Deep Dive into Pex
How Pex works, implications for design of Code Hunt puzzles

Nikolai Tillmann
Principal Software Engineering Manager
Microsoft, Redmond, USA
Agenda

• Dynamic Symbolic Execution with Pex
  • Symbolic state representation, constraint solving, the environment
• The Pex Family
• Code Hunt Deep Dive
  • How the game works
  • Pex in the cloud @ api.codehunt.com
  • Inputs and outputs, assumptions, overflows
  • Path explosion
  • Sandbox, and how to peek below
  • Side effects
  • Forcing values by branching
  • Back end: Public REST APIs
Dynamic Symbolic Execution with Pex

Symbolic state representation, constraint solving, the environment
Concolic Execution [Godefroid et al. 05][Sen et al. 05][Tillmann et al. 08]

Combines concrete and symbolic execution.

Algorithm:

\[ \text{Set } J := \emptyset \]  (J is set of already analyzed program inputs)

\[ \text{Loop} \]

\[ \text{Choose program input } i \notin J \quad \text{(stop if no such } i \text{ can be found)} \]

\[ \text{Output } i \]

\[ \text{Execute } P(i); \text{ record path condition } C \]  (in particular, \( C(i) \) holds)

\[ \text{Set } J := J \cup C \quad \text{(viewing } C \text{ as the set } \{ i \mid C(i) \} ) \]

\[ \text{End loop} \]

Loop does not terminate if number of execution paths is infinite (in the presence of loops/recursion)

Dynamic Symbolic Execution

• This choice decides search order
• Search order decides how quick we can achieve high code coverage!
• Incomplete constraint-solver leads to under-approximation
void CoverMe(int[] a)
{
    if (a == null) return;
    if (a.Length > 0)
        if (a[0] == 1234567890)
            throw new Exception("bug");
}

---

**Constraints to solve**

<table>
<thead>
<tr>
<th>Constraints to solve</th>
<th>Input</th>
<th>Observed constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>a==null</td>
<td>null</td>
<td>a==null</td>
</tr>
<tr>
<td>a!=null &amp;&amp; a.Length&gt;0</td>
<td>{}</td>
<td>a!=null &amp;&amp; !(a.Length&gt;0)</td>
</tr>
<tr>
<td>a==null &amp;&amp; a.Length&gt;0</td>
<td>{0}</td>
<td>a==null &amp;&amp; a.Length&gt;0 &amp;&amp; a[0]!=1234567890</td>
</tr>
<tr>
<td>a==null &amp;&amp; a.Length&gt;0 &amp;&amp; a[0]==1234567890</td>
<td>{123}</td>
<td>a==null &amp;&amp; a.Length&gt;0 &amp;&amp; a[0]==1234567890</td>
</tr>
</tbody>
</table>

**Observed constraints**

- a==null
- a!=null && !(a.Length>0)
- a==null && a.Length>0 && a[0]!=1234567890
- a==null && a.Length>0 && a[0]==1234567890

**Done:** There is no path left.

http://pex4fun.com/CoverMe
Symbolic State Representation

Representation of symbolic values and state is similar to the ones used to build verification conditions in ESC/Java, Spec#, ...

Terms for
- Primitive types (integers, floats, ...), constants, expressions
- Struct types by tuples
- Instance fields of classes by mutable "mapping of references to values"
- Elements of arrays, memory accessed through unsafe pointers by mutable "mapping of integers to values"

Efficiency by
- Many reduction rules, including reduction of ground terms to constants
- Sharing of syntactically equal sub-terms
- BDDs over if-then-else terms to represent logical operations
- Patricia Trees to represent AC1 operators (including parallel array updates)
Constraint Solving

• 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
• Limitations
  • We are not using decision procedure for floating point arithmetic and strings
  • Instead, heuristic search-based approaches
• Pex uses Z3: http://research.microsoft.com/z3
Dynamic Symbolic Execution Exercises

CodeMe
All explicit branches.

ArrayIndexLength
Pex knows about all implicit, exception-throwing control-flow branches

ArrayHeap
Pex models the heap

Assert, Assert123
Assertions connect code coverage and correctness
Note: Pex actually runs your code

- **Dynamic** symbolic execution
- Behavior of environment is unknown
- Pex comes with a built-in way to isolate code from environment dependencies ("Moles")

```csharp
void CoverMe()
{
    var lines = File.ReadAllLines("a.txt");
    if (lines[0] == "[complicated]")
        throw new Exception("bug");
    if (lines[1] == "[clear]")
        Disk.Format("c:");
}
```
The Pex Family
Timeline + Impact
• **Pex** (released May 2008, Microsoft Research download)
  • 30,388 downloads (20 months, Feb 08-Oct 09)
  • **Ships** with Visual Studio 2015 Ultimate “in the box” as **Smart Unit Tests**

