Leonardo de Moura and Nikolaj Bjørner
Microsoft Research

Tutorial overview

- Appetizers
  - SMT solving
  - Applications

- Applications at Microsoft Research

- Background
  - Basics, DPLL(∅), Equality, Arithmetic, DPLL(T), Arrays, Matching

- Z3 – An Efficient SMT solver
SMT Appetizer

Domains from programs

- **Bits and bytes**: \( 0 = ( ( x - 1 ) \& x ) \iff x = 00100000..00 \)
- **Arithmetic**: \( x + y = y + x \)
- **Arrays**: \( \text{read}(\text{write}(a,i,4),i) = 4 \)
- **Records**: \( \text{mkpair}(x,y) = \text{mkpair}(z,u) \Rightarrow x = z \)
- **Heaps**: \( n \rightarrow^{*} n' \land m = \text{cons}(a,n) \Rightarrow m \rightarrow^{*} n' \)
- **Data-types**: \( \text{car}(\text{cons}(x,nil)) = x \)
- **Object inheritance**: \( B <: A \land C <: B \Rightarrow C <: A \)
Satisfiability Modulo Theories (SMT)

\[ x + 2 = y \Rightarrow f(\text{read}(\text{write}(a, x, 3), y - 2)) = f(y - x + 1) \]

Arithmetic  Arrays  Free Functions

Applications Appetizer
Some takeaways from Applications

- SMT solvers are used in several applications:
  - Program Verification
  - Program Analysis
  - Program Exploration
  - Software Modeling

- SMT solvers are
  - directly applicable, or
  - disguised beneath a transformation

- Theories and quantifiers supply abstractions
  - Replace ad-hoc, often non-scalable, solutions

Program Verification

Hyper-V
Win. Modules
VCC
HAVOC
Boogie
Z3

Bug path

Rustan Leino, Mike Barnet, Michal Moskal, Shaz Qadeer, Shuvendu Lahiri, Herman Venter, Peter Muller, Wolfram Schulte, Ernie Cohen
**Test case generation**

- Run Test and Monitor
- Execution Path
- Path Condition
- Test Inputs
- Known Paths
- Unexplored path
- Solve

- Seed
- New input

**Z3**

Nikolai Tillmann, Peli de Halleux, Patrice Godefroid
Aditya Nori, Jean Philippe Martin, Miguel Castro,
Manuel Costa,Lintao Zhang

Vigilante

**Static Driver Verifier**

- Z3 is part of SDV 2.0 (Windows 7)
- It is used for:
  - Predicate abstraction (c2bp)
  - Counter-example refinement (newton)
More applications

- Bounded model-checking of model programs
- Termination
- Security protocols, F#/7
- Business application modeling
- Cryptography
- Model Based Testing (SQL-Server)
- Verified garbage collectors
Program Exploration with *Pex*

Nikolai Tillmann, Peli de Halleux

http://research.microsoft.com/Pex

---

**What is Pex**

- **Test input generator**
  - Pex starts from parameterized unit tests
  - Generated tests are emitted as traditional unit tests

- **Dynamic symbolic execution framework**
  - Analysis of .NET instructions (bytecode)
  - Instrumentation happens automatically at JIT time
  - Using SMT-solver Z3 to check satisfiability and generate models = test 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; }
...

© 2007 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries.
The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.
MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
class ArrayListTest {
   [PexMethod]
   void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.AddItem();
      Assert(list[0] == item); }
}

class ArrayList {
   object[] items;
   int count;

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

   void Add(object item) {
      // 0 == c  true
      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();
         Assert(list[0] == item); }
   }

   class ArrayList {
      object[] items;
      int count;

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

      void Add(object item) {
         // 0 == c  true
         if (count == items.Length)
            ResizeArray();
         items[this.count++] = item; }
      ...
   }
ArrayList: Run 1, (0,null)

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

```csharp
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; }
}
```

