Large-Scale API Protocol Mining for Automated Bug Detection

Michael Pradel
Department of Computer Science
ETH Zurich
Motivation

`LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while ( i.hasNext() ) {
    PinLink curr = (PinLink) i.next();
    if ( ... ) {
        pinConnections.remove(curr);
    }
}"

(from DaCapo benchmarks)
Motivation

Don’t modify a collection while iterating over it!

(from DaCapo benchmarks)
API Usage Constraints

Program

Constraints

call x before y

eventually call x
don’t call x while calling y and z

API
API Usage Constraints

Program

call x before y

eventually call x

don’t call x while calling y and z

Protocols!

API
Protocol Mining

Problem: No protocols specified

Training Programs

API
Protocol Mining

Problem: No protocols specified

Training Programs

Mining

API
Protocol Mining

Problem: No protocols specified

[Diagram showing the process of Protocol Mining]

- **Training Programs**
- **API**
- **Mining**
- **Target Program**
- **Bug finding**

The diagram illustrates the process of Protocol Mining, which involves training programs to mine APIs, followed by bug finding in the target program.
Big Picture

Protocol mining → Bug finding
Big Picture

Protocol mining

Bug finding

Other applications
Big Picture

Protocol mining → Bug finding → Other applications
Big Picture

Protocol mining

- static vs. dynamic
- single- vs. multi-object

API-based vs. client-based

Bug finding

Other applications
Big Picture

Protocol mining

- static vs. dynamic
- single- vs. multi-object

API-based vs. client-based

Bug finding

- verification
- static testing
- dynamic testing

Other applications
Part 1: Protocol Mining
Protocol Mining - Overview

Program & Input

Execution trace

Subtraces

Protocols
Protocol Mining - Overview

- Program & Input
- Execution trace
- Subtraces
- Protocols
List l = new LinkedList();
l.add(new Foo());
Iterator i = l.iterator();
OutputStream s
    = new FileOutputStream("f");
while (i.hasNext()) {
    Foo f = i.next();
    if (f.isOK())
        s.write(f.getData());
}
s.close();
Execution Traces

List l = new LinkedList();
l.add(new Foo());
Iterator i = l.iterator();
OutputStream s
    = new FileOutputStream("f");
while (i.hasNext()) {
    Foo f = i.next();
    if (f.isOK())
        s.write(f.getData());
}
s.close();

AspectJ instrumentation
List l = new LinkedList();
    l.add(new Foo());
Iterator i = l.iterator();
OutputStream s
    = new FileOutputStream("f");
while (i.hasNext()) {
    Foo f = i.next();
    if (f.isOK())
        s.write(f.getData());
}
s.close();

AspectJ instrumentation
new LinkedList → 1
1.add(2) → 3
1.iterator → 4
new FileOutputStream(6) → 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close
Protocol Mining - Overview

Program & Input

Execution trace

Subtraces

Protocols
Protocol Mining - Overview

Program & Input

Execution trace

Subtraces

Protocols
Subtraces

new LinkedList → 1
1.add(2) → 3
1.iterator → 4
new FileOS(6) → 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close
Subtraces

```
new LinkedList → 1
1.add(2) → 3
1.iterator → 4
new FileOS(6) → 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close
```

Different API usages intermingled!
Subtraces

new LinkedList → 1
1.add(2) → 3
1.iterator → 4
new FileOS(6) → 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close

Core object x:
- Calls to x
- Calls to parameters passed to x
- Calls to objects returned by x
Subtraces

new LinkedList → 1
1.add(2) → 3
1.iterator → 4
new FileOS(6) → 5
4.hasNext
4.next
5.write(7)
4.hasNext
5.close

new LinkedList → 1
1.add(2) → 3
1.iterator → 4
4.hasNext
4.next
4.hasNext
4.hasNext
4.hasNext
new FileOS(6) → 5
5.write(7)
5.close
Protocol Mining - Overview

