SMT@Microsoft: Max-Planck-Institut Informatik, 2009

Leonardo de Moura
Microsoft Research
Verification/Analysis tools need some form of Symbolic Reasoning.
A formula \( F \) is valid

If and only if

\( \neg F \) is unsatisfiable

Theorem Prover/Satisfiability Checker

- Satisfiable Model
- Unsatisfiable Proof
A formula $F$ is valid iff $\neg F$ is unsatisfiable.
Verification/Analysis Tool: “Template”

1. Problem
2. Verification/Analysis Tool
3. Logical Formula
4. Theorem Prover/Satisfiability Checker

- Satisfiable
- Unsatisfiable (Counter-example)
unsigned GCD(x, y) {
    requires(y > 0);
    while (true) {
        unsigned m = x % y;
        if (m == 0) return y;
        x = y;
        y = m;
    }
}

We want a trace where the loop is executed twice.
We want a trace where the loop is executed twice.
unsigned GCD(x, y) {
    requires(y > 0);
    while (true) {
        unsigned m = x % y;
        if (m == 0)
            return y;
        x = y;
        y = m;
    }
}

(y_0 > 0) and
(m_0 = x_0 \% y_0) and
not (m_0 = 0) and
(x_1 = y_0) and
(y_1 = m_0) and
(m_1 = x_1 \% y_1) and
(m_1 = 0)

x_0 = 2
y_0 = 4
m_0 = 2
x_1 = 4
y_1 = 2
m_1 = 0

We want a trace where the loop is executed twice.
Type checking

Signature:

\[ \text{div} : \text{int}, \{ x : \text{int} \mid x \neq 0 \} \rightarrow \text{int} \]

Call site:

if \( a \leq 1 \) and \( a \leq b \) then

return div(a, b)

Verification condition

\( a \leq 1 \) and \( a \leq b \) implies \( b \neq 0 \)
Logic is “The Calculus of Computer Science”
Zohar Manna
High computational complexity
Logic is "The Calculus of Computer Science"

Zohar Manna

High computational complexity

(Skeptical Person)

These problems are intractable!
Yes, we cannot solve **arbitrary** problems from the "complexity ladder".

But,...
Problems in verification/analysis are big, but shallow.

Small unsatisfiable cores:

F

F'
We need provers that try to avoid the worst-case complexity.

Example of bad prover/solver:
The **truth-table** method is a decision procedure for SAT (it always take exponential time)
We need provers that try to avoid the worst-case complexity.

Example of bad prover/solver:
The **truth-table** method is a decision procedure for SAT (it always take exponential time)

**Disclaimer:**
Even $O(n^2)$ behavior “kills” a prover.
Is formula $F$ satisfiable modulo theory $T$?
Is formula $F$ satisfiable modulo theory $T$?

Arithmetic,
Bit-vectors,
Arrays,
Inductive data-types,
....
Satisfiability Modulo Theories (SMT)

Example:

1 > 2

Satisfiable if the symbols 1, 2 and > are uninterpreted.

| M | = { • }
M(1) = M(2) = •
M(<) = { (•, •) }

Unsatisfiable modulo the theory arithmetic
SMT solvers have efficient engines for reasoning modulo theory $T$!
SMT is also about Combining Different Engines
Combining Engines

SMT
- DPLL
- Simplex
- Grobner Basis
- Superposition
- Congruence Closure
- Simplification
- KB Completion
- ∀∃-elimination
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a, b, 3), c - 2)) \neq f(c - b + 1) \]
Satisfiability Modulo Theories (SMT)

\[ b + 2 = c \text{ and } f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]
$b + 2 = c$ and $f(\text{read}(\text{write}(a, b, 3), c-2) \neq f(c-b+1)$
\( b + 2 = c \) and \( f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \)
Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing
Some Applications @ Microsoft

- Spec# Programming System
- HAVOC
- Hyper-V
- Terminator T-2
- VCC
- NModel
- SpecExplorer
- SAGE
- Vigilante
- F7

Microsoft Research
Z3 is a new solver developed at Microsoft Research.
Development/Research driven by internal customers.
Free for academic research.
Interfaces:

http://research.microsoft.com/projects/z3
Test case generation
Test case generation

- Test (correctness + usability) is 95% of the deal:
  - Dev/Test is 1-1 in products.
  - Developers are responsible for unit tests.
- Tools:
  - Annotations and static analysis (SAL + ESP)
  - File Fuzzing
  - Unit test case generation
Security is critical

- Security bugs can be very expensive:
  - Cost of each MS Security Bulletin: $600k to $Millions.
  - Cost due to worms: $Billions.
  - The real victim is the customer.

