Object Usage Models

Andrzej Wasylkowsk
Saarland University
A bug in AspectJ

<table>
<thead>
<tr>
<th>[Tools] Bug#</th>
<th>165631</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>AspectJ</td>
</tr>
<tr>
<td>Component</td>
<td>Compiler</td>
</tr>
<tr>
<td>Status</td>
<td>NEW</td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
</tr>
<tr>
<td>Assigned To</td>
<td>aspectj inbox <a href="mailto:aspectj-inbox@eclipse.org">aspectj-inbox@eclipse.org</a></td>
</tr>
<tr>
<td>Reporter</td>
<td>Andrzej Wasylkowski</td>
</tr>
<tr>
<td>Add CC</td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>Macintosh</td>
</tr>
<tr>
<td>OS</td>
<td>MacOS X</td>
</tr>
<tr>
<td>Version</td>
<td>DEVELOPMENT</td>
</tr>
<tr>
<td>Priority</td>
<td>P3</td>
</tr>
<tr>
<td>Severity</td>
<td>normal</td>
</tr>
<tr>
<td>Target Milestone</td>
<td></td>
</tr>
</tbody>
</table>

**Summary:** It is possible to mark a class as implementing multiple parametrizations of a generic type.

**QA Contact:**

**URL:**

**Status Whiteboard:**

**Keywords:**
A bug in AspectJ

```java
private boolean verifyNoInheritedAlternateParameterization(ResolvedType typeToVerify, ResolvedType newParent, World world) {
    if (typeToVerify.equals(ResolvedType.OBJECT)) return true;
    ResolvedType newParentGenericType = newParent.getGenericType();
    Iterator iter = typeToVerify.getDirectSupertypes();
    while (iter.hasNext()) {
        ResolvedType supertype = (ResolvedType) iter.next();
        if ((supertype.isRawType() && newParent.isParameterizedType()) ||
            (supertype.isParameterizedType() && newParent.isRawType()) &&
            newParentGenericType.equals(supertype.getGenericType())) {
            // new parent is a parameterized type, but this is a raw type
            world.getMessageHandler().handleMessage(new Message(
                WeaverMessages.format(WeaverMessages.CANT_DECYP_MULTIPLE_PARAMETERIZATIONS, 
                    newParent.getName(), typeToVerify.getName(), supertype.getName()),
                getSourceLocation(), true, new ISourceLocation[] { typeToVerify.getSourceLocation() }));
            return false;
        }
        if (supertype.isParameterizedType()) {
            ResolvedType genericType = supertype.getGenericType();
            // If the generic types are compatible but the parameterizations aren't then we have a problem
            if (genericType.isAssignableFrom(newParentGenericType) &&
                supertype.isAssignableFrom(newParentGenericType)) {
                world.getMessageHandler().handleMessage(new Message(
                    WeaverMessages.format(WeaverMessages.CANT_DECYP_MULTIPLE_PARAMETERIZATIONS, 
                        newParent.getName(), typeToVerify.getName(), supertype.getName()),
                    getSourceLocation(), true, new ISourceLocation[] { typeToVerify.getSourceLocation() }));
                return false;
            }
        }
    }
    return true;
}
```
A bug in AspectJ

private boolean verify(...) {
    ...
    Iterator iter = ...;
    while (iter.hasNext()) {
        ... = iter.next();
        ...
        return verify(...) ;
    }
    return true;
}

How can we find such a bug automatically?
The approach

Modeling objects’ behavior using finite state automata
How to create models?

- Use method calls as transitions
- How to define states?

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```
How to define states?

I. Anonymous states via grammar inference

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```
How to define states?

