Statically Checking API Protocol Conformance with Mined Multi-Object Specifications

Michael Pradel\textsuperscript{1}, Ciera Jaspan\textsuperscript{2}, Jonathan Aldrich\textsuperscript{3}, and Thomas R. Gross\textsuperscript{1}

\textsuperscript{1} ETH Zurich
\textsuperscript{2} Cal Poly Pomona
\textsuperscript{3} Carnegie Mellon University
Motivating Example

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

(from DaCapo benchmarks)
Motivating Example

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

(from DaCapo benchmarks)
Motivating Example

```java
LinkedList pinConnections = ...;
Iterator i = pinConnections.iterator();

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

(from DaCapo benchmarks)
Motivating Example

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

(from DaCapo benchmarks)
Motivating Example

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

(from DaCapo benchmarks)
Motivating Example

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

Don’t modify a collection while iterating over it!

(from DaCapo benchmarks)
The Problem

FSMs that capture constraints like:

- Call x before y
- Don’t call y after calling x
- Don’t call x while calling y and z
Multi-Object Protocols

Many protocols involve multiple objects
Multi-Object Protocols

Many protocols involve multiple objects

Collection + Iterator
Multi-Object Protocols

Many protocols involve multiple objects

- Collection + Iterator
- Condition + ReentrantLock
Multi-Object Protocols

Many protocols involve multiple objects

- Collection + Iterator
- Condition + ReentrantLock
- FileReader + BufferedReader
Many protocols involve multiple objects

How to find incorrect API usages that involve multiple objects?

- Collection + Iterator
- FileReader + BufferedReader
- Condition + ReentrantLock
This Talk

Automatic detection of multi-object protocol bugs
This Talk

Don’t require API specs

Automatic detection of multi-object protocol bugs
This Talk

Automatic detection of multi-object protocol bugs

Warnings about:
- Missing calls
- Incorrect calls
State of the Art

Type state checking
DeLine + Fähndrich ’04
Fink et al., ’08
Bierhoff + Aldrich ’07
Naeem + Lhotak, ’08
Bodden ’10

Anomaly Detection
Whaley et al. ’02
Gabel + Su ’10
Wasylkowski + Zeller ’09
Nguyen et al. ’09
Open Issues

Type state checking:  
- Needs API specs
- Sound but incomplete

Anomaly detection:  
- Single-object
- Missing calls
  (not: incorrect calls)
Open Issues

Type state checking:
- Needs API specs
- Sound but incomplete

Anomaly detection:
- Single-object
- Missing calls
  (not: incorrect calls)

Our contributions:
- Multi-object
- Missing calls and incorrect calls
- Complete checking
Approach Overview

Training programs

Dynamic protocol miner

Target programs

Static checker

Protocols

Warnings
Approach Overview

- Training programs
- Dynamic protocol miner
- Static checker

- Target programs
- Dynamic protocol miner
- Static checker

Problem: No complete static checker for multi-object protocols
Approach Overview

Training programs

Dynamic protocol miner

Protocols

Target programs

Static checker

Relationship annotations

Use relationship-based API usage checker

Warnings
Approach Overview

Challenge: Map protocols to relationships

Dynamic protocol miner

Protocols

Relationship annotations

Warnings

Static checker

Target programs

Training programs
Approach Overview

- Training programs
  - Dynamic protocol miner
  - Protocols
  - Translation
  - Relationship annotations
  - Static checker
  - Warnings
- Target programs
Approach Overview

Training programs

Dynamic protocol miner

Translation

Static checker

Target programs

Protocols

Relationship annotations

Challenge: False positives

Warnings
Approach Overview

- Training programs
  - Dynamic protocol miner
    - Protocols
  - Translation
    - Relationship annotations
  - Static checker
    - Potential warnings
- Target programs
  - Pruner
    - Reported warnings
Approach Overview

Training programs

Dynamic protocol miner

Protocols

[Pradel et al., ASE’09, ICSM’10]

Protocol miner

Translation

Relationship annotations

[Jaspan and Aldrich, ECOOP’09]

Static checker

Potential warnings

Translation Pruner

Pruner

Reported warnings
Approach Overview

Training programs

Dynamic protocol miner

Protocols

Translation

Relationship annotations

Static checker

Potential warnings

Pruner

Reported warnings

[This paper]
Example: Inferred Protocol

Two objects

`LinkedList` + `Iterator`
Example: Inferred Protocol

- **bind protocol parameters**
  - new LinkedList → list
  - list.update
  - list.iterator → iter
  - iter.hasNext
  - iter.next

- Iteration process:
  - list.update
  - iter.hasNext
  - iter.next
Example: Inferred Protocol

new LinkedList \rightarrow \text{list} \rightarrow \text{iter} \rightarrow \text{iter}.hasNext \rightarrow \text{iter}.next \rightarrow \text{list.update} \rightarrow \text{valid iterator}
Example: Inferred Protocol

1. new LinkedList
2. list.update
3. list.iterator
4. iter.hasNext
5. iter.hasNext
6. iter.hasNext

iterator invalidated
Example: Checking

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

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

Initially:

All objects in state 1
Example: Checking

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

Bind list to protocol parameter list
Example: Checking

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

New state: 2
Example: Checking

```java
LinkedList pinConnections = new LinkedList(...);