- Most security exploits are initiated via files or packets.
  - Ex: Internet Explorer parses dozens of file formats.

- Security testing: **hunting for million dollar bugs**
  - Write A/V
  - Read A/V
  - Null pointer dereference
  - Division by zero
Two main techniques used by “black hats”:
- Code inspection (of binaries).
- **Black box fuzz testing.**

**Black box** fuzz testing:
- A form of black box random testing.
- Randomly *fuzz* (=modify) a well formed input.
- Grammar-based fuzzing: rules to encode how to fuzz.

**Heavily** used in security testing
- At MS: several internal tools.
- Conceptually simple yet effective in practice
Directed Automated Random Testing (DART)

- Run Test and Monitor
- Test Inputs
- Constraint System
- Known Paths
- Execution Path
- Path Condition

Seed
New input
Solve
Z3
DARTish projects at Microsoft

- **PEX**: Implements DART for .NET.
- **SAGE**: Implements DART for x86 binaries.
- **YOGI**: Implements DART to check the feasibility of program paths generated statically.
- **Vigilante**: Partially implements DART to dynamically generate worm filters.
Test input generator

- Pex starts from parameterized unit tests
- Generated tests are emitted as traditional unit tests
ArrayList: The Spec

**ArrayList.Add Method**

Adds an object to the end of the **ArrayList**.

**Namespace:** System.Collections

**Assembly:** mscorlib (in mscorlib.dll)

**Remarks**

- **ArrayList** accepts a null reference (Nothing in Visual Basic) as a valid value and allows duplicate elements.

  If **Count** already equals **Capacity**, the capacity of the **ArrayList** is increased by automatically reallocating the internal array, and the existing elements are copied to the new array before the new element is added.

  If **Count** is less than **Capacity**, this method is an O(1) operation. If the capacity needs to be increased to accommodate the new element, this method becomes an O(n) operation, where n is **Count**.
class ArrayListTest {
[PexMethod]  
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.AddItem(item);
    Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }
    ...
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
...
class ArrayList
{
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
}
class ArrayList { 
object[] items; 
int count; 

ArrayList(int capacity) { 
if (capacity < 0) throw ...; 
items = new object[capacity]; 
} 

void Add(object item) { 
if (count == items.Length) 
ResizeArray(); 
items[this.count++] = item; } 
...

class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.AddItem(item);
    Assert(list[0] == item); }
}
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length) ResizeArray();
        items[this.count++] = item;
    }
    ...

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
</tr>
</tbody>
</table>

$0 == c \rightarrow true$
ArrayList: Run 1, (0,null)

class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem(item);
        Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();

        items[this.count++] = item; }
...

Inputs          Observed Constraints
(0,null)       !(c<0) && 0==c

item == item → true

This is a tautology, i.e. a constraint that is always true, regardless of the chosen values. We can ignore such constraints.
class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.Add(item);
    Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }
    ...

<table>
<thead>
<tr>
<th>Constraints to solve</th>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
<td></td>
</tr>
<tr>
<td>![c&lt;0) &amp;&amp; 0!=c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
class ArrayListTest {
[PexMethod]
void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.AddItem(item);
    Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item; }

...
class ArrayListTest {  
[PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item);  
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)  
            ResizeArray();
        items[this.count++] = item;  
    }
    ...