• **Moles** (released September 2009, Microsoft Research download)
  • **Shipped** with Visual Studio 2012 Ultimate “in the box” as **Fakes**

• **Pex4Fun** website (released June 2010)
  • 1.6 million user interactions (clicks on “Ask Pex”)

• **Code Digger** (simplified Pex, released on April 2013 as VS Gallery download)
  • 22,466 downloads (10 months, Apr 13-Jan 14)

• **Code Hunt** website (released May 2014)
Code Hunt Deep Dive

How the game works, Pex in the cloud @ api.codehunt.com, Inputs and outputs, assumptions, overflows, Path explosion, Sandbox, and how to peek below, Side effects, Forcing values by branching, Back end: Public REST APIs
Code Hunt programming game
Code Hunt programming game
Code Hunt programming game

SECTOR 00

00.01 TUTORIAL
00.02
00.03
00.04
public class Program {
    public static int Puzzle(int x) {
        return 0;
    }
}
Discover the arithmetic operation applied to 'x'.

public class Program {
    public static int Puzzle(int x) {
        return 0;
    }
}

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>EXPECTED RESULT</th>
<th>YOUR RESULT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>✗</td>
<td>0</td>
<td>1</td>
<td>Mismatch</td>
</tr>
<tr>
<td>2</td>
<td>✓</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

(Level: 00.02)
Discover the arithmetic operation applied to 'x'.

```java
public class Program {
    public static int Puzzle(int x) {
        return 1;
    }
}
```
Discover the arithmetic operation applied to 'x'.

```java
public class Program {
    public static int Puzzle(int x) {
        if (x == -1) {
            return 0;
        } else if (x == 0) {
            return 1;
        } else if (x == 1) {
            return 2;
        } else {
            return 0;
        }
    }
}
```
Discover the arithmetic operation applied to 'x'.

```java
public class Program {
    public static int Puzzle(int x) {
        return x+1;
    }
}
```
You repaired and captured the code fragment.

**SKILL RATING:**

you wrote elegant code!

TOTAL SCORE: 6

KEEP TRYING  NEXT
More difficult level

SELECT SECTOR

- 00 TRAINING
- 01 ARITHMETIC
- 02 LOOPS
- 03 LOOPS 2
- 04 CONDITIONALS
- 05 CONDITIONALS 2
- 06 STRINGS
- 07 STRINGS 2
- 08 NESTED LOOPS
- 09 1D ARRAYS
- 10 JAGGED ARRAYS
- 11 ARRAYS 2
- 12 SEARCH SORT
- 13 CYPHERS
- 14 PUZZLES
Try to capture the code fragment!

```java
class Program {
    public static int Puzzle(int lowerBound, int upperBound) {
        return lowerBound * upperBound;
    }
}
```
public class Program {
    public static int Puzzle(int lowerBound, int upperBound) {
        return lowerBound * upperBound;
    }
}

Try to capture the code fragment!

[Table]

<table>
<thead>
<tr>
<th>LOWERBOUND</th>
<th>UPPERBOUND</th>
<th>EXPECTED RESULT</th>
<th>YOUR RESULT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>40.320</td>
<td>8</td>
<td>Mismatch</td>
</tr>
<tr>
<td>15</td>
<td>24</td>
<td>24496.3328</td>
<td>360</td>
<td>Mismatch</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>272</td>
<td>272</td>
<td></td>
</tr>
</tbody>
</table>

You may find a loop useful on this level.
public class Program {
    public static int Puzzle(int lowerBound, int upperBound) {
        int r = 1;
        for(int i = lowerBound; i < upperBound; i++)
            r *= i;
        return r;
    }
}
It’s a game!

iterative gameplay
adaptive
personalized
no cheating
clear winning criterion
class Secret {
    public static int Puzzle(int x) {
        if (x <= 0) return 1;
        return x * Puzzle(x - 1);
    }
}

class Player {
    public static int Puzzle(int x) {
        return x;
    }
}

class Test {
    public static void Driver(int x) {
        if (Secret.Puzzle(x) != Player.Puzzle(x))
            throw new Exception("Mismatch");
    }
}
Pex in the cloud @ api.codehunt.com

• Pex performs dynamic symbolic execution in a sandbox
• 32KB compressed code limit (deflate compression of UTF8-encoded program text)
• Single-threaded code only
• Default Pex search strategy for path selection
• 2s timeout for each Z3 query, 30s overall timeout
• If any discovered path exceeds some instruction limit (~100,000), you lose (likely termination issue)
• If Pex doesn’t find counterexample, you win
• Secret program can trim / shape input space via assumptions
Inputs and outputs

• The puzzle signature (parameters and result) may refer to...
  • simple datatypes (byte, bool, char, int, double, string, ...)
    (however, avoid floating point number computations)
  • arrays of simple data types
  • that’s it.

