Coverage-Driven Test Generation for Thread-Safe Classes via Parallel and Conflict Dependencies

IEEE TCSE Distinguished Paper Award

Valerio Terragni*
Mauro Pezzè*◇
Francesco Bianchi*

* USI Università della Svizzera italiana
Switzerland
◇ Università di Milano Bicocca
Italy

ICST 2019
Multi-Core Era

Trend of Desktop CPUs

- Transistors (thousands)
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Typical Power (Watts)
- Number of Logical Cores

Plot from: https://goo.gl/MJALxM
Non-Deterministic Thread Interleavings

Execution orders of shared-memory accesses among threads

Thread 1

\[
\text{temp} = x; \quad x = \text{temp} + 1;
\]

Thread 2

\[
\text{temp} = x; \quad x = \text{temp} + 1;
\]

interleaving #1

Thread 1

\[
\text{temp} = x; \quad x = \text{temp} + 1;
\]

Thread 2

\[
\text{temp} = x; \quad x = \text{temp} + 1;
\]

interleaving #2
Non-Deterministic Thread Interleavings

Execution orders of shared-memory accesses among threads

Thread 1

x : 0

temp = x;
x = temp + 1;

Thread 2

x : 0

temp = x;
x = temp + 1;

interleaving #1

x : 2

Thread 1

x : 0

temp = x;
x = temp + 1;

Thread 2

x : 1

temp = x;
x = temp + 1;

interleaving #2
Thread Synchronization

e.g., lock and unlock operations

Thread 1

lock(A)

\[
temp = x; \\
x = temp + 1; \\
unlock(A)
\]

Thread 2

lock(A)

\[
temp = x; \\
x = temp + 1; \\
unlock(A)
\]
Synchronization is Challenging
Thread-safe Classes

“A class that encapsulates synchronizations that ensure a correct behavior when the same instance of the class is accessed from multiple threads”

```java
public class C1 {
    private int x;
    private int y;

    public C1() { ... }

    public synchronized void m1(int k, C2 a) { ... }

    public void m2() {
        ...  
        synchronized(this){...}
        ...
    }
}
```
Thread-Safe Classes are Buggy
Thread-Safe Classes are Buggy

Oracle Technology Network > Java > Java SE > Community > Bug Database

JDK-4728096: java.io.BufferedInputStream has no synchronization on close operation

Type: Bug
Component: core-libs
Sub-Component: java.io
Affected Version: 1.4.0, 1.4.1, 1.4.2

Priority: P4
Status: Resolved
Resolution: Fixed
OS: generic, linux, linux_redhat_6.1, ...
CPU: generic, x86, sparc
Thread-Safety Violation (Example)

JDK-4728096: java.io.BufferedInputStream has no synchronization on close operation

Thread 1

```java
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}
```

NullPointerException

Thread 2

```java
// missing synchronization
public void close() {
    if (in == null)
        return;
    in.close();
    in = null;
    buf = null;
}
```

failure-inducing thread interleaving
Automated Concurrent Test Generation

Valerio Terragni and Mauro Pezzè
Effectiveness and Challenges in Generating Concurrent Tests for Thread-Safe Classes.
ASE 2018

General Framework
Automated Concurrent Test Generation

StringBufferInputStream var0 = new StringBufferInputStream("v;");
BufferedInputStream sout = new BufferedInputStream(var0);
sout.close();

...

