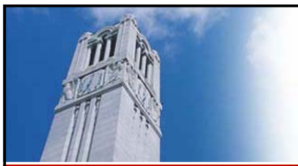


Automated Software Engineering Research Group @NCSU

NC STATE UNIVERSITY Computer Science

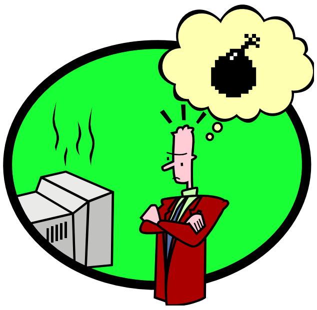
Fitness-Guided Path Exploration in Automated Test Generation

Tao Xie
Department of Computer Science
North Carolina State University
http://ase.csc.ncsu.edu/

Joint work with Nikolai Tillmann, Peli de Halleux, Wolfram Schulte from Microsoft Research

Motivation

- Testing is tedious
- Too easy to miss cases
- Old tests get stale
- Too much legacy code – what does it do?



Automated Software Testing to help

NC STATE UNIVERSITY Computer Science

Outline

- Parameterized Unit Tests and Pex
- Dynamic Symbolic Execution
- Fitness-Guided Path Exploration
- Evaluation
- Conclusion

NC STATE UNIVERSITY Computer Science

Unit Testing Today

A unit test is a small program with assertions.

```
void AddTest()
{
    HashSet set = new HashSet();
    set.Add(7);
    set.Add(3);

    Assert.IsTrue(set.Count == 2);
}
```

Many developers write such unit tests by hand.

NC STATE UNIVERSITY Computer Science 4

Parameterized Unit Testing

```
void AddSpec(int x, int y)
{
    HashSet set = new HashSet();
    set.Add(x);
    set.Add(y);

    Assert.AreEqual(x == y, set.Count == 1);
    Assert.AreEqual(x != y, set.Count == 2);
}
```

Parameterized Unit Tests separate two concerns:
(1) The specification of externally visible behavior (assertions)
(2) The selection of internally relevant test inputs (coverage)

