Bugs, Moles and Skeletons: Symbolic Reasoning for Software Development

UCAR 2010, Edinburgh

Leonardo de Moura
Microsoft Research
Symbolic Reasoning

Software analysis tools need some form of symbolic reasoning
Z3: Our Symbolic Reasoning Engine

Joint work with Nikolaj Bjorner.

Z3 is a Satisfiability Modulo Theories (SMT) Solver.

http://research.microsoft.com/projects/z3
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(read(write(a, b, 3), c-2)) ≠ f(c-b+1)
b + 2 = c and \( f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \)
\[ b + 2 = c \text{ and } f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]
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
SDV: The Static Driver Verifier
PREfix: The Static Analysis Engine for C/C++. 
PEX: Program Exploration for .NET
SAGE: Scalable Automated Guided Execution
Yogi: Dynamic Symbolic Execution + Abstraction
Symbolic Reasoning @ Microsoft

SPEC#: C# + Contracts
VCC: Verifying C Compiler
Many others:

HAVOC: Heap-Aware Verification of C-code.
SpecExplorer: Model-based testing of protocol specs.
FORMULA: Model-based Design
F7: Refinement types for security protocols
M3: Model Program Modeling
VS3: Abstract interpretation and Synthesis

...
Many others:
HAVOC: Heap-Aware Verification of C-code.
SpecExplorer: Model-based testing of protocol specs.
FORMULA: Model-based Design
F7: Refinement types for security protocols
M3: Model Program Modeling
VS3: Abstract interpretation and Synthesis

... They all use Z3
Software Analysis Tool: “Template”

Problem

Verification/Analysis Tool

Logical Formula

Satisfiability Checker

Satisfiable (Model)

Unsatisfiable (Proof)
Directed Automated Random Testing (DART)

Run Test and Monitor → Execution Path → Path Condition

seed → Test Inputs

New input → Logical Formula → Solve

Known Paths
<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEX</td>
<td>Implements DART for .NET.</td>
</tr>
<tr>
<td>SAGE</td>
<td>Implements DART for x86 binaries.</td>
</tr>
<tr>
<td>YOGI</td>
<td>Implements DART to check the feasibility of program paths generated statically.</td>
</tr>
<tr>
<td>Vigilante</td>
<td>Partially implements DART to dynamically generate worm filters.</td>
</tr>
</tbody>
</table>
What is Pex?

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

.NET Framework Class Library
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.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 ArrayListTest 

Inputs
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>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>
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>(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.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

<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>
<tr>
<td>!(c&lt;0) &amp;&amp; 0!=c (1,null)</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.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);
  }
}

Constraints to solve | Inputs | Observed Constraints
--- | --- | ---
(0,null) | !(c<0) && 0==c
!(c<0) && 0!=c | (1,null) | !(c<0) && 0!=c
c<0 | (-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 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 → 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
public ResourceReader(Stream stream)
{
    if (stream == null)
        throw new ArgumentNullException("stream");
    if (!stream.CanRead)
        throw new ArgumentException(Environme...)

    _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
            // Not ResourceWriter & InternalResCan use different Savings
```
White box testing in practice

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

    _objFormatter = bf;
    try {
        // Read ResourceManager header
        // Check for magic number
        int magicNum = _store.ReadInt32();
        if (public virtual int ReadInt32() {
            if (m_isMemoryStream) {
                // read directly from MemoryStream buffer
                // MemoryStream mStream = m_stream as MemoryStream;
                BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");
                int mInt;
                return mStream.InternalReadInt32();
            } else {
                FillBuffer(4);
            }
        }
        // Read in type name for a suitable ResourceReader
        // New ResourceWriters & ResourceResGen use different walkers
    }
```
Pex — Test Input Generation

```
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
```
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

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

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

Linear arithmetic

Bitvector

Arrays

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.
The Three Main Applications

- Bug finding
- Test case generation
- Verification
Bug finding and Test-case generation tools are successful because Bugs and moles can be easily digested.

Evidence is easy to understand and check.
Bugs and Moles can be checked using the **ULTIMATE ORACLE** (the actual program)
Verification from a Skeptical Point of View

How do I trust the verifier?