StringBufferInputStream var0 = new StringBufferInputStream("v;");
BufferedInputStream sout = new BufferedInputStream(var0);
sout.read();
Automated Concurrent Test Generation

StringBufferInputStream var0 = new StringBufferInputStream("v;");
BufferedInputStream sout = new BufferedInputStream(var0);

sout.close();
sout.read();

Set of method call sequences that exercise the public interface of a class from multiple threads
Automated Concurrent Test Generation

StringBufferInputStream var0 = new StringBufferInputStream("v;");
BufferedInputStream sout = new BufferedInputStream(var0);

sout.close();

sout.read();

interleaving exploration
Random – Systematic - Selective
Automated Concurrent Test Generation

```
if (pos >= count)
    ...

buf[pos++] & 0xff;
```

**NullPointer Exception**

Thread Safety Oracle: **Linearizability** (Herlihy@TOPLAS `90)
Challenges

# 1 – Step 3 Interleaving exploration is expensive!

Implication: we cannot generate and explore the interleaving space of many concurrent tests

\[
\frac{(N_1+N_2)!}{N_1! N_2!} \quad \text{# possible interleavings}
\]

2 threads

50  \( N_1 = \) #shared memory accesses thread 1
50  \( N_2 = \) #shared memory accesses thread 2

\[9.2 \cdot 10^{128}\]
Challenges

**# 1 – Step 3** Interleaving exploration is expensive!

**Implication:** we cannot generate and explore the interleaving space of many concurrent tests

**# 2 – Step 2** Huge space of concurrent tests!

\[
\# \text{ of possible concurrent tests} = M^T L
\]

\[M = 10, L = 10, T = 3\]

\[10^{30}\]
Challenges

# 1 – Step 3 Interleaving exploration is expensive!
# 2 – Step 2 Huge space of concurrent tests!

How can we generate fewer tests that are likely to expose thread-safety violations?
Avoid generating concurrent tests that are redundant

Coverage-driven concurrent test generation
Choudhary @ ICSE 2017
Terragni @ ICSE 2016
Steenbuck @ ICST 2013
Our Intuition

Avoid generating concurrent tests that are redundant and irrelevant for exposing thread-safety violations

Coverage-driven concurrent test generation
Choudhary @ ICSE 2017
Terragni @ ICSE 2016
Steenbuck @ ICST 2013

DepCon
Conflict Dependency: method1 and method2 access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution method1 and method2 can interleave
Conflict Dependency: \texttt{read()} and \texttt{close()} access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution \texttt{read()} and \texttt{close()} can interleave

\begin{center}
\begin{tikzcd}
\text{Thread 1} & \cdots & \text{Thread 2} \\
\text{read()} & \text{close()}
\end{tikzcd}
\end{center}

\section*{EXAMPLE 1}

```
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}
```

```
public void close() {
    if (in == null)
        return;
    in.close();
in = null;
buf = null;
}
```
Conflict Dependency: read() and close() access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution read() and close() can interleave

```
Thread 1       Thread 2
read()     ...     close()
```

```java
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}
```

```java
public void close() {  
    if (in == null)
        return;
    in.close();
    in = null;
    buf = null;
}
```
Conflict Dependency: read() and close() access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution read() and close() can interleave

```java
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}

public void close() {
    if (in == null)
        return;
    in.close();
    in = null;
    buf = null;
}
```
Conflict Dependency: \textit{method1} and \textit{method2} access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution \textit{method1} and \textit{method2} can interleave

Theorem 1
Having both conflict and parallel dependencies is a \textbf{necessary condition} for exposing a thread-safe violation
Conflict Dependency: read() and mark() access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution read() and mark() can interleave

EXAMPLE 2

```java
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}
```

```java
public synchronized void mark(int readlimit) {
    marklimit = readlimit;
    markpos = pos;
}
```
Conflict Dependency: \texttt{read()} and \texttt{mark()} access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution \texttt{read()} and \texttt{mark()} can interleave

```
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}
```

```
public synchronized void mark(int readlimit) {
    marklimit = readlimit;
    markpos = pos;
}
```
**Conflict Dependency:** read() and mark() access at least one same shared-memory location (W-R, R-W)

**Parallel Dependency:** the execution read() and mark() can interleave

```java
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}

public synchronized void mark(int readlimit) {
    marklimit = readlimit;
    markpos = pos;
}
```

**DepCon will NOT generate concurrent tests that execute read() and mark() concurrently**
Conflict Dependency: `close()` and `mark()` access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution `close()` and `mark()` can interleave