NC STATE UNIVERSITY Computer Science

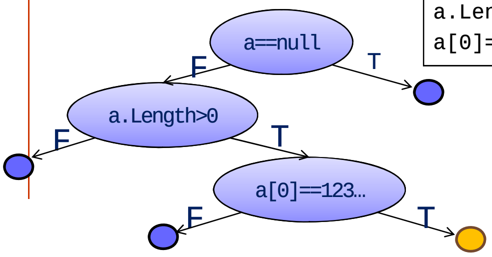
Dynamic Symbolic Execution

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");
}
```

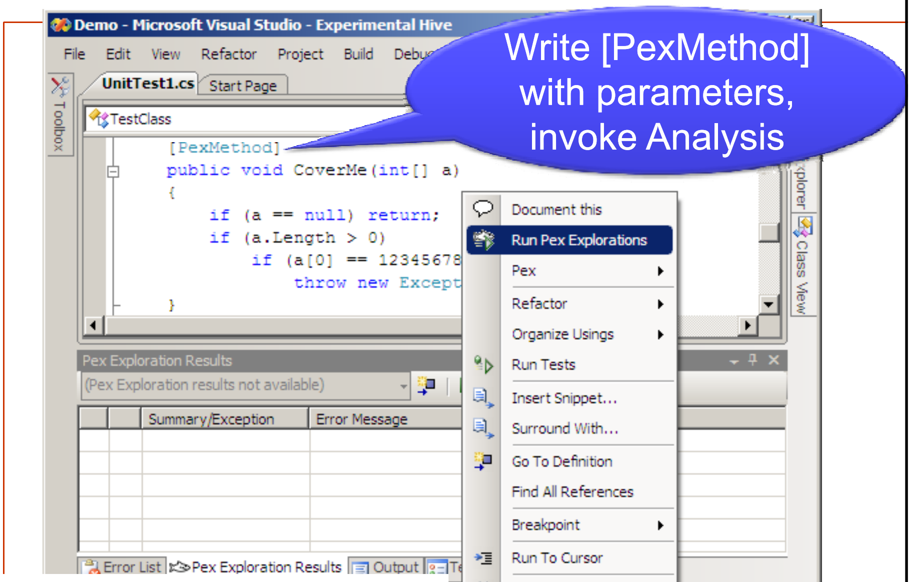
Constraints to solve	Input	Observed constraints
	null	a==null
a!=null	{}	a!=null &&
a!=null a.Length>0		a.Length>0 && a[0]!=1234567890
a!=null && a.Length>0 && a[0]==123456890	{123..}	a==null && a.Length>0 && a[0]==1234567890

Negated condition



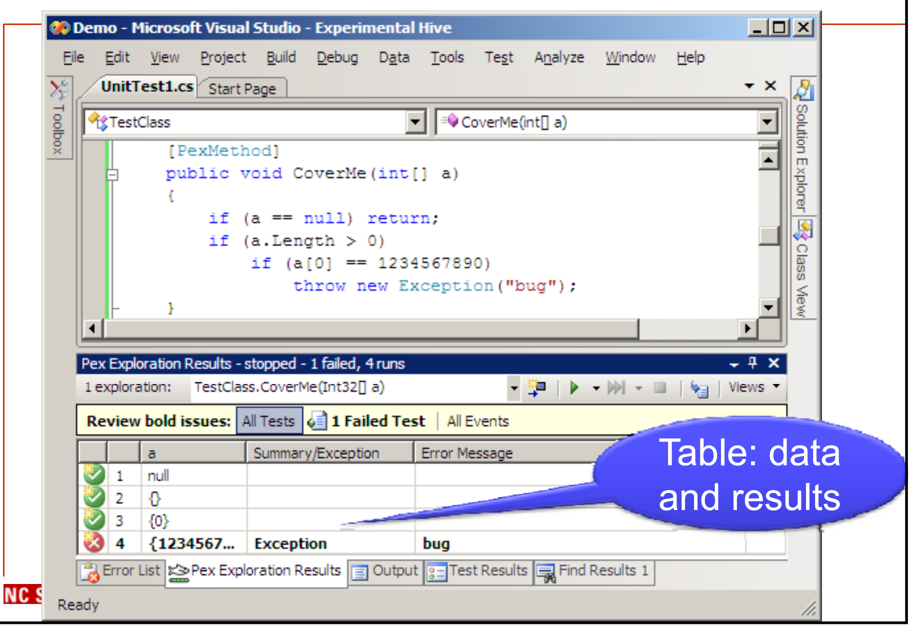
Done: There is no path left.

Parameterized Unit Test in Pex

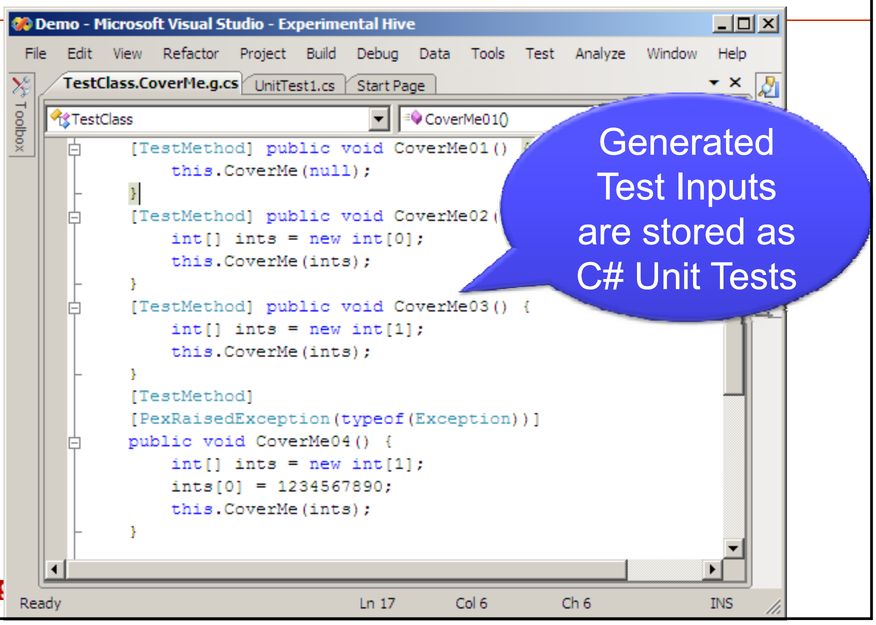


http://research.microsoft.com/Pex

Pex Exploration Results



Generated Unit Tests



Real World Example: ResourceReader

Actual code from .NET base class libraries

Takes stream of bytes, extracts 'resource' chunks

