Mining Operational Preconditions

Andrzej Wasylkowski • Andreas Zeller
Saarland University
aspectj  

crosscutting objects for better modularity
@interface A {
}

aspect Test {
  declare @field : @A int var* : @A;
  declare @field : int var* : @A;

  interface Subject {
    public int Subject.vara;
    public int Subject.varb;
  }
}

class X implements Test.Subject {
}
java.util.NoSuchElementException
at java.util.AbstractList$Itr.next(AbstractList.java:427)
at org.aspectj.weaver.bcel.BcelClassWeaver.
    weaveAtFieldRepeatedly(BcelClassWeaver.java:1016)
...

java.util.NoSuchElementException
  at java.util.AbstractList$Itr.next(AbstractList.java:427)
  at org.aspectj.weaver.bcel.BcelClassWeaver.
      weaveAtFieldRepeatedly(BcelClassWeaver.java:1016)
...

jc Stack Trace
weaveAtFieldRepeatedly

```java
for (Iterator it = c1.iterator(); it.hasNext();) {
    E e1 = (E) it.next();
    ...
    for (Iterator it2 = c2.iterator(); it.hasNext();) {
        E e2 = (E) it2.next();
        ...
    }
}
```
weaveAtFieldRepeatedly

```java
for (Iterator it = c1.iterator(); it.hasNext();) {
    E e1 = (E) it.next();
    ...
    for (Iterator it2 = c2.iterator(); it2.hasNext();) {
        E e2 = (E) it2.next();
        ...
    }
}
```
weaveAtFieldRepeatedly

```java
for (Iterator it = c1.iterator(); it.hasNext();) {
    E e1 = (E) it.next();
    ...
    for (Iterator it2 = c2.iterator(); it.hasNext();) {
        E e2 = (E) it2.next();
        ...
    }
}
```

- Invalid iterator usage: `hasNext()` should precede `next()`
Preconditions

• Invoking next() with no next element violates a precondition

• Traditional preconditions are axiomatic – describing the state of the system

• How do we reach that state?
Preconditions

close (int fildes)
Preconditions

close (int fildes)

- **Axiomatic**: fildes is a valid file descriptor
Preconditions

close (int fildes)

- **Axiomatic**: `fildes` is a valid file descriptor
- **Operational**: `fildes` stems from a call to `open()` with `read()` and `write()` calls in between
Preconditions

- **Axiomatic:** `fildes` is a valid file descriptor
- **Operational:** `fildes` stems from a call to `open()` with `read()` and `write()` calls in between
- Can we learn and check operational preconditions?

```c
close (int fildes)
```
OP-Miner

**Object usage models**

- `iter.hasNext ()`
- `iter.next ()`

**Sequential constraints**

- `hasNext < next`
- `hasNext < hasNext`
- `next < hasNext`
- `next < next`

**Violations**

- `hasNext < next`
- `hasNext < hasNext`

**Operational preconditions**

- `hasNext < next`
- `hasNext < hasNext`
OP-Miner

Object usage models

iter.hasNext () iter.next ()

Sequential constraints

hasNext < next
hasNext < hasNext
next < hasNext
next < next

Violations

hasNext < next
hasNext < hasNext

Operational preconditions

hasNext < next
hasNext < hasNext
public List gPL (Set ps) {
    log ("IN: gPL: " + ps);
    List l = new ArrayList ();
    createPList (this.cl, l);
    Iterator it = ps.iterator ();
    while (it.hasNext ()) {
        Property p = it.next ();
        addP (p, l);
    }
    reapPList (l);
    log ("OUT: gPL: " + l);
    return l;
}
public List gPL(Set ps) {
    log("IN: gPL: " + ps);
    List l = new ArrayList();
    createPList(this.cl, l);
    Iterator it = ps.iterator();
    while (it.hasNext()) {
        Property p = it.next();
        addP(p, l);
    }
    reapPList(l);
    log("OUT: gPL: " + l);
    return l;
}
Creating a usage model

```java
log("IN: gPL: " + ps);
List l = new ArrayList();
createPList(this.cl, l);
Iterator it = ps.iterator();

addP(p, l);
reapPList(l);
log("OUT: gPL: " + l);
```
Creating a usage model

```java
Iterator it = ps.iterator ();
```

```
Property p = it.next ();
```

```
it.hasNext ()
```

```
it.hasNext ()
```
Creating a usage model

```
RETVAL: Set.iterator ()

it.hasNext ()

it.next ()
```
Creating a usage model

```java
log ("IN: gPL: " + ps);

List l = new ArrayList ();
createPList (this.cl, l);

Iterator it = ps.iterator ();
it.hasNext ();

addP (p, l);

reapPList (l);

Property p = it.next ();

log ("OUT: gPL: " + l);
```
Creating a usage model

```java
List l = new ArrayList();
createPList(this.cl, l);
addP(p, l);
reapPList(l);
log("OUT: gPL: " + l);
```
Creating a usage model

l.<init> ()

ASTNode.createPList (... , l)

ASTNode.addP (... , l)

ASTNode.reapPList (l)

StringBuffer.add (l)
Creating a usage model

```
ASTNode.<init> ()
ASTNode.createPList (..., l)
ASTNode.addP (... , l)
ASTNode.reapPList (l)
StringBuffer.add (l)
```
OP-Miner