EXAMPLE 3

```java
public void close() {
    if (in == null)
        return;
    in.close();
    in = null;
    buf = null;
}

public synchronized void mark(int readlimit) {
    marklimit = readlimit;
    markpos = pos;
}
```
Conflict Dependency: close() and mark() access at least one same shared-memory location (W-R, R-W)

Parallel Dependency: the execution close() and mark() can interleave

```java
public void close() {
    if (in == null)
        return;
    in.close();
    in = null;
    buf = null;
}
```

```java
public synchronized void mark(int readlimit) {
    marklimit = readlimit;
    markpos = pos;
}
```

DepCon will NOT generate concurrent tests that execute close() and mark() concurrently.
DepCon

Input
Class Under Test (CUT)

Static parallel and conflict dependency analysis

Coverage-Driven
Concurrent Test Generation

Concurrent Test
S1
Thread 1
S2
Thread 2
S3

Parallel and conflict dependencies

Interleaving Exploration

Output
Thread-safety Violations
Computing the Dependencies

Method summaries

**ACCESS SUMMARY**: it represents an over-approximation of all the possible accesses of shared-memory locations

{R(in), R(count), R(pos)}

**LOCK SUMMARY**: *set of locks that always protect every shared-memory accesses*

{this}

```java
private void ensureOpen() {
    if (in == null)
        throw new IOException("Stream closed");
}

public synchronized int available() {
    ensureOpen();
    return (count - pos) + in.available();
}
```
Computing the Dependencies

Challenges

- Efficiency (overhead should be low)
- Completeness (no missed dependencies)
- High precision (effective search space pruning)

Solution

- Novel and effective combination of classic static analysis techniques

```java
public synchronized int read() {
    ensureOpen();
    if (pos >= count) {
        fill();
        if (pos >= count)
            return -1;
    }
    return buf[pos++] & 0xff;
}
```
Evaluation

RQ1 Effectiveness
Can DepCon effectively generate concurrent tests that expose thread-safety violations?

RQ2 Comparison
Is DepCon more effective than state-of-the-art concurrent test generation?

RQ3 Static Analysis
What is the efficiency, completeness and precision of DepCon’s Static Analysis?
<table>
<thead>
<tr>
<th>Code Base</th>
<th>Class Name</th>
<th>LOC</th>
<th># public methods</th>
<th>fault type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Math</td>
<td>IntRange</td>
<td>276</td>
<td>26</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td>Apache DBCP</td>
<td>PerUserPoolDataSource</td>
<td>719</td>
<td>66</td>
<td>Data race</td>
</tr>
<tr>
<td></td>
<td>SharedPoolDataSource</td>
<td>546</td>
<td>52</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td>HQSQLDB</td>
<td>DoubleIntIndex</td>
<td>966</td>
<td>34</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td>JDK</td>
<td>BufferedInputStream</td>
<td>239</td>
<td>10</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td></td>
<td>Vector</td>
<td>786</td>
<td>45</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td></td>
<td>Day</td>
<td>267</td>
<td>26</td>
<td>Data race</td>
</tr>
<tr>
<td></td>
<td>NumberAxis</td>
<td>1,662</td>
<td>111</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td></td>
<td>PeriodAxis</td>
<td>1,975</td>
<td>126</td>
<td>Data race</td>
</tr>
<tr>
<td></td>
<td>TimeSeries</td>
<td>359</td>
<td>41</td>
<td>Data race</td>
</tr>
<tr>
<td></td>
<td>XYPlot</td>
<td>3,080</td>
<td>218</td>
<td>Data race</td>
</tr>
<tr>
<td></td>
<td>XYSeries</td>
<td>200</td>
<td>25</td>
<td>Data race</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>FileAppender</td>
<td>369</td>
<td>21</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td></td>
<td>WriterAppender</td>
<td>317</td>
<td>24</td>
<td>Atomicity violation</td>
</tr>
<tr>
<td>XStream</td>
<td>XStream</td>
<td>926</td>
<td>66</td>
<td>Data race</td>
</tr>
</tbody>
</table>

