Fully Automatic and Precise Detection of Thread Safety Violations

Michael Pradel and Thomas R. Gross
Department of Computer Science
ETH Zurich

thread-safe.org
Motivation

Thread-safe classes:
Building blocks for concurrent programs
Motivation

Thread-safe classes:
Building blocks for concurrent programs
Motivation

Thread-safe classes:
Building blocks for concurrent programs
Example from JDK

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

Thread 1

```java
b.insert(1, b)
```

Thread 2

```java
b.deleteCharAt(1)
```
Example from JDK

StringBuffer b = new StringBuffer()
b.append("abc")

Thread 1

b.insert(1, b)

Thread 2

b.deleteCharAt(1)

IndexOutOfBoundsException

Confirmed as bug: Issue #7100996
Example from JDK

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

How to test thread safety?

IndexOutOfBoundsException

Confirmed as bug: Issue #7100996
Goal

Automatic and precise bug detection

Class -> Tool -> Thread safety violations
Goal

Automatic and precise bug detection

Tests
Class
Formal specs

Tool

Thread safety violations
False positives
Goal

Automatic and precise bug detection

Class → Tool  → Thread safety violations
Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”
Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”
Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”
Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”

“operations ... behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads”

StringBuffer API documentation, JDK 6
Thread-Safe Classes

“behaves correctly when accessed from multiple threads ... with no additional synchronization ... (in the) calling code”

“operations ... behave as if they occur in some serial order that is consistent with the order of the method calls made by each of the individual threads”

StringBuffer API documentation, JDK 6
Example

```java
StringBuffer b = new StringBuffer();

b.append("a")
Thread 1
b.append("b")
Thread 2
b.append("c")
```


Example

StringBuffer b = new StringBuffer()

b.append("a")

b.append("b")

b.append("c")

Thread 1

Thread 2

"abc" ✓  "cab" ✓  "acb" ✓  "ac" ✗
Example

```java
StringBuffer b = new StringBuffer()
b.append("a")
b.append("b")
b.append("c")
```

Threads:
- Thread 1:
  - `b.append("a")`
  - `b.append("b")`
- Thread 2:
  - `b.append("c")`

Results:
- "abc" [✓]
- "cab" [✓]
- "acb" [✓]
- "ac" [✗]
Example

```
StringBuffer b = new StringBuffer()
    b.append("a")
    b.append("b")
    b.append("c")
```

Thread 1
```
b.append("a")
```

Thread 2
```
b.append("c")
```

Valid: "abc", "cab", "acb", "acb"

Invalid: "ac"
Example

```java
StringBuffer b = new StringBuffer()
b.append("a")
b.append("b")
```

Thread 1
```
b.append("a")
```

Thread 2
```
b.append("c")
b.append("b")
```

Valid results:
- "abc"
- "cab"
- "acb"

Invalid result:
- "ac"
Example

```java
StringBuffer b = new StringBuffer();

b.append("a")

b.append("b")

b.append("c")

"abc"  ✓
"cab"  ✓
"acb"  ✓
"ac"   ✗
```
Approach

Class under test (CUT)

Generate a concurrent test

Execute

Thread safety oracle

Bug
Approach

Class under test (CUT)

Generate a concurrent test

Execute

Thread safety oracle

Bug
Approach

Class under test (CUT) →

Generate a concurrent test

→ Execute

Thread safety oracle

Bug
Generating Concurrent Tests

Example:

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(1, b)
```

Thread 1
```
b.insert(1, b)
```

Thread 2
```
b.deleteCharAt(1)
```
Generating Concurrent Tests

Example:

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

Sequential prefix: Create and set up CUT instance

Thread 1

Thread 2

```java
b.insert(1, b)  b.deleteCharAt(1)
```
Generating Concurrent Tests

Example:

```java
StringBuffer b = new StringBuffer()
b.append("abc")

Thread 1
Thread 2

b.insert(1, b) b.deleteCharAt(1)

