Inner Oracles: Input-Specific Assertions on Internal States

Yingfei Xiong, Dan Hao, Lu Zhang, Tao Zhu, Muyao Zhu, Tian Lan

Peking University, China
2015
How a Test Detects a Bug

1. Trigger a Bug
2. Buggy State
3. Propagate the state
4. Buggy Output
5. Check the oracle
6. Find the bug!
How a Test Not Detects a Bug

1. **Test Input**
2. **Trigger a Bug**
3. **Buggy State**
4. **Propagate the state**
5. **Buggy Output**
6. **Check the oracle**
7. **Find the bug!**

Given list: `a = {-2, -1, 0, 1, 2, 3, 4};`

```java
int compare(List<Integer> a) {
    int pos = 0, neg = 0;
    for (int i : a) {
        if (i > 0) pos++;
        else neg++;
    } //buggy
    if (pos > neg) return 1;
    else if (pos == neg) return 0;
    else return -1;
}
```

Test Input:
```
a = {-2, -1, 0, 1, 2, 3, 4};
```

Test Output:
```
assert(compare(a)==1); //No bug
```

State:
- **Buggy**
  - `pos = 4`
  - `neg = 3`

- **not buggy**
  - `compare(a)=1`
Traditional Oracles

- Test Input
- Trigger a Bug
- Buggy State
- Propagate the state
- Buggy Output
- Check the oracle

Find the bug!

int compare(List<Integer> a) {
    int pos = 0, neg = 0;
    for (int i : a) {
        if (i > 0) pos++;
        else neg++;
        //buggy
    }
    if (pos > neg) return 1;
    else if (pos == neg) return 0;
    else return -1;
}

Test Input
a = {-2, -1, 0, 1, 2, 3, 4};

pos = 4
neg = 3

compare(a)=1
assert(compare(a)==1);

No bug

Traditional Oracles
- Specific to one test input
- Declared on the output of the execution
Assertions on Internal State

Test Input

Trigger a Bug

Buggy State

Propagate the state

Buggy Output

Check the oracle

Find the bug!

Standard Assertions
• on internal states
• common to all input
  • Not easy to write
  • Programmers may make the same mistake

```
int compare(List<Integer> a) {
  int pos = 0, neg = 0;
  for (int i : a) {
    if (i > 0) pos++;
    else neg++;  //buggy
  }

  // pos is the number of positives, neg is the number of negatives
  if (pos > neg) return 1;
  else if (pos == neg) return 0;
  else return -1;
}
```
Inner Oracles

Test Input

Trigger a Bug

Buggy State

Propagate the state

Buggy Output

Check the oracle

Find the bug!

int compare(List<Integer> a) {
    int pos = 0, neg = 0;
    for (int i : a) {
        if (i > 0) pos++;
        else neg++; //buggy
    }
    assert_for_this_test(neg == 2);
    if (pos > neg) return 1;
    else if (pos == neg) return 0;
    else return -1;
}

Test Input

a = {-2, -1, 0, 1, 2, 3, 4};

pos = 4
neg = 3

Inner Oracles
- Declared on internal states
- Specific to a test input
Inner oracles can also be written by weaving, similar to AOP.
Check at: http://ayzk.github.io/InnerTest/
How much can we gain with inner oracles? – Enhancing tests

<table>
<thead>
<tr>
<th>Subject</th>
<th>KLOC</th>
<th>#Method</th>
<th>#Class</th>
<th>#Test</th>
<th>#Mutant</th>
<th>Pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>jodatime</td>
<td>25.8</td>
<td>3276</td>
<td>198</td>
<td>3417</td>
<td>26</td>
<td>88842</td>
</tr>
<tr>
<td>timeandmoney</td>
<td>1.1</td>
<td>262</td>
<td>30</td>
<td>104</td>
<td>39</td>
<td>4056</td>
</tr>
<tr>
<td>barbecue</td>
<td>8.0</td>
<td>283</td>
<td>55</td>
<td>51</td>
<td>50</td>
<td>2550</td>
</tr>
<tr>
<td>xmlsec</td>
<td>16.2</td>
<td>1213</td>
<td>181</td>
<td>97</td>
<td>22</td>
<td>2134</td>
</tr>
</tbody>
</table>

In 30.72%-69.65% pairs, fault cannot be captured by traditional oracles on output, but only by inner oracles.

- The buggy state is not propagated into a buggy output (294/1369)
- The buggy part in the output state cannot be accessed by a test, e.g., a private member (1075/1369)
How much can we gain with inner oracles? – Reducing test suites

<table>
<thead>
<tr>
<th>Subject</th>
<th>jodatime</th>
<th>timeandmoney</th>
<th>barbecue</th>
<th>xmlsec</th>
</tr>
</thead>
<tbody>
<tr>
<td>#With traditional oracles</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>#With inner oracles</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

Test suites are further reduced by 14.3%-50.0% with inner oracles.
Applications and Implications
--- Testing Optimization

```c
int times(int a, int b) {
    if (b % 2 == 0) {
        while (b >>= 1)
            return a << 1;
    }
    else
        return a * b;
}
```

How do we know the first branch is executed when \( b \) is 8?
Applications and Implications
--- Testing Optimization

```c
int times(int a, int b) {
    if (b % 2 == 0) {
        while (b >>= 1)
            return a << 1;
    } else {
        assert(!test1);
        return a * b;
    }
}
```
test1 = true;
times(2, 8);
test1 = false;
Applications and Implications --- Debugging

Traditional Oracles
- Any executed statements may be buggy

Inner Oracles
- Only the statements executed before the inner oracle may be buggy
Applications and Implications
--- Regression Test Generation

doStuff(X x, int n, int m) {
    Y y = x.doSth(n);
    Z z = y.doSthElse(m);
    z.field = n+m;
    return;
}

@Test
...  
doSstuff(x, 1, 2);
assert(z.field == 3);

• How do we know which object z is?
• How do we access this object?

[Xie et al., ECOOP06], [Taneja et al., ASE08]
Applications and Implications
--- Regression Test Generation

doStuff(X x, int n, int m) {
    Y y = x.doSth(n);
    Z z = y.doSthElse(m);
    z.field = n+m;
    assert(!test1 || z.field==3);
    return;
}

@test
...

  test1 = true;
doStuff(x, 1, 2);
test1 = false;
Application and Implication
--- Invariant Discovery

• Tools like Daikon discovers invariants (oracles on internal states for all inputs)
• Sometimes very few invariants can be discovered if we use too many inputs
• Let Daikon discover inner oracles for some inputs instead

```java
test1 = true;
doStuff(1);
test1 = false;
```

```java
test1 = true;
doStuff(2);
test1 = false;
```

```java
test1 = true;
doStuff(3);
test1 = false;
```
Summary

• Inner Oracles
  • declared on internal states
  • specific to one test input
• Has a lot of applications/implications
• Ignored in existing literatures
• Worth putting more weights on