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 {}
private static void a jc$postClinit() org.aspectj.weaver.AjAttribute$AjSynthetic@abfbc6 :

    NEW Test (line 1)
    DUP
    INVOKEVIRTUAL Test.<init> ()V
    PUTSTATIC Test.a jc$perSingletonInstance LTest;
    RETURN

    end private static void a jc$postClinit()
end public class Test

Exception thrown from AspectJ 1.5.3

This might be logged as a bug already -- find current bugs at

Bugs for exceptions thrown have titles File:line from the top stack,
e.g., "SomeFile.java:243"

If you don't find the exception below in a bug, please add a new bug
at http://bugs.eclipse.org/bugs/enter_bug.cgi?product=AspectJ
To make the bug a priority, please include a test program
that can reproduce this exception.
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)
weaveAtFieldRepeatedly

for (Iterator iter = itdFields.iterator();
     iter.hasNext();)
{
  ...
  for (Iterator iter2 = worthRetrying.iterator();
       iter.hasNext();)
  {
    ...
  }
}
weaveAtFieldRepeatedly

for (Iterator iter = itdFields.iterator();
    iter.hasNext());) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
        iter2.hasNext();) {
        ...
        should be iter2
    }
}
weaveAtFieldRepeatedly

for (Iterator iter = itdFields.iterator();
     iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
         iter.hasNext();) {
        ...
        should be iter2
    }
}

• Invalid iterator usage:
  hasNext() should precede next()
Preconditions
Preconditions

🌟 Invoking `next()` with no next element violates a **precondition**
Preconditions

- Invoking `next()` with no next element violates a precondition

- Traditional preconditions are axiomatic – describing the state of the system
Preconditions

- Invoking next() with no next element violates a precondition.
- Traditional preconditions are axiomatic – describing the state of the system.
- How do we reach this state?
Preconditions
Preconditions

close(int fildes)
Preconditions

close(int fildes)

• **Axiomatic:** *fildes* is a valid file descriptor
Axiomatic: `fildes` is a valid file descriptor

Operational: `fildes` stems from a call to `open()` with `read()` and `write()` calls in between
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

• Can we check operational preconditions?
OP-Miner
OP-Miner

Usage Models

Program

iter.hasNext ()
iter.next ()
OP-Miner

Usage Models

iter.hasNext () iter.next ()

Temporal Properties

hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next
OP-Miner

Usage Models

iter.hasNext () iter.next ()

Temporal Properties

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

Patterns

hasNext < next
hasNext < hasNext
OP-Miner

Usage Models

iter.hasNext () iter.next ()

Temporal Properties

hasNext ≺ next
hasNext ≺ hasNext
next ≺ hasNext
next ≺ next

Anomalies

hasNext ≺ next
✓ hasNext ≺ hasNext
✓ hasNext ≺ hasNext
✓ hasNext ≺ hasNext

Patterns

hasNext ≺ next
hasNext ≺ hasNext
OP-Miner

Usage Models

Usage Models

Temporal Properties

Temporal Properties

Anomalies

Anomalies

Patterns

Patterns
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i ++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
public Stack createStack () {
    Random r = new Random ();
    int n = r.nextInt ();
    Stack s = new Stack ();
    int i = 0;
    while (i < n) {
        s.push (rand (r));
        i++;
    }
    s.push (-1);
    return s;
}
Random r = new Random();

int n = r.nextInt();

Stack s = new Stack();

int i = 0;

i < n

s.push (-1);

s.push (rand (r));

i++;

Usage Models
Stack s = new Stack();

s.push(-1);
s.push(rand(r));
Usage Models

s.<init>()

s.push (_)

s.push (_)

s.push (_)

s.push (_)

s.<init>()
Random r = new Random ();
int n = r.nextInt ();
Stack s = new Stack ();
int i = 0;
i < n
s.push (-1);
s.push (rand (r));
i++;

Random r = new Random ();

int n = r.nextInt ();

s.push (rand (r));
Usage Models

\[ r.\text{init}() \]

\[ r.\text{nextInt}() \]

\[ \text{Utils.rand}(r) \]
JPanel.add()
ASTNode.reapPropertyList()

list.<init>

ASTNode.createPropertyList (... , list)

ASTNode.addProperty (... , list)

ASTNode.reapPropertyList (list)
Resource.getFlags()
OP-Miner

Usage Models

iter.hasNext ()  iter.next ()

Temporal Properties

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

Anomalies

hasNext < next ✓
hasNext < hasNext ✓

Patterns

hasNext < next
hasNext < hasNext
## Methods vs. Properties

### Temporal Properties

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>start ≺</td>
<td>lock ≺</td>
</tr>
<tr>
<td>stop</td>
<td>eof ≺</td>
</tr>
<tr>
<td>lock</td>
<td>close</td>
</tr>
<tr>
<td>unlock</td>
<td></td>
</tr>
</tbody>
</table>
Methods vs. Properties