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

ArrayList: Picking the next branch to cover

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

```csharp
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; }
}
```
**ArrayList: Solve constraints using SMT solver**

```csharp
class ArrayListTest { [PexMethod] void AddItem(int c, object item) { var list = new ArrayList(c); list.AddItem(); 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; 0==c</td>
</tr>
<tr>
<td>(1,null)</td>
<td>!(c&lt;0) &amp; 0!=c</td>
</tr>
</tbody>
</table>

**Z3**

Constraint solver

Z3 has decision procedures for
- Arrays
- Linear integer arithmetic
- Bitvector arithmetic
- ...
- (Everything but floating-point numbers)

**ArrayList Run 2, (1, null)**

```csharp
class ArrayListTest { [PexMethod] void AddItem(int c, object item) { var list = new ArrayList(c); list.AddItem(); 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; 0==c</td>
</tr>
<tr>
<td>(1,null)</td>
<td>!(c&lt;0) &amp; 0!=c</td>
</tr>
</tbody>
</table>

8/19/2008
**ArrayList: Pick new branch**

```csharp
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem();
        Assert(list[0] == 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</td>
<td></td>
</tr>
<tr>
<td>(1,null)</td>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
</tr>
<tr>
<td>c&lt;0</td>
<td></td>
</tr>
</tbody>
</table>

**ArrayList: Run 3, (-1, null)**

```csharp
class ArrayListTest {
    [PexMethod]
    void AddItem(int c, object item) {
        var list = new ArrayList(c);
        list.AddItem();
        Assert(list[0] == 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</td>
<td></td>
</tr>
<tr>
<td>(1,null)</td>
<td>!(c&lt;0) &amp;&amp; 0!=c</td>
</tr>
<tr>
<td>c&lt;0 (-1,null)</td>
<td></td>
</tr>
</tbody>
</table>
ArrayList: Run 3, (-1, null)

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

ArrayList: Run 3, (-1, null)

class ArrayListTest {  
    [PexMethod]  
    void AddItem(int c, object item) {  
        var list = new ArrayList(c);  
        list.AddItem();  
        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  
c<0  (-1,null)  c<0  

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
Pex – Test more with less effort

- Reduce testing costs
  - Automated analysis, reproducible results
- Produce more secure software
  - White-box code analysis
- Produce more reliable software
  - Analysis based on contracts written as code

White box testing in practice

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 ArgumentException("stream");
    if (!stream.CanRead)
        throw new ArgumentException(Env.DOSResourceString("Argument_StreamNotReadable"));

    _resCache = new Dictionary<string, ResourceLocator>(FastResourceComparer.Default);
    _store = new BinaryReader(stream, Encoding.UTF8);
    // We have a faster path for reading resource files from an assembly.
    _ums = stream as UnmanagedMemoryStream;
    ReadResources();
}
```
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()
{
    BinaryFormatter bf = new BinaryFormatter(null, new StreamingContext(StreamingContextStates.File);
    if (FEATURE_WAL
        _objFormatter = new TypeLimitingDeserializationBinder();
    bfBinder = _objFormatter;
    try
    {
        // Read ResourceManager header
        // Check for magic number
        srg = new ReadInt32();
        if (PUBLIC virtual int ReadInt32();
            if (m_objectBinaryStream)
            {
                // Read directory from MemoryStream
                // BCLDebug.Assert(mStream == null, "m_stream as MemoryStream != null");
                return mStream.InternalReadInt32();
            }
            else
            {
            }
        }
    }
}
```

---

Pex – Test Input Generation tomorrow

Test input, generated by Pex
```csharp
byte[] a = new byte[14];
a[0] = 204;
a[1] = 202;
a[2] = 2396;
a[3] = 190;
a[4] = 64;
```
Test Input Generation by Dynamic Symbolic Execution

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

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

Initially, choose Arbitrary
Finds only real bugs
No false warnings

\[
\begin{align*}
a[0] &= 0; \\
a[1] &= 0; \\
a[2] &= 0; \\
a[3] &= 0;
\end{align*}
\]
Test Input Generation by Dynamic Symbolic Execution

Result: small test suite, high code coverage
Finds only real bugs
No false warnings
Test Input Generation by Dynamic Symbolic Execution

\[
a[0] = 206; \\
a[1] = 202; \\
a[2] = 239; \\
a[3] = 190;
\]

