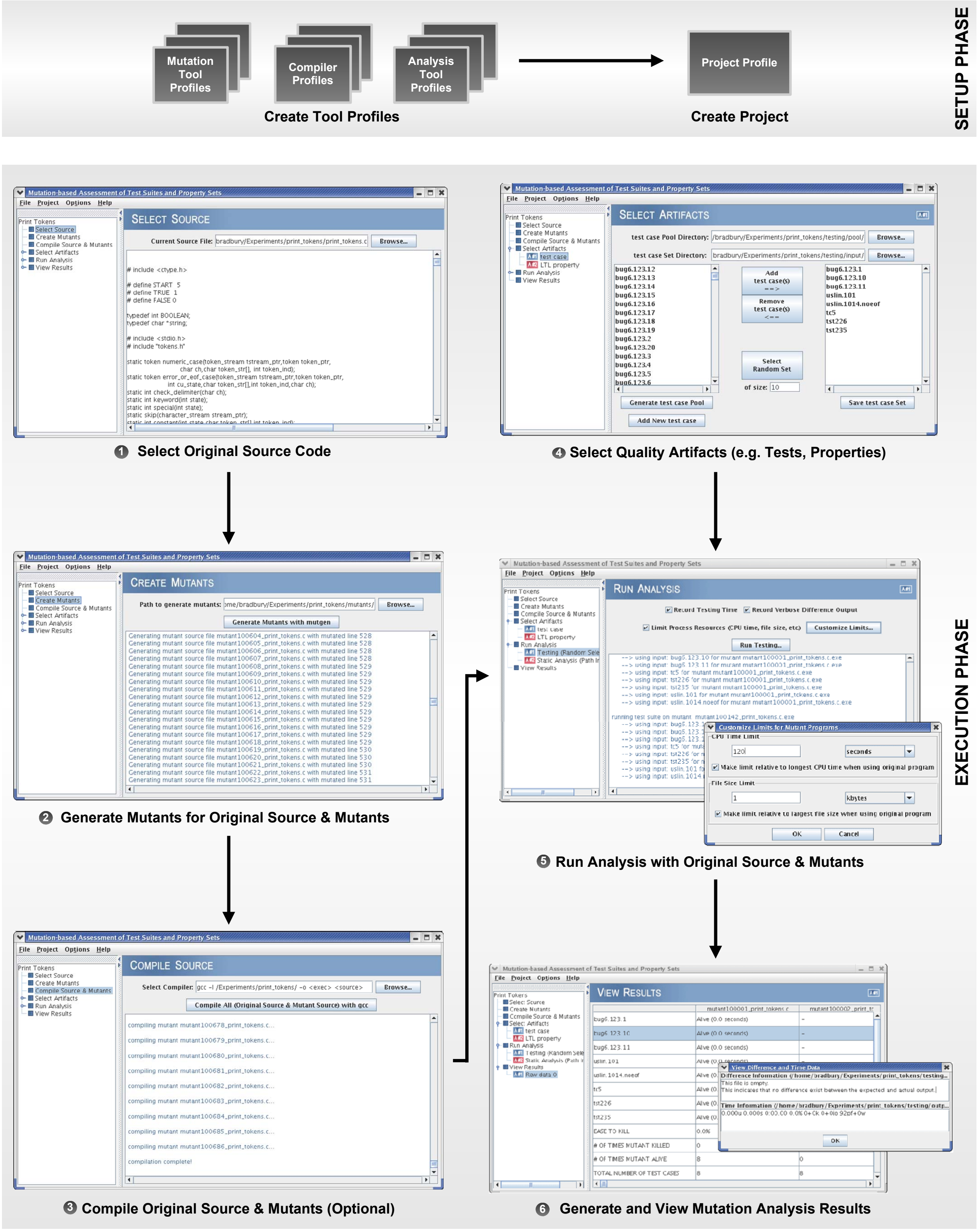
Mutation Tool Profiles
Compiler Profiles
Analysis Tool Profiles
Create Tool Profiles
Project Profile
Create Project
SETUP PHASE
❶ Select Original Source Code
❷ Generate Mutants for Original Source & Mutants
❸ Compile Original Source & Mutants (Optional)
❹ Select Quality Artifacts (e.g. Tests, Properties)
❺ Run Analysis with Original Source & Mutants
❻ Generate and View Mutation Analysis Results
EXECUTION PHASE

Figure 2: The ExMAn process

6. Comparing Testing and Model Checking

- ExMAn is customized to compare testing with ConTest and model checking with Java PathFinder or Bandera/Bogor
- The ConMAn plug-in is used with ExMAn to generate the faulty program versions
- Our current experiment uses a set of 7 programs from a benchmark of concurrent Java applications maintained at the IBM Haifa Labs

7. References

[1] ExMAn Framework website (http://www.cs.queensu.ca/~bradbury/exman/)

[2] ConMAn Operators website (http://www.cs.queensu.ca/~bradbury/conman/)

[3] **"Mutation Operators for Concurrent Java (J2SE 5.0)"**, J.S. Bradbury, J.R. Cordy, and J. Dingel, In *Proc. of the* 2nd *Workshop on Mutation Analysis (Mutation 2006)*, Nov. 2006, 10 pp. *(to appear)*

[4] **"ExMAn: A Generic and Customizable Framework for Experimental Mutation Analysis"**, J.S. Bradbury, J.R. Cordy, and J. Dingel, In *Proc. of the* 2nd *Workshop on Mutation Analysis (Mutation 2006)*, Nov. 2006, 6 pp. *(to appear)*

[5] **"Using Mutation for the Assessment and Optimization of Tests and Properties"**, J.S. Bradbury, *Doctoral Symposium being held in conjunction with the International Symposium on Software Testing and Analysis (ISSTA 2006)*, Jul. 2006, 4 pp.

[6] **"An Empirical Framework for Comparing Effectiveness of Testing and Property-Based Formal Analysis"**, J.S. Bradbury, J.R. Cordy, and J. Dingel, In *Proc. of the* 6th *International ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE 2005)*, pages 2–5, Sept. 2005.

NSERC CRSNG