<table>
<thead>
<tr>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>start ≺ lock ≺ eof</td>
</tr>
<tr>
<td>stop ≺ unlock ≺ close</td>
</tr>
</tbody>
</table>

Methods

- get()
## Methods vs. Properties

<table>
<thead>
<tr>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>start ≺ stop</td>
</tr>
<tr>
<td>lock ≺ unlock</td>
</tr>
<tr>
<td>eof ≺ close</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>open()</td>
</tr>
<tr>
<td>get()</td>
</tr>
</tbody>
</table>
Methods vs. Properties

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
<td>start &lt; stop</td>
</tr>
<tr>
<td>open()</td>
<td>lock &lt; unlock</td>
</tr>
<tr>
<td>hello()</td>
<td>eof &lt; close</td>
</tr>
</tbody>
</table>
## Methods vs. Properties

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
<td></td>
</tr>
<tr>
<td>open()</td>
<td>start ≺ lock ≺ eof</td>
</tr>
<tr>
<td>hello()</td>
<td>stop</td>
</tr>
<tr>
<td>parse()</td>
<td>unlock</td>
</tr>
<tr>
<td></td>
<td>close</td>
</tr>
</tbody>
</table>
Methods vs. Properties

Temporal Properties

<table>
<thead>
<tr>
<th></th>
<th>start &lt;</th>
<th>lock &lt;</th>
<th>eof &lt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unlock</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>close</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Methods

- get()
- open()
- hello()
- parse()
# Methods vs. Properties

<table>
<thead>
<tr>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>start ≺ stop</td>
</tr>
<tr>
<td>lock ≺ unlock</td>
</tr>
<tr>
<td>eof ≺ close</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
</tr>
<tr>
<td>open()</td>
</tr>
<tr>
<td>hello()</td>
</tr>
<tr>
<td>parse()</td>
</tr>
</tbody>
</table>

The diagram illustrates the relationships between various methods and temporal properties.
Methods vs. Properties

Temporal Properties

<table>
<thead>
<tr>
<th>start</th>
<th>lock</th>
<th>eof</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>unlock</td>
<td>close</td>
</tr>
</tbody>
</table>

Methods

- get()
- open()
- hello()
- parse()
Methods vs. Properties

Temporal Properties

- start ≺ lock ≺ eof ≺ stop ≺ unlock ≺ close

Methods
- get()
- open()
- hello()
- parse()
Discovering Anomalies

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>start ≺ lock ≺ eof</td>
</tr>
<tr>
<td>get()</td>
<td>stop</td>
</tr>
<tr>
<td>open()</td>
<td>unlock</td>
</tr>
<tr>
<td>hello()</td>
<td>close</td>
</tr>
<tr>
<td>parse()</td>
<td></td>
</tr>
</tbody>
</table>
Discovering Anomalies

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
<td>start ≺ stop</td>
</tr>
<tr>
<td></td>
<td>lock ≺ unlock</td>
</tr>
<tr>
<td></td>
<td>eof ≺ close</td>
</tr>
<tr>
<td>open()</td>
<td></td>
</tr>
<tr>
<td>hello()</td>
<td></td>
</tr>
<tr>
<td>parse()</td>
<td></td>
</tr>
</tbody>
</table>

✘
Discovering Anomalies

<table>
<thead>
<tr>
<th>Methods</th>
<th>get()</th>
<th>open()</th>
<th>hello()</th>
<th>parse()</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Temporal Properties</th>
<th>start</th>
<th>lock</th>
<th>eof</th>
<th>stop</th>
<th>unlock</th>
<th>close</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>&lt;</td>
<td>lock</td>
<td>&lt;</td>
<td>eof</td>
<td>&lt;</td>
<td>close</td>
</tr>
</tbody>
</table>
Discovering Anomalies

<table>
<thead>
<tr>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>start &lt; lock &lt; eof</td>
</tr>
<tr>
<td>stop</td>
</tr>
<tr>
<td>unlock</td>
</tr>
<tr>
<td>close</td>
</tr>
</tbody>
</table>

### Methods

- `get()`: Implies
- `open()`: Implied
- `hello()`: Implied
- `parse()`: Implied
Discovering Anomalies

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
<td>start ≺ stop</td>
</tr>
<tr>
<td>open()</td>
<td>lock ≺ unlock</td>
</tr>
<tr>
<td>hello()</td>
<td>eof ≺ close</td>
</tr>
<tr>
<td>parse()</td>
<td></td>
</tr>
</tbody>
</table>
Discovering Anomalies

<table>
<thead>
<tr>
<th>Methods</th>
<th>Temporal Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>get()</td>
<td>start ≺ lock ≺ eof</td>
</tr>
<tr>
<td>open()</td>
<td>stop</td>
</tr>
<tr>
<td>hello()</td>
<td>unlock</td>
</tr>
<tr>
<td>parse()</td>
<td>close</td>
</tr>
</tbody>
</table>
Case Study: AspectJ
Case Study: AspectJ