```
[SecurityPermissionAttribute(SecurityAction.LinkDemand, Flags=SecurityPermissionFlag.SerializationFormatter)]
public ResourceReader(Stream stream)
{
    if (stream==null)
        throw new ArgumentNullException("stream");
    if (!stream.CanRead)
        throw new ArgumentException(Environment.GetResourceString("Argument_StreamNotReadable"));

    _resCache = new Dictionary<String, ResourceLocator>(FastResourceComparer.Default);
    _store = new BinaryReader(stream, Encoding.UTF8);
    // We have a faster code path for reading resource files from an assembly.
    _ums = stream as UnmanagedMemoryStream;

    BCLDebug.Log("RESMGRFILEFORMAT", "ResourceReader .ctor(Stream).  UnmanagedMemoryStream: "+(_ums!=null));
    ReadResources();
}
```

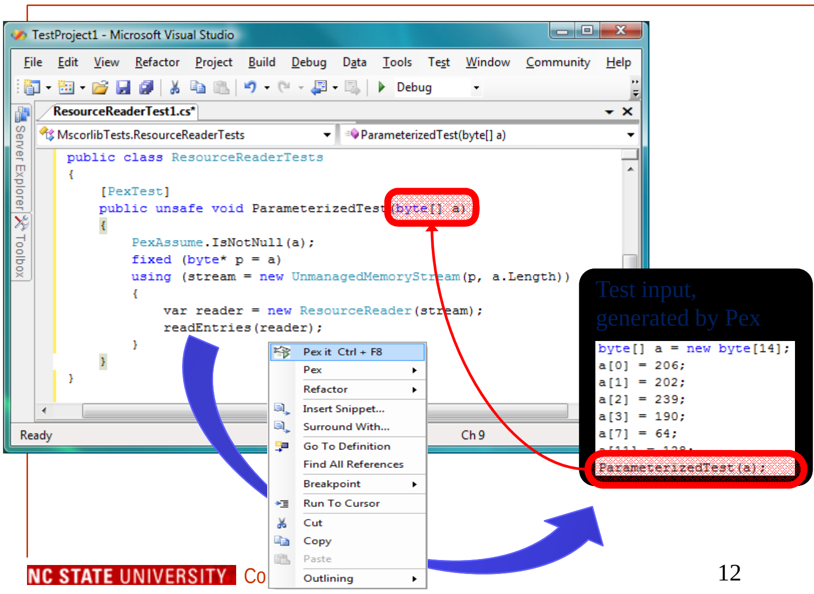
ResourceReader

```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader is closed!");
            BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File |
#if !FEATURE_PAL
            _typeLimitingBinder = new TypeLimitingDeserializationBinder();
            bf.Binder = _typeLimitingBinder;
#endif
            _objFormatter = bf;
            try {
                // Read ResourceManager header
                // Check for magic number
                int magicNum = _store.ReadInt32();
                if
```

