Program Analysis 2.0
Thomas Ball
Microsoft Research
Program Analysis 1.0 (1999-200?)

1. Legacy focus
2. The static analysis genie
3. Scale via aggressive abstraction
4. The genie out of the bottle → false alarms
Program Analysis 2.0

1. Constraints are empowering
2. Diversification $\rightarrow$ growth
3. Scale by decomposition
4. Rebottle the genie!
Some Program Analysis 2.0 Tools

- **Code contracts for .NET**
  - MSIL rewriting, dynamic + static checking

- **Automatic test data generation (Pex)**
  - Symbolic execution with SMT solvers (Z3)

- **Systematic concurrency testing (CHESS)**
  - Direct model checking of code
Available for Academic/Commercial Use

• Academic
  ◦ http://research.microsoft.com/contracts/
  ◦ http://research.microsoft.com/pex/
  ◦ http://research.microsoft.com/chess/

• Commercial
  ◦ http://msdn.com/devlabs/
Code Contracts for .NET 4.0

- Mike Barnett
- Manuel Fähndrich
- Francesco Logozzo
class Rational {
    public Rational(int n, int d) {
        Contract.Requires(0 < d);
        this.N = n;
        this.D = d;
    }
}
What Contracts Can I Write?

- **Requires**
  - What must be true at method entry
- **Ensures**
  - What must be true at method exit
- **Invariants**
  - What must be true at all method exits
- **Assertions**
  - What must be true at a particular point
- **Assumptions**
  - What should be true at a particular point
What Can I Put In A Contract?

- Any Boolean expression
  - In your favorite programming language!
  - Including method calls (but must be marked Pure)
- Quantifiers
  - `Contract.ForAll(0,A.Length,i => A[i] > 0);`
  - `Contract.Exists(0,A.Length,i => A[i] > 0);`
- `Contract.Result`
  - refer to the return value of the method
- `Contract.OldValue`
  - refer to values at method entry
**How Do I Write A Contract?**

```csharp
public virtual int Add(object value) {
    Contract.Requires(value != null);
    Contract.Ensures(Count == Contract.OldValue(Count) + 1);
    Contract.Ensures(Contract.Result<int>() == Contract.OldValue(Count));
    if (count == items.Length) EnsureCapacity(count + 1);
    items[count] = value;
    return count++;
}
```

```csharp
void ObjectInvariant() {
    Contract.Invariant(items != null);
}
```

**Features**
- Declarative
- Language expression syntax
  - Type checking / IDE
- Special Encodings
  - Result and OldValue
public virtual int Add(object value) {
    Contract.Requires(value != null);
    Contract.Ensures(Count == Contract.OldValue(Count) + 1);
    Contract.Ensures(Contract.Result<int>() == Contract.OldValue(Count));
    if (_size == _items.Length) EnsureCapacity(_size + 1);
    _items[_size] = value;
    return _size++;
}
Demo of Heap example
Code Contracts Summary

- Enables contracts in all .NET languages
  - No restrictions on what can be expressed

- Contract library a core component of .NET 4.0

- Same contracts used for
  - Runtime checking
  - Static checking
  - Documentation generation
White Box Test Generation for .NET

Nikolai Tillmann, Peli de Halleux
Microsoft Research
Purpose: Test input generator
- Start from unmodified code (or code with contracts)
- Generated tests emitted as traditional unit tests
- Goal: test suite that covers all reachable statements

Technology: Dynamic symbolic execution
- Whole-program, white-box code analysis
- At the level of the .NET instructions (bytecode)
- Symbolic execution based on monitoring and re-execution
- Constraint solver (Z3) determines test inputs for new paths
void CoverMe(int[] a) {
    if (a == null) return;
    if (a.Length > 0) {
        if (a[0] == 1234567890)
            throw new Exception("bug");
    }
}
How to test this code?
(Actual code from .NET base class libraries.)
Real World (II)

```csharp
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");
                if (magicNum == 0x1) {
                    return mStream.InternalReadInt32();
                }
            }
        } else {
        FillBuffer(4);
        }
        // Read in type name for a suitable ResourceReader
    } // New Resource Virtual Functions - Resource-specific Reader
```
Demos
- test generation of Heap with contracts-
- test the ResourceReader-
Summary

- **Pex** automates test input generation for .NET programs
- **Pex** enables Parameterized Unit Testing
- Used in Microsoft to test core .NET components

http://research.microsoft.com/pex
Concurrency Analysis Platform And Tools For Finding Concurrency Bugs

Thomas Ball
Principal Researcher
Microsoft Corporation

Sebastian Burckhardt
Researcher
Microsoft Corporation

Madan Musuvathi
Researcher
Microsoft Corporation

Shaz Qadeer
Senior Researcher
Microsoft Corporation
Thread Interleavings

Thread 1
- x++; 
- x++;

Thread 2
- x*=2; 
- x*=2; 

Diagram:
```
  0
 / \
1   2
|   |
2   3
|   |
4   5
|   |
8   6
```
Concurrency Analysis Platform (CAP)

- **Goal:** Drive a program along an interleaving of choice
  - Interleaving decided by user or by a program/tool

- **Today:** Controlling/observing concurrency is difficult
  - Manual and intrusive process

- Enables lots of concurrency tools:
  - Test a program along a set of interleavings
  - Reproduce Heisenbugs
  - Program understanding / debugging
  - ...
Record the interleaving executed
Drive the program along an interleaving
CAP Specifics

- Ability to explore all interleavings
  - Need to understand complex concurrency APIs (Win32 and System.Threading)
  - Threads, threadpools, locks, semaphores, async I/O, APCs, timers, ...

- Does not introduce false behaviors
  - Any interleaving produced by CAP is possible on the real scheduler
CHESS: Find And Reproduce Heisenbugs

While(not done) {
    TestScenario()
    ...
}

Win32 API

CHESS runs the scenario in a loop
- Every run takes a different interleaving
- Every run is repeatable

Uses the CAP scheduler
- To control and direct interleavings

Detect
- Assertion violations
- Deadlocks
- Dataraces
- Livelocks

Kernel:
Threads, Scheduler, Synchronization Objects
CHESS Internal Customers

- Parallel Computing Platform
  - PLINQ: Parallel LINQ
  - CDS: Concurrent Data Structures
  - STM: Software Transactional Memory
  - TPL: Task Parallel Library
  - ConcRT: Concurrency RunTime
  - CCR: Concurrency Coordination Runtime

- Dryad/Cosmos

- Singularity (Research OS from MSR)
  - CHESS can systematically test the boot and shutdown process
Hi Tom, today one of our CCR bvts failed (and they have not failed in a long time) which means there is some very rare race in either the MultipItemReceive primitive or something more fundamental.


The test above was the one that did not terminate. This is something you can throw CHESS at to see if it catches anything.

Thanx
g
Program Analysis 2.0

1. Constraints are empowering
2. Diversification $\rightarrow$ growth
3. Scale by decomposition
4. Rebottle the genie!