Using Clone Detection to Identify Bugs in Concurrent Software

Kevin Jalbert, Jeremy S. Bradbury
Software Quality Research Group
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
{kevin.jalbert, jeremy.bradbury}@uoit.ca
http://faculty.uoit.ca/bradbury/sqrg/
Concurrent Software

• Concurrent software has multiple threads that can be interleaved in many different ways
• The different interleavings make concurrent software difficult to test and debug
Concurrent Software

• Concurrent software has multiple threads that can be interleaved in many different ways.

• The different interleavings make concurrent software difficult to test and debug.
  • **Data Races** – two or more threads access unprotected shared data, resulting in inconsistent access to the shared data.
  • **Deadlock** – the order of lock acquisition prevents other threads from acquiring the needed lock.
Concurrency Bug Detection

- Concurrency Testing
  - Costly dynamic analysis tools
  - Trade-off between effectiveness and efficiency

Example Testing Tools:
- IBM ConTest
- Microsoft CHESS
- NASA Java Pathfinder
  ...
Concurrency Testing with IBM ConTest

- A typical testing process using ConTest [EFN+02]

Active testing uses a randomized thread scheduler to verify if warnings reported by a predictive program analysis are real bugs.

- P. Joshi, M. Naik, C.-S. Park, and K. Sen [JNPS09]

Example: CalFuzzer
What kind of predictive program analysis can we use to improve testing with ConTest?
What kind of predictive program analysis can we use to improve testing with ConTest?
Clone Detection

• Ability to find similar code fragments within source code

• Able to find Type I-III clones
  I. Exact
  II. Near-exact
  III. Gapped
Goal

- Identify potential concurrency bugs in software using clone detection to localize testing effort
Key Tasks

1. Identification of concurrency bugs
2. Using clone detection of existing bugs (and bug patterns)
3. Localize testing efforts within the thread interleaving space
Identification of Concurrency Bugs

• An identified bug is abstracted to create a bug pattern

• Concurrency bug patterns require:
  • Code fragments involved in the bug
  • Interaction between the code fragments that causes the bug
    • Specifically, we are interested in the interaction between objects in the code fragments
Bug Patterns and Clone Detection

- **Clone detection** is used to identify clones of a bug pattern’s code fragments.
- The results of clone detection is a set of clones for each code fragment.
- We **classify** a set of clones that match a bug pattern’s code fragments as either **high-** or **low-potential** for being an actual concurrency bug
  - (high-potential bug matches also satisfy **rules** that define the interactions between the code fragments of the bug pattern)
Walkthrough

- Pattern Knowledge
- User knowledge
- User experience
Walkthrough

- Bug Pattern Creation
  - Easy way to **specify** and **maintain** bug patterns using the Bug Pattern Creator
Bug Pattern Creator

- General bug pattern information
Bug Pattern Creator

- **Code fragments** required for this bug pattern
- **Ability to highlight terms** (objects that interact)
Bug Pattern Creator

- Terms from code fragments are combined into a rule
- Defines the interactions between code fragments
- Uses Boolean operators and properties
Walkthrough

- Bug Patterns
  - Contains bug pattern information that is represented in XML
Example Data Race Bug Pattern

```
<bugPattern id="0" sourcePath="/bp/bug_pattern_0.xml">
  <type>Data race - no locks</type>
  <tester>John Smith</tester>
  <description>This bug exhibits inconsistent data of the obj object.</description>
  <solution>Synchronize both code fragments.</solution>
  <originalFragment sourcePath="/src/bp_code/bug_pattern_code_0_0.java" countLines="0"
    patternId="0" fragmentId="0">
    <term id="F0.obj" line="0" tokenPosition="3"/>
  </originalFragment>
  <originalFragment sourcePath="/src/bp_code/bug_pattern_code_0_1.java" countLines="0"
    patternId="0" fragmentId="1">
    <term id="F1.obj" line="0" tokenPosition="0"/>
  </originalFragment>
  <rule>(F0.obj == F1.obj && !F0.obj.IS_SYNCHED && !F1.obj.IS_SYNCHED)</rule>
</bugPattern>
```
Example Data Race Bug Pattern

```xml
<bugPattern id="0" sourcePath="/bp/bug_pattern_0.xml">
  <type>Data race - no locks</type>
  <tester>John Smith</tester>
  <description>This bug exhibits inconsistent data of the obj object.</description>
  <solution>Synchronize both code fragments.</solution>

  <originalFragment sourcePath="/src/bp_code/bug_pattern_code_0_0.java" countLines="0"
    patternId="0" fragmentId="0">
    <term id="F0.obj" line="0" tokenPosition="3"/>
  </originalFragment>

  <originalFragment sourcePath="/src/bp_code/bug_pattern_code_0_1.java" countLines="0"
    patternId="0" fragmentId="1">
    <term id="F1.obj" line="0" tokenPosition="0"/>
  </originalFragment>

  <rule>(F0.obj == F1.obj && !F0.obj.IS_SYNCHED
   && !F1.obj.IS_SYNCHED)</rule>
</bugPattern>
```
Example Data Race Bug Pattern

```
<bugPattern id="0" sourcePath="/bp/bug_pattern_0.xml">
  <type>Data race - no locks</type>
  <tester>John Smith</tester>
  <description>This bug exhibits inconsistent data of the obj object.</description>
  <solution>Synchronize both code fragments.</solution>
  <originalFragment sourcePath="/src/bp_code/bug_pattern_code_0_0.java" countLines="0"
    patternId="0" fragmentId="0">
    <term id="F0.obj" line="0" tokenPosition="3"/>
  </originalFragment>
  <originalFragment sourcePath="/src/bp_code/bug_pattern_code_0_1.java" countLines="0"
    patternId="0" fragmentId="1">
    <term id="F1.obj" line="0" tokenPosition="0"/>
  </originalFragment>
  <rule>(F0.obj == F1.obj &amp;&amp; !F0.obj.IS_SYNCHED &amp;&amp; !F1.obj.IS_SYNCHED)</rule>
</bugPattern>
```
Example Deadlock Bug Pattern

