Applications of SMT

Lecture 3 2012

Nikolaj Bjørner
Microsoft Research
DTU Winter course January 4th 2012
Plan

- Selected SMT applications
  - Smart white-box fuzzing with Pex and SAGE
  - Software model checking with SLAM/SDV
  - Bit-precise static analysis with PREfix
  - Program verification with Boogie
Program analysis, verification and test tools solve problems that can be reduced to logical formulas and transformations between logical formulas at their core.

Modern SMT solvers are a often good fit as they:
- Are better tuned than ad-hoc solvers
- Often handle domains found in programs directly.

The selected examples are intended to show instances where sub-tasks are reduced to SMT/Z3.
Tools using Z3

Property Driven
- Vcc
- F7
- HAVOC
- SLAyer
- BOOGIE

Execution Guided
- SLAM
- Yogi

Model Based
- Spec#
- M3
- BEK
- FORMULA
- SAGE

Over-Approximation
Under-Approximation

Type Safety
Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing
A formula $F$ is valid if and only if

$\neg F$ is unsatisfiable
A formula $F$ is valid
iff
$\neg F$ is unsatisfiable
Verification/Analysis Tool: “Template”

- Problem
- Verification/Analysis Tool
- Logical Formula
- Theorem Prover/Satisfiability Checker
- Satisfiable
- Unsatisfiable
  - (Counter-example)
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
SAGE by numbers

100+ CPU-years - largest dedicated fuzz lab in the world

100s apps - fuzzed using SAGE

100s previously unknown bugs found

1,000,000,000+ computers updated with bug fixes

Millions of $ saved for Users and Microsoft

10s of related tools (incl. Pex), 100s DART citations

100,000,000+ constraints - largest usage for any SMT solver

adapted from [Patrice Godefroid, ISSTA 2010]
Directed Automated Random Testing (DART)

- Run Test and Monitor
- Execution Path
- Path Condition
- Test Inputs
- Known Paths
- Z3
  - Solve
  - New input
  - seed
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.
What is Pex?

- 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.
```csharp
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 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 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)</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.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>

item == item \(\rightarrow\) 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; 
    }

    ...
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 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.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>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;
    }
...

<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>(-1,null)</td>
<td></td>
</tr>
</tbody>
</table>

ArrayList: Run 3, (-1, null)
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.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>

Once again, Pex blows my mind. It's utterly amazing the bugs that it can find :).
White box testing in practice

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

```csharp
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("RESMGRFILEFORMAT", "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(Environment.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
        // Note ResourceWriter & InternalResCon use different strings
    }
    catch (Exception e)
    {
        // Handle exceptions
    }
    finally
    {
        // Clean up
    }
}
```
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");
                return mStream.InternalReadInt32();
            } else {
                FillBuffer(4);
            }
        }
```
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;
ParameterizedTest(a);
```
Test Input Generation by Dynamic Symbolic Execution

Initially, choose Address

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.
Rich Combination

Linear arithmetic

Bitvector

Arrays

Free Functions

∀-Quantifier

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

Undecidable (in general)
Undecidable (in general)

Solution:

Return “Candidate” Model
Check if trace is valid by executing it
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