Concurrent suffixes:
Use shared CUT instance
```
Test Generation Algorithm

1. Create prefix
   - Instantiate CUT
   - Call methods

2. Create suffixes for prefix
   - Call methods on shared CUT instance

3. Prefix + two suffixes = test
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

Randomly select a constructor

```java
StringBuffer b = new StringBuffer()
```
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

StringBuffer b = new StringBuffer()

After adding a call: Execute
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

After adding a call:
Execute

StringBuffer b = new StringBuffer()
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

Randomly select a method

```java
StringBuffer b = new StringBuffer();
```
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

StringBuffer b = new StringBuffer()

Randomly select a method

b.append(\/* String */\)
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

Arguments:
   a) Take available object
   b) Call method returning required type
   c) Random value

```java
StringBuffer b = new StringBuffer();
b.append(/* String */);
```
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

Arguments:
- a) Take available object
- b) Call method returning required type
- c) Random value

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

After adding a call:
Execute
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

After adding a call:
Execute
Creating a Prefix

1. Create prefix
   - Instantiate CUT
   - Call methods

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```
Creating Suffixes

2. Create suffixes
   for prefix

   ■ Call methods on
     shared CUT instance

   StringBuffer b = new StringBuffer()
   b.append("abc")
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert( /* int */ , /* CharSequence */ )
```
2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer();
b.append("abc")
```

Arguments:

a) Take available object
b) Call method returning required type
c) Random value

```java
b.insert(/* int */ , /* CharSequence */)
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

Arguments:

- a) Take available object
- b) Call method returning required type
- c) Random value

```java
StringBuffer b = new StringBuffer();
b.append("abc");
b.insert(-5, b);
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(-5, b)
```

After adding a call: Execute
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(-5, b)
```

After adding a call: Execute
2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(/* int */ , /* CharSequence */)
```
Creating Suffixes

2. Create suffixes for prefix
   - Call methods on shared CUT instance

   Arguments:
   a) Take available object
   b) Call method returning required type
   c) Random value

```java
StringBuffer b = new StringBuffer()
b.append("abc")

b.insert(1, b)
```
2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")

b.insert(1, b)
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(1, b)
```
2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

```java
b.insert(1, b)
```
2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")

b.insert(1, b)  // b.deleteCharAt(1)
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

After adding a call:

Execute

```java
b.insert(1, b)  // b.deleteCharAt(1)
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

- After adding a call:
  - Execute

```java
b.insert(1, b)
b.deleteCharAt(1)
```
Creating Suffixes

2. Create suffixes for prefix

- Call methods on shared CUT instance

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

```java
b.insert(1, b)  // b.deleteCharAt(1)
```
Creating a Test

3. Prefix + two suffixes = test
Creating a Test

3. Prefix + two suffixes = test

StringBuffer b = new StringBuffer()
b.append("abc")

b.insert(1, b)    b.deleteCharAt(1)
Creating a Test

3. Prefix + two suffixes = test

```java
StringBuffer b = new StringBuffer();
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

Spawn new thread for each suffix

Thread 1
```
b.insert(1, b)
```

Thread 2
```
b.deleteCharAt(1)
```
Approach

Class under test (CUT) →

Generate a concurrent test

Execute

Thread safety oracle

Bug
Approach

Class under test (CUT) → Generate a concurrent test → Execute → Thread safety oracle → Bug
Thread Safety Oracle

Does the test execution expose a thread safety violation?

- Focus on exceptions and deadlocks
- Compare concurrent execution to linearizations
Assumptions

Concurrency-only crashes are undesired

- Matches definition of thread safety

Control over all input to tests

- Sequential execution: Deterministic
Linearizations

- Put all calls into one thread
- Preserve order of calls within a thread
Linearizations

- Put all calls into one thread
- Preserve order of calls within a thread
The Oracle

Execute concurrently

Exception or deadlock?

Execute linearization

Same failure?

All linearizations checked

Thread safety violation
The Oracle

Execute concurrently

Exception or deadlock?

Yes

Execute linearization

Same failure?

Yes

No

All linearizations checked

Thread safety violation
The Oracle

Execute concurrently

Exception or deadlock?

Yes

Execute linearization

Same failure?

Yes

No

All linearizations checked

Thread safety violation
The Oracle

Execute concurrently

Exception or deadlock?

Yes

Execute linearization

Same failure?

Yes

Yes

All linearizations checked

No

Thread safety violation
The Oracle

- Execute concurrently
  - Exception or deadlock?
    - Yes: Execute linearization
    - No: All linearizations checked
      - Same failure?
        - Yes: Thread safety violation
        - No: Thread safety violation
The Oracle

Execute concurrently

Exception or deadlock?

Yes

Execute linearization

Same failure?

Yes

No

All linearizations checked

Thread safety violation
The Oracle

Execute concurrently

Exception or deadlock?

Yes

Execute linearization

Same failure?

Yes

No

All linearizations checked

Thread safety violation
The Oracle

1. Execute concurrently
2. Exception or deadlock?
   - No
   - Yes
3. Execute linearization
4. Same failure?
   - Yes
   - No
5. All linearizations checked
6. Thread safety violation
Example

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

Thread 1

