Directed Random Testing*

Wolfram Schulte
Microsoft Research

Soqua 11/2006

Was formerly announced as: “Challenge Problems in Testing”

What my team does

- Static program verification & language design
  - Verifying multi-threaded OO programs (Spec#)
  - Verifying message passing contracts (Sing#)
  - Integration of data via structural types and monads (Xen,Cω,C# V3)
- Runtime systems
  - Task concurrency (Futures)
  - Memory resilience (DieHard)
- Development systems
  - Build/version/deploy
- Modeling and test
  - Model-based testing (Spec Explorer)
  - **White-box testing** (Mutt / Unit Meister/ PUT / PEX)

Why testing is hard…

```
void AddTest() {
    ArrayList a = new ArrayList(1);
    object o = new object();
    a.Add(o);
    Assert.IsTrue(a[0] == o);
}
```

Writing a test involves

- determining a meaningful sequence of method calls,
- selecting exemplary argument values (the test input values),
- stating assertions.

A test states both the intended behavior, and achieves certain code coverage.

Outline

- Input generation
- Mock object generation
- Sequence generation
- Compositional testing

Test input generation

Problem definition

- Test Input Generation
  - Given a statement *s* in program *P*, compute input *i*, such that *P*(*i*) executes *s*

- Test Suite Generation
  - Given a set of statements *S* in *P*, compute inputs *I*, such that forall *s* in *S*, exists *i* in *I*: *P*(*i*) executes *s*

Existing test generation techniques

```
void Obscure(int x, int y){
   if (x==crypt(y)) error(); return 0;
}
```

- Static test case generation via symbolic execution often cannot solve constraints (assumes error)
- Random testing via concrete execution often cannot find interesting value (misses errors)
- Directed Random Testing/ Conc(rete & symb)olic execution finds error: take random y, solve for x

Concolic execution

Generate a test suite for program *P.*

Algorithm for test suite generation:
We use a dynamic predicate *Q* over the program input.

0. set Q := true
1. choose inputs *i* such that *Q*(*i*) holds
2. execute *P*(*i*) and build up path condition P(*i*)
3. set Q := (Q and not P)
4. if Q <> false, goto (1.)

Remark: The choice in (1.) is the cornerstone of concolic execution. It can be implemented in a variety of ways: as a random choice (e.g. for the initial inputs), or as a depths-first/iterative deepening/breadth first/… search over the logical structure of the constructed predicate Q, or using any existing constraint solver.

Example: Concolic execution

```
class List {
   int head;
   List tail;
}

static bool Find(List xs,
                 int x){
  while (xs!=null) {
    if (xs.head == x)
       return true;
      xs = xs.tail;
   }
  return false;
}
```

Concrete values (Assignments)	Symbolic constraints (Predicates)

1. Choose arbitrary value for x, choose null for xs

x = 517; xs = null;	xs== null

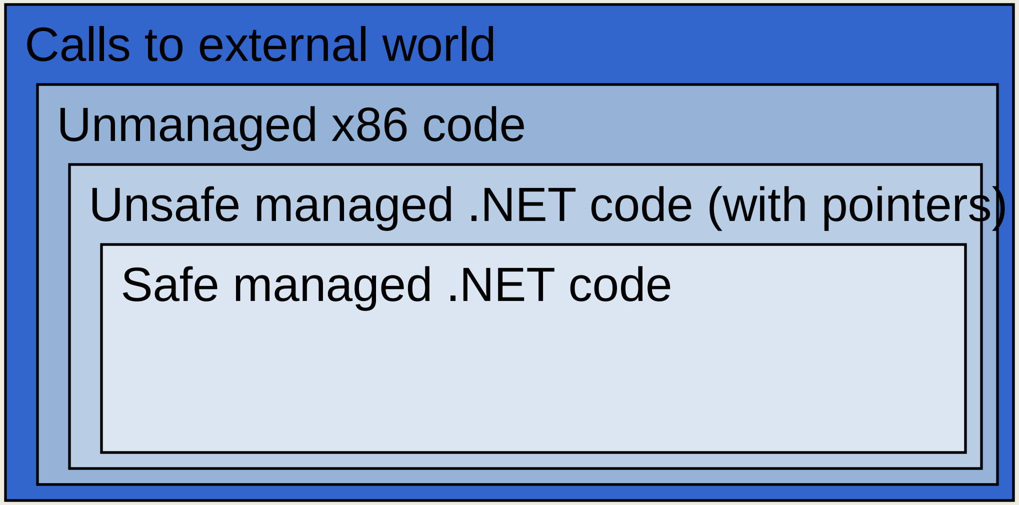
2. Negate predicate (xs == null)
   → choose new list with new arb. head

x = 517;
xs.head = -3;
xs.tail = null;
	xs!=null &&
	xs.head != x &&
	xs.tail == null

3. Negate both predicates, equivalent to
   xs!=null && (xs.head == x || xs.tail != null)
   → let's choose xs.head != x, thus xs.tail == xs

x = 517;
xs.head =-3;
xs.tail = xs;
	CRASH!
	→ Cyclic list

Why concolic execution is needed



- Most .NET programs use unsafe/unmanaged code for legacy and performance reasons
- Combining *concrete execution and symbolic reasoning* still works: all conditions that can be monitored will be systematically explored