Constraints to solve | Inputs | Observed Constraints
----------------------|--------|----------------------
(0, null)             | !(c<0) && 0==c  
!(c<0) && 0!=c         | (1, null) | !(c<0) && 0!=c  

0 == c → false
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
    ...
}

<table>
<thead>
<tr>
<th>Constraints to solve</th>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
<td></td>
</tr>
<tr>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
<td>(1,null)</td>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
</tr>
<tr>
<td>c&lt;0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem(item);
        Assert(list[0] == item); }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...;
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();

        items[this.count++] = item; }

    ...
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.Add(item);
        Assert(list[0] == item);
    }
}

class ArrayList {
    object[] items;
    int count;

    ArrayList(int capacity) {
        if (capacity < 0) throw ...
        items = new object[capacity];
    }

    void Add(object item) {
        if (count == items.Length)
            ResizeArray();
        items[this.count++] = item;
    }
    ...

<table>
<thead>
<tr>
<th>Constraints to solve</th>
<th>Inputs</th>
<th>Observed Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
<td>(0, null)</td>
<td>!(c&lt;0) &amp;&amp; 0==c</td>
</tr>
<tr>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
<td>(1, null)</td>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
</tr>
<tr>
<td>c&lt;0</td>
<td>(-1, null)</td>
<td>c&lt;0</td>
</tr>
</tbody>
</table>

\[ c < 0 \Rightarrow true \]
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
    var list = new ArrayList(c);
    list.Add(item);
    Assert(list[0] == item); }
}

class ArrayList {
  object[] items;
  int count;

  ArrayList(int capacity) {
    if (capacity < 0) throw ...;
    items = new object[capacity];
  }

  void Add(object item) {
    if (count == items.Length)
      ResizeArray();
    items[this.count++] = item; }

  // ...
How to test this code?
(Real code from .NET base class libraries.)

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

    _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("RES\$RF\$FORMAT", "ResourceReader .ctor(Stream). UnmanagedMemoryStream: "+(_ums!=null));
    ReadResources();
}
```
White box testing in practice

```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 (magicNum != ResourceManager.MagicNumber)
            throw new ArgumentException(ResourceManager.GetResourceString("Resources_StreamNotValid"));

        // Assuming this is ResourceManager header v1 or greater, hopefully
        // after the version number there is a number of bytes to skip
        // to bypass the rest of the ResMgr header.
        int resMgrHeaderVersion = _store.ReadInt32();
        if (resMgrHeaderVersion > 1) {
            int numBytesToSkip = _store.ReadInt32();
            BCLDebug.Assert(numBytesToSkip >= 0, "numBytesToSkip in ResMgr header should be positive!");
            _store.BaseStream.Seek(numBytesToSkip, SeekOrigin.Current);
        }
        else {
            BCLDebug.Log("RESMGRFILEFORMAT", "ReadResources: Parsing ResMgr header v1.");
            SkipInt32();  // We don't care about numBytesToSkip.
        // Read in type name for a suitable ResourceReader
        // New ResourceWriter & InternalDeserializer use different Strin
```
White box testing in practice

```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");
                int
                return mStream.InternalReadInt32();
            }
        } else
        { FillBuffer(4);
        }
    }
// Read in type name for a suitable ResourceReader
// Note: ResourceReader & ResourceParser use different classes
```

Pex — Test Input Generation

```csharp
public class ResourceReaderTests {
    [PexTest]
    public unsafe void ParameterizedTest(byte[] a) {
        PexAssume.IsNotNull(a);
        fixed (byte* p = a)
        using (stream = new UnmanagedMemoryStream(p, a.Length))
            var reader = new ResourceReader(stream);
            readEntries(reader);
    }
}
```

Test input, generated by Pex:

```csharp
byte[] a = new byte[14];
a[0] = 206;
a[1] = 202;
a[2] = 239;
a[3] = 190;
a[7] = 64;
a[11] = 120;
ParameterizedTest(a);
```
Test Input Generation by Dynamic Symbolic Execution

- **Constraint System**
- **Test Inputs**
- **Execution Path**
- **Known Paths**

Result: small test suite, high code coverage
Finds only real bugs, No false warnings
Rich Combination

- Linear arithmetic
- Bitvector
- Arrays
- Free Functions

Models

Model used as test inputs

∀-Quantifier

Used to model custom theories (e.g., .NET type system)

API

Huge number of small problems. Textual interface is too inefficient.
Undecidable (in general)
Undecidable (in general)

Solution:
- Return “Candidate” Model
- Check if trace is valid by executing it
Rich Combination
Linear arithmetic
Bitvector
Arrays
Free Functions

∀-Quantifier
Used to model custom theories (e.g., .NET type system)

**Undecidable** (in general)

Refined solution:
Support for decidable fragments.
Apply DART to large applications (not units).
Start with well-formed input (not random).
Combine with generational search (not DFS).
  - Negate 1-by-1 each constraint in a path constraint.
  - Generate many children for each parent run.
Apply DART to large applications (not units).

Start with well-formed input (not random).

Combine with generational search (not DFS).
  - Negate 1-by-1 each constraint in a path constraint.
  - Generate many children for each parent run.
Starting with 100 zero bytes ...

SAGE generates a crashing test for Media1 parser

Generation 0 – seed file

00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ........
Starting with 100 zero bytes ...

SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00
```