- 2,954 classes
Case Study: AspectJ

- 2,954 classes
- 36,045 methods
Case Study: AspectJ

- 2,954 classes
- 36,045 methods
- 1,154 methods with OP support ≥ 20
Case Study: AspectJ

- 2,954 classes
- 36,045 methods
- 1,154 methods with OP support $\geq 20$
- 300 violations found in 8 minutes
Case Study: AspectJ

• 2,954 classes
• 36,045 methods
• 1,154 methods with OP support ≥ 20
• 300 violations found in 8 minutes
• Examined every single one
A Defect

```java
for (Iterator iter = itdFields.iterator();
    iter.hasNext();)
    {
        ...
        for (Iterator iter2 = worthRetrying.iterator();
            iter.hasNext();)
            {
                ...
            }
    }
```
A Defect

```java
for (Iterator iter = itdFields.iterator(); iter.hasNext();) {
    ...
    for (Iterator iter2 = worthRetrying.iterator(); iter.hasNext();) {
        ...
        should be iter2
    }
}
```
A Defect

for (Iterator iter = itdFields.iterator();
    iter.hasNext());) {
    ...
    for (Iterator iter2 = worthRetrying.iterator();
        iter.hasNext());) {
        ...
        should be iter2
    }
}
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+' (...)")
    }
}
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+\" (...");
    }
}
A False Positive

```java
Name internalNewName (String[] identifiers)

... for (int i = 1; i < count; i++) {
    SimpleName name = new SimpleName(this);
    name.internalSetIdentifier(identifiers[i]);
    ...
}
...}
```
A False Positive

```java
Name internalNewName (String[] identifiers)
    ...
    for (int i = 1; i < count; i++) {
        SimpleName name = new SimpleName(this);
        name.internalSetNameIdentifier(identifiers[i]);
        ...
    }
    ...
```
A Code Smell

class

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

- Defects
- Code smells
- False positives
## More Results

<table>
<thead>
<tr>
<th>Program</th>
<th># Violations</th>
<th># Defects</th>
<th># Code smells</th>
<th># False positives</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>Investigated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACT-Rbot 0.8.2</td>
<td>25</td>
<td>25</td>
<td>2</td>
<td>13</td>
<td>60%</td>
</tr>
<tr>
<td>Apache Tomcat 6.0.16</td>
<td>55</td>
<td>55</td>
<td>0</td>
<td>9</td>
<td>16%</td>
</tr>
<tr>
<td>ArgoUML 0.24</td>
<td>305</td>
<td>28</td>
<td>0</td>
<td>12</td>
<td>43%</td>
</tr>
<tr>
<td>AspectJ 1.5.3</td>
<td>300</td>
<td>300</td>
<td>16</td>
<td>42</td>
<td>19%</td>
</tr>
<tr>
<td>Azureus 2.5.0.0</td>
<td>315</td>
<td>85</td>
<td>1</td>
<td>26</td>
<td>32%</td>
</tr>
<tr>
<td>Columba 1.2</td>
<td>57</td>
<td>57</td>
<td>4</td>
<td>15</td>
<td>33%</td>
</tr>
<tr>
<td>jEdit 4.2</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>4</td>
<td>36%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1,068</strong></td>
<td><strong>562</strong></td>
<td><strong>23</strong></td>
<td><strong>121</strong></td>
<td><strong>26%</strong></td>
</tr>
</tbody>
</table>
Future Work
Future Work

• Procedural languages
  leveraging appropriate static analysis frameworks
Future Work

• Procedural languages
  leveraging appropriate static analysis frameworks

• Interprocedural analysis
  but does this make sense for learning usage?
Future Work

• **Procedural languages**
  leveraging appropriate static analysis frameworks

• **Interprocedural analysis**
  but does this make sense for learning usage?

• **Ranking violations**
  in particular in presence of low support
Future Work

• Procedural languages
  leveraging appropriate static analysis frameworks

• Interprocedural analysis
  but does this make sense for learning usage?

• Ranking violations
  in particular in presence of low support

• Early programmer support
  in terms of recommendations and documentation
OP-Miner
OP-Miner learns *operational preconditions*
i.e., how to typically construct arguments
OP-Miner

★ OP-Miner learns *operational preconditions*
  i.e., how to typically construct arguments

★ Learns from normal argument usage
  for specific projects or across projects
OP-Miner

★ OP-Miner learns operational preconditions
  i.e., how to typically construct arguments

★ Learns from normal argument usage
  for specific projects or across projects

★ Fully automatic
OP-Miner

- OP-Miner learns operational preconditions, i.e., how to typically construct arguments
- Learns from normal argument usage for specific projects or across projects
- Fully automatic
- Found dozens of verified defects