Subjects

- 7 popular Java code bases
- 15 known concurrency bugs
- Subjects used in the evaluation of previous work
## RQ1 Effectiveness

- Build on top of CovCon (Choudhary@ICSE2017)
- Interleaving Explorer: Stress testing (100 iterations)
- Time budget of 1 hour
- 5 runs

<table>
<thead>
<tr>
<th>Class name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>276</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>719</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>546</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>966</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>239</td>
</tr>
<tr>
<td>Vector</td>
<td>786</td>
</tr>
<tr>
<td>Day</td>
<td>267</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>1,662</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>1,975</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>359</td>
</tr>
<tr>
<td>XYPlot</td>
<td>3,080</td>
</tr>
<tr>
<td>XYSeries</td>
<td>200</td>
</tr>
<tr>
<td>FileAppender</td>
<td>369</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>317</td>
</tr>
<tr>
<td>XStream</td>
<td>926</td>
</tr>
</tbody>
</table>

**AVG** 68%
### RQ1 Effectiveness

- Build on top of CovCon (Choudhary@ICSE2017)
- Interleaving Explorer: Stress testing (100 iterations)
- Time budget of 1 hour
- 5 runs

<table>
<thead>
<tr>
<th>Class name</th>
<th>LOC</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>276</td>
<td>100%</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>719</td>
<td>40%</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>546</td>
<td>100%</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>966</td>
<td>60%</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>239</td>
<td>100%</td>
</tr>
<tr>
<td>Vector</td>
<td>786</td>
<td>20%</td>
</tr>
<tr>
<td>Day</td>
<td>267</td>
<td>100%</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>1,662</td>
<td>40%</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>1,975</td>
<td>100%</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>359</td>
<td>100%</td>
</tr>
<tr>
<td>XYPlot</td>
<td>3,080</td>
<td>40%</td>
</tr>
<tr>
<td>XYSeries</td>
<td>200</td>
<td>100%</td>
</tr>
<tr>
<td>FileAppender</td>
<td>369</td>
<td>20%</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>317</td>
<td>20%</td>
</tr>
<tr>
<td>XStream</td>
<td>926</td>
<td>80%</td>
</tr>
</tbody>
</table>

**AVG 68%**
<table>
<thead>
<tr>
<th>Class name</th>
<th>LOC</th>
<th>Success Rate</th>
<th>AVG Detection Time (hh:mm:ss)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>276</td>
<td>100%</td>
<td>00:01:21</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>719</td>
<td>40%</td>
<td>00:43:57</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>546</td>
<td>100%</td>
<td>00:11:51</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>966</td>
<td>60%</td>
<td>00:34:13</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>239</td>
<td>100%</td>
<td>00:00:07 MIN</td>
</tr>
<tr>
<td>Vector</td>
<td>786</td>
<td>20%</td>
<td>00:51:11</td>
</tr>
<tr>
<td>Day</td>
<td>267</td>
<td>100%</td>
<td>00:01:24</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>1,662</td>
<td>40%</td>
<td>00:39:50</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>1,975</td>
<td>100%</td>
<td>00:03:15</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>359</td>
<td>100%</td>
<td>00:02:19</td>
</tr>
<tr>
<td>XYPlot</td>
<td>3,080</td>
<td>40%</td>
<td>00:43:54</td>
</tr>
<tr>
<td>XYSeries</td>
<td>200</td>
<td>100%</td>
<td>00:00:33</td>
</tr>
<tr>
<td>FileAppender</td>
<td>369</td>
<td>20%</td>
<td>00:48:18 MAX</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>317</td>
<td>20%</td>
<td>00:48:06</td>
</tr>
<tr>
<td>XStream</td>
<td>926</td>
<td>80%</td>
<td>00:11:44</td>
</tr>
</tbody>
</table>