Code instrumentation for symbolic analysis

```
class  Point { int x; int y;
  public static int GetX(Point p) {
    if (p != null) return p.X;
    else return -1; } }
```

```
      ldtoken  Point::GetX
      call     __Monitor::EnterMethod
      brfalse  L0
      ldarg.0
      call     __Monitor::NextArgument<Poi
L0:   .try {
        .try {
          call    __Monitor::LDARG_0
          ldarg.0
          call    __Monitor::LDNULL
          ldnull
          call    __Monitor::CEQ
          ceq
          call    __Monitor::BRTRUE
          brtrueL1
          call    __Monitor::BranchFallthrough
          call    __Monitor::LDARG_0
          ldarg.0
          …
```

```
ldtoken          Point::X
call   __Monitor::LDFLD_REFERENCE
ldfld  Point::X
call   __Monitor::AtDereferenceFallthrough
br     L2

call   __Monitor::AtBranchTarget
       C_I4_M1

       T

                 L4
       llReferenceException {
call   __Monitor::AtNullReferenceException
rethrow
      }
L4:   leave  L5
    } finally {
       _Monitor::LeaveMethod
       y
L5:  ldloc.0
     ret
```

Prologue

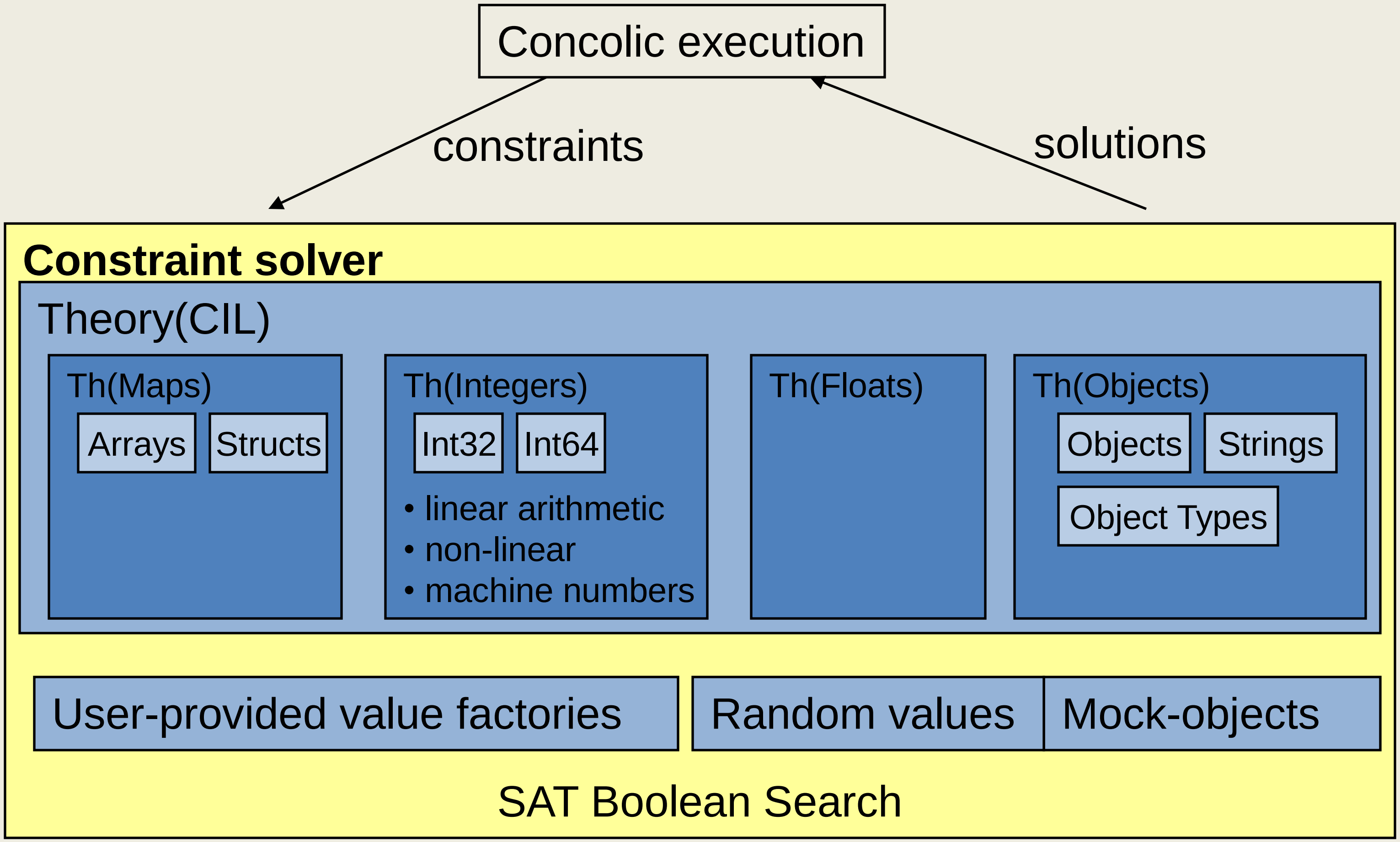
Record concrete values
... all information
... method is called
... er context

(The real C# compiler output is actually more complicated.)