```
public virtual int ReadInt32() {
    if (m_isMemoryStream) {
        // read directly from MemoryStream buffer
        MemoryStream mStream = m_stream as MemoryStream;
        BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");

        return mStream.InternalReadInt32();
    }
    else
    {
        FillBuffer(4);
        return (int)(m_buffer[0] | m_buffer[1] << 8 | m_buffer[2] << 16 | m_buffer[3] << 24);
    }
}
```


Pexing ResourceReader



Division of Testing Labor

Parameterized Unit Tests (PUTs) separate two concerns:

- The specification of external behavior (i.e., assertions)

- The selection of internal test inputs (i.e., coverage)

NC STATE UNIVERSITY Computer Science

PUTs == Algebraic Specifications

- A PUT can be read as a universally quantified, conditional axiom.

  ∀ int name, int data.
      name ≠ null ∧ data ≠ null ⇒
      equals(
          ReadResource(name,
              WriteResource(name, data)),
          data)

- Teaching/training of writing specs is challenging but we do have success with teaching PUT/Pex

NC STATE UNIVERSITY Computer Science http://sites.google.com/site/teachpex

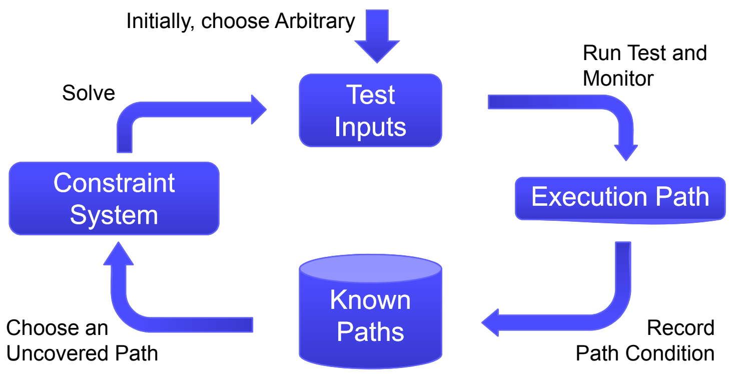
Dynamic Symbolic Execution

Dynamic symbolic execution (DSE) combines static and dynamic analysis:

- Execute a program multiple times with different inputs
  - build ***path condition***: input constraints for the execution path on the side
  - plug in ***concrete results*** of operations which cannot reasoned about symbolically
- Use a constraint solver to obtain new inputs
  - solve a constraint system that represents an execution path not seen before

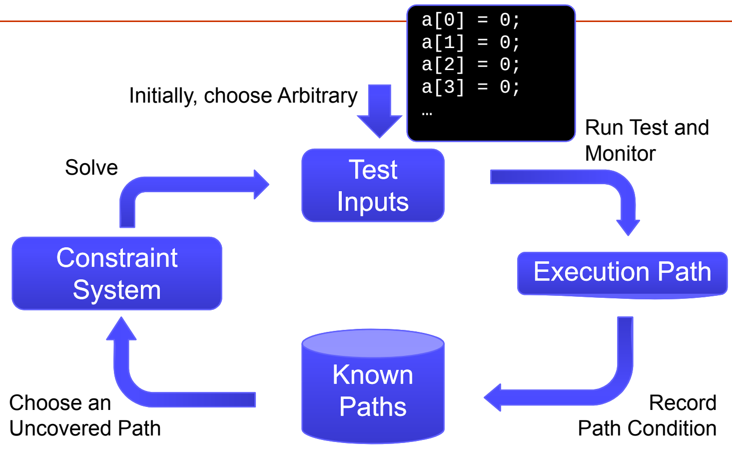
NC STATE UNIVERSITY Computer Science

Dynamic Symbolic Execution



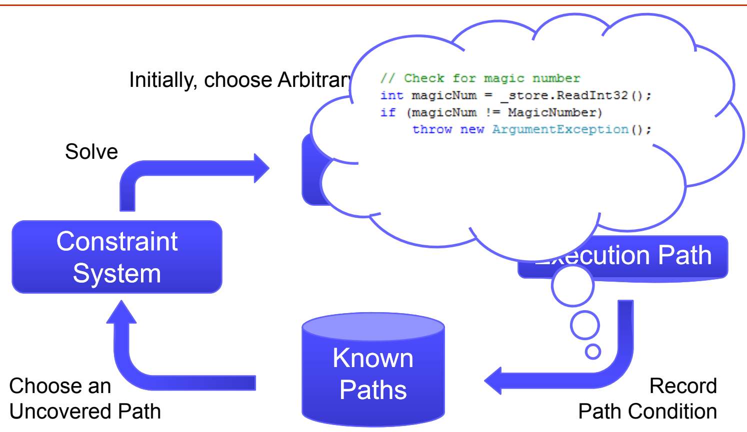
NC STATE UNIVERSITY Computer Science

Dynamic Symbolic Execution



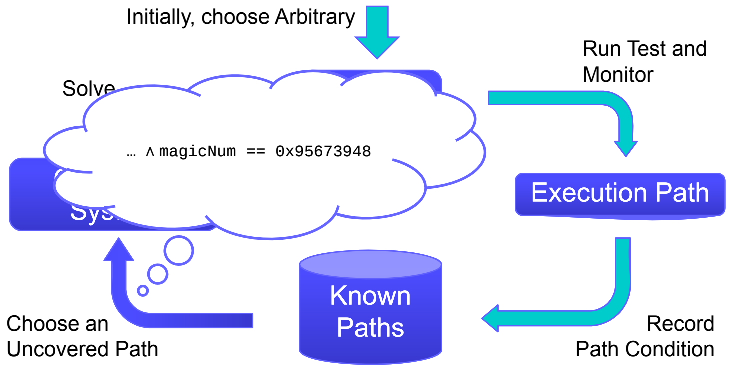
NC STATE UNIVERSITY Computer Science

Dynamic Symbolic Execution



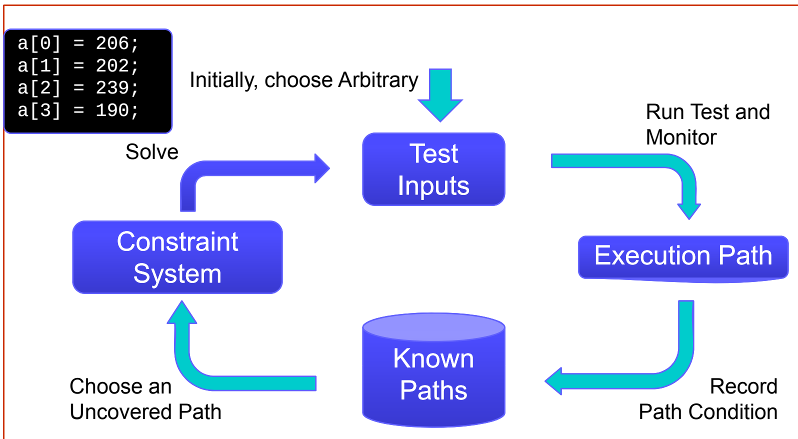
NC STATE UNIVERSITY Computer Science

Dynamic Symbolic Execution



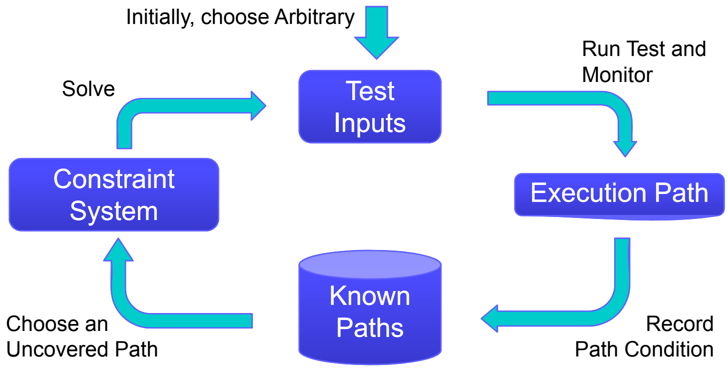
NC STATE UNIVERSITY Computer Science

Dynamic Symbolic Execution



NC STATE UNIVERSITY Computer Science

Dynamic Symbolic Execution



DSE Example - Loop