**AVG** 68% 00:22:48

**RQ1 Effectiveness**

- Build on top of CovCon (Choudhary@ICSE2017)
- Interleaving Explorer: Stress testing (100 iterations)
- Time budget of 1 hour
- 5 runs
<table>
<thead>
<tr>
<th>Class name</th>
<th>LOC</th>
<th>Success Rate</th>
<th>AVG Detection Time (hh:mm:ss)</th>
<th>AVG # Generated Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>276</td>
<td>100%</td>
<td>00:01:21</td>
<td>188</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>719</td>
<td>40%</td>
<td>00:43:57</td>
<td>2,810</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>546</td>
<td>100%</td>
<td>00:11:51</td>
<td>1,221</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>966</td>
<td>60%</td>
<td>00:34:13</td>
<td>2,617</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>239</td>
<td>100%</td>
<td>00:00:07</td>
<td>18 MIN</td>
</tr>
<tr>
<td>Vector</td>
<td>786</td>
<td>20%</td>
<td>00:51:11</td>
<td>1,174</td>
</tr>
<tr>
<td>Day</td>
<td>267</td>
<td>100%</td>
<td>00:01:24</td>
<td>255</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>1,662</td>
<td>40%</td>
<td>00:39:50</td>
<td>5,860</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>1,975</td>
<td>100%</td>
<td>00:03:15</td>
<td>438</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>359</td>
<td>100%</td>
<td>00:02:19</td>
<td>370</td>
</tr>
<tr>
<td>XYPlot</td>
<td>3,080</td>
<td>40%</td>
<td>00:43:54</td>
<td>4,113 MAX</td>
</tr>
<tr>
<td>XYSeries</td>
<td>200</td>
<td>100%</td>
<td>00:00:33</td>
<td>116</td>
</tr>
<tr>
<td>FileAppender</td>
<td>369</td>
<td>20%</td>
<td>00:48:18</td>
<td>2,622</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>317</td>
<td>20%</td>
<td>00:48:06</td>
<td>2,609</td>
</tr>
<tr>
<td>XStream</td>
<td>926</td>
<td>80%</td>
<td>00:11:44</td>
<td>222</td>
</tr>
<tr>
<td><strong>AVG</strong></td>
<td></td>
<td><strong>68%</strong></td>
<td><strong>00:22:48</strong></td>
<td><strong>1,642</strong></td>
</tr>
</tbody>
</table>

**RQ1 Effectiveness**

- Build on top of CovCon (Choudhary@ICSE2017)
- Interleaving Explorer: Stress testing (100 iterations)
- Time budget of 1 hour
- 5 runs
<table>
<thead>
<tr>
<th>Class name</th>
<th>Success Rate</th>
<th>AVG Detection Time (hh:mm:ss)</th>
<th># Generated Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>100%</td>
<td>00:01:21</td>
<td>188</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>40%</td>
<td>00:43:57</td>
<td>2,810</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>100%</td>
<td>00:11:51</td>
<td>1,221</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>60%</td>
<td>00:34:13</td>
<td>2,617</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>100%</td>
<td>00:00:07</td>
<td>18</td>
</tr>
<tr>
<td>Vector</td>
<td>20%</td>
<td>00:51:11</td>
<td>1,174</td>
</tr>
<tr>
<td>Day</td>
<td>100%</td>
<td>00:01:24</td>
<td>255</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>40%</td>
<td>00:39:50</td>
<td>5,860</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>100%</td>
<td>00:03:15</td>
<td>438</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>100%</td>
<td>00:02:19</td>
<td>370</td>
</tr>
<tr>
<td>XYPlot</td>
<td>40%</td>
<td>00:43:54</td>
<td>4,113</td>
</tr>
<tr>
<td>XYSeries</td>
<td>100%</td>
<td>00:00:33</td>
<td>116</td>
</tr>
<tr>
<td>FileAppender</td>
<td>20%</td>
<td>00:48:18</td>
<td>2,622</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>20%</td>
<td>00:48:06</td>
<td>2,609</td>
</tr>
<tr>
<td>XStream</td>
<td>80%</td>
<td>00:11:44</td>
<td>222</td>
</tr>
<tr>
<td><strong>AVG</strong></td>
<td><strong>68%</strong></td>
<td><strong>00:22:48</strong></td>
<td><strong>1,642</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CovCon (Choudhary@ICSE2017)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Success Rate</strong></td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td>40%</td>
</tr>
<tr>
<td>40%</td>
</tr>
<tr>
<td>20%</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>20%</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>0%</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>20%</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>0%</td>
</tr>
<tr>
<td>0%</td>
</tr>
<tr>
<td>80%</td>
</tr>
<tr>
<td><strong>AVG</strong></td>
</tr>
</tbody>
</table>
### CovCon (Choudhary@ICSE2017)

