Tao Xie
North Carolina State University
Nikolai Tillmann, Peli de Halleux, Wolfram Schulte
Microsoft Research

Outline

- Unit Testing
- Parameterized Unit Testing (PUT)
- Mutation Analysis for PUT

Unit Under Test

```
public class IntStack {
   public IntStack() { … }
   public void Push(int value) {
      if (value < 0) return;
      …
   }
   public int Pop() { … }
   public bool IsEmpty() { … }
   public bool Equals(Object other) { … }
}
```

Unit Testing

- A unit test is a small program with assertions
- Test a single (small) unit of code

```
void TestPushPop() {
 IntStack s = new IntStack();
 s.Push(3);
 s.Push(5);
 Assert.IsTrue(s.Pop() == 5);
}
```

- Happy path only
- New code with old tests
- Redundant tests

The Recipe of Unit Testing

- Three ingredients:
  - Data
  - Method Sequence
  - Assertions

```
void TestPushPop() {
    int item1 = 3, item2 = 5;
```

```
    IntStack s = new IntStack();
    s.Push(item1);
    s.Push(item2);
```

```
    Assert.IsTrue(s.Pop() == item2);
}
```

The (problem with) Data

```
s.Push(5);
```

- Which value matters?
  - Redundant, Incomplete Test Suites
- Does not evolve with the code under test.

Parameterized Unit Test

- *Parameterized* Unit Test =

  Unit Test with *Parameters*
- Separation of concerns
  - Data is generated by a tool
  - Human takes care of the Functional Specification

```
void TestPushPopPUT4(IntStack s, int i) {
   PexAssume.IsTrue(s != null);
   PexAssume.IsTrue(i >= 0);
   s.Push(i);
   PexAssert.IsTrue(s.Pop() == i);
}
```

Parameterized Unit Tests are Algebraic Specifications

- A Parameterized Unit Test can be read as a universally quantified, conditional axiom.

```
void TestPushPopPUT4(IntStack s, int i) {
   PexAssume.IsTrue(s != null);
   PexAssume.IsTrue(i >= 0);
   s.Push(i);
   PexAssert.IsTrue(s.Pop() == i);
}
```

```
∀ IntStack s, int i:
  s ≠ null ∧ i >= 0 ⇒
    equals(
     Pop(Push(s, i)),
     i)
```

Test Generation WorkFlow

Code Under Test Project

Test Project

Pex

Parameterized Unit Tests

Generated Tests

Automatic test input generation with Pex

- Pex is a test input generator
  - Pex starts from parameterized unit tests
  - Generated tests are emitted as traditional unit tests

- Pex analyzes execution paths
  - Analysis at the level of the .NET instructions (MSIL)
  - Dynamic symbolic execution (i.e., directed random testing in DART, concolic testing in CUTE, …)

http://research.microsoft.com/Pex/

Parameterized Unit Testing/Pex

- Pex is being used both inside and outside of Microsoft
- Publicly available with both commercial and academic licenses
- Being integrated into Visual Studio
- Being taught at NCSU graduate testing course
  - http://sites.google.com/site/teachpex/
- ICSE 2009 Tutorial on PUT
  - http://ase.csc.ncsu.edu/put/
- …

4A Patterns for Parameterized Unit Tests

- Assume, Arrange, Act, Assert

```
void TestPushPopPUT3(int i) {
   PexAssume.IsTrue(i >= 0);         // assume
   IntStack s = new IntStack();      // arrange
   s.Push(i);                        // act
   PexAssert.IsTrue(s.Pop() == i); // assert
}
```

Writing (Good) Parameterized Unit Tests is Challenging

- *Stronger assumptions (not good)*
- *Weaker assertions (not good)*

Detecting them is challenging too

```
void TestPushPopPUT3(int i) {
   PexAssume.IsTrue(i >= 0);
   IntStack s = new IntStack();
   s.Push(i);
   PexAssert.IsTrue(s.Pop() == i);
}
```

```
void TestPushPopPUT4(IntStack s, int i) {
   PexAssume.IsTrue(s != null);
   PexAssume.IsTrue(i >= 0);
   s.Push(i);
   PexAssert.IsTrue(s.Pop() == i);
}
```

Analysis of Parameterized Unit Tests (PUTs)