Epilogue

Calls to build path condition

Finding solutions of constraint systems



Closing the environment: Generating mock objects

Testing with interfaces

Example

```
AppendFormat(null, "{0} {1}!", "Hello", "Microsoft");
```

BCL Implementation

```
public StringBuilder AppendFormat(
    IFormatProvider provider,
    char[] chars, params object[] args) {

    if (chars == null || args == null)
        throw new ArgumentNullException(…);
    int pos = 0;
    int len = chars.Length;
    char ch = '\x0';
    ICustomFormatter cf = null;
    if (provider != null)
        cf = (ICustomFormatter)provider.GetFormat( typeof(ICustomFormatter));
  …
```

Generating mock objects

- Introduce a mock class implementing the interface.
- Let an oracle provide the behavior of the mock methods.

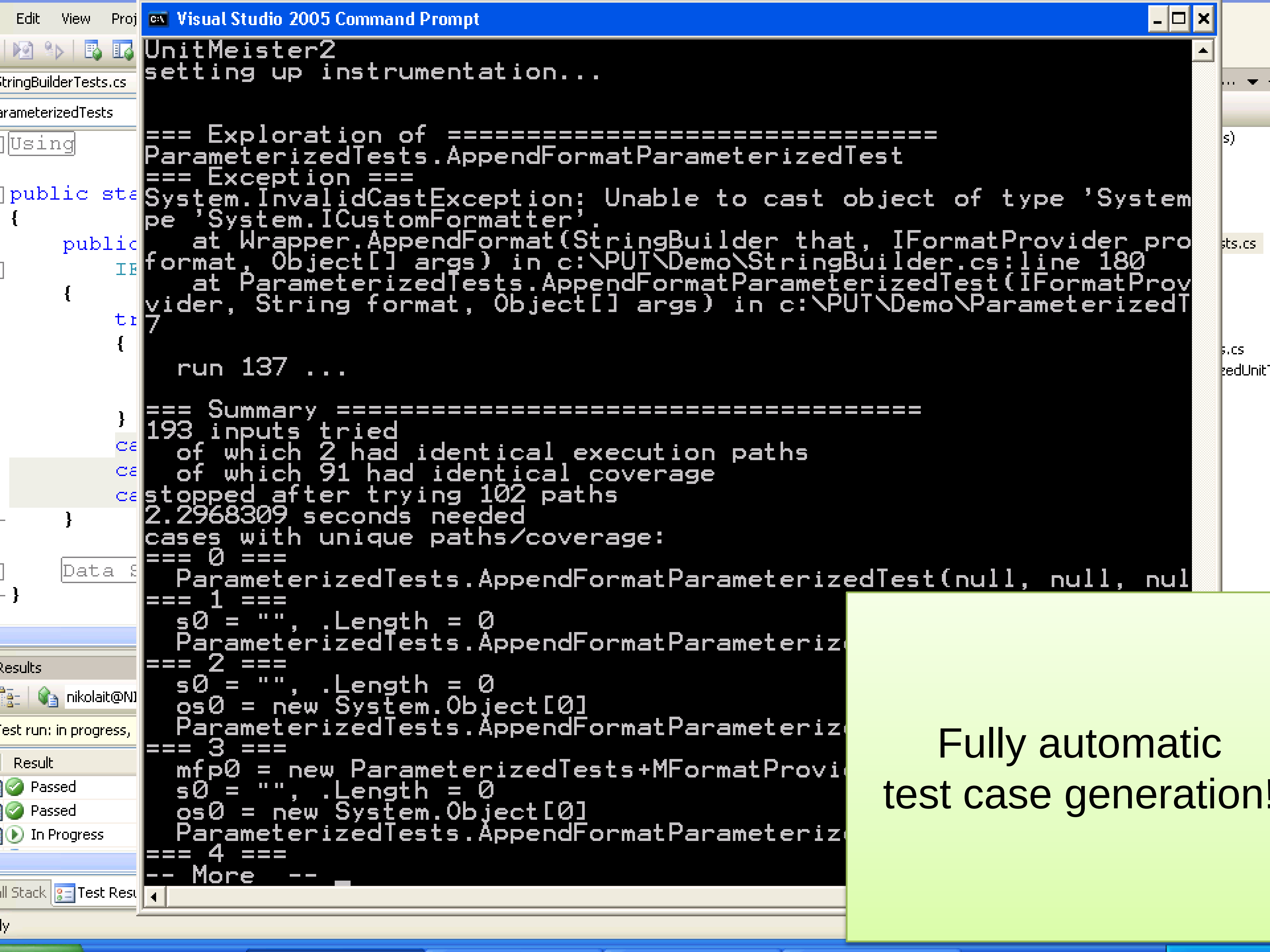
```
public class MFormatProvider : IFormatProvider {

    public object GetFormat(Type formatType) {
        …
        object o = call.ChooseResult<object>();
        Assume.IsTrue(o is IFormatProvider );
        return o;
    }
}
```

- During symbolic execution, pick a new symbol to represent unknowns
- Collect constraints over symbols along each execution path
- Solve the constraints to obtain concrete values for each execution path
- During concrete execution, choose these concrete values

```csharp
public static class ParameterizedTests
{
    public static void AppendFormatParameterizedTest(
        IFormatProvider formatProvider, String format, object[] args)
    {
        try
        {
            StringBuilder sb = new StringBuilder();
            Wrapper.AppendFormat(sb, formatProvider, format, args);
        }
        catch (ArgumentNullException) { }
        catch (ArgumentOutOfRangeException) { }
        catch (FormatException) { }
    }

    public class MFormatProvider : IFormatProvider
    {
        public object GetFormat(Type formatType)
        {
            IMethodCallOracle call = Oracle.MethodCall(this,
            call.ParameterValue(0, formatType);
            //if (call.ChooseThrowException()) throw call.Cho
            return call.ChooseResult<object>();
        }
    }

    Data Sources
}
```