Buggy axiomatizations.

Wrong properties.

Hidden assumptions.
How do I trust the verifier?

Z3 may be buggy!
How do I trust the verifier?

Z3 may be buggy!

Standard Solution
Certificate (aka proof) generation
Z3 may be buggy!

Standard Solution
Certificate (aka proof) generation

Problems
Overhead in terms of memory and time.
How easy is to check the certificate?
The Ultimate Distraction

Verifying the Verifier
Wrong axiomatization

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)
Wrong properties

How should we interpret the statement “I proved program X correct”?
Hidden Assumptions

Author: “I proved that Program X can’t crash.”

... 

Audience: “What happens when X runs out of memory?”
Author: “X crashes...”
Verification tools are bug-finding tools!

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

We add Loop invariants to speed up 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.

“It is not about proving.”
Bugs and Moles are extracted from models produced by SMT solvers.

“In symbolic reasoning we need attractive models”
SAT/SMT Solvers are successful because they produce models.
Quantifier Reasoning in SMT is a long-standing challenge
Modeling the runtime

∀ h,o,f:
    IsHeap(h) ∧ o ≠ null ∧ read(h, o, alloc) = t
⇒
    read(h,o, f) = null ∨ read(h, read(h,o,f),alloc) = t
Quantifiers in Practice

Frame axioms

∀ o, f:
  o ≠ null ∧ read(h₀, o, alloc) = t ⇒
  read(h₁,o,f) = read(h₀,o,f) ∨ (o,f) ∈ M
Quantifiers in Practice

User provided assertions

\[ \forall i, j: i \leq j \Rightarrow \text{read}(a, i) \leq \text{read}(b, j) \]
Axiomatizing Theories

\( \forall x: p(x,x) \)

\( \forall x,y,z: p(x,y), p(y,z) \implies p(x,z) \)

\( \forall x,y: p(x,y), p(y,x) \implies x = y \)
Candidate Models

\[
F = \forall i, j. \ i \leq j \rightarrow f(i) \leq f(j) \land w \geq v + 2 \land f(v) \leq 1 \land f(w) \leq 3
\]
$\forall i, j. \ i \leq j \rightarrow f(i) \leq f(j) \land w \geq v + 2 \land f(v) \leq 1 \land f(w) \leq 3$

$v \leftarrow 0, \ w \leftarrow 2, \ f \leftarrow [0 \leftarrow 1, \ 2 \leftarrow 3, \ else \leftarrow 4]$
Candidate Models in Practice

can be extracted from candidate models
can be extracted from candidate models

Ultimate Oracle
(the actual program)
Streams of Candidate Models

$M_1, M_2, M_3, M_4, \ldots$

Sequence of more and more refined candidate models
Model Checking Candidate Models

Candidate Model

$\forall$-formula

Model Checker

(Counter-example)
Representing Candidate Models

Structure that satisfies Background theory $T$

"candidate model"

Structure that satisfies $T \cup F$
Representing Candidate Models

Structure that satisfies Background theory $T$

"candidate model"

Structure that satisfies $T \cup F$

Interpretation of uninterpreted symbol $f$ is an expression $I_f[x]$
Representing Candidate Models

\[
F
\begin{align*}
\forall i, j. \ i \leq j \to f(i) & \leq f(j) \land \\
w \geq v + 2 \land f(v) & \leq 1 \land f(w) \leq 3
\end{align*}
\]

\[
G
\]

\[
v \mapsto 0, \ w \mapsto 2, \ f \mapsto [0 \mapsto 1, \ 2 \mapsto 3, \ \text{else} \mapsto 4]
\]
Representing Candidate Models

\[ F \]
\[ \forall i, j. \ i \leq j \rightarrow f(i) \leq f(j) \land \]
\[ w \geq v + 2 \land f(v) \leq 1 \land f(w) \leq 3 \]

\[ G \]

\[ v \mapsto 0, \ w \mapsto 2, \ f(x) \mapsto \text{ite}(x = 0, 1, \text{ite}(x = 2, 3, 4)) \]
Candidate Model $M$