- Key idea for detecting stronger assumptions
  - weakening assumptions while violating no assertions in the PUT
- Key idea for detecting weaker assertions
  - strengthening assertions while still being satisfied by the generated test inputs

Mutation Analysis of Parameterized Unit Tests (PUTs)

- Key idea for detecting stronger assumptions
  - weakening assumptions (producing a mutant PUT) while violating no assertions in the PUT (being a live mutant or not being killed)

- Key idea for detecting weaker assertions
  - strengthening assertions (producing a mutant PUT) while still being satisfied by the generated test inputs PUT (being a live mutant or not being killed)

Mutation Killing

- A mutant PUT is *live* if **no** test inputs can be generated (by a test generation tool) to
  - violate specified assertions
  - satisfy the specified assumptions

- A live mutant PUT indicates likely PUT improvement
  - generalization on assumptions
  - specialization on assertions

Mutation Operators

- **Assumption Weakening:** weaken constraints specified in assumptions
- **Assertion Strengthening:** strengthen constraints specified in assertions

- **Primitive-Value Generalization:** replace a primitive value with an additional parameter (related to assumption weakening)
- **Method-Invocation Deletion:** Delete a method invocation (related to assumption weakening)

Assumption Weakening

Deleting an assumption from the PUT

```
void TestPushPopPUT1(int j) {
  PexAssume.IsTrue(j >= 0);
  IntStack s = new IntStack();
  s.Push(j);
  s.Push(5);
  PexAssert.IsTrue(s.Pop() == 5);
}
```

Weakening a clause in an assumption: P > Q ➔ P >= Q

```
void TestPushPopPUT2(int i) {
  PexAssume.IsTrue(i > 0); ➔ PexAssume.IsTrue(i >= 0);
  IntStack s = new IntStack();
  s.Push(i);
  PexAssert.IsTrue(s.Pop() == i);
}
```

Assertion Strengthening

Strengthen a clause

strengthen P > Q to P > (Q + const)

strengthen P > Q to P == (Q+const)),

```
void TestPushPopPUT1(int j) {
  IntStack s = new IntStack();
  s.Push(j);
  s.Push(5);
  PexAssert.IsTrue(s.Pop() > -1);
  PexAssert.IsTrue(s.Pop() > 0);
  PexAssert.IsTrue(s.Pop() == 5);
}
```

(On the slide, the lines `PexAssert.IsTrue(s.Pop() > -1);` and `PexAssert.IsTrue(s.Pop() > 0);` are struck through.)

Primitive-Value Generalization

```
void TestPushPopPUT() {
   IntStack s = new IntStack();
   s.Push(3);
   s.Push(5);
   PexAssert.IsTrue(s.Pop() == 5);
}
```

```
void TestPushPopPUT1(int j) {
   IntStack s = new IntStack();
   s.Push(j);
   s.Push(5);
   PexAssert.IsTrue(s.Pop() == 5);
}
```

Method-Invocation Deletion

```
void TestPushPopPUT3(int i) {
   PexAssume.IsTrue(i >= 0);
   IntStack s = new IntStack();
   s.Push(i);
   PexAssert.IsTrue(s.Pop() == i);
}
```

```
void TestPushPopPUT4(IntStack s, int i) {
   PexAssume.IsTrue(s != null);
   PexAssume.IsTrue(i >= 0);
   s.Push(i);
   PexAssert.IsTrue(s.Pop() == i);
}
```

Conclusion

- Writing good PUTs is challenging
  - Stronger assumptions
  - Weaker assertions

Mutation analysis of PUTs

- Mutation killing
- Mutation operators
  - Assumption weakening
  - Assertion strengthening
  - Primitive-value generalization
  - Method-invocation deletion