Result: small test suite, high code coverage

Finds only real bugs
No false warnings

Test Input Generation by Dynamic Symbolic Execution

Result: small test suite, high code coverage

Finds only real bugs
No false warnings
Independent constraint optimization + Constraint caching (similar to EXE)

- Idea: Related execution paths give rise to "similar" constraint systems

- Example: Consider $x>y \land z>0$ vs. $x>y \land z<=0$

- If we already have a cached solution for a "similar" constraint system, we can reuse it
  - $x=1, y=0, z=1$ is solution for $x>y \land z>0$
  - we can obtain a solution for $x>y \land z<=0$ by
    - reusing old solution of $x>y$: $x=1, y=0$
    - combining with solution of $z<=0$: $z=0$
Constraint Solving: Z3

- **Rich Combination**: Solvers for uninterpreted functions with equalities, linear integer arithmetic, bitvector arithmetic, arrays, tuples
- Formulas may be a big conjunction
  - Pre-processing step
  - Eliminate variables and simplify input format
- **Universal quantifiers**
  - Used to model custom theories, e.g. .NET type system
- **Model generation**
  - Models used as test inputs
- **Incremental** solving
  - Given a formula $F$, find a model $M$, that minimizes the value of the variables $x_0 \ldots x_n$
  - **Push / Pop** of contexts for model minimization
- **Programmatic** API
  - For small constraint systems, text through pipes would add huge overhead

---

Monitoring by Code Instrumentation

```csharp
class Point { int x; int y;
    public static int GetX(Point p) {
        if (p != null) return p.X;
        else return -1;
    }
}
```

**Prologue**
- Record concrete values when all information about a method is called
- Calls to build path condition

**Epilogue**
- Calls to build path condition
- (The real C# compiler output is actually more complicated.)

© 2007 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries.
The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.
MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
Spec# and Boogie

Rustan Leino & Mike Barnett

A verifying compiler uses *automated reasoning* to check the correctness of a program that is compiled.

Correctness is specified by *types, assertions, . . . and other redundant annotations* that accompany the program.

Tony Hoare 2004
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),
  - automatic theorem proving Z3

---

Basic verifier architecture

Source language

Intermediate verification language

Verification condition (logical formula)
Verification architecture

Spec# compiler
MSIL
Bytecode translator
Boogie
V.C. generator
Inference engine
Z3
 Verification condition

Spec#

C

C

Dafny

Dafny verifier

Modeling execution traces

terminates
diverges
goes wrong
States and execution traces

- **State**
  - Cartesian product of variables (x: int, y: int, z: bool)

- **Execution trace**
  - Nonempty finite sequence of states
  - Infinite sequence of states
  - Nonempty finite sequence of states followed by special error state

Command language

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

- havoc x

- assert P
  - ¬P

- assume P
  - P
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 \( Q \)

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

- $wp(\ x := E,\ Q\ ) = Q[\ E / x ]$
- $wp(\ havoc\ x,\ Q\ ) = (\forall x \bullet Q)$
- $wp(\ assert\ P,\ Q\ ) = P \land Q$
- $wp(\ assume\ P,\ Q\ ) = P \Rightarrow Q$
- $wp(\ S ; T,\ Q\ ) = wp(\ S,\ wp(\ T,\ Q))$
- $wp(\ S \parallel T,\ Q\ ) = wp(\ S,\ Q\ ) \land wp(\ T,\ Q)$

Structured if statement

if $E$ then $S$ else $T$ end =

\[-\text{assume } E;\ S\]
\[-\text{assume } \neg E;\ T\]
Dijkstra's guarded command

if \( E \rightarrow S \mid F \rightarrow T \) fi =

\[
\text{assert } E \lor F; \\
( \\
\text{assume } E; S \\
\text{assume } F; T \\
) 
\]

Picking any good value

assign \( x \) such that \( P = \) havoc \( x; \) assume \( P \)

\[
\text{assign } x \text{ such that } x^2 = y
\]
A **procedure** is a user-defined command