Program & Input

Execution trace

Subtraces

Protocols
Protocol Mining - Overview

Program & Input

Execution trace

Subtraces

Protocols
Group Subtraces

Group by set of involved types

LinkedList, Iterator

Iterator

FileOS
Generate Protocols

Finite state machine

- Method $\rightarrow$ state
- Consecutive call $\rightarrow$ transition
Generate Protocols

Finite state machine

- **Method** → state
- **Consecutive call** → transition

```
new LinkedList  → l
l.add
l.add
l.add
l.add
l.iterator  → i
i.hasNext
i.hasNext
i.hasNext
i.next
l.add
```
Generate Protocols

Finite state machine

- Method $\rightarrow$ state
- Consecutive call $\rightarrow$ transition
Scalability

Bottleneck: Large execution traces
Scalability

Bottleneck: Large execution traces

Pass 1:
Find core objects and associated objects
Scalability

Bottleneck: Large execution traces

Pass 1:
Find core objects and associated objects

Pass 2:
Extract calls for each subtrace
Scalability

Bottleneck: Large execution traces

Pass 1:
Find core objects and associated objects

Pass 2:
Extract calls for each subtrace

Mine millions of events in a few minutes
Examples

ZipFile and Enumeration

new ZipFile → f
f.entries → e
e.hasMoreElements

f.close

e.hasMoreElements

e.nextElement
Examples

URL and InputStream

new URL → u

u.openStream → s

s.close
Evaluation

Are examples convincing enough?

20+ mining approaches

OK/ Not OK

Use for some task
Evaluation

Are examples convincing enough?

Neither reproducible nor comparable!
Evaluation Framework

Precision and recall
Evaluation Framework

Precision and recall

Method constraint groups

OK
Example

Mined protocol M

Reference protocol

locale, out, ioException

Formatter

format

flush

close

format

close

printIn

Formatter

close

printIn

Formatter

close

Formatter
Example

Mined protocol M

Reference protocol

precision:

how much of M is correct? → 23%
Example

Mined protocol M

Reference protocol

Recall:
How complete is M?
→ 9%
12 training programs
32 reference protocols
Learn More from Many Teachers?

Empirical study:

How does mining more programs influence the results?

12 programs
API Coverage

Types and methods covered in mined protocols

![Graph showing API coverage](image-url)
API Coverage

Types and methods covered in mined protocols

More programs
Higher coverage
Recall

Recall of mined protocols

![Graph showing recall of mined protocols.](graph.png)
Recall

Recall of mined protocols

More programs ↓
Higher recall
Part 2: Bug Finding
Bug Finding

Training Programs

API

Mining

Target Program

Bug finding

x

y

z

25
Bug Finding

Training Programs

API

Mining

Target Program

Bug finding

x

y

z
Overview

Analysis
Overview

Analysis

Runtime verification

Static checking
Overview

What is a protocol violation?

Runtime verification

Static checking
Overview

What is a protocol violation?

Runtime verification

Static checking

Analysis
Incomplete Specifications

Protocols = Incomplete specifications
Incomplete Specifications

Protocols = Incomplete specifications

```
new LinkedList → l
l.add
l.iterator → i
i.hasNext
i.next
i.hasNext
i.hasNext
```

27
What is a Protocol Violation?

Setup phase:
- bind parameters

Liable phase:
- all parameters bound
- violation:
  - take non-existing transition
  - end in non-final state
State Partitioning

