STAR: Stack-Trace based Automatic crash Reproduction

Sung Kim <hunkim@cse.ust.hk>
The Hong Kong University of Science and Technology
with Ning Chen and Hojun Jaygarl

ROSAEC Workshop Jan 6, 2011
Sung’s research areas

• MSR: Mining Software Repositories
  - Defect prediction (learning from repositories)
  - Bug triage/bug report mining
  - Crash report/stack trace mining
  - Code clones

• Static Analysis
  - Unit test generation
  - Crash reproduction
  - Patch generation
Mining Software Repositories

Produce

- Specs
- E-mails
- Objects
- Changes
- Traces
- Bug reports

Mine

Feedback

- Bug Prediction
- Change Impact Analysis
- Resource Allocation
- Software Understanding

History Information
Objects

Stack traces

Testing

Crash repro
ROSAEC Workshop 2010

Sequence (Randoop)

Object Space

Test run & System tests

OCAT
Reproducing Crashes

• Must be able to reproduce crashes for debugging
  - To fix bugs and validate fixes

• Reproducing crashes (faults) is hard!
  - Require the exact configuration of crash (in field)
ReCrash

Program

Crash

Unit tests

reproduce

generate

6-64% performance overhead
Crash Reporting System

A Crash Reporting System is a software application that captures and collects information about crashes or errors encountered by users. The system typically prompts users to report the error, which is then sent to a database for analysis. This database is used to diagnose and improve the software by analyzing the collected error reports.
Crash Reporting System

[Diagram showing the process of crash reporting and reproduction]
Crash Stack Traces

```
org.apache.bcel.classfile.ClassFormatException
at org.apache.bcel.classfile.ClassParser.readClassInfo(ClassParser.java:242)
at org.apache.bcel.classfile.ClassParser.parse(ClassParser.java:165)
at org.aspectj.weaver.bcel.Utility.makeJavaClass(Utility.java:358)
at org.aspectj.weaver.bcel.UnwovenClassFile.getJavaClass(UnwovenClassFile.java:63)
at org.aspectj.weaver.bcel.UnwovenClassFile.getClassName(UnwovenClassFile.java:147)
```

Exception
Frame 1

Call Stack
Frame 5
Frame n
Direction?

at org.apache.bcel.classfile.ClassParser.readClassInfo(ClassParser.java:242)
Crash Inputs

Frame 1:

ClassParser.readClassInfo(ClassInfo x)
(1) Receiver

ClassParser.readClassInfo(ClassInfo x)
(2) Arguments

1. `ClassParser.readClassInfo(ClassInfo x)`
Problem Definition

```java
void testCase() {
    ClassParser cp = ?
    ClassInfo x = ?
    cp.readClassInfo(x)
}
```
void testCase() {
    ClassParser cp = ?
    cp.readClassInfo(x)
}
void testCase() {
    ClassParser cp = ?
    ClassInfo x = ?
    cp.readClassInfo(x)
}
Three Approaches to Find Crash Inputs

• Feedback based random approach (feed)
• Object-capture based (objcap)
• Static analysis (precondition)
Feedback (Randoop)

- Find methods that return Bar
  - Bar foo() {..}
  - Bar getBar(List x) {..}
Feedback (Randoop)

• Find methods that return $Bar$
  - $Bar$ foo() {..}
  - $Bar$ getBar(List x) {..}

• Generates object instances recursively
  - foo = getFoo()
  - bar = get(foo)
Feedback (Randoop)

• Find methods that return \textit{Bar}
  - \textit{Bar} \textit{foo}() \{..\}
  - \textit{Bar} \textit{getBar} (\textit{List} \textit{x}) \{..\}

• Generates object instances recursively
  - \textit{foo} = \textit{getFoo}()
  - \textit{bar} = \textit{get} (\textit{foo})

• Mutate objects using method sequences
  - \textit{bar} = \textit{get} (\textit{Foo})
  - \textit{setBar} (\textit{bar})
  - ...
OCAT

Sequence (Randoop)

Object Space

Test run & System tests
OCAT

Sequence (Randooop)