```
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 ; ....
```

Generation 0 – seed file
Starting with 100 zero bytes ...

SAGE generates a crashing test for Media1 parser

**Screenshot:**
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 ; ....str²uv:(...  
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 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
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.
do {
    KeAcquireSpinLock();

    nPacketsOld = nPackets;

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

KeReleaseSpinLock();
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++;
    } else {
        b = b ? false : *
    }
} while (nPackets != nPacketsOld);

KeReleaseSpinLock();
do {
    KeAcquireSpinLock();

    b = true;

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

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

Model Checking
Refined Program

b: (nPacketsOld == nPackets)
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
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 $e$.
  - $\text{ImpliedBy}_F(e) = \text{not } \text{Implies}_F(\text{not } e)$
Implies$_F(e)$ and ImpliedBy$_F(e)$
Computing $\text{Implies}_F(e)$

- minterm $m = l_1 \text{ and } ... \text{ and } l_n$, 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 $m \text{ implies } e$.
- 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 $\text{Imp}_F(e)$

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

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 $F$
- $\neg x < y, \neg x=2$ 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 $\text{Imp}_F(e)$

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

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

$\text{Imp}_F(y>1) = b_1 \land b_2$
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
    ....

Package: java.util.Arrays
Function: binary_search

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)
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
    ....

Package: java.util.Arrays
Function: binary_search

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)
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
    ....
}

Package: java.util.Arrays
Function: binary_search

-INT_MIN= INT_MIN

3(INT_MAX+1)/4 + (INT_MAX+1)/4 = INT_MIN

Book: Kernighan and Ritchie
Function: itoa (integer to ascii)
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**

**model for function init_name**
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)
C/C++ functions

int init_name(char **outname, uint n)
{
    if (n == 0) return 0;
    else if (n > UINT16_MAX) exit(1);
    else if (*((char **)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;
}

model for function init_name
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)

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)
Overflow on unsigned addition

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

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

- FORMULA
FORMULA: Design Space Exploration

Use Design Space Exploration to identify valid candidate architectures
FORMULA: Diversified Search

- Subtract all isomorphic solutions
- Diversify and Constrain Search Space
- Remember this model

SMT Formula

Z3 Solver

\[ \text{SMT Formula} \]

\[ \text{Z3 Solver} \]

1 11 48 11163

\[ \emptyset \{f_0\} \{f_1\} \{f_0, f_1\} \]
Verifying Compilers

Annotated Program

Verification Condition $F$

pre/post conditions
invariants
and other annotations
Building Verve

- Source file
- Verification tool
- Compilation tool

Kernel.cs

C# compiler

Nucleus.bpl (x86)

Boogie/Z3

Translator/Assembler

Kernel.obj (x86)

TAL checker

Linker/ISO generator

Verve.iso

9 person-months

Safe to the Last Instruction / Jean Yang & Chris Hawbliztl PLDI 2010
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

**Diagram:**
- Spec# Compiler
- Spec# (annotated C#)
- Boogie PL
- VC Generator
- Formulas
- SMT Solver
Command language

- $x := E$
  - $x := x + 1$
  - $x := 10$

- havoc $x$

- assert $P$
  - assume $P$
  - $S ; T$
Reasoning about execution traces

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 *weakest precondition* of \( S \) with respect to \( Q \), written \( \text{wp}(S, Q) \).

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

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

if E then S else T end =

assume E; S

¬E; T
While loop with loop invariant

while E
  invariant J
do
  S
end

=  assert J;
  havoc x; assume J;
  ( assume E; S; assert J; assume false
    □ assume ¬E
  )

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 $IsNotNull(this, Chunker)) returns ($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]) || $Heap[$Heap[this, $ownerRef], $localinv] == $BaseClass($Heap[this, $ownerFrame]) && (forall $pc: ref :: $pc != null && $Heap[$pc, $allocated] && $Heap[$pc, $ownerRef] == $Heap[this, $ownerRef] && $Heap[$pc, $ownerFrame] == $Heap[this, $ownerFrame]) ==> $Heap[$pc, $inv] == $typeof($pc) && $Heap[$pc, $localinv] == $typeof($pc));

    // out-parameter: return value
    free ensures $Heap[$result, $allocated];

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

    // frame condition
    modifies $Heap;
    free ensures (forall $o: ref, $f: name :: ($Heap[$o, $f], $inv) ) $f != $inv & $f != $localinv & $f != $FirstConsistentOwner & (!IsStaticField($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 || !$Heap[$Heap[$o, $ownerRef], $inv] < $Heap[$o, $allocated]) && $Heap[$o, $allocated] == $Heap[$o, $allocated] & $Heap[$o, $allocated] == $PeerGroupPlaceholder) || !($Heap)[$Heap[$o, $inv] == $Heap[$o, $inv] && $Heap[$o, $allocated] && $Heap[$o, $allocated] == $PeerGroupPlaceholder && $Heap[$o, $allocated] == $PeerGroupPlaceholder) && old($o != this || !$IsStaticField($f)) && old($o != this || $f != $exposeVersion) ==> old($Heap[$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], $exposeVersion] && old($o != this || $Heap[$o, $FirstConsistentOwner]) == $Heap[$o, $FirstConsistentOwner] && old($o != this || $Heap[$o, $FirstConsistentOwner]) == $Heap[$o, $FirstConsistentOwner]);

    free ensures (forall $o: ref :: ( $Heap[$o, $localinv] ) $Heap[$o, $inv] ) old($Heap[$o, $allocated]) == old($Heap[$o, $inv])$Heap[$o, $allocated] && old($o != this || $Heap[$o, $localinv] == $Heap[$o, $localinv]);

    free ensures (forall $o: ref :: ( $Heap[$o, $allocated] ) old($Heap[$o, $allocated]) && ($forall $o: ref :: ( $Heap[$o, $ownerFrame] ) $Heap[$o, $allocated] && old($o != this || $PeerGroupPlaceholder) && old($Heap[$o, $ownerRef] == $Heap[$o, $ownerRef] && old($Heap[$o, $ownerFrame] == $Heap[$o, $ownerFrame]) && old($o != this || $Heap[$o, $allocated] && $NonNullFieldsAreInitialized) == $Heap[$BeingConstructed, $NonNullFieldsAreInitialized]);
Verification conditions: Structure

∀ Axioms (non-ground)

BIG and-or tree (ground)

Control & Data Flow
**Hypervisor: A Manhattan Project**

- **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
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, rundowns, 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
Challenges for Verification of Concurrent C

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)
VCC Architecture