Generation 10 – CRASH
SAGE is very effective at finding bugs.
Works on large applications.
Fully automated
Easy to deploy (x86 analysis – any language)
Used in various groups inside Microsoft
Powered by Z3.
Formulas are usually big conjunctions.

SAGE uses only the bitvector and array theories.

Pre-processing step has a huge performance impact.
- Eliminate variables.
- Simplify formulas.

Early unsat detection.
Static Driver Verifier

SLAM
Z3 is part of SDV 2.0 (Windows 7)

It is used for:

- Predicate abstraction (c2bp)
- Counter-example refinement (newton)
Overview

- http://research.microsoft.com/slam/
- **SLAM/SDV** is a software model checker.
- Application domain: *device drivers*.
- Architecture:
  - **c2bp** C program → boolean program (*predicate abstraction*).
  - **bebop** Model checker for boolean programs.
  - **newton** Model refinement (check for path feasibility)
- SMT solvers are used to perform predicate abstraction and to check path feasibility.
- c2bp makes several calls to the SMT solver. The formulas are relatively small.
Example

Do this code obey the looking rule?

do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
}
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
Example

Model checking Boolean program

```c
do {
    KeAcquireSpinLock();

    if(*){
        KeReleaseSpinLock();
    }

    } while (*);

KeReleaseSpinLock();
```
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    nPacketsOld = nPackets;
    b = true;
    if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
Example

Model Checking
Refined Program

\( b: (nPacketsOld == nPackets) \)

do {
    KeAcquireSpinLock();

    b = true;

    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while (!b);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();
    
    b = true;
    
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
    } while (!b);
    
    KeReleaseSpinLock();

Example

Model Checking
Refined Program

\[ b: (n\text{PacketsOld} == n\text{Packets}) \]

```c
do {
    KeAcquireSpinLock();
    b = true;
    if(*){
        KeReleaseSpinLock();
        b = b ? false : *;
    }
} while (!b);
KeReleaseSpinLock();
```
Automatic discovery of invariants
- driven by property and a finite set of (false) execution paths
- predicates are not invariants, but observations
- abstraction + model checking computes inductive invariants (Boolean combinations of observations)

A hybrid dynamic/static analysis
- newton executes path through C code symbolically
- c2bp+bebop explore all paths through abstraction

A new form of program slicing
- program code and data not relevant to property are dropped
- non-determinism allows slices to have more behaviors
Predicate Abstraction: c2bp

- **Given** a C program $P$ and $F = \{p_1, \ldots, p_n\}$.
- **Produce** a Boolean program $B(P, F)$
  - Same control flow structure as $P$.
  - Boolean variables $\{b_1, \ldots, b_n\}$ to match $\{p_1, \ldots, p_n\}$.
  - Properties true in $B(P, F)$ are true in $P$.
- Each $p_i$ is a pure Boolean expression.
- Each $p_i$ represents set of states for which $p_i$ is true.
- Performs modular abstraction.
Abstracting Expressions via $F$

- $\text{Implies}_F (e)$
  - Best Boolean function over $F$ that implies $e$.

- $\text{ImpliedBy}_F (e)$
  - Best Boolean function over $F$ that is implied by $e$.
  - $\text{ImpliedBy}_F (e) = \text{not} \text{Implies}_F (\text{not} e)$
$\text{Implies}_F(e)$ and $\text{ImpliedBy}_F(e)$
Computing $\text{Implies}_F(e)$

- minterm $m = l_1$ and ... and $l_n$, where $l_i = p_i$, or $l_i = \text{not } p_i$.
- $\text{Implies}_F(e)$: disjunction of all minterms that imply $e$.