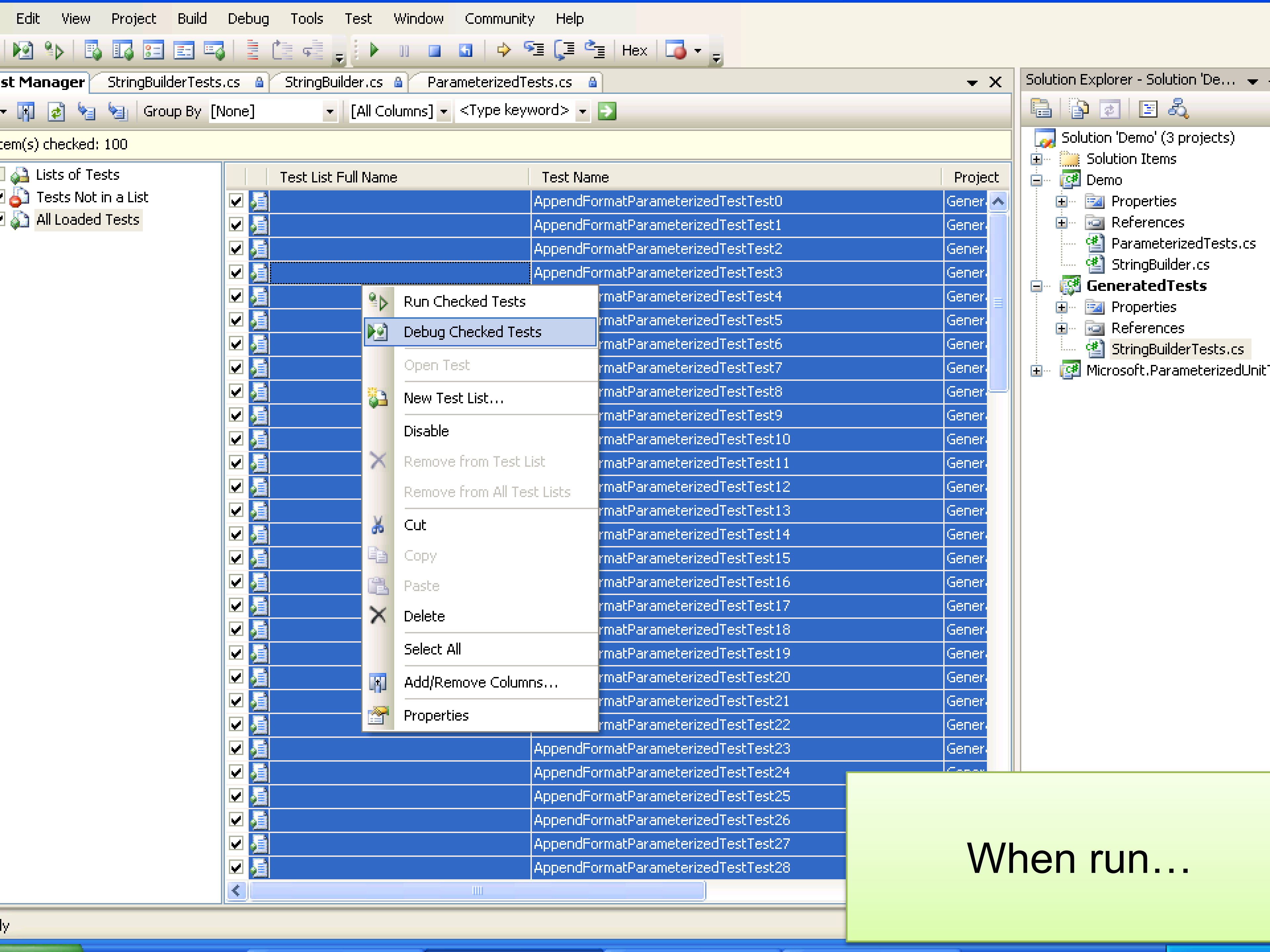
DEMO

Here is a simple test which catches all documented exception and uses a mock MFormatProvider