```
public bool TestLoop(int x, int[] y) {
   if (x == 90) {
         for (int i = 0; i < y.Length; i++)
                if (y[i] == 15)
                     x++;
         if (x == 110)
                return true;
   }
   return false;
}
```

TestLoop(0, {0})

Path condition:
!(x == 90)
↓
New path condition:
(x == 90)
↓
New test input:
TestLoop(90, {0})

DSE Example - Loop

```
public bool TestLoop(int x, int[] y) {
   if (x == 90) {
        for (int i = 0; i < y.Length; i++)
             if (y[i] == 15)
                  x++;
        if (x == 110)
             return true;
   }
   return false;
}
```

TestLoop(90, {0})

Path condition:
(x == 90) && !(y[0] ==15)
↓
New path condition:
(x == 90) && (y[0] ==15)
↓
New test input:
TestLoop(90, {15})

NC STATE UNIVERSITY Computer Science

Challenge in DSE - Loop

```
public bool TestLoop(int x, int[] y) {
   if (x == 90) {
        for (int i = 0; i < y.Length; i++)
             if (y[i] == 15)
                  x++;
        if (x == 110)
             return true;
   }
   return false;
}
```

TestLoop(90, {15})

Path condition:
(x == 90) && (y[0] ==15)
&& !(x+1 == 110)
↓
New path condition:
(x == 90) && (y[0] ==15)
&& (x+1 == 110)
↓
New test input:
No solution!?

NC STATE UNIVERSITY Computer Science

A Closer Look

```
public bool TestLoop(int x, int[] y) {
   if (x == 90) {
         for (int i = 0; i < y.Length; i++)
               if (y[i] == 15)
                  x++;
         if (x == 110)
               return true;
   }
   return false;
}
```

TestLoop(90, {15})

Path condition:
(x == 90) && (y[0] ==15)
&& (0 < y.Length)
&& !(1 < y.Length)
&& !(x+1 == 110)
↓
New path condition:
(x == 90) && (y[0] ==15)
&& (0 < y.Length)
&& (1 < y.Length)
→ **Expand array size**

A Closer Look

```
public bool TestLoop(int x, int[] y) {
   if (x == 90) {
         for (int i = 0; i < y.Length; i++)
               if (y[i] == 15)
                  x++;
         if (x == 110)
               return true;
   }
   return false;
}
```

TestLoop(90, {15})

We can have infinite paths!

Manual analysis → need at least 20 loop iterations to cover the target branch

Exploring all paths up to 20 loop iterations is infeasible: 2^{20} paths

Fitnex: Fitness-Guided Exploration

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
   for (int i = 0; i < y.Length; i++)
      if (y[i] == 15)
        x++;
   if (x == 110)
      return true;
 }
 return false;
}
```

TestLoop(90, {15, 0})
TestLoop(90, {15, 15})

Key observations: with respect to the coverage target
- not all paths are equally promising for branch-node flipping
- not all branch nodes are equally promising to flip

- Our solution:
  - Prefer to flip branch nodes on the most ***promising*** paths
  - Prefer to flip the most ***promising*** branch nodes on paths
  - Fitness function to measure "promising" extents

Fitness Function

- Compute fitness value (distance between the current state and the goal state)
- Search tries to minimize fitness value

Predicate	Fitness function	
	True	False
$F(a == b)$	0	$\|a - b\|$
$F(a > b)$	0	$(b - a) + K$
$F(a >= b)$	0	$(b - a)$
$F(a < b)$	0	$(a - b) + K$
$F(a <= b)$	0	$(a - b)$
$F(P_1 \&\& P_2)$	0	$F(P_1) + F(P_2)$
$F(P_1 \|\| P_2)$	0	$(F(P_1) * F(P_2))/(F(P_1) + F(P_2))$

[Tracey et al. 98, Liu at al. 05, …]

Fitness Function for (x == 110)