• Generated driver code...
  • First calls secret program, then user program
  • Passes inputs to both
  • Compares results: values have to be equal
    (deep equality for arrays / strings), exceptions types (if any) have to match

```java
class Secret {
    public static int Puzzle(int x) {
        if (x <= 0) return 1;
        return x * Puzzle(x - 1);
    }
}

class Player {
    public static int Puzzle(int x) {
        return x;
    }
}

class Test {
    public static void Driver(int x) {
        // simplified, similar for exceptions
        if (Secret.Puzzle(x) != Player.Puzzle(x))
            throw new Exception("Mismatch");
    }
}
```
Assumptions

using Microsoft.Pex.Framework;
[...]  
PexAssume.IsTrue(…)

• Assumptions act as filters on input values
• Only the secret code is allowed to contain assumptions
Overflows

• Pex faithfully models the default behavior of C# / .NET.

```csharp
public static void Puzzle(int x) {
    if (x + 10 < x) throw new Exception("what?");
}
```

Use an assumption to limit input space if you don’t want to confuse people.

```csharp
PexAssume.IsTrue(x <= int.MaxValue - 10);
```
Path explosion

- Pex tries to flip execution at each MSIL branching instruction => avoid branches!
- Prefer strict Boolean expressions over short circuit
- Use PexAssume to impose bounds.

```csharp
public static void Puzzle(int[] a) {
    PexAssume.IsTrue(a != null && a.Length == 100);
    bool condition = true;
    for (int i=0; i<100; i++) {
        condition = condition & a[i] > i;
    }
    PexAssume.IsTrue(condition);
}
```
Reminder: Pex actually runs your code...

• In Code Hunt, white-list of APIs for sandboxing
  
  System.IO.Directory.Delete("c:\");
  
  => “Disallowed dependencies”

• Interaction with PexSymbolicValue
  
  using Microsoft.Pex.Framework;
  
  [...]
  
  var pc = PexSymbolicValue.GetPathConditionString();
  
  Console.WriteLine(pc);
Side effects

• White-listed APIs: many non-deterministic APIs excluded (e.g., System.Random)

• In code:
  • Avoid static fields (except possible for deterministic initialize-once cases)
  • Do not mutate incoming arrays: effects are visible to user code
Forcing values

• You can introduce benign branches to force Pex to generate certain test cases

```csharp
public static void Puzzle(int[] a) {
    if (a != null && a.Length == 10 && a[3] == 27) {} // Forcing value
    if (a != null && a.Length == 20 && a[13] == 42) {} // Forcing value
}
```
Back end @ api.codehunt.com

Code Hunt Services

**Running** capacity 12, load 1%

Email [codehunt@microsoft.com](mailto:codehunt@microsoft.com) for more information.

**overview**

api.codehunt.com is a cloud service that provides remote access to the code exploration and test generation functionality of [Pex](http://pex.codeplex.com) for .NET programs.

In particular, api.codehunt.com powers the [Code Hunt](http://codehunt.codeplex.com) game.

**disclaimer**

The APIs are subject to change. The state managed by api.codehunt.com may reset at any time.
Statistics
Users: 2044477
User Programs: 9458818
User Explorations: 7490881
Programs: 5309700
Explorations: 4099675

APIs
Each API is given via its method (GET or POST), its path (/api/something), an optional request body, and a list of possible response codes and bodies. Requests and response bodies, if any, are in JSON format, and are specified using TypeScript interface syntax.

authorization
Most APIs require a special authorization header with a bearer token. If you do not send the following header with the APIs, you will get a 401 Unauthorized status code.

Authorization: Bearer $ACCESS_TOKEN

There are two ways to get an access token: 1) anonymously (which creates a new anonymous user account on-the-fly), or 2) by referring to a regular user account. To get an anonymous $ACCESS_TOKEN, do the following.

POST /api/token?grant_type=client_credentials&client_id=anonymous&client_secret=anonymous
   response 200 OR
   body: TokenInfo

interface TokenInfo {
   access_token: string;
}

Anonymously obtained access tokens have severe usage restrictions. If you want to obtain a regular user account (represented by the pair client_id and client_secret), send a request by email to codehunt@microsoft.com.

merging
You can merge all data from an anonymously obtained account into a regular user account. You need to obtain an access token for the anonymous account from which data will be taken; then call the following API authorized by the regular target user to whom the data is copied.

POST /api/merge
Summary

Code Hunt is a serious programming game powered by Pex, an industrial-strength dynamic symbolic execution engine.

www.codehunt.com
api.codehunt.com
research.microsoft.com/Pex
research.microsoft.com/CodeHunt