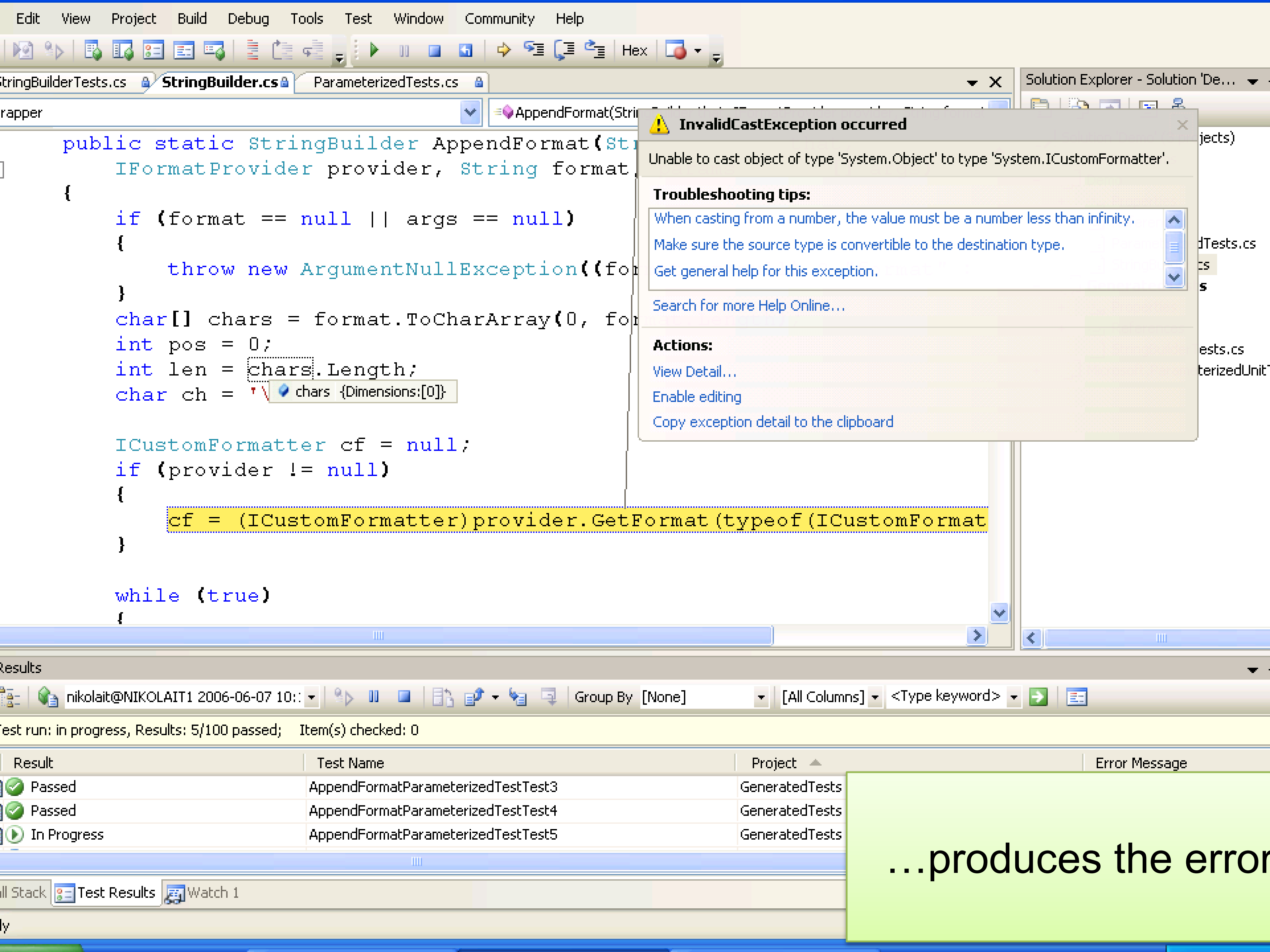

Visual Studio 2005 Command Prompt
UnitMeister2
setting up instrumentation...
=== Exploration of ==================================
ParameterizedTests.AppendFormatParameterizedTest
=== Exception ===
System.InvalidCastException: Unable to cast object of type 'System
pe 'System.ICustomFormatter'.
at Wrapper.AppendFormat(StringBuilder that, IFormatProvider pro
format, Object[] args) in c:\PUT\Demo\StringBuilder.cs:line 180
at ParameterizedTests.AppendFormatParameterizedTest(IFormatProv
vider, String format, Object[] args) in c:\PUT\Demo\ParameterizedT
7
run 137 ...
=== Summary ======================================
193 inputs tried
of which 2 had identical execution paths
of which 91 had identical coverage
stopped after trying 102 paths
2.2968309 seconds needed
cases with unique paths/coverage:
=== 0 ===
ParameterizedTests.AppendFormatParameterizedTest(null, null, nul
=== 1 ===
s0 = "", .Length = 0
ParameterizedTests.AppendFormatParameteriz
=== 2 ===
s0 = "", .Length = 0
os0 = new System.Object[0]
ParameterizedTests.AppendFormatParameteriz
=== 3 ===
mfp0 = new ParameterizedTests+MFormatProvi
s0 = "", .Length = 0
os0 = new System.Object[0]
ParameterizedTests.AppendFormatParameteriz
=== 4 ===
-- More --
Fully automatic
test case generation!
Edit View Proj
StringBuilderTests.cs
arameterizedTests
Using
public sta
public
Data S
Results
nikolait@N
Test run: in progress,
Result
Passed
Passed
In Progress
Test Resu

```
        object[] os0 = new object[0];
        ParameterizedTests.AppendFormatParameterizedTest(mfp0, s0, os0
    }

    [TestMethod()]
    public void AppendFormatParameterizedTestTest4() {
        string s0 = "a";
        object[] os0 = new object[0];
        ParameterizedTests.AppendFormatParameterizedTest(null, s0, os0
    }

    [TestMethod()]
    public void AppendFormatParameterizedTestTest5() {
        ParameterizedTests.MFormatProvider mfp0 = new ParameterizedTes
        string s0 = "";
        object[] os0 = new object[0];
        Microsoft.ParameterizedUnitTesting.MethodCallExpectationsAndRe
        Microsoft.ParameterizedUnitTesting.Oracle.Initialize(oraclePro
        System.Reflection.MethodBase GetFormat = typeof(Para
        object o0 = new object();
        oracleProvider.GetOracleBuilder(0, GetFormat).SetRe
        ParameterizedTests.AppendFormatParameterizedTest(mf
    }

    [TestMethod()]
    public void AppendFormatParameterizedTestTest6() {
        ParameterizedTests.MFormatProvider mfp0 = new Parame
        string s0 = "a";
        object[] os0 = new object[0];
        Microsoft.ParameterizedUnitTesting.MethodCallExpecta
```