1. Anonymous states via grammar inference

```java
q = new Queue();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```

Execution:

```java
q = new
q.offer
q.poll
q.offer
q.poll
q.offer
...
q.poll
q.offer
```
How to define states?

1. Anonymous states via grammar inference

```java
q = new Queue();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```
I. Anonymous states via grammar inference

```java
q = new
q.offer
q.poll
q.offer
q.poll
...
q.poll
q.offer
```

Grammar inference

```
q = new
q.offer
q.poll
```
I. Anonymous states via grammar inference

```
q = new
q.offer
q.poll
q.offer
q.poll
q.offer
.
q.poll
```

```
q = new
q.offer
q.poll
q.offer
.
q.poll
```

```
q = new
q.offer
q.poll
q.offer
...
q.poll
q.offer
```
How to define states?

1. Anonymous states via grammar inference

Which model represents actual behavior?
I. Anonymous states via grammar inference

- Can uncover hidden structure
- Difficult to assign meaning
- NP-hard
- Does not handle ambiguity well

Which model represents actual behavior?
How to define states?

II. Based on object’s state

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```
II. Based on object’s state

```java
q = new Queue();
q.offer(...);
while (...) {
    ...
    e = q.poll();
    ...
    q.offer(...);
}
```

![Diagram](Diagram.png)
How to define states?

II. Based on object’s state

```java
q = new Queue ();
q.offer (...);
while (...)
{
...
e = q.poll ();
...
q.offer (...);
}
```

✓ Expresses the true meaning of the state

✗ Needs abstraction to keep models small
How to define states?

III. Based on the way the object is used

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```
III. Based on the way the object is used

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```

Static analysis
III. Based on the way the object is used

q = new Queue();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
How to define states?

III. Based on the way the object is used

```java
q = new Queue ();
q.offer (...);
while (...) {
    ... 
    e = q.poll () ;
    ... 
    q.offer (...) ;
}
```
III. Based on the way the object is used

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    q.offer (...);
}
```

- ✓ User-centered view of the object
- ✗ May be too specific
How to define states?

I. Anonymous states via grammar inference

II. Based on object’s state

III. Based on the way the object is used
How to define states?

I. Anonymous states via grammar inference
II. Based on object’s state
III. Based on the way the object is used
External method calls

```java
q = new Queue();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    refresh (q);
}
```
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    refresh (q);
}
Inlining models

```java
q = new Queue ();
q.offer (...);
while (...) {
    ...
    e = q.poll ();
    ...
    refresh (q);
}

void refresh (Queue q) {
    if (!q.isEmpty ()) {
        e = q.poll ();
        ...
        q.offer (e);
    }
}
```
Inlining models

```java
void refresh(Queue q) {
    if (!q.isEmpty()) {
        e = q.poll();
        ...
        q.offer(e);
    }
}
```