Naive approach
- Generate all $2^n$ possible minterms.
- For each minterm $m$, use SMT solver to check validity of $m$ implies $e$.

Many possible optimizations
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$
- $\neg x < y, \neg x = 2$ implies $y > 1$
- $x < y, \neg x = 2$ implies $y > 1$
- $\neg x < y, x = 2$ implies $y > 1$
- $x < y, x = 2$ implies $y > 1$
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$
- $\neg x<y, \neg x=2 \text{ implies } y>1$
- $x<y, \neg x=2 \text{ implies } y>1$
- $\neg x<y, x=2 \text{ implies } y>1$
- $x<y, x=2 \text{ implies } y>1$

- $\neg x<y, \neg x=2 \text{ implies } y>1$
- $x<y, \neg x=2 \text{ implies } y>1$
- $\neg x<y, x=2 \text{ implies } y>1$
- $x<y, x=2 \text{ implies } y>1$
Computing $\text{Implies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$
- $\neg x < y, \neg x=2$ implies $y>1$
- $x < y, \neg x=2$ implies $y>1$
- $\neg x < y, x=2$ implies $y>1$
- $x < y, x=2$ implies $y>1$

$\text{Implies}_F(y>1) = x < y \land x=2$
Computing $\text{Imp}l\text{ies}_F(e)$

- $F = \{ x < y, x = 2 \}$
- $e : y > 1$

Minterms over $F$

- $\neg x < y, \neg x = 2 \implies y > 1$
- $x < y, \neg x = 2 \implies y > 1$
- $\neg x < y, x = 2 \implies y > 1$
- $x < y, x = 2 \implies y > 1$

$\text{Imp}l\text{ies}_F(y > 1) = b_1 \land b_2$
Given an error path $p$ in the Boolean program $B$.
Is $p$ a feasible path of the corresponding C program?
  Yes: found a bug.
  No: find predicates that explain the infeasibility.
Execute path symbolically.
Check conditions for inconsistency using SMT solver.
Z3 & Static Driver Verifier

- All-SAT
  - Better (more precise) Predicate Abstraction
- Unsatisfiable cores
  - Why the abstract path is not feasible?
- Fast Predicate Abstraction
Bit-precise Scalable Static Analysis

PREfix [Moy, Bjorner, Sielaff 2009]
int binary_search(int[] arr, int low, int high, int key)
while (low <= high)
{
    // Find middle value
    int mid = (low + high) / 2;
    int val = arr[mid];
    if (val == key) return mid;
    if (val < key) low = mid + 1;
    else high = mid - 1;
}
return -1;

void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
int binary_search(int[] arr, int low, int high, int key) {
    while (low <= high) {
        int mid = (low + high) / 2;
        int val = arr[mid];
        if (val == key) return mid;
        if (val < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
}
int binary_search(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = (low + high) / 2;
        int val = arr[mid];
        if (val == key) return mid;
        if (val < key) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}

void itoa(int n, char* s) {
    if (n < 0) {
        *s++ = '-';
        n = -n;
    }
    // Add digits to s
    ....
```c
int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if (((*outname = malloc(n))) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size)
{
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
error:
    return status;
}
```
int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if ((*outname = malloc(n)) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size)
{
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
error:
    return status;
}
```c
int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if (((*outname = malloc(n)) == NULL) {
        return 0xC0000095; // NT_STATUS_NO_MEM;
    }
    return 0;
}

int get_name(char* dst, uint size)
{
    char* name;
    int status = 0;
    status = init_name(&name, size);
    if (status != 0) {
        goto error;
    }
    strcpy(dst, name);
error:
    return status;
}
```

**models**

- **outcome init_name_0**: guards: $n == 0$
  results: result == 0
- **outcome init_name_1**: guards: $n > 0$, $n <= 65535$
  results: result == 0xC0000095
- **outcome init_name_2**: guards: $n > 0$, $n <= 65535$
  constraints: valid(outname)
  results: result == 0; init(*outname)