<table>
<thead>
<tr>
<th>Class name</th>
<th>Success Rate</th>
<th>AVG Detection Time (hh:mm:ss)</th>
<th># Generated Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>40%</td>
<td>00:36:15</td>
<td>6,660</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>40%</td>
<td>00:52:36</td>
<td>9,055</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>20%</td>
<td>00:56:00</td>
<td>8,947</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>100%</td>
<td>00:10:38</td>
<td>1,055</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>100%</td>
<td>00:00:34</td>
<td>161</td>
</tr>
<tr>
<td>Vector</td>
<td>20%</td>
<td>00:54:52</td>
<td>5,010</td>
</tr>
<tr>
<td>Day</td>
<td>100%</td>
<td>00:03:03</td>
<td>640</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>0%</td>
<td>01:00:00</td>
<td>12,368</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>100%</td>
<td>00:09:52</td>
<td>1,720</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>100%</td>
<td>00:15:56</td>
<td>2,895</td>
</tr>
<tr>
<td>XYPlot</td>
<td>20%</td>
<td>00:57:56</td>
<td>4,320</td>
</tr>
<tr>
<td>XYSeries</td>
<td>100%</td>
<td>00:07:47</td>
<td>1,842</td>
</tr>
<tr>
<td>FileAppender</td>
<td>0%</td>
<td>01:00:00</td>
<td>16,264</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>0%</td>
<td>01:00:00</td>
<td>15,911</td>
</tr>
<tr>
<td>XStream</td>
<td>80%</td>
<td>00:26:54</td>
<td>568</td>
</tr>
</tbody>
</table>

### DepCon (this work)

<table>
<thead>
<tr>
<th>Class name</th>
<th>Success Rate</th>
<th>AVG Detection Time (hh:mm:ss)</th>
<th># Generated Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRange</td>
<td>100%</td>
<td>00:01:21</td>
<td>26.72x</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>40%</td>
<td>00:43:57</td>
<td>1.20x</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>100%</td>
<td>00:11:51</td>
<td>4.73x</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>60%</td>
<td>00:34:13</td>
<td>0.31x</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>100%</td>
<td>00:00:07</td>
<td>5.21x</td>
</tr>
<tr>
<td>Vector</td>
<td>20%</td>
<td>00:51:11</td>
<td>1.07x</td>
</tr>
<tr>
<td>Day</td>
<td>100%</td>
<td>00:01:24</td>
<td>2.17x</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>40%</td>
<td>00:39:50</td>
<td>1.51x</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>100%</td>
<td>00:03:15</td>
<td>3.04x</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>100%</td>
<td>00:02:19</td>
<td>6.88x</td>
</tr>
<tr>
<td>XYPlot</td>
<td>40%</td>
<td>00:43:54</td>
<td>1.32x</td>
</tr>
<tr>
<td>XYSeries</td>
<td>100%</td>
<td>00:00:33</td>
<td>14.05x</td>
</tr>
<tr>
<td>FileAppender</td>
<td>20%</td>
<td>00:48:18</td>
<td>1.24x</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>20%</td>
<td>00:48:06</td>
<td>1.25x</td>
</tr>
<tr>
<td>XStream</td>
<td>80%</td>
<td>00:11:44</td>
<td>2.29x</td>
</tr>
</tbody>
</table>