Thank you

http://research.microsoft.com/pex

http://ase.csc.ncsu.edu/put/

Automated Software Engineering Research Group @NCSU

RiSE

Pex in Visual Studio

Leveraging the Visual Studio integration

Dynamic Symbolic Execution

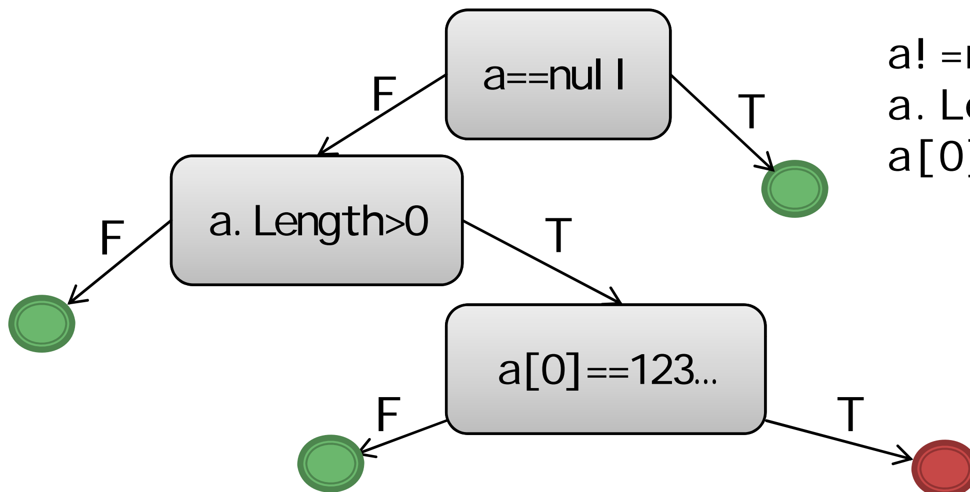
- Code to generate inputs for:

```
void CoverMe(int[] a)
{
  if (a == null) return;
  if (a.Length > 0)
    if (a[0] == 1234567890)
      throw new Exception("bug");
}
```

Choose next path → Solve → Execute&Monitor

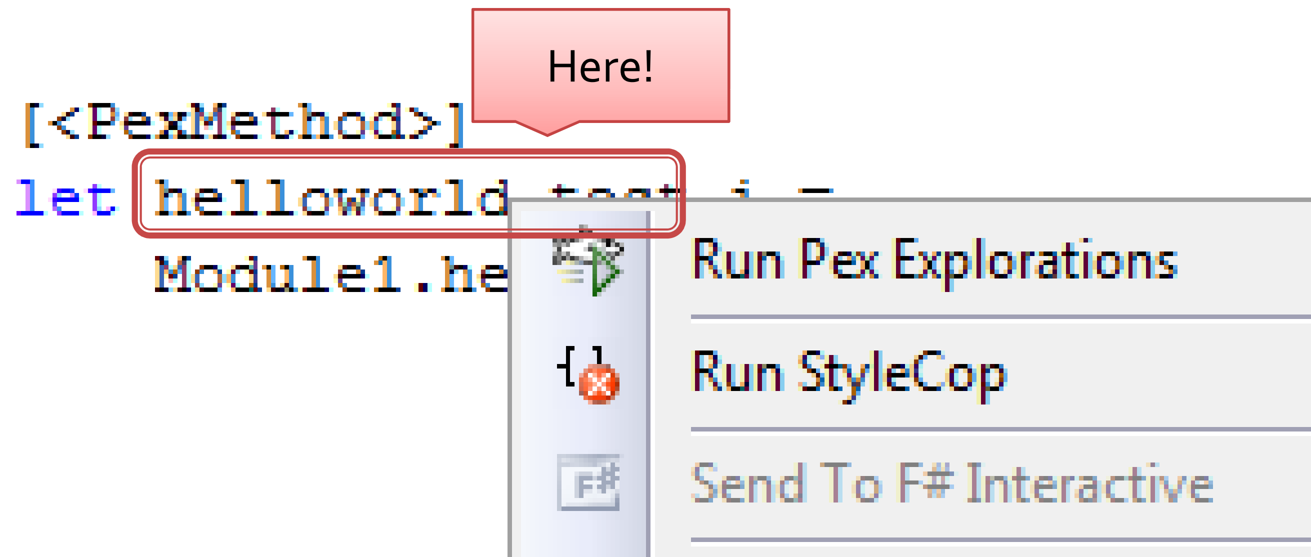
Constraints to solve	Data	Observed constraints
	null	a==null
a!=null	{}	a!=null && !(a.Length>0)
a!=null && a.Lengt…		…& …90
a!=null && a.Length>0 && a[0]==1234567890	{123..}	a!=null && a.Length>0 && a[0]==1234567890

Negated condition



Running Pex from the Editor

- Right-click on the **method name**
- Select Run Pex Explorations



Exploration Result View

Input/Output table
Row = generated test
Column = parameterized test input or output

ue bar
portant messages
re !!!