Protocol transformation:
Setup states vs. liable states
State Partitioning

Protocol transformation:
Setup states vs. liable states
State Partitioning

Protocol transformation: Setup states vs. liable states
State Partitioning

Protocol transformation:
Setup states vs. liable states

new LinkedList → l

ambiguous → split

l.add → i

l.iterator → i

i.hasNext

l.add

i.hasNext

i.next

l.add
State Partitioning

Protocol transformation:
Setup states vs. liable states

new LinkedList → l
l.add
l.iterator → i
i.hasNext

l.add
i.hasNext
i.next

l.add
l.add

l.add
State Partitioning

Protocol transformation:
Setup states vs. liable states
Overview

Analysis

Runtime verification

Static checking
Overview

Analysis

Runtime verification

Static checking
Dynamic Protocol Checking

Input → Runtime verification (JavaMOP) → Output
Dynamic Protocol Checking

Challenge 1:
Check many different execution paths
Dynamic Protocol Checking

Challenge 1:
Check many different execution paths

Challenge 2:
Monitoring mined protocols

Runtime verification (JavaMOP)
Randomly Generated Input

Challenge 1:
Check many different execution paths

Input ..........................
Randomly Generated Input

Challenge 1:
Check many different execution paths

Input \longrightarrow \text{Random test generation}
Randomly Generated Input

Challenge 1:
Check many different execution paths

Input $\rightarrow$ Random test generation $\rightarrow$ Call sequences that trigger an exception
Randomly Generated Input

Challenge 1:
Check many different execution paths

Input

Non-exceptional sequences  Random test generation  Call sequences that trigger an exception
Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```
l = new LinkedList();
```
Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```java
l = new LinkedList();
```
Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```java
l = new LinkedList()
```
Protocol Monitoring

Challenge 2: Monitoring mined protocols

```java
l = new LinkedList();
```

Violation!?
Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```java
l = new LinkedList()
i1 = l.iterator()
i2.next()
```
Protocol Monitoring

Challenge 2: Monitoring mined protocols

```java
l = new LinkedList()
i1 = l.iterator()
i2.next()
```
Challenge 2: Monitoring mined protocols

```java
l = new LinkedList();
i1 = l.iterator();
i2.next();
```
Protocol Monitoring

Challenge 2:
Monitoring mined protocols

```java
l = new LinkedList()
i1 = l.iterator()
i2.next()
```

Violation!?
Protocol Monitoring

Challenge 2: Monitoring mined protocols

```java
l = new LinkedList()
i1 = l.iterator()
i2 = i1.next()
```

*Naive approach gives too many violations*
Explicit Fail Transitions

Fail only in liable states

- new LinkedList → i
- i.iterator → i
- i.hasNext
- i.next
- F
- i.hasNext
- i.next
Explicit Fail Transitions

Fail only in liable states

Violation:
- Reach fail state
- End in non-final, liable state
Evaluation

Questions

■ Find relevant issues by monitoring mined protocols?

■ How useful is generated input?

Setup:
DaCapo benchmarks, 1.6 MLOC Java
## Results

### Randomly generated

<table>
<thead>
<tr>
<th>Program</th>
<th>Test cases</th>
<th>Total</th>
<th>Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>avrora</td>
<td>15,753</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>batik</td>
<td>3,477</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>daytrader</td>
<td>32,446</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eclipse</td>
<td>816</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fop</td>
<td>6,536</td>
<td>52</td>
<td>50</td>
</tr>
<tr>
<td>h2</td>
<td>7,584</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>lucene</td>
<td>1,985</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pmd</td>
<td>1,286</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sunflow</td>
<td>4,300</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tomcat</td>
<td>14,627</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>xalan</td>
<td>21,083</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>160,857</strong></td>
<td><strong>74</strong></td>
<td><strong>63</strong></td>
</tr>
</tbody>
</table>

**Bug (exception, unexpected behavior) or code smell (performance/maintainability problem)**
## Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Test cases</th>
<th>Total</th>
<th>Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>avrora</td>
<td>15,753</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>batik</td>
<td>3,477</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>daytrader</td>
<td>32,446</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eclipse</td>
<td>816</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fop</td>
<td>6,536</td>
<td>52</td>
<td>50</td>
</tr>
<tr>
<td>h2</td>
<td>7,584</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>lucene</td>
<td>1,985</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pmd</td>
<td>1,286</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sunflow</td>
<td>4,300</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tomcat</td>
<td>14,627</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>xalan</td>
<td>21,083</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Sum</strong></td>
<td><strong>160,857</strong></td>
<td><strong>74</strong></td>
<td><strong>63</strong></td>
</tr>
</tbody>
</table>