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
   for (int i = 0; i < y.Length; i++)
     if (y[i] == 15)
       x++;
   if (x == 110)
     return true;
 }
 return false;
}
```

Fitness function: |110 - x |

NC STATE UNIVERSITY Computer Science

Compute Fitness Values for Paths

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
   for (int i = 0; i < y.Length; i++)
     if (y[i] == 15)
       x++;
   if (x == 110)
     return true;
 }
 return false;
}
```

Fitness function: |110 - x |

(x, y)	Fitness Value
(90, {0})	20
(90, {15})	19
(90, {15, 0})	19
(90, {15, 15})	18
(90, {15, 15, 0})	18
(90, {15, 15, 15})	17
(90, {15, 15, 15, 0})	17
(90, {15, 15, 15, 15})	16
(90, {15, 15, 15, 15, 0})	16
(90, {15, 15, 15, 15, 15})	15
…	

Give preference to flip paths with better fitness values

We still need to address which branch node to flip on paths …

NC STATE UNIVERSITY Computer Science

Compute Fitness Gains for Branches

```
public bool TestLoop(int x, int[] y) {
 if (x == 90) {
   for (int i = 0; i < y.Length; i++)
       if (y[i] == 15)
         x++;
   if (x == 110)
       return true;
 }
 return false;
}
```

Fitness function: |110 - x |

Branch b1: i < y.Length
Branch b2: i >= y.Length
Branch b3: y[i] == 15
Branch b4: y[i] != 15

(x, y)	Fitness Value
(90, {0})	20
(90, {15}) ← flip b4	19
(90, {15, 0}) ← flip b2	19
(90, {15, 15}) ← flip b4	18
(90, {15, 15, 0}) ← flip b2	18
(90, {15, 15, 15}) ← flip b4	17
(90, {15, 15, 15, 0}) ← flip b2	17
(90, {15, 15, 15, 15}) ← flip b4	16
(90, {15, 15, 15, 15, 0}) ← flip b2	16
(90, {15, 15, 15, 15, 15}) ← flip b4	15
…	

•Flipping Branch b4 (b3) gives us average 1 (-1) fitness gain (loss)
•Flipping branch b2 (b1) gives us average 0 fitness gain (loss)

Compute Fitness Gain for Branches cont.

- For a flipped node leading to *Fnew*, find out the old fitness value *Fold* before flipping
  - Assign Fitness Gain (*Fold - Fnew*) for the branch of the flipped node
  - Assign Fitness Gain (*Fnew - Fold*) for the other branch of the branch of the flipped node
- Compute the average fitness gain for each branch over time

Search Frontier

- Each branch node candidate for being flipped is prioritized based on its composite fitness value:
  - (Fitness value of node – Fitness gain of its branch)
- Select first the one with the best composite fitness value
- To avoid local optimal or biases, the fitness-guided strategy is integrated with Pex's previous search strategies

NC STATE UNIVERSITY Computer Science

Evaluation Subjects

- A collection of micro-benchmark programs routinely used by the Pex developers to evaluate Pex's performance, extracted from real, complex C# programs

Ranging from string matching like
if (value.StartsWith("Hello") &&
value.EndsWith("World!") &&
value.Contains(" "))
to a small parser for a Pascal-like language where the target is to create a legal program

subject	#basic blocks	subject	#basic blocks
1	9	16	9
2	16	17	40
3	29	18	18
4	40	19	11
5	20	20	18
6	28	21	25
7	21	22	25
8	34	23	19
9	29	24	16
10	25	25	44
11	27	26	11
12	27	27	9
13	27	28	9
14	39	29	21
15	34	30	62