1 explor... dule1.helloworld_test(String i) Views

Review bo... ues: All Tests | All Events | **1 Uninstrumented Method**

		i	result	Summary/Exception	Error Message
	1	null	"not found"		

Events View

Event filtering

Select and see details

Review bold ... Events | 1 Uninstrumented Method

Event

Module1.hello_world(String)

Details...

Stack trace:

at Module1.helloworld_test(String) | Module1.fs(6)

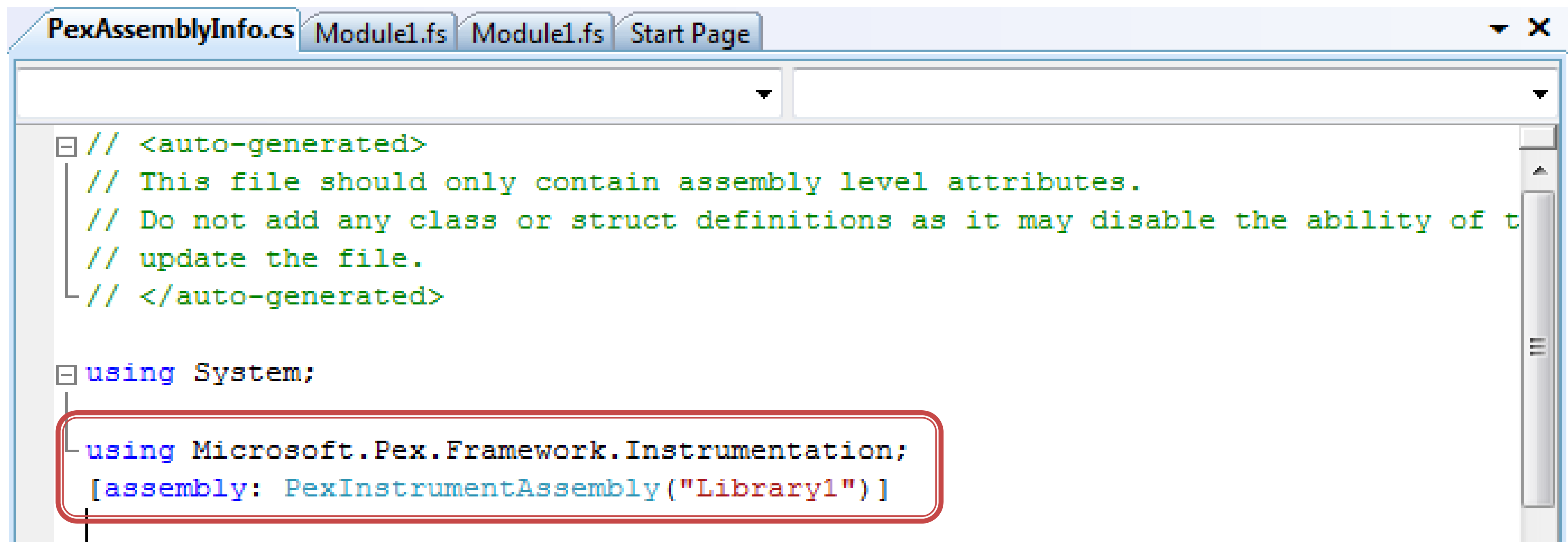
Apply Pex suggested fix

Instrument assembly... | Ignore uninstrumented method...

Send To | Help

Suggested Fixes

- Attributes



Input/Output table

Test outcome filtering

Review bold issues:	All Tests	1 Failed Test	All Events

		i	result	Summary/Exception	Error Message
	1	null		ption	Exception of type 'System.Exception' was thrown.
	2	"hello v			

New Test ailable

Input/Output table

Review bold issues: All Tests | 1 Failed Test | All Events

		i	result	Summary/Exception
	1	null		Exception
	2	"hello world"	"found it!"	

Details...

Stack trace:

System.Exception, FSharp.Core	Exception of type 'System
at Operators.raise(Exception)	
at Module1.hello_world(String)	Module1.fs(6)
at Module1.helloworld_test(String)	Module1.fs(6)

Add Precondition...

Allow Exception... Send To

Exception
Stack trace

Revi

Allow exception