**paths**

- **path for function get_name**: guards: $size == 0$
  constraints:
  facts: init(dst); init(size); status == 0
- **pre-condition for function strcpy**: init(dst) and valid(name)

**warnings**

- Pre-condition for function strcpy
Overflow on unsigned addition

```
iElement = m_nSize;
if( iElement >= m_nMaxSize )
{
    bool bSuccess = GrowBuffer( iElement+1 );
    ...
}
::new( m_pData+iElement ) E( element );
m_nSize++;
```

- `m_nSize == m_nMaxSize == UINT_MAX`
- `iElement + 1 == 0`
- Write in unallocated memory
- Code was written for address space < 4GB
ULONG AllocationSize;
while (CurrentBuffer != NULL) {
    if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
        return NULL;
    }
    NumberOfBuffers++;
    CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER) * NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);
Verifying Compilers

Annotated Program $\rightarrow$ Verification Condition $F$

pre/post conditions
invariants
and other annotations
class C {
    private int a, z;
    invariant z > 0

    public void M()
        requires a != 0
    {
        z = 100/a;
    }
}
**Spec# Approach for a Verifying Compiler**

- **Source Language**
  - C# + goodies = Spec#

- **Specifications**
  - method contracts,
  - invariants,
  - field and type annotations.

- **Program Logic:**
  - *Dijkstra’s weakest preconditions.*

- **Automatic Verification**
  - type checking,
  - verification condition generation (VCG),
  - SMT
Command language

- $x := E$
  - $x := x + 1$
  - $x := 10$
- havoc $x$
- $S ; T$
- assert $P$
- assume $P$
- $S \square T$
Hoare triple \( \{ P \} \ S \ \{ Q \} \) says that every terminating execution trace of S that starts in a state satisfying P does not go wrong, and terminates in a state satisfying Q.
Hoare triple \( \{ P \} \ S \ \{ Q \} \) says that every terminating execution trace of \( S \) that starts in a state satisfying \( P \)

- does not go wrong, and
- terminates in a state satisfying \( Q \)

Given \( S \) and \( Q \), what is the weakest \( P' \) satisfying \( \{ P' \} \ S \ \{ Q \} \) ?

\( P' \) is called the \textit{weakest precondition} of \( S \) with respect to \( Q \), written \( \text{wp}(S, Q) \)

to check \( \{ P \} \ S \ \{ Q \} \), check \( P \Rightarrow P' \)
Weakest preconditions

\[
\begin{align*}
\text{wp}( x := E, \ Q ) &= \ Q[ E / x ] \\
\text{wp}( \text{havoc} \ x, \ Q ) &= \ (\forall x \cdot Q) \\
\text{wp}( \text{assert} \ P, \ Q ) &= \ P \land Q \\
\text{wp}( \text{assume} \ P, \ Q ) &= \ P \Rightarrow Q \\
\text{wp}( S ; T, \ Q ) &= \ \text{wp}( S, \ \text{wp}( T, Q ) ) \\
\text{wp}( S \boxplus T, \ Q ) &= \ \text{wp}( S, Q ) \land \text{wp}( T, Q )
\end{align*}
\]
Structured if statement

if \( E \) then \( S \) else \( T \) end =

\[
\text{assume } E; \ S
\]

\[
\square
\]

\[
\text{assume } \neg E; \ T
\]
While loop with loop invariant

while E
  invariant J
do
  S
end

where \( x \) denotes the assignment targets of \( S \)

- check that the loop invariant holds initially
- “fast forward” to an arbitrary iteration of the loop
- check that the loop invariant is maintained by the loop body
procedure Chunker.NextChunk(this: ref where $IsNotNull(null, This, Chunker)) returns ($result: ref where $IsNotNull($result, System.String));

// in-parameter: target object
free requires $Heap[this, $allocated];
requires ($Heap[this, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[this, $ownerRef], $inv] <: $Heap[this, $ownerFrame]) ||

// out-parameter: return value
free ensures $Heap[$result, $allocated];
ensures ($Heap[$result, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[$result, $ownerRef], $inv] <: $Heap[$result, $ownerFrame]) ||