Randomly generated

<table>
<thead>
<tr>
<th>Protocol violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bug (exception, unexpected behavior) or code smell (performance/maintainability problem)</td>
</tr>
</tbody>
</table>

85% true positives
try {
    is = u.openStream();
    r = new InputStreamReader(is, "UTF-8");
    br = new BufferedReader(r);
} finally {
    if ( is != null ){
        try { is.close(); } catch ( IOException ignored ){
            is = null;
        }
    }
    if ( r != null ){
        try{ r.close(); } catch ( IOException ignored ){
            r = null;
        }
    }
    if ( br == null ){
        try{ br.close(); } catch ( IOException ignored ){
            br = null;
        }
    }
}
try {
    is = u.openStream();
    r = new InputStreamReader(is, "UTF-8");
    br = new BufferedReader(r);
} finally {
    if (is != null) {
        try { is.close(); } catch (IOException ignored) {}
        is = null;
    }
    if (r != null) {
        try { r.close(); } catch (IOException ignored) {}
        r = null;
    }
    if (br == null) {
        try { br.close(); } catch (IOException ignored) {}
        br = null;
    }
}

Reader never closed
Examples

Iterator i = pinConnections.iterator();

PinLink currLink = (PinConnect.PinLink) i.next();
currLink.propagateSignals();

while (i.hasNext()) {
    currLink = (PinConnect.PinLink) i.next();
currLink.propagateSignals();
}


Examples

Iterator i = pinConnections.iterator();

PinLink currLink = (PinConnect.PinLink) i.next();
currLink.propagateSignals();

while (i.hasNext()) {
    currLink = (PinConnect.PinLink) i.next();
    currLink.propagateSignals();
}

Incorrect iterator usage
Normal vs. Generated Input

Benchmark input only
Both
Randomly generated input only

Violations

Reader Writer Input Stream Output Stream CollIter JDBC1 JDBC2 Zip
Overview

Analysis

Runtime verification

Static checking
Overview

Analysis

Runtime verification

Static checking
State of the Art

+ specification

Typestate checking

Anomaly detection
State of the Art

Typestate checking

+ Precise
- Needs specification

Anomaly detection

+ Automatic
- Imprecise
State of the Art

+ specification

Typestate checking

-Anomaly detection

+ Precise
- Needs specification

+ Automatic
- Imprecise

Combine both!
Precise checker for mined multi-object protocols
Static Protocol Checking

Joint work with Ciera Jaspan and Jonathan Aldrich (ISR, CMU)
Static analysis to check API usages

Good match with mined protocols:

- Reasons about *interacting objects*
- Distinguishes setup from checking
Relationship-based Analysis

Relationships = Tuples of objects

```c
void m() {
    Effects: Add/remove objects
    ...
}

Requirements: Check before call
```
Relationship-based Analysis

Relationships = Tuples of objects

void m() {
  
  Effects: Add/remove objects

  ..

}  

Requirements: Check before call

Keep track of protocol execution (e.g., current state)

Check protocol constraints if in liable state
Example

```java
LinkedList l = new LinkedList();

Iterator i = l.iterator();

i.next();
```
Example

```java
LinkedList l = new LinkedList();

l ∈ r_{state2}, l ∈ r_{iterator}

Iterator i = l.iterator();

i.next();
```
Example

**Example**

```java
LinkedList l = new LinkedList();

l ∈ r_{state2}, l ∈ r_{iterator}

Iterator i = l.iterator();

l ∈ r_{state3}, i ∈ r_{state3}, i ∈ r_{hasNext}, (l, i) ∈ r_{protocol}

i.next();
```

![Diagram showing the example with states and transitions]
Example

```java
LinkedList l = new LinkedList();

l ∈ r_{state2}, l ∈ r_{iterator}

Iterator i = l.iterator();

l ∈ r_{state3}, i ∈ r_{state3}, i ∈ r_{hasNext}, (l, i) ∈ r_{protocol}

i.next();

i ∉ r_{next}
```