$f \rightarrow l_f[x]$  

Replace $f(t)$ with $l_f[t]$  

$\forall \bar{x}. F[\bar{x}]$
Candidate Model $M$

\[ f \rightarrow I_f[x] \]

Contains only interpreted symbols and variables

\[ \forall \overline{x}. F[\overline{x}] \]

Replace $f(t)$ with $I_f[t]$

\[ \forall \overline{x}. F'[\overline{x}] \]
Candidate Model $M$

$$f \rightarrow I_f[x]$$

Contains only interpreted symbols and variables

Is satisfied by $M$ if negation is unsatisfiable.

$$\forall \overline{x}. \ F[\overline{x}]$$

Replace $f(t)$ with $I_f[t]$

$$\forall \overline{x}. \ F^I[\overline{x}]$$

$$\neg F^I[\overline{s}]$$
\[ \forall i, j. \ i \leq j \rightarrow f(i) \leq f(j) \]

\[ v \mapsto 0, \ w \mapsto 2, \ f(x) \mapsto \text{ite}(x = 0, 1, \text{ite}(x = 2, 3, 4)) \]

\[ s_1 \leq s_2 \land \]

\[ \neg(\text{ite}(s_1 = 0, 1, \text{ite}(s_1 = 2, 3, 4)) \leq \text{ite}(s_2 = 0, 1, \text{ite}(s_2 = 2, 3, 4))) \]
\( \forall i, j. \ i \leq j \rightarrow f(i) \leq f(j) \)

\[ v \leftarrow 0, \ w \leftarrow 2, \ f(x) \leftarrow \text{ite}(x = 0, 1, \text{ite}(x = 2, 3, 4)) \]

\[ s_1 \leq s_2 \land \\
\neg(\text{ite}(s_1 = 0, 1, \text{ite}(s_1 = 2, 3, 4))) \leq \text{ite}(s_2 = 0, 1, \text{ite}(s_2 = 2, 3, 4))) \]

is satisfied by

\[ s_1 \leftarrow 1, \ s_2 \leftarrow 2 \]
Model Checking Candidate Models

∀i, j. i ≤ j → f(i) ≤ f(j)
ν → 0, w → 2, f(x) → ite(x ≤ 0, 1, ite(x ≤ 2, 3, 4))

{s_1} ≤ {s_2} \land

\neg(ite(s_1 ≤ 0, 1, ite(s_1 ≤ 2, 3, 4)) ≤ ite(s_2 ≤ 0, 1, ite(s_2 ≤ 2, 3, 4)))

is unsatisfiable
Synthesis a “new” application

Bug finding

Synthesis

Verification
Loop Invariants

Ranking Functions (Termination)

Code
Synthesis: Loop Invariants

\( pre \)

\textbf{while} \ (c) \ { \\
\hspace{1cm} T \\
\}

\( post \)

\( \forall s. \ pre[s] \rightarrow I(s) \)
\( \forall s, s'. \ I(s) \land c[s] \land T[s, s'] \rightarrow I(s') \)
\( \forall s. \ I(s) \land \neg c[s] \rightarrow post[s] \)
Synthesis: Ranking Functions

\[ \text{pre} \]
\[ \text{while } (c) \{ \]
\[ T \]
\[ \} \]
\[ \text{post} \]

\[ \forall s. \ rank(s) \geq 0 \]
\[ \forall s, s'. c[s] \land T[s, s'] \rightarrow rank(s') < rank(s) \]
**Synthesis: Loop Invariants**

```plaintext
assert (n >= 0);
x = 0; y = 0;
while (x < n) {
x = x + 1;
y = y + 1;
}
assert (y == n);

∀x, y, n. n ≥ 0 ∧ x = 0 ∧ y = 0 → I(x, y, n)
∀x, y, n, x', y', n'. I(x, y, n) ∧ x < n ∧ x' = x + 1 ∧ y' = y + 1 ∧ n' = n → I(x', y', n')
∀x, y, n. I(x, y, n) ∧ ¬(x < n) → y = n

I(x, y, n) ← x = y ∧ x ≤ n
```
Most SMT Solvers will return “unknown” for this formula.
Basic Idea for synthesising a function:

Use a (set of) skeletons.