// user-declared postconditions
ensures $StringLength($result) <= $Heap[this, Chunker.ChunkSize];

// frame condition
modifies $Heap;
free ensures (forall $o: ref, $f: name :: { $Heap[$o, $f] } $f != $inv & $f != $localinv & $f != $FirstConsistentOwner & (!$StaticField($f)) ||
!$IsDirectlyModifiableField($f)) && $o != null && old($Heap[$o, $allocated]) && (old($Heap[$o, $ownerFrame] == $PeerGroupPlaceholder ||
(old($Heap)[old($Heap][$o, $ownerRef], $inv] <: old($Heap[$o, $ownerFrame]) || old($Heap)[old($Heap][$o, $ownerRef], $localinv] ==
$BaseClass(old($Heap][$o, $ownerFrame])) && old($o != this || (!Chunker <= DeclType($f)) || $IncludedInModifiesStar($f) && old($o != this || $f != $exposeVersion) ==> old($Heap)[old($o, $f] == $Heap[$o, $f]);

// boilerplate
free requires $BeingConstructed == null;
free ensures (forall $o: ref :: { $Heap[$o, $localinv] } { $Heap[$o, $inv] } $o != null && old($Heap[$o, $allocated]) && $Heap[$o, $allocated] ==> $Heap[$o, $inv] == $typeof($o) && $Heap[$o, $localinv] == $typeof($o));
free ensures (forall $o: ref :: { $Heap[$o, $FirstConsistentOwner] } old($Heap)[old($Heap][$o, $FirstConsistentOwner], $exposeVersion] ==
$Heap[old($Heap][$o, $FirstConsistentOwner], $exposeVersion] ==> old($Heap)[old($Heap][$o, $FirstConsistentOwner] == $Heap[$o, $FirstConsistentOwner]);
free ensures (forall $o: ref :: { $Heap[$o, $localinv] } { $Heap[$o, $inv] } old($Heap)[old($o, $allocated]) == old($Heap)[old($o, $inv]) == $Heap[$o, $inv] &&
old($Heap)[old($o, $localinv]) == $Heap[$o, $localinv]);
free ensures (forall $o: ref :: { $Heap[$o, $allocated] } old($Heap)[old($o, $allocated]) == old($Heap)[old($o, $allocated)] && (forall $ot: ref :: { $Heap[$ot, $ownerFrame] } { $Heap[$ot, $ownerRef] } old($Heap)[old($ot, $allocated) && old($Heap)[old($ot, $ownerFrame] == $PeerGroupPlaceholder ==> old($Heap)[old($ot, $ownerRef] == $Heap[$ot, $ownerRef] && old($Heap)[old($ot, $ownerFrame] == $Heap[$ot, $ownerFrame]) &&
old($Heap)[old($BeingConstructed, $NonNullFieldsAreInitialized] == $Heap[$BeingConstructed, $NonNullFieldsAreInitialized];
∀ Axioms (non-ground) + BIG and-or tree (ground)

Control & Data Flow
Meta OS: small layer of software between hardware and OS

Mini: 100K lines of non-trivial concurrent systems C code

Critical: must provide functional resource abstraction

Trusted: a verification grand challenge
A partition cannot distinguish (with some exceptions) whether a machine instruction is executed

a) through the HV  OR  b) directly on a processor
Hypervisor Implementation

- real code, as shipped with Windows Server 2008
- ca. 100,000 lines of C, 5,000 lines of x64 assembly
- concurrency
  - spin locks, r/w locks, rundown, turnstiles
  - lock-free accesses to volatile data and hardware covered by implicit protocols
- scheduler, memory allocator, etc.
- access to hardware registers (memory management, virtualization support)
Partners:

- European Microsoft Innovation Center
- Microsoft Research
- Microsoft’s Windows Div.
- Universität des Saarlandes

co-funded by the German Ministry of Education and Research

http://www.verisoftxt.de
1. Memory model that is adequate and efficient to reason about
2. Modular reasoning about concurrent code
3. Invariants for (large and complex) C data structures
4. Huge verification conditions to be proven automatically
5. “Live” specifications that evolve with the code
The Microsoft Verifying C Compiler (VCC)