Diagram:

1. `new LinkedList` → l
2. i.iterator → i
3. i.hasNext
4. i.next
# Results

## Program Warnings

<table>
<thead>
<tr>
<th>Program</th>
<th>Total</th>
<th>Bugs</th>
<th>Code smells</th>
<th>True pos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>avrora</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>69%</td>
</tr>
<tr>
<td>batik</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>daytrader</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>eclipse</td>
<td>15</td>
<td>2</td>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>fop</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>69%</td>
</tr>
<tr>
<td>h2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>jython</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>43%</td>
</tr>
<tr>
<td>lucene</td>
<td>13</td>
<td>3</td>
<td>3</td>
<td>46%</td>
</tr>
<tr>
<td>pmd</td>
<td>15</td>
<td>2</td>
<td>8</td>
<td>67%</td>
</tr>
<tr>
<td>sunflow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>tomcat</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>xalan</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>81</strong></td>
<td><strong>26</strong></td>
<td><strong>15</strong></td>
<td><strong>51%</strong></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Program</th>
<th>Total</th>
<th>Bugs</th>
<th>Code smells</th>
<th>True pos.</th>
</tr>
</thead>
<tbody>
<tr>
<td>avrora</td>
<td>13</td>
<td>9</td>
<td>0</td>
<td>69%</td>
</tr>
<tr>
<td>batik</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>daytrader</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>eclipse</td>
<td>15</td>
<td>2</td>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>fop</td>
<td>13</td>
<td>8</td>
<td>1</td>
<td>69%</td>
</tr>
<tr>
<td>h2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>jython</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td>43%</td>
</tr>
<tr>
<td>lucene</td>
<td>13</td>
<td>3</td>
<td>3</td>
<td>46%</td>
</tr>
<tr>
<td>pmd</td>
<td>15</td>
<td>2</td>
<td>8</td>
<td>67%</td>
</tr>
<tr>
<td>sunflow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>tomcat</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>xalan</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>81</strong></td>
<td><strong>26</strong></td>
<td><strong>15</strong></td>
<td><strong>51%</strong></td>
</tr>
</tbody>
</table>

Few false positives!
Examples

```java
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while (i.hasNext()) {
    PinLink curr = (PinLink) i.next();
    if (...)
        pinConnections.remove(curr);
}
```
Examples

```java
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();
while ( i.hasNext() ) {
    PinLink curr = (PinLink) i.next();
    if ( ... ) {
        pinConnections.remove(curr);
    }
}
```

Concurrent modification
Examples

BufferedReader in = null;

try {
    in = new BufferedReader(...);
    ...
    in.close();
} finally {
    if (in != null) {
        try { in.close(); }
        catch (IOException e) { ... }
    }
}
Examples

BufferedReader in = null;
try {
    in = new BufferedReader(...);
    ...
    in.close();
} finally {
    if (in != null) {
        try { in.close(); } catch (IOException e) { ... }
    }
}

Duplicate close
Summary: Bug Finding

Clear definition of protocol violation

Static and dynamic checking:
- Both are practical
- Complement each other
Conclusion

An attractive tool to help programmers

- understand large systems
- pinpoint problem areas

Supplements other approaches:

- Reveals multi-object bugs
- Easy to use
Thank you!
michael@binaervarianz.de
What is a Violation?

Negative protocols vs. Positive protocols
What is a Violation?

Negative protocols

- Reaching final state

Positive protocols

- Taking non-existing transition
- Not reaching final state
Related Work (Mining)

- **Gabel & Su, FSE 2008**
  - Language learning algorithm with pre-defined micro-patterns
  - Don’t consider dataflow

- **Lee, Chen & Rosu, ICSE 2011**
  - First, learn related events; then, mine sliced trace with PFSA learner
  - Require unit tests for first step
Template
Template
Template
Template