On Test Repair Using Symbolic Execution

Brett Daniel
Darko Marinov

Tihomir Gvero

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

ISSTA 2010: International Symposium on Software Testing and Analysis
July 15, 2010
Trento, Italy
Passing Unit Tests

```java
public class Cart {
    ...
    public double getTotalPrice() {...}
    public String getPrintedBill() {...}
    ...
}
```

```java
public void testAddTwoDifferentProducts() {
    Cart cart = ...
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals(
        "Discount: -$3.00, Total: $3.00",
        cart.getPrintedBill());
}
```
Requirements Change

```java
public class Cart {
    ...  
    public double getTotalPrice() {...}
    public String getPrintedBill() {...}
    ...
}
```

```java
public void testAddTwoDifferentProducts() {
    Cart cart = ...
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals(
            "Discount: -$3.00, Total: $3.00",
            cart.getPrintedBill());
}
```
Delete Broken Tests?

But that reduces the quality of the test suite.
Repairing Tests is Preferable

But that requires a lot of time and effort
ReAssert Suggests Repairs

ReAssert: Suggesting Repairs for Broken Unit Tests
Brett Daniel, Vilas Jagannath, Danny Dig, Darko Marinov
ASE 2009, Auckland, New Zealand
Confirm or Reject Suggestions

Original Test File

```java
public void testRedPenCoupon() {
    Cart cart = new Cart();
    cart.addProduct(new RedPen());
    cart.addProduct(new RedPen());
    cart.addCoupon(new AnniversaryCoupon());
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals("Discount: -$3.00, Total: $3.00", cart.getPrintedBill());
}
```

Fixed Test File

```java
public void testRedPenCoupon() {
    Cart cart = new Cart();
    cart.addProduct(new RedPen());
    cart.addProduct(new RedPen());
    cart.addCoupon(new AnniversaryCoupon());
    assertEquals(6.0, cart.getTotalPrice());
    assertEquals("Discount: -$6.00, Total: $6.00", cart.getPrintedBill());
}
```
ReAssert Reduces Effort
Does ReAssert Work?

Case Studies

100% (37 of 37) Repaired
78% (29 of 37) Useful

User Study

97% (131 of 135) Repaired
86% (113 of 131) Useful
3% (4 of 135) Not Repaired

Open Source

45% (75 of 167) Repaired
55% (92 of 167) Not Repaired
Many failures can be repaired by changing literal values in test code.

ReAssert could not determine which literals needed to change and how.

Symbolic execution can discover literals that cause a test to pass.
Simple Assertion Failure

```java
assertEquals(3.0, cart.getTotalPrice());
```
Replace Literal

```java
replaceAll
```

Replace in code

Record actual value

```java
assertEquals(6.0, cart.getTotalPrice());
```
double expTotal = 3.0;

...  

assertEquals(expTotal, cart.getTotalPrice());
double expTotal = 6.0;

...
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
} else {
    expTotal = 3.0;
}

assertEquals(expTotal, cart.getTotalPrice());
Multiple Expected Values

double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}

assertEquals(expTotal, cart.getTotalPrice());
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}
...

assertEquals(6.0, cart.getTotalPrice());
ReAssert's Limitations

- **Multiple Expected Values**
  ```java
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
} else {
    expTotal = 3.0;
}
assertEquals(expTotal, cart.getTotalPrice());
```

- **Computed Expected Value**
  ```java
double total = 3.0;
String expBill = "Total: $" + total;
assertEquals(expBill, cart.getPrintedBill());
```

- **Expected Object Comparison**
  ```java
Product expProduct = new Product("Red pen", 3.0);
assertEquals(expProduct, cart.getItem(0));
```
Symbolic Execution

Dynamic symbolic execution

```
int input = PexChoose.Value<int>("i");
if (input < 5) {
    throw new Exception();
}
```

Branches introduce path constraints

Nondeterministic choice generator produces concrete values

Solve constraints to execute alternate paths

Symbolic Execution in Testing

Test Generation

Find values that make a program fail
(or achieve coverage)

Test Repair

Find values that make a test pass
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```java
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}
assertEquals(
    expTotal,
    cart.getTotalPrice());
```
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
} else {
    expTotal = 3.0;
}
assertEquals( expTotal, cart.getTotalPrice());
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```java
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}
assertEquals(
    expTotal,
    cart.getTotalPrice());
```
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```java
double expTotal;
if (HAS_TAX) {
    expTotal = PexChoose.Value<double>("e1");
} else {
    expTotal = PexChoose.Value<double>("e2");
}
assertEquals(
    expTotal,
    cart.getTotalPrice());
```
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

declare expTotal;
if (HAS_TAX) {
    expTotal = PexChoose.
        Value<double>("e1");
} 
else {
    expTotal = PexChoose.
        Value<double>("e2");
} 
assertEquals( 
    expTotal,
    cart.getTotalPrice());

\text{e2} == 6.0
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 6.0;
}
assertEquals(expTotal, cart.getTotalPrice());
Implementation Mismatch

- Java
- Eclipse

- .NET
- Visual Studio
Evaluation

Q1: How many failures can ideal literal replacement repair?

Q2: How do ReAssert and literal replacement compare?

Q3: Can symbolic execution discover literals?
Failures in Open-Source Software
Open Source Software - Java

- Checkstyle
- JDepend
- JFreeChart
- Lucene
- PMD
- XStream

Legend:
- Red: Failures
- Green: ReAssert
- Light Green: Literal Repl.
Open Source Software - .NET

The chart shows the number of failures and repairs for various open source software projects using different tools.

- **Fudge-C#**
- **GCaIExSync**
- **Json.NET**
- **NGChart**
- **NHaml**
- **ProjectPilot**
- **AdblockIE**
- **Markdown#**

The y-axis represents different projects, and the x-axis represents failures and repairs. Different colors indicate the use of various tools:
- Red: Failures
- Green: ReAssert
- Blue: Literal Repl.
- Greenish-blue: Pex

The data suggests that **ProjectPilot** has the highest number of failures and repairs, followed by **Json.NET** and then **Markdown#**.
ReAssert vs. Literal Replacement

Java
- ReAssert: 14% (24 of 167)
- Neither: 34% (56 of 167)
- Both: 22% (36 of 167)
- Literal Repl.: 31% (51 of 167)

.NET
- ReAssert: 12% (8 of 68)
- Neither: 35% (24 of 68)
- Both: 12% (8 of 68)
- Literal Repl.: 41% (28 of 68)
Recreate Literals

- AdblockIE
- CSHgCmd
- Json.NET
- NerdDinner
- SharpMap

Literals

- Passing
- Matching
- Unsolved

- 77% (564 of 734)
- 8% (60 of 734)
- 15% (110 of 734)
Results

Q1: How many failures can ideal literal replacement repair?

About half

Q2: How do ReAssert and literal replacement compare?

12% to 22% improvement when combined

Q3: Can symbolic execution discover literals?

Yes: 52% to 92% of literals
Thanks

- Nikolai Tillmann for help with Pex
- Milos Gligoric, Vilas Jagannath, and Viktor Kuncak for valuable comments
- Anonymous colleagues for potential evolutions
- Anonymous reviewers

- Supported in part by the US National Science Foundation under Grant No. CCF-0746856
ReAssert

http://mir.cs.illinois.edu/reassert