Reduce the search to the space of instances.
An Skeleton is just an expression $t[x, c]$.

Example:

$ax + b$, where $a$ and $b$ are fresh constants

Instances:

$x + 1$ ($a \mapsto 1, b \mapsto 1$) and $2x$ ($a \mapsto 2, b \mapsto 0$)
Template Binding (TB):
associate a template with each uninterpreted function
\[ f \mapsto t[x, c] \]

SMF procedure for instantiating templates.
Given quantifier-free formula \( \varphi \) and TB

\[ \varphi \wedge \bigwedge_{f(r) \in \varphi} f(r) = t[r, c] \]
Skeleton Based Model Finding

\[ f(a_1) \geq 10 \land f(a_2) \geq 100 \land f(a_3) \geq 1000 \land \]
\[ a_1 = 0 \land a_2 = 1 \land a_3 = 2 \]
\[ f \mapsto c_1 x + c_2 \]

\[ f(a_1) \geq 10 \land f(a_2) \geq 100 \land f(a_3) \geq 1000 \land \]
\[ a_1 = 0 \land a_2 = 1 \land a_3 = 2 \land \]
\[ f(a_1) = c_1 a_1 + c_2 \land f(a_2) = c_1 a_2 + c_2 \land \]
\[ f(a_3) = c_1 a_3 + c_2 \]

\[ c_1 \rightarrow 1, \ c_2 \rightarrow 1000 \]
Quantified Bit-Vector Logic (QBVF)

Joint work with C. Wintersteiger and Y. Hamadi
FMCAD’10

Ingredients:
\( \forall \exists \)

Bit-vectors

Uninterpreted Function Symbols
Quantified Bit-Vector Logic (QBVF)

Very attractive for software analysis & synthesis.

Decidable.

Arrays (Memory) can be easily encoded.

NEXPTIME-Complete
QBF (PSPACE-complete)
Example: array updates

\[ f' = \text{write}(f, a+1, 0) \]

\[ f'(a + 1) = 0 \land (\forall x : 8. \ x = a + 1 \lor f'(x) = f(x)) \]
New (prototype) Solver:

First-order techniques:

- Mini-scoping, Skolemization, DER,
- Rewriting, ...

Theory rewriting

Quasi-Macro detection

Skeleton Based Model Finding
Putting everything together

\textbf{solver}(\varphi, \text{TB})

\begin{align*}
\varphi & := \text{Simplify}(\varphi) \\
\text{w.l.o.g. assume } \varphi \text{ is of the form } \forall \overline{x}. \phi[\overline{x}] \\
\rho & := \text{HeuristicInst}(\phi[\overline{x}])
\end{align*}

\textbf{loop}

\begin{align*}
\text{if } \text{SMT}(\rho) = \textit{unsat} & \textbf{ return unsat} \\
M & := \text{SMF}(\rho, \text{TB}) \\
\text{if } M = \bot & \textbf{ return unsat modulo TB} \\
V & := \text{MC}(\varphi, M) \\
\text{if } V = \top & \textbf{ return (sat, M)} \\
\rho & := \rho \land \bigwedge_{\overline{v} \in V} \phi[\overline{v}]
\end{align*}
If we replace every $f(s)$ with $t[s,c]$, we are essentially reducing QBVF to QBF.

\[ \text{NEXTPTIME} \rightarrow \text{PSPACE} \]
Very good experimental results.

Hardware Fixpoint Checks.
Given: $I[x]$ and $T[x, x']$

$$\forall x, x'. \ I[x] \land T^k[x, x'] \rightarrow \exists y, y'. I[y] \land T^{k-1}[y, y']$$

Ranking function synthesis.
Hardware Fixpoint Checks
Ranking Function Synthesis
Symbolic Reasoning is extensively used at Microsoft.

Model generation is a “must” for most applications.

Bug finding and Mole generation tools are the most successful.

Verification is bug-finding with better coverage.

Synthesis is a new hot area.

Skeleton Based Model Finding makes intractable problems tractable.