Using Bug Patterns in the Regression Testing of Concurrent Software

Kevin Jalbert, Jeremy S. Bradbury ● Software Quality Research Group ● Faculty of Science ● University of Ontario Institute of Technology ● Oshawa, ON, Canada
kevin.jalbert@mycampus.uoit.ca, jeremy.bradbury@uoit.ca

1. Motivation

- In general, it is challenging to develop and test high quality concurrent software
  - **Why?** Possibly many different thread interleavings
  - Tools already exist to test and debug these different schedules
- We believe combining static analysis with regression testing can ensure that previously fixed concurrency bugs do not reoccur in future versions

Research Goal:
1. Statically detect potential concurrency bugs in a software system using bug patterns
2. Use potential concurrency bugs to optimize the testing of concurrent software

2. Background

- During maintenance software can regress and previous bugs can reoccur as a result of many different reasons
  - e.g., same programmer writing habitually incorrect code, code that contained a bug is copied & pasted with potential modifications, etc.
- Our approach uses clone detection as the foundation for detecting bug patterns in concurrent software
  - Clone detection is a process of finding source code that is duplicated (with possible modifications)
- We use the ConQAT[1] clone detection framework because it is designed to be easily customizable and extensible – it is capable of detecting exact, near-exact and gapped clones
  - Exact clones are exactly the same textually
  - Near-exact clones are the same in structure though textually they are different
  - Gapped clones have some statements that have been added or removed
- ConTest[2] is a testing tool that is capable of instrumenting source code to expose concurrency bugs

3. Defining Bug Patterns

What is a bug pattern?

- A bug pattern can consist of several fragments of code as well as rules about how the fragments interact to cause a bug [3]
- A potential bug is identified if all fragments are present and the rules are satisfied

How are bug patterns created and managed?

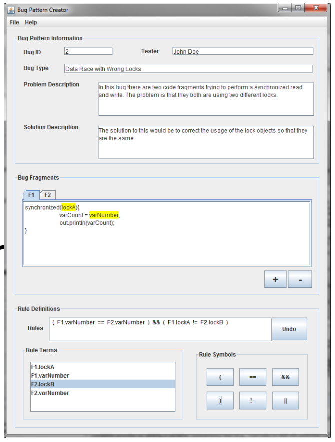
- Bug patterns are identified by developers when a bug is found
- A bug pattern can be created using the Bug Pattern Creator (see Figure 1)
- A collection of user-created bug patterns are stored as XML files and managed using Bug Pattern Creator

Bug Pattern

Bug Fragment 1 (B1):
```
synchronized(<t name="F1.lockA">lockA</t>{
   varCount = <t name="F1.varNumber">varNumber</t>;
   out.println(varCount);
}
```

Bug Fragment 2 (B2):
```
synchronized(<t name="F2.lockB">lockB</t>){
   out.println(varCount);
   <t name="varNumber">varNumber</t> = varCount;
}
```

Rule: (F1.varNumber == F2.varNumber) && (F1.lockA != F2.lockB)



Bug Pattern Information

Bug ID: 2
Tester: John Doe
Bug Type: Data Race with Wrong Locks
Problem Description: In this bug there are two code fragments trying to perform a synchronized read and write. The problem is that they both are using two different locks.
Solution Description: The solution to this would be to correct the usage of the lock objects so that they are the same.

Bug Fragments

F1 | F2

```
synchronized(lockA){
        varCount = varNumber;
        out.println(varCount);
}
```

Rule Definitions

Rules: (F1.varNumber == F2.varNumber) && (F1.lockA != F2.lockB)