Generated tests exercise different path of the implementation


Edit View Project Build Debug Tools Test Window Community Help
Hex
st Manager
StringBuilderTests.cs
StringBuilder.cs
ParameterizedTests.cs
Group By [None]
[All Columns]
<Type keyword>
tem(s) checked: 100
Lists of Tests
Tests Not in a List
All Loaded Tests
Test List Full Name
Test Name
Project
AppendFormatParameterizedTestTest0
AppendFormatParameterizedTestTest1
AppendFormatParameterizedTestTest2
AppendFormatParameterizedTestTest3
AppendFormatParameterizedTestTest23
AppendFormatParameterizedTestTest24
AppendFormatParameterizedTestTest25
AppendFormatParameterizedTestTest26
AppendFormatParameterizedTestTest27
AppendFormatParameterizedTestTest28
Run Checked Tests
Debug Checked Tests
Open Test
New Test List…
Disable
Remove from Test List
Remove from All Test Lists
Cut
Copy
Paste
Delete
Select All
Add/Remove Columns…
Properties
Solution Explorer - Solution 'De…
Solution 'Demo' (3 projects)
Solution Items
Demo
Properties
References
ParameterizedTests.cs
StringBuilder.cs
GeneratedTests
Properties
References
StringBuilderTests.cs
Microsoft.ParameterizedUnit

When run…



Edit View Project Build Debug Tools Test Window Community Help
Hex
tringBuilderTests.cs
StringBuilder.cs
ParameterizedTests.cs
Solution Explorer - Solution 'De...
public static StringBuilder AppendFormat(St
IFormatProvider provider, String format,
{
if (format == null || args == null)
{
throw new ArgumentNullException((fo
}
char[] chars = format.ToCharArray(0, fo
int pos = 0;
int len = chars.Length;
char ch = '\
chars {Dimensions:[0]}
ICustomFormatter cf = null;
if (provider != null)
{
cf = (ICustomFormatter)provider.GetFormat(typeof(ICustomFormat
}
while (true)
{
InvalidCastException occurred
Unable to cast object of type 'System.Object' to type 'System.ICustomFormatter'.
Troubleshooting tips:
When casting from a number, the value must be a number less than infinity.
Make sure the source type is convertible to the destination type.
Get general help for this exception.
Search for more Help Online...
Actions:
View Detail...
Enable editing
Copy exception detail to the clipboard
esults
nikolait@NIKOLAIT1 2006-06-07 10:
Group By [None]
[All Columns]
<Type keyword>
est run: in progress, Results: 5/100 passed; Item(s) checked: 0
Result
Test Name
Project
Error Message
Passed
AppendFormatParameterizedTestTest3
GeneratedTests
Passed
AppendFormatParameterizedTestTest4
GeneratedTests
In Progress
AppendFormatParameterizedTestTest5
GeneratedTests
ll Stack
Test Results
Watch 1
…produces the error

Method sequence generation

Problem definition

Given a class *C* with methods *M*.

Test Sequence Generation

- Given a statement *s* in a method of *M*, compute a sequence of method calls *c*, such that *c* executes *s*

Test Sequence Suite Generation

- Given a set of statements *S* occurring in *M*, compute a set of sequence of method calls *C*, such that forall *s* in *S*, exists *c* in *C*: *c* executes *s*

Observation

We can only reach a statement *s* in a method *m* if we have proper states and arguments available, so that the execution of *m* on that state and argument triggers the execution of *s*

```
List l = new List();
object o = new object();
l.Append(o);
object p = l[l.Count-1];
```

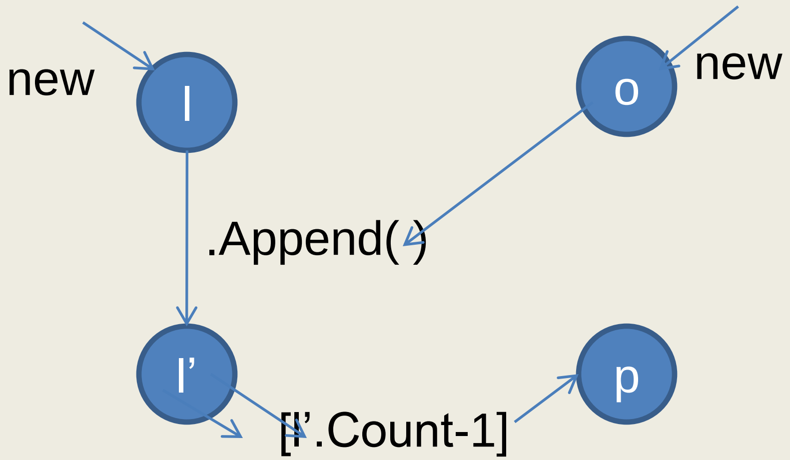
We create new states of objects by calling