```java
q = new
q.offer
q.poll
refresh (q)
```
Inlining models
Inlining models

- `q = new` 
- `q.offer`
- `q.poll`
- `q.isEmpty`
### Subject: AspectJ

<table>
<thead>
<tr>
<th>jar file</th>
<th>Size (kB)</th>
<th>Classes</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspectjlib.jar</td>
<td>8</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>aspectjrt.jar</td>
<td>110</td>
<td>68</td>
<td>496</td>
</tr>
<tr>
<td>aspectjweaver.jar</td>
<td>1800</td>
<td>956</td>
<td>9999</td>
</tr>
<tr>
<td>aspectjtools.jar</td>
<td>8100</td>
<td>2980</td>
<td>36372</td>
</tr>
</tbody>
</table>
## Time & Models

<table>
<thead>
<tr>
<th>jar file</th>
<th>Classes</th>
<th>Models</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspectjlib.jar</td>
<td>3</td>
<td>116</td>
<td>0:01</td>
</tr>
<tr>
<td>aspectjrt.jar</td>
<td>68</td>
<td>2865</td>
<td>0:04</td>
</tr>
<tr>
<td>aspectjweaver.jar</td>
<td>956</td>
<td>74236</td>
<td>1:23</td>
</tr>
<tr>
<td>aspectjtools.jar</td>
<td>2980</td>
<td>243804</td>
<td>7:33</td>
</tr>
</tbody>
</table>
Stack s

in ThreadStackImpl11.getThreadStack()
StringTokenizer st

in AdviceSignatureImpl.toAdviceName()
StringTokenizer st
in Factory.makeConstructorSig()

st = new
st.countTokens
st.nextToken
Method m
in AjTypeImpl.asPointcut()

m.getDeclaringClass  m.getName  m.getAnnotation

new PointcutImpl (...,..., m, ..., ...)

23
Class c
in SignatureImpl.shortTypeName()
Class c
in AjTypeSystem.getAjType()
Infrequent models (1)

Occurrences of models: SystemColor

- Model 1: 3
- Model 2: 4
Infrequent models (2)

Occurrences of models: RuntimeException

Model 1
Model 2
Model 3
Model 4

174
Infrequent models (3)

The most frequently occurring RuntimeException model
Infrequent models (3)

Anomalous RuntimeException models

- RuntimeException thrown
- re = new Trace.exit(..., re)
- re = new List.add(re)
- RuntimeException thrown
## Infrequent models statistics

<table>
<thead>
<tr>
<th>jar file</th>
<th>Total Classes</th>
<th>Anomalies Classes</th>
<th>Models</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspectjlib.jar</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0:02</td>
</tr>
<tr>
<td>aspectjrt.jar</td>
<td>68</td>
<td>0</td>
<td>0</td>
<td>0:05</td>
</tr>
<tr>
<td>aspectjweaver.jar</td>
<td>956</td>
<td>8</td>
<td>75</td>
<td>1:16</td>
</tr>
<tr>
<td>aspectjtools.jar</td>
<td>2980</td>
<td>20</td>
<td>249</td>
<td>9:06</td>
</tr>
</tbody>
</table>
Subsequences

q = new

q.offer

q.poll

q.offer
Subsequences

\[ q = \text{new} \]

\[ q.\text{offer} \]

\[ q.\text{poll} \]

Subsequences extraction

\[ (\text{new, offer}) \]
\[ (\text{new, poll}) \]
\[ (\text{offer, offer}) \]
\[ (\text{offer, poll}) \]
\[ (\text{poll, offer}) \]
\[ (\text{poll, poll}) \]
## Frequent subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify…</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get…</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>

32
Frequent subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify…</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get…</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>

Subsequence 48 forms an itemset with support 4
## Frequent subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify…</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get…</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>
Frequent subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify…</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get…</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>

Subsequences 45 and 48 form an itemset with support 3
## Missing subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify…</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get…</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>
## Missing subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify…</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get…</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>

Missing subsequences point to anomalies
## Missing subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Ids of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Dump.println</code></td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td><code>ClassPathManager.&lt;init&gt;</code></td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td><code>DeclareParents.verify…</code></td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td><code>BcelObjectType.get…</code></td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>

Subsequence 45: `(hasNext, hasNext)`

Subsequence 48: `(hasNext, next)`

Missing subsequences point to anomalies
The anomalous method

/** *
 * This method looks through the type hierarchy for some target type - it is attempting to *