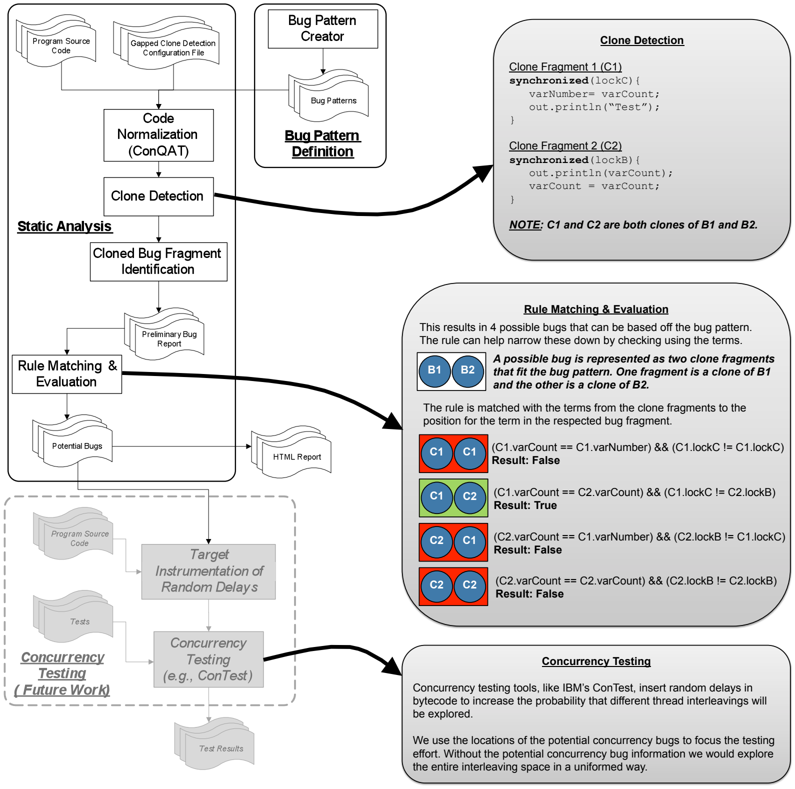
Rule Terms
F1.lockA
F1.varNumber
F2.lockB
F2.varNumber

Rule Symbols

Figure 1: Screenshot of Bug Pattern Creator that contains an example bug pattern

4. Process

- Figure 2 demonstrates how bug patterns are used to identify potential concurrency bugs and how these potential bugs can be used to optimize the testing effort


Program Source Code
Gapped Clone Detection Configuration File
Bug Pattern Creator
Bug Patterns
Bug Pattern Definition
Code Normalization (ConQAT)
Clone Detection
Static Analysis
Cloned Bug Fragment Identification
Preliminary Bug Report
Rule Matching & Evaluation
Potential Bugs
HTML Report
Program Source Code
Target Instrumentation of Random Delays
Tests
Concurrency Testing (e.g., ConTest)
Concurrency Testing (Future Work)
Test Results
B1
B2
C1
C2

Clone Detection

Clone Fragment 1 (C1)
```
synchronized(lockC){
   varNumber= varCount;
   out.println("Test");
}
```

Clone Fragment 2 (C2)
```
synchronized(lockB){
   out.println(varCount);
   varCount = varCount;
}
```

NOTE: C1 and C2 are both clones of B1 and B2.

Rule Matching & Evaluation

This results in 4 possible bugs that can be based off the bug pattern. The rule can help narrow these down by checking using the terms.

B1 B2 ***A possible bug is represented as two clone fragments that fit the bug pattern. One fragment is a clone of B1 and the other is a clone of B2.***

The rule is matched with the terms from the clone fragments to the position for the term in the respected bug fragment.

C1 C1 (C1.varCount == C1.varNumber) && (C1.lockC != C1.lockC)
Result: False

C1 C2 (C1.varCount == C2.varCount) && (C1.lockC != C2.lockB)
Result: True

C2 C1 (C2.varCount == C1.varNumber) && (C2.lockB != C1.lockC)
Result: False

C2 C2 (C2.varCount == C2.varCount) && (C2.lockB != C2.lockB)
Result: False

Concurrency Testing

Concurrency testing tools, like IBM's ConTest, insert random delays in bytecode to increase the probability that different thread interleavings will be explored.

We use the locations of the potential concurrency bugs to focus the testing effort. Without the potential concurrency bug information we would explore the entire interleaving space in a uniformed way.

Figure 2: Process Overview

5. Conclusions & Future Work

- The first research goal was achieved since bug patterns can be used to find potential concurrency bugs
- Future work includes:
  - Using a dynamic concurrency tool (e.g., ConTest) to test the potential concurrency bugs
  - Empirically studying and evaluating the benefits of our approach in comparison to other regression testing techniques for concurrency

[1] TUM's ConQAT website (http://conqat.cs.tum.edu/index.php/ConQAT)
[2] IBM's ConTest website (http://www.haifa.ibm.com/projects/verification/contest/)
[3] J.S. Bradbury and K. Jalbert. "Defining a Catalog of Programming Anti-Patterns for Concurrent Java", In *Proc. of the 3rd Int. Workshop on Software Patterns and Quality (SPAQu'09)*, pages 6-11, Orlando, Florida, USA, Oct. 2009.

CASCON 2009
Technology Showcase

Funding provided by: NSERC CRSNG