Iterator i = pinConnections.iterator();

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

Bind iterator to protocol parameter iter
Example: Checking

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

New state: 3
Example: Checking

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

All objects bound:
Start checking
Example: Checking

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

Call is legal in state 3
Example: Checking

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

New state: 4
Example: Checking

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

Call is legal in state 4
Example: Checking

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

New state: 3
Example: Checking

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

Call is legal in state 3
Example: Checking

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

New state: 5
Example: Checking

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

Call is illegal in state 5
Protocol violation!
LinkedList pinConnections = new LinkedList(...);
Iterator i = pinConnections.iterator();
while (i.hasNext()) {
    PinLink curr = (PinLink) i.next();
    if (...) {
        pinConnections.remove(curr);
    }
}

**Multi-Object: Essential (1)**

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

**Only LinkedList: No warning**
```java
LinkedList pinConnections = new LinkedList(...);
Iterator i = pinConnections.iterator();
while (i.hasNext()) {
    PinLink curr = (PinLink) i.next();
    if (...) {
        pinConnections.remove(curr);
    }
}
```

**Only Iterator: No warning**
Multi-Object: Essential (1)

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

Need LinkedList and Iterator to find bug
```
`LinkedList l = new LinkedList(...);
Iterator i = l.iterator();
if (l.size() > 0) {
    .. = i.next();
}

Linked List l = new LinkedList(...);
Iterator i = l.iterator();
if (l.size() > 0) {
    .. = i.next();
}

Only Iterator: False warning
Multi-Object: Essential (2)

```java
LinkedList l = new LinkedList(...);
Iterator i = l.iterator();
if (l.size() > 0) {
    .. = i.next();
}
```

**LinkedList and Iterator:** No warning
Multi-Object Protocols

Benefits of multi-object protocols

1. Find bugs missed by single-object protocols
2. Reduce false positives
Implementation

Automatic tool to check Java programs

- Currently, two object-protocols only
- Prototype: Not tuned for performance
- 2,500 LoC (Scala)
  + existing miner and checker
Evaluation

Effective in finding protocol violations

Importance of multi-object protocols

Comparison with existing approaches
Setup

**APIs:** java.* and javax.*

**Protocol mining**
- 223 automatically mined protocols
- 146 types, 26 packages

**Checking**
- DaCapo benchmarks (1.6 MLoC)
- Warning classification:
  Bug — code smell — false positive
Setup

**APIs:** java.* and javax.*

**Protocol mining**
- 223 automatically mined protocols
- 146 types, 26 packages

**Checking**
- DaCapo benchmarks (1.6 MLoC)
- Warning classification:
  - Bug — code smell — false positive
Setup

**APIs**: java.* and javax.*

**Protocol mining**
- 223 automatically mined protocols
- 146 types, 26 packages

**Checking**
- DaCapo benchmarks (1.6 MLoC)
- Warning classification:
  Bug — code smell — false positive
Setup

**APIs:** java.* and javax.*

**Protocol mining**
- 223 automatically mined protocols
- 146 types, 26 packages

**Checking**
- DaCapo benchmarks (1.6 MLoC)
- Warning classification:
  - Bug — code smell — false positive
Example: Bug

Map comparators = ...

Iterator i = comparators.values().iterator();
for (Comparator c = (Comparator) i.next();
c != null;
c = (Comparator) i.next()) {
	...
}
Example: Bug

Map comparators = ...

Iterator i = comparators.values().iterator();
for (Comparator c = (Comparator) i.next(); c != null;
    c = (Comparator) i.next()) { ... }

Illegal iterator usage

(from Avrora)
Example: Code Smell

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

(from PMD)
Example: Code Smell

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

Duplicate close

(from PMD)
81 reported warnings, 51% true positives

- 26 bugs
- 15 code smells
- 40 false warnings
Precision

- Incomplete protocols
- Special protocol semantics
- Imprecise points-to analysis

40 false warnings
Which Bugs Do We Miss?

50 randomly seeded protocol bugs

Find 35 of them (70%)

15 missed bugs:
- Conservatism of static checker (6)
- Protocol too permissive (5)
- Protocol not expressive enough (4)
Are multi-object protocols important?

- **Protocols:**
  61% involve multiple objects

- **Violations:**
  44% only found with multi-object protocol
Comparison to Prior Work

Bugs found:

Nguyen et al., ESEC/FSE’09 *

Wasylkowski +Zeller, ASE’09 *

Our work

* java.* and javax.* bugs only
Comparison to Prior Work

Bugs found:

Nguyen et al., ESEC/FSE’09 *

- Our work: 4
- Our work: 6

Wasylkowski +Zeller, ASE’09 *

- 2 (missing calls)
- 3 (incorrect calls)
- 9 (our work)

* java.* and javax.* bugs only

Why?

- Missing calls vs. incorrect calls
- Must infer the ”right” specification
Conclusion

Mined protocols: Good basis for automatic bug finding

Multi-object: Important in practice

API usage: (Still) a difficult problem
Thank you!

Statically Checking API Protocol Conformance with Mined Multi-Object Specifications
Michael Pradel, Ciera Jaspan, Jonathan Aldrich, and Thomas R. Gross

Bugs and protocols for download:
http://mp.binaervarianz.de/icse2012-statically/