Object usage models

Sequential constraints

Operational preconditions

Violations
OP-Miner

Object usage models

iter.hasNext ()
iter.next ()

Sequential constraints

hasNext < next
hasNext < hasNext
next < hasNext
next < next

Violations

hasNext < next
hasNext < hasNext

Operational preconditions

hasNext < next
hasNext < hasNext
Extracting sequential constraints

l.<init> ()
ASTNode.createPList (..., l)
ASTNode.addP (... , l)
ASTNode.reapPList (l)
Extracting sequential constraints

l.<init> ()

ASTNode.createPList (..., l)

ASTNode.addP (..., l)

ASTNode.reapPList (l)

l.<init> () < createPList (..., l)

l.<init> () < addP (..., l)

l.<init> () < reapPList (l)

createPList (..., l) < addP (..., l)

createPList (..., l) < reapPList (l)

addP (..., l) < addP (..., l)

addP (..., l) < reapPList (l)
### Callers vs. constraints

<table>
<thead>
<tr>
<th>Callers</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Callers vs. constraints

This forms an operational precondition
Detecting violations

<table>
<thead>
<tr>
<th>Callers</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential constraints</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>S4</td>
</tr>
</tbody>
</table>

...
Detecting violations

<table>
<thead>
<tr>
<th>Callers</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Sequential constraints
Detecting violations

<table>
<thead>
<tr>
<th>Sequential constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Callers</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
</tr>
<tr>
<td>C2</td>
</tr>
<tr>
<td>C3</td>
</tr>
<tr>
<td>C4</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

...
Detecting violations

<table>
<thead>
<tr>
<th>Callers</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sequential constraints
Detecting violations

Sequential constraints

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Callers

S1  S2  S3  S4  ...

...
Detecting violations

<table>
<thead>
<tr>
<th>Callers</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sequential constraints

![Image]

The diagram illustrates the relationships between callers and sequential constraints. A red 'X' indicates a violation.
Case study: AspectJ

- Compiler for the AspectJ language
- 36,045 methods in 2,957 classes
- 1,154 methods with OP support ≥ 20
- 300 violations found in 8 minutes
A defect

```java
for (Iterator it = c1.iterator(); it.hasNext();) {
    E e1 = (E) it.next();
    ...
    for (Iterator it2 = c2.iterator(); it2.hasNext();) {
        E e2 = (E) it2.next();
        ...
    }
    ...
}
```
A defect

```java
for (Iterator it = c1.iterator(); it.hasNext();) {
    E e1 = (E) it.next();
    ...
    for (Iterator it2 = c2.iterator(); it.hasNext();) {
        E e2 = (E) it2.next();
        ...
    }
}```
Another defect

```java
public void visitNEWARRAY (NEWARRAY o) {
    byte t = o.getTypecode ();
    if (! ( (t == Constants.T_BOOLEAN) ||
            (t == Constants.T_CHAR) ||
            ...
            (t == Constants.T_LONG) ) ) {
        constraintViolated (o, "(...) '+t+' (...)");}
}
```
Another defect

public void visitNEWARRAY (NEWARRAY o) {
  byte t = o.getTypecode ();
  if (! ( (t == Constants.T_BOOLEAN) ||
         (t == Constants.T_CHAR) ||
         ...
         (t == Constants.T_LONG) ) ) {
    constraintViolated (o, "(...) +t+ (...)");
  }
}

should be using double quotes
A false positive

Name internalNewName (String[] identifiers) {
    ...
    for (int i = 1; i < count; i++) {
        SimpleName name = new SimpleName (this);
        name.internalSetIdentifier (identifiers[i]);
        ...
    }
    ...
}
A false positive

Name internalNewName (String[] identifiers) {
    ...
    for (int i = 1; i < count; i++) {
        SimpleName name = new SimpleName (this);
        name.internalSetIdentifier (identifiers[i]);
        ...
    }
    ...
}
public String getRetentionPolicy () {
    ...
    for (Iterator it = ...; it.hasNext();)
        ...
            = it.next();
            ...
    return retentionPolicy;
}
A code smell

```java
public String getRetentionPolicy () {
    ...
    for (Iterator it = ...; it.hasNext();)
    {
        ... = it.next ();
    ...
    return retentionPolicy;

    } should be fixed

    }
}
```
AspectJ violations

- Defects
- Code smells
- False positives
AspectJ violations

- Defects: 16
- Code smells: 42
- False positives
AspectJ violations

- Defects: 16
- Code smells: 42
- False positives: 242
More results

- AspectJ 1.5.3
- Azureus 2.5.0.0
- Apache Tomcat 6.0.16
- ArgoUML 0.24
- Columba 1.2
- jEdit 4.2
- Act-Rbot 0.8.2

- Defects
- Code smells
- False positives
Future work
Future work

• Procedural languages
Future work

• Procedural languages
• Interprocedural analysis
Future work

- Procedural languages
- Interprocedural analysis
- Ranking violations
Future work

• Procedural languages
• Interprocedural analysis
• Ranking violations
• Early programmer support
OP-Miner

• OP-Miner learns *operational preconditions*
• Learns from normal argument usage
• Fully automatic
• Found dozens of verified defects