```c
#include <vcc2.h>

typedef struct _BITMAP {
    UINT32 Size; // Number of bits
    PUINT32 Buffer; // Memory to store
    ...
    // private invariants
    invariant(Size > 0 && Size % 32 == 0)
    ...
}

:assumption
(forall (?x Int) (?y Int)
    (iff
        (= (IntEqual ?x ?y) boolTrue
        (= ?x ?y)))
    :formula
    (flatten ...

:cc
$ref_cnt(olds, #p) == $ref_cnt(olds, #p)
& $ite.bool($set_in(#p, #owns(olds, owner)),
    $ite.bool($set_in(#p, owns),
    $st_eq(olds, $s, #p),
    $wrapped($s, #p, $typ(#p)) &&
    $timestamp_is_now($s, #p)),
    $ite.bool($set_in(#p, owns),
    $owner($s, #p) == owner && $closed($s);

:boogie
owner),
    $ite.bool($set_in(#p, owns),
    $st_eq(olds, $s, #p),
    $wrapped($s, #p, $typ(#p)) &&
    $timestamp_is_now($s, #p)),
    $ite.bool($set_in(#p, owns),
    $owner($s, #p) == owner &&
    $closed($s);

:smt
$ref_cnt(olds, #p) == $ref_cnt(olds, #p)
& $ite.bool($set_in(#p, #owns(olds, owner)),
    $ite.bool($set_in(#p, owns),
    $st_eq(olds, $s, #p),
    $wrapped($s, #p, $typ(#p)) &&
    $timestamp_is_now($s, #p)),
    $ite.bool($set_in(#p, owns),
    $owner($s, #p) == owner &&
    $closed($s);

:z3

Available at http://vcc.codeplex.com/
• 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
Users 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)
Why did my proof attempt fail?

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.
Modification in invariant checking

Switch to Z3 v2

Z3 v2 update

Switch to Boogie2

Attempt to improve Boogie/Z3 interaction
The Importance of Speed

Subject: FW: Der neue Z3 ist höllisch schnell (und ich meine kein Auto)

Fyi.

I have released the new VCC once on my example has produced any result otherwise after 50000 seconds. Now, I receive the first error already after 200-300 seconds. That is why I am very happy and satisfied! This is huge progress.


Viel Spaß und liebe Grüße an Lieven,
Markus
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 ≠ null ∧ read(h₀, o, alloc) = t ⇒
  read(h₁, o, f) = read(h₀, o, f) ∨ (o, f) ∈ 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
  \begin{align*}
  \forall x : p(x,x) \\
  \forall x,y,z: p(x,y), p(y,z) & \implies p(x,z) \\
  \forall x,y: p(x,y), p(y,x) & \implies x = y
  \end{align*}
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!
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
Challenge: Modeling Runtime

- Is the axiomatization of the runtime consistent?
- False implies everything
- Partial solution: SMT + Saturation Provers
- Found many bugs using this approach
**Challenge: Robustness**

- Standard complain
  
  "I made a small modification in my Spec, and Z3 is timingout"

- 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.
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 speed up the process.

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

They are successful if they expose bugs not exposed by regular testing.
Powerful, mature, and versatile tools like SMT solvers can now be exploited in very useful ways.

The construction and application of satisfiability procedures is an active research area with exciting challenges.

SMT is hot at Microsoft.

Z3 is a new SMT solver.

Main applications:

- Test-case generation.
- Verifying compiler.
- Model Checking & Predicate Abstraction.