```
b.insert(1, b)  b.deleteCharAt(1)
```
Example

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```
Example

```java
StringBuffer b = new StringBuffer()
b.append("abc")

// Thread 1
b.insert(1, b)

// Thread 2
b.deleteCharAt(1)

// Incorrect approach
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

⚠️
Example

```java
StringBuffer b = new StringBuffer()
b.append("abc")
```

![Diagram showing thread 1 inserting into the buffer and thread 2 deleting a character]

```java
b.insert(1, b)   b.deleteCharAt(1)
```

```java
StringBuffer b = ..
b.append("abc")
b.insert(1, b)   b.deleteCharAt(1)
```

```java
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
```
Example

```java
StringBuffer b = new StringBuffer()
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

Thread safety violation

```java
StringBuffer b = ..
b.append("abc")
b.insert(1, b)
b.deleteCharAt(1)
```

```java
StringBuffer b = ..
b.append("abc")
b.deleteCharAt(1)
b.insert(1, b)
```
Properties of the Oracle

Sound but incomplete
- All reported violations are real
- Cannot guarantee thread safety

Independent of bug type
- Data races
- Atomicity violations
- Deadlocks

* with respect to incorrectness
Implementation

thread-safe.org

Automatic and precise thread safety checking
Evaluation

1. Effectiveness in finding bugs

2. Performance

Setup:
- Thread-safe classes from six Java libraries (e.g., JDK, Apache DBCP)
- Intel Xeon (8x3GHz)
Bugs

Found **15 bugs and 0 false positives**

- **9 known bugs**
- **6 previously unknown bugs**
  - E.g., in JDK and Apache DBCP
Example: Apache DBCP

```java
DataSource ds = new DataSource()

Thread 1         Thread 2

ds.setDataSourceName("a")   ds.close()
```
Example: Apache DBCP

```
DataSource ds = new DataSource()

Thread 1

ds.setDataSourceName("a")

ds.close()

Thread 2

```

⚠️ ConcurrentModificationException

Reason: Unsynchronized use of thread-unsafe collection

Confirmed as bug: Issue #369
Kinds of Failures

12 of 15 failures are implicit (VM or JDK)

Most common:

- NullPointerException
- ConcurrentModificationException
Performance

The graph shows the relationship between the number of bugs and the time taken to resolve them. The y-axis represents time in minutes, ranging from 0.01 to 1000, and the x-axis represents the number of bugs. The data points are clustered above the line indicating > 1 hour and below the line indicating < 1 minute.
Performance

![Graph showing time (min/avg/max minutes) vs. bugs]

- Time (min/avg/max minutes) range from 0.01 to 1000
- Bugs range from 0 to 8 hours
- Significant number of data points indicating time over 1 hour and less than 1 minute
Conclusion

Concurrency: More and more important

Need tools to test thread-safe classes

This work:

- Fully automatic testing
- Only real bugs reported
Thank you!

Try it:  
thread-safe.org

Fully Automatic and Precise Detection of Thread Safety Violations
Michael Pradel and Thomas R. Gross, ETH Zurich