- constructors
- methods, if they
  - modify this
  - modify any other formal parameter
  - return a new result

Plans

Plans are DAGs (They shows how to manufacture new objects, arrays, boxed values, and mock objects for interfaces and generics)

- Its nodes are objects
- Its edges are calls to constructors, methods, static fields, whenever they return a new o




```
List l = new List();
object o = new object();
Append(o);
object p = l[l.Count-1];
```

Tests are concrete instances of plans

Plans

Call a method

- With symbol for primitive argument types
- Using other plans for reference argument types

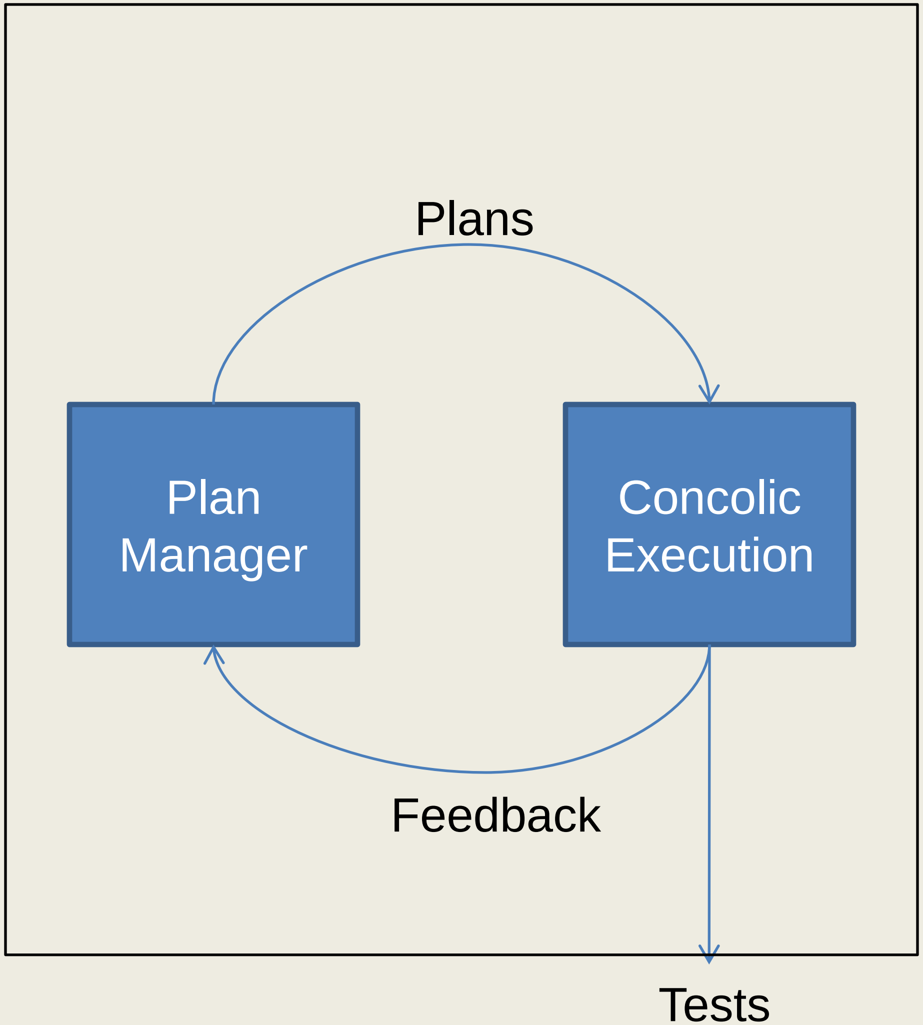
to provide objects

Tests

Call a method

- With concrete values for primitive argument types
- Using simpler tests to build objects

to observe behavior



Observation

During execution we monitor

- what fields a method *actually* reads and write
- what other methods a method *actually* calls
- which arguments *actually* matter
- which instructions are *actually* covered

Method sequence suite generation

(i) Phase: Learn dynamic behavior

- touch all methods once
- gives basic coverage

(ii) Phase: Apply strategies

- order plans so that
  - readers appear after writers
  - methods with coverage potential (transitively) are preferred
- prune plans: Don’t use
  - pure methods to extend plans, unless they return hidden objects
  - methods that throw exceptions to extend plans

Evaluation

- Between 30% and 85% branch coverage on all dlls studied so far
- Found many errors: Nullreferences, IndexOutOfRange , InvalidCasts, Non termination
- Easy to combine with other dynamic checkers: found many resource leaks, incorrect exception handlings (by using fault injection), to be continued…

Compositional Testing

1) Via Parameterized Unit Tests
2) Via Synthesized Specs

V1. Parameterized Unit Tests (PUT)