<bugPattern id="1">
  ...
  <originalFragment fragmentId="0">
    <term id="F0.lock1"/>
    <term id="F0.lock2"/>
  </originalFragment>
  <originalFragment fragmentId="1">
    <term id="F1.lock2"/>
    <term id="F1.lock1"/>
  </originalFragment>
  <rule>(F0.lock1 == F1.lock1 && F0.lock2 == F1.lock2)</rule>
</bugPattern>
Walkthrough

- Bug Pattern Code Fragments
  - The actual code fragments that composes the bug pattern
Example Deadlock Code Fragments

```java
synchronized (lock1) {
    synchronized (lock2) {
        var1 = obj.read();
    }
}
```

```java
synchronized (lock2) {
    synchronized (lock1) {
        var1 = obj.read();
    }
}
```
Walkthrough

- Source Code
- The source code of the system under observation
Walkthrough

- **Clone Detection (ConQAT[JDH09])**
  - Designed for research
  - Detects type I-III clones between source code and bug pattern code fragments

---

Walkthrough

- **Cloned Fragment Grouping**
  - Forms **valid** bug pattern combinations using found **clones** of bug patterns
Walkthrough

- All Possible Bug Pattern Matches
- Possible concurrency bugs
Walkthrough

- Finding Terms in Cloned Fragments
  - Type II and III clone’s terms must be mapped to the appropriate terms
Finding Terms in Cloned Fragments

synchronized ( lock1 ){
    synchronized ( lock2 ){
        var1 = obj.read();
    }
}

synchronized ( lockB ){
    synchronized ( lockA ){
        a.add(a);
        newVar7 = a.read();
    }
}
Walkthrough

- Rule Evaluation
  - The rule is evaluated to categories the possible bugs into high- and low-potential bugs
Rule Evaluation

1. ( F0.lock1 == F1.lock1 && F0.lock2 == F1.lock2 )
   • Original bug pattern rule

2. ( F0.lockB == F1.lockB && F0.lockA == F1.lockA )
   • Replace terms with source code clone match terms

3. ( true == true )
   • Evaluate
Walkthrough

- Potential Bugs
  - A XML list of high-potential bugs, along with source code location
Walkthrough

- Report Generation
  - Process to transform XML list of potential bugs into an HTML report
Walkthrough

- **HTML Report**
  - A readable report of the potential bugs
HTML Report

- **Summary statistics**
- **High-level view of potential bugs**

**Bug Pattern 0**

- **Id:** 0
- **Type:** Deadlock - Wrong Locks
- **Tester:** John Smith
- **Description:** This bug exhibits the classic deadlocking situation that occurs with the wrong locks being used
- **Solution:** Switch the locks to the right ones, exchanging the nested lock objects
- **Rule:**
  
  \[(F0.lock1 \equiv F1.lockA \& \& F0.lock2 \equiv F1.lockB)\]

**High-Potential Bug Matches 1/2.0 (50.0%)**

**Low-Potential Bug Matches 1/2.0 (50.0%)**

**High-Potential Match 0**

**Code Fragment 0**

- **Source:** /home/jalbert/workspace/conhpl/test-
- **Path:** data/example/src/bug.java
- **Start Line:** 22
- **Line Count:** 12

<table>
<thead>
<tr>
<th>Term Id</th>
<th>Term Line Number</th>
<th>Term Line Position</th>
<th>Term Token Position</th>
<th>Term Score</th>
<th>Term Line Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Walkthrough

• Test Cases
  • A testing suite that covers the area of the concurrency bug
Walkthrough

• Test Potential Bugs
• Using a dynamic testing technique like ConTest
• Explore thread interleaving space to verify potential bugs
Walkthrough

- **Real Bugs**
  - A report of real found bugs is formulated
Proposed Experimental Evaluation

• In order to comprehensively evaluate our active testing research we need to satisfy the following three goals:
  • Ensure that our specification notation for concurrency bug patterns is expressive enough to handle many different types of concurrency bugs.
  • Assess our bug detection process and the use of clone detection with finding concurrency bugs.
  • Evaluate the benefits of using the high-potential bugs to localize testing effort.
Conclusion

• The use of clone detection and bug patterns should **increase testing effectiveness** by **reducing the search space**, even with the possibility of false positives.
Future Work

• Additional work is needed to finish the active testing process.
• Experimentation is needed to assess the benefits of our tool when compared to existing active testing tools such as CalFuzzer.
Using Clone Detection to Identify Bugs in Concurrent Software

Kevin Jalbert, Jeremy S. Bradbury
Software Quality Research Group
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
{kevin.jalbert, jeremy.bradbury}@uoit.ca
http://faculty.uoit.ca/bradbury/sqrg/