### AVG

<table>
<thead>
<tr>
<th></th>
<th>Success Rate</th>
<th>AVG Detection Time (hh:mm:ss)</th>
<th># Generated Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>68%</td>
<td>00:22:48</td>
<td>4.87x</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>00:34:10</td>
<td>5,828</td>
</tr>
</tbody>
</table>
### Concurrent Function Pairs

<table>
<thead>
<tr>
<th>Class name</th>
<th>Concurrent Function Pairs</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL</strong></td>
<td><strong>DepCon</strong></td>
<td><strong>Reduction</strong></td>
</tr>
<tr>
<td>IntRange</td>
<td>351</td>
<td>21</td>
</tr>
<tr>
<td>PerUserPoolDataSource</td>
<td>2,211</td>
<td>66</td>
</tr>
<tr>
<td>SharedPoolDataSource</td>
<td>1,378</td>
<td>52</td>
</tr>
<tr>
<td>DoubleIntIndex</td>
<td>595</td>
<td>297</td>
</tr>
<tr>
<td>BufferedInputStream</td>
<td>55</td>
<td>22</td>
</tr>
<tr>
<td>Vector</td>
<td>1,035</td>
<td>51</td>
</tr>
<tr>
<td>Day</td>
<td>351</td>
<td>70</td>
</tr>
<tr>
<td>NumberAxis</td>
<td>6,216</td>
<td>292</td>
</tr>
<tr>
<td>PeriodAxis</td>
<td>8,001</td>
<td>278</td>
</tr>
<tr>
<td>TimeSeries</td>
<td>861</td>
<td>296</td>
</tr>
<tr>
<td>XYPlot</td>
<td>23,871</td>
<td>844</td>
</tr>
<tr>
<td>XYSeries</td>
<td>325</td>
<td>114</td>
</tr>
<tr>
<td>FileAppender</td>
<td>231</td>
<td>53</td>
</tr>
<tr>
<td>WriterAppender</td>
<td>300</td>
<td>37</td>
</tr>
<tr>
<td>XStream</td>
<td>2,211</td>
<td>427</td>
</tr>
<tr>
<td><strong>AVG</strong></td>
<td><strong>3,199</strong></td>
<td><strong>195</strong></td>
</tr>
</tbody>
</table>

### RQ3 Static Analysis

- **Complete:** all bugs were always detected
- **(mostly) Precise:** **13.89x** of reduction (average)
- **Efficient:** **1,519 ms** on average
  
  (1.66% of the detection time)
Conclusion

Synchronization is Challenging

DepCon

Automated Concurrent Test Generation

Cost-effective Static Analysis

RQ3 Static Analysis

java.io.BufferedInputStream JDK-4728096

Efficient: it takes 1 second
Effective: it reduces by 8.77x the number of generated tests
it exposes the bug 5.21x faster (on average)
Thank you!

Questions?
Computing the Dependencies

```java
public class A {
    private B field1 = new B();
    private int field2 = 0;

    public A() {
        ...
    }

    public void m1() {
        B lock = field1;
        synchronized(lock){
            int k = m2();
        }
    }

    private int m2() {
        field2++;
    }
}
```

Method summaries

ACCESS SUMMARY: it represents an over-approximation of all the possible accesses of shared-memory locations performed by all possible invocations of under all possible execution paths.

\{R(field1

LOCK SUMMARY: set of locks that always protect every shared-memory accesses that can be triggered by an invocation of m:
Concurrent Test for Thread-Safe Classes

Set of method call sequences that exercise the public interface of a class from multiple threads

Concurrent Test

StringBufferInputStream var0 = new StringBufferInputStream("v;");
BufferedInputStream sout = new BufferedInputStream(var0);

Thread 1

sout.close();

Thread 2

sout.read();

interleaving exploration
Random – Systematic - Selective