Adding parameters turns unit tests into specifications

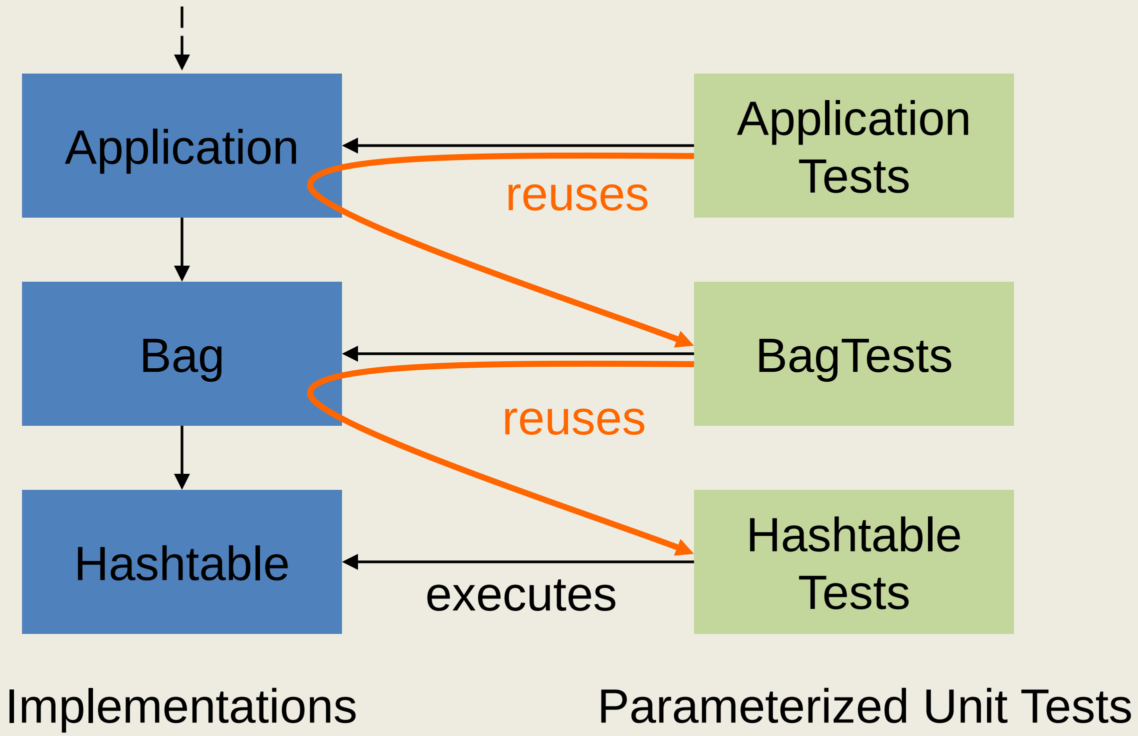
```
void AddAxiom(ArrayList a, object o) {
    Assume.IsTrue(a != null);
    int len = a.Count;
    a.Add(o);
    Assert.IsTrue(a[len] == o);
}
```

Allows to interpret PUT as axioms

$\forall$ ArrayList a, object o.
a!=null $\rightarrow$ let len = a.Count in a.Add(o) $\circ$ a[len] == o

V1. Scale up

- Interpret functions in PUT as uninterpreted symbols
- Use PUTs as rewrite rules for theorem prover



V1. Evaluation

Datatype	# Ops	Input size	Normal PUTs	Excpt. PUTs	# Cases	Time /s	Bugs found
ArrayList	10	3	8	4	34	3.6	1
Enumerator	4	4	4	6	67	9.8	1
Hashtable	9	2	6	5	30	29.9	
Bag (deep)	3	any	3	3	20	37.2	
Bag (shallow)	**3**	**any**	**3**	**3**	**9**	**2.3**	
LinkedList	3	10	3	0	64	3.6	1
RedBlackTree	3	8	3	0	457	427	

V2. Compute Summaries

```
int isPositive(int x) {
  if (x>0) return 1;
  return 0;
}
```

Compute summary in terms of input and state:

- $x>0 \Rightarrow ret = 1$
- $x \leq 0 \Rightarrow ret = 0$

```
int g(int x) {
  if (x<0) return 0;
  int y = crypt();
  if (y == 100) return 0;
  if (x<= 10) return 2;
}
```

Use only functions that prover can decide:

- $x < 0 \Rightarrow ret = 0$
- $x \geq 0 \wedge x \leq 10 \Rightarrow ret = 2$

V2. Algorithm

Compute summaries on the fly in a top down fashion

- Execute f until reaches first function g
- Backtrack over g and compute summary for g
- Continue f with summary of g

Complexity

number of functions in program * number of paths pro function

Summary: Concolic execution has its limitations

- If there are >> 20 methods, don’t test all combinations
  - provide *API protocol* or parameterized scenarios for the possible use
- If a complex function takes a complex data structure as input, then either
  - provide an *invariant* (don’t use the API to generate the data-structure), or
  - (automatically) *partition* the function (based on cohesion) into smaller units that can be tested independently
- If the constraint solver times out, then reduce the number of paths for which constraints have to be solved, ie.
  - apply *compositional* testing, i.e. generate summaries of used methods and then use the summaries for solving constraints

Summary: Concolic execution works!

- Follows the small scope hypothesis; it generates
  - small error revealing data-structures for test inputs
  - short sequences of methods
- Works
  - for TDD, DbC, and also for traditional test
  - for mixed managed/unmanaged setting
  - even when the constraint solver times out
  - compositionally
- Only reports real errors

Thank you

References

- DART: P. Godefroid et al
- Cute: K. Sen et al.
- PUT/Unit Meister: N. Tillmann et al.
- D. Engler et al.

My address

http://research.microsoft.com/~schulte