 * find an existing parameterization that clashes with the new parent that the user *
 * wants to apply to the type. If it finds an existing parameterization that matches the *
 * new one, it silently completes, if it finds one that clashes (e.g. a type already has *
 * A<String> when the user wants to add A<Number>) then it will produce an error. *
 * It uses recursion and exits recursion on hitting 'jObject' *
 */

private boolean verifyNoInheritedAlternateParameterization(ResolvedType typeToVerify, ResolvedType newParent, World world) {
    if (typeToVerify.equals(ResolvedType.OBJECT)) return true;
    ResolvedType newParentGenericType = newParent.getGenericType();
    Iterator<ResolvedType> iter = typeToVerify.getDirectSupertypes();
    while (iter.hasNext()) {
        ResolvedType supertype = (ResolvedType) iter.next();
        if (supertype.isParameterizedType() ||
            (supertype.isParameterizedType() && newParent.isRawType()) &&
            newParentGenericType.equals(supertype.getGenericType())) {
            // new parent is a parameterized type, but this is a raw type
            world.getMessageHandler().handleMessage(new Message(
                WeaverMessages.format(WeaverMessages.CANT_DECP_MULTI_PARAMETERIZATIONS, newParent.getName(), typeToVerify.getName(), supertype.getName()),
                getSourceLocation(), true, new ISourceLocation[] {typeToVerify.getSourceLocation()}));
            return false;
        }
        if (supertype.isParameterizedType()) {
            ResolvedType genericType = supertype.getGenericType();
            // If the generic types are compatible but the parameterizations aren't then we have a problem
            if (genericType.isAssignableFrom(newParentGenericType) &&
                !supertype.isAssignableFrom(newParentGenericType)) {
                world.getMessageHandler().handleMessage(new Message(
                    WeaverMessages.format(WeaverMessages.CANT_DECP_MULTI_PARAMETERIZATIONS, newParent.getName(), typeToVerify.getName(), supertype.getName()),
                    getSourceLocation(), true, new ISourceLocation[] {typeToVerify.getSourceLocation()}));
                return false;
            }
        }
    }
    return verifyNoInheritedAlternateParameterization(supertype, newParent, world);
}
The anomalous method

This method is in fact buggy
Another anomaly

List values = ...;
for (Iterator it = values.iterator();
     it.hasNext();)
{
    ... return ...;
}
...

The list holds in fact at most one element
### Frequent subsequences statistics

<table>
<thead>
<tr>
<th>jar file</th>
<th>Classes</th>
<th>Rules</th>
<th>Flaws</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspectjlib.jar</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0:03</td>
</tr>
<tr>
<td>aspectjrt.jar</td>
<td>68</td>
<td>2</td>
<td>0</td>
<td>0:04</td>
</tr>
<tr>
<td>aspectjweaver.jar</td>
<td>956</td>
<td>38</td>
<td>26</td>
<td>1:06</td>
</tr>
<tr>
<td>aspectjtools.jar</td>
<td>2980</td>
<td>161</td>
<td>71</td>
<td>6:50</td>
</tr>
</tbody>
</table>
Frequent subsequences statistics

✓ Fast enough to be practically useful
✓ Found previously unknown bug in AspectJ
✓ Found violations of convention in AspectJ
✗ Many false positives
Suggesting method calls

In Eclipse
Suggesting method calls

Using models
Other ideas (1)

- Use conditions in code to enhance models
Other ideas (2)

- Suggest changes based on model’s evolution

Evolution of a model of class A originating from method B
Other ideas (2)

- Suggest changes based on model’s evolution
Other ideas (2)

- Suggest changes based on model’s evolution
Other ideas (2)

- Suggest changes based on model’s evolution

Suggest the change
Conclusion

The approach

Modeling objects’ behavior using finite state automata

How to define states?

III. Based on the way the object is used

q = new Queue();
q.offer (x);
while (!q.isEmpty()) {
    e = q.poll();
    q.offer (e);
}

Infrequent models (2)

Occurrences of models: RuntimeException

Missing subsequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Lds of subsequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dump.println</td>
<td>28, 45, 46, 47, 48, 4386</td>
</tr>
<tr>
<td>ClassPathManager.&lt;init&gt;</td>
<td>45, 46, 47, 48</td>
</tr>
<tr>
<td>DeclareParents.verify</td>
<td>4, 48, 10537</td>
</tr>
<tr>
<td>BcelObjectType.get</td>
<td>45, 48, 12028</td>
</tr>
</tbody>
</table>

Subsequence 45: (hasNext, hasNext)
Subsequence 48: (hasNext, next)
Missing subsequences point to anomalies

Suggesting method calls

Using models

Other ideas (2)

- Suggest changes based on model’s evolution
- Suggest the change

53