NC STATE UNIVERSITY Computer Science

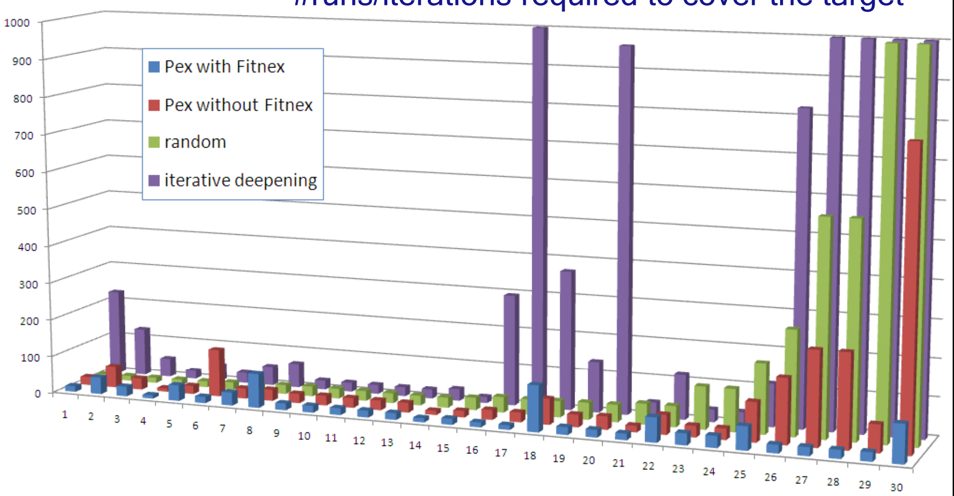
Techniques under Comparison

- Pex with the Fitnex strategy
- Pex without the Fitnex strategy
  - Pex's previous default strategy
- Random
  - a strategy where branch nodes to flip are chosen randomly in the already explored execution tree
- Iterative Deepening
  - a strategy where breadth-first search is performed over the execution tree

NC STATE UNIVERSITY Computer Science

Evaluation Results

#runs/iterations required to cover the target



Pex w/o Fitnex on average by a factor of 1.9 improvement over *Random*
Pex w/ Fitnex on average by a factor of 5.2 improvement over *Random*

NC STATE UNIVERSITY Computer Science

Impact

- Since Sept 17, 2008, Pex releases' default exploration strategy integrates Fitnex
  - http://research.microsoft.com/Pex
- Fitnex is released as open source
  - http://www.codeplex.com/Pex
- Download counts of Pex in early Nov 2008
  - About 4000 after available for about half a year.
  - About 1000 of Pex for Visual Studio 2010 Community Technology Preview (Microsoft Incubation Software) after available for about two weeks

NC STATE UNIVERSITY Computer Science

Case Study on Pex [TAP 2008]

- A previous version of Pex was applied on a core .NET component
  - Already extensively tested for several years
  - Assertions written by developers
  - >10,000 public methods
  - >100,000 basic blocks
- Found a significant number of benign bugs, e.g. NullReferenceException, IndexOutOfRangeException, …
- 17 unique bugs involving
  - violation of developer-written assertions,
  - exhaustion of memory,
  - other serious issues.

NC STATE UNIVERSITY Computer Science

Ongoing/Future Work

- Method sequence generation
- Regression test generation
- String generation (e.g., regular expressions)
- Environment mocking
- Test generalization
- Guidance from tool users
- …

Conclusion

- Parameterized Unit Tests separate
  - Manual specification of external behavior
  - Pex's selection of internal test inputs
- Dynamic Symbolic Execution enables Pex to deal with various complications
- Real-world challenges of path explosion call for guided path exploration
  - Fitness values of explored paths
  - Fitness gains of branches' past flipping
- Evaluation results show the effectiveness of the new Fitnex strategy
- Fitnex has been integrated in Pex' default strategy

Questions?

http://ase.csc.ncsu.edu/
http://research.microsoft.com/Pex
http://www.codeplex.com/Pex

Constraint Solving: Z3

- SMT-Solver ("Satisfiability Modulo Theories")
  - Decides logical first order formulas with respect to theories
  - SAT solver for Boolean structure
  - Decision procedures for relevant theories: uninterpreted functions with equalities, linear integer arithmetic, bitvector arithmetic, arrays, tuples
- Model generation for satisfiable formulas
  - Models used as test inputs
- Incremental solving
  - Enables efficient model minimization