Object Space

Test run & System tests

STAR
Mutating Object (precondition)

```java
foo (Object o) {
    if (o.x>o.y) {
        o.x = o.x + o.y;
        o.y = o.x - o.y;
        o.x = o.x - o.y;
        if (o.x - o.y > 0) {
            // throw exception
        }
    }
}
```
Mutating Object (precondition)

- Identify crash condition (postcondition)
- Compute weakest precondition (wp)
- There is a wp rule for each statement in the programming language
wp rules: assignment

// precondition: ??
\( x = e; \)
// postcondition: \( Q \)

Precondition = \( Q \) with all (free) occurrences of \( x \) replaced by \( e \)

Example:
// assert: ??
\( x = x + 1; \)
// assert \( x > 0 \)

Precondition = \((x+1) > 0\)

We write this as \( \text{wp} \) for “weakest precondition”
\( \text{wp}("x=e;", Q) = Q \) with \( x \) replaced by \( e \)
wp: if statement

// precondition: ??
if (b) S1 else S2
// postcondition: Q

Essentially case analysis
wp("if (b) S1 else S2", Q) =
    ( b ⇒ wp("s1", Q)
    ∧ ¬b ⇒ wp("s2", Q) )
wp: composition

// precondition: ??
S1; // some statement
S2; // another statement
// postcondition: Q

Work from back to front
Postcondition = wp(“s1; s2;”, Q) = wp(“s1;”, wp(“s2;”, Q))

Example:
// precondition: ??
x = 0;
y = x+1;
// postcondition: y > 0
wp example

```java
foo (Object o) {
    wp: o.x > o.y & o.y - o.x > 0
    if (o.x > o.y) {
        wp: x > y & ((x+y) - ((x+y) - y)) - ((x+y) - y) > 0
        o.x = o.x + o.y; wp: ((x+y) - ((x+y) - y)) - ((x+y) - y) > 0
        o.y = o.x - o.y; wp: (x - (x-y)) - (x-y) > 0
        o.x = o.x - o.y; wp: (x-y) - y > 0
        if (o.x - o.y > 0) {
            Q: x-y > 0
            throw exception
        }
    }
}
```
Three Approaches to Find Crash Inputs

- Feedback based random approach (feed)
- Object-capture based (objcap)
- Static analysis (precondition)
void testCase() {
    ClassParser cp = createCP(); // random
    ClassInfo x = loadClassInfo(); // object-capture
    x.b = false; // based on wp
    cp.readClassInfo(x)
}
STAR Approach

• **Challenge I**: Crash points?
  - Crash reporting system (MSR)

• **Challenge II**: missing objects
  - Collect from normal execution (MSR)

• **Challenge III**: not suitable objects
  - Mutate objects (Static Analysis)
### Experiments

<table>
<thead>
<tr>
<th>system</th>
<th># of bug reports</th>
<th># of bug reports with stack traces</th>
<th># of valid stack traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJDT</td>
<td>461</td>
<td>162</td>
<td>83</td>
</tr>
<tr>
<td>ACC</td>
<td>97</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ACM</td>
<td>116</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>Total</td>
<td>674</td>
<td>184</td>
<td>101</td>
</tr>
</tbody>
</table>
### Results

<table>
<thead>
<tr>
<th>System</th>
<th># of Bug Reports</th>
<th># of Reproducible Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJDT 1.1.</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>AJDT 1.2.</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>AJDT 1.5.</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>ACC</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>ACM</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>45</strong></td>
<td><strong>45</strong></td>
</tr>
</tbody>
</table>

- **Percent**: 44.6%
Summary

• STAR approaches
  - Feedback based random approach (feed)
  - Object-capture based (objcap)
  - Static analysis (precondition)

• 45 % crash reproduction (with 0 overhead)

• Repository data (captured objects, crash traces) help static analysis
Future Work

• Common change patterns + autofix?
  - Most autofix approaches are random mutation based

• Translation + Static analysis
  - “press x% when %x is on”

• Any other combinations with MSR?
STAR: Stack-Trace based Automatic crash Reproduction

Sung Kim <hunkim@cse.ust.hk>
http://www.cse.ust.hk/~hunkim
http://www.se.or.kr (Korean Blog)