- **Source Language**
  - ANSI C +
  - Design-by-Contract Annotations +
  - Ghost state +
  - Theories +
  - Metadata Annotations

- **Program Logic**
  - Dijkstra’s weakest preconditions

- **Automatic Verification**
  - verification condition generation (VCG)
  - automatic theorem proving (SMT)
int foo(int x)
    requires(x > 5) // precond
    ensures(result > x) // postcond
{
    ...
}

void bar(int y; int *z)
    writes(z) // framing
    requires(y > 7)
    maintains(*z > 7) // invariant
{
    *z = foo(y);
    assert(*z > 7);
}

• function contracts: pre-/postconditions, framing
• modularity: bar only knows contract (but not code) of foo

advantages:
• modular verification: one function at a time
• no unfolding of code: scales to large applications
VCs have several Mb
Thousands of non ground clauses
Developers are willing to wait at most 5 min per VC
VCs have several Mb
Thousands of non ground clauses
Developers are willing to wait at most 5 min per VC

Are you willing to wait more than 5 min for your compiler?
Verification Attempt Time vs. Satisfaction and Productivity

By Michal Moskal (VCC Designer and Software Verification Expert)
1. My annotations are not strong enough!
weak loop invariants and/or contracts

2. My theorem prover is not strong (or fast) enough.
Send “angry” email to Nikolaj and Leo.
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime

\( \forall h,o,f:\)

\[ \text{IsHeap}(h) \land o \neq \text{null} \land \text{read}(h, o, \text{alloc}) = t \]

\[ \Rightarrow \]

\[ \text{read}(h,o,f) = \text{null} \lor \text{read}(h, \text{read}(h,o,f), \text{alloc}) = t \]
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms

∀ o, f:

\[ o \neq \text{null} \land \text{read}(h_0, o, \text{alloc}) = t \Rightarrow \]
\[ \text{read}(h_1, o, f) = \text{read}(h_0, o, f) \lor (o, f) \in M \]
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions

\( \forall i,j: i \leq j \Rightarrow \text{read}(a,i) \leq \text{read}(b,j) \)
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
  - $\forall x: p(x, x)$
  - $\forall x, y, z: p(x, y), p(y, z) \Rightarrow p(x, z)$
  - $\forall x, y: p(x, y), p(y, x) \Rightarrow x = y$
Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
- Solver must be fast in satisfiable instances.

We want to find bugs!
Bad news

There is no sound and refutationally complete procedure for
linear integer arithmetic + free function symbols
Many Approaches

- Heuristic quantifier instantiation
- Combining SMT with Saturation provers
- Complete quantifier instantiation
- Decidable fragments
- Model based quantifier instantiation
Is the axiomatization of the runtime consistent?
False implies everything
Partial solution: SMT + Saturation Provers
Found many bugs using this approach
Standard complain

“I made a small modification in my Spec, and Z3 is timing out”

This also happens with SAT solvers (NP-complete)

In our case, the problems are undecidable

Partial solution: parallelization
Joint work with Y. Hamadi (MSRC) and C. Wintersteiger
Multi-core & Multi-node (HPC)
Different strategies in parallel
Collaborate exchanging lemmas
Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.
Hey, I don’t trust these proofs

Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.

Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)
Hey, I don’t trust these proofs

Z3 may be buggy.
Solution: proof/certificate generation.
Engineering problem: these certificates are too big.
The Axiomatization of the runtime may be buggy or inconsistent.
Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)
The VCG generator is buggy (i.e., it makes the wrong assumptions)
Do you trust your compiler?
These are bug-finding tools!

When they return “Proved”, it just means they cannot find more bugs.

I add Loop invariants to speedup the process.

I don’t want to waste time analyzing paths with 1,2,...,k,... iterations.

They are successful if they expose bugs not exposed by regular testing.
Logic as a platform

Most verification/analysis tools need symbolic reasoning

SMT is a hot area

Many applications & challenges

http://research.microsoft.com/projects/z3
Logic as a platform
Most verification/analysis tools need symbolic reasoning
SMT is a hot area
Many applications & challenges
http://research.microsoft.com/projects/z3

Thank You!