**procedure** M(x, y, z) **returns** (r, s, t)
- requires P
- modifies g, h
- ensures Q

**Procedure example**

**procedure** Inc(n) **returns** (b)
- requires 0 ≤ n
- modifies g
- ensures g = old(g) + n
Procedures

- A **procedure** is a user-defined command
- procedure \( M(x, y, z) \) returns \((r, s, t)\) requires \(P\) modifies \(g, h\) ensures \(Q\)
- call \(a, b, c := M(E, F, G)\)
  \[
  = x' := E; \ y' := F; \ z' := G; \\
  \text{assert } P'; \\
  g0 := g; \ h0 := h; \\
  \text{havoc } g, h, r', s', t'; \\
  \text{assume } Q'; \\
  a := r'; \ b := s'; \ c := t'
  \]

Procedure implementations

- procedure \( M(x, y, z) \) returns \((r, s, t)\) requires \(P\) modifies \(g, h\) ensures \(Q\)
- implementation \( M(x, y, z) \) returns \((r, s, t)\) is \(S\)
  \[
  = \text{assume } P; \\
  g0 := g; \ h0 := h; \\
  S; \\
  \text{assert } Q'
  \]

Where:
- \(x', y', z', r', s', t', g0, h0\) are fresh names
- \(P'\) is \(P\) with \(x', y', z'\) for \(x, y, z\)
- \(Q'\) is \(Q\) with \(x', y', z', r', s', t', g0, h0\) for \(x, y, z, r, s, t, \text{old}(g), \text{old}(h)\)
- syntactically check that \(S\) assigns only to \(g, h\)
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
  )

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

where x denotes the assignment targets of S

Properties of the heap

- introduce:
  
  axiom (∀ h: HeapType, o: Ref, f: Field Ref •
    o ≠ null ∧ h[o, alloc]
    ⇒
    h[o, f] = null ∨ h[ h[o,f], alloc ] );
Properties of the heap

- introduce:
  
  \[
  \text{function } \text{IsHeap}(\text{HeapType}) \text{ returns } (\text{bool});
  \]

- introduce:
  
  \[
  \text{axiom } (\forall h: \text{HeapType}, o: \text{Ref}, f: \text{Field Ref} \bullet \\
  \text{IsHeap}(h) \land o \neq \text{null} \land h[o, \text{alloc}] \\
  \implies \\
  h[o, f] = \text{null} \lor h[h[o,f], \text{alloc}]);
  \]

- introduce: assume \text{IsHeap}(\text{Heap}) after each Heap update; for example:

  \[
  \text{Tr}[[\ E.x := F ]] = \\
  \text{assert } \ldots; \text{Heap}[[\ldots]] := \ldots;
  \]

  assume \text{IsHeap}(\text{Heap})

---

Methods

- method \text{M}(x: X) \text{ returns } (y: Y)
  
  requires \text{P}; \text{modifies } S; \text{ensures } Q;
  
  \{
  \text{Stmt }
  \}

- procedure \text{M}(\text{this}: \text{Ref}, x: \text{Ref}) \text{ returns } (y: \text{Ref})
  
  free requires \text{IsHeap}(\text{Heap});
  
  free requires \text{this} \neq \text{null} \land \text{Heap}[\text{this}, \text{alloc}];
  
  free requires \text{x} = \text{null} \lor \text{Heap}[\text{x}, \text{alloc}];

  requires \text{Df[[ P ]]} \land \text{Tr[[ P ]]};

  requires \text{Df[[ S ]]};

  modifies \text{Heap};

  ensures \text{Df[[ Q ]]} \land \text{Tr[[ Q ]]};

  ensures (\forall \alpha) \ o: \text{Ref}, f: \text{Field } \alpha \bullet 
  
  o \neq \text{null} \land \text{old}(\text{Heap})[o, \text{alloc}] \Rightarrow 
  
  \text{Heap}[o,f] = \text{old}(\text{Heap})[o,f] \lor 
  
  (o,f) \in \text{old}(\text{Tr[[ S ]]});

  free ensures \text{IsHeap}(\text{Heap});
  
  free ensures \text{y} = \text{null} \lor \text{Heap}[\text{y}, \text{alloc}];

  free ensures (\forall \text{o: Ref} \bullet \text{old}(\text{Heap})[o,\text{alloc}] \Rightarrow \text{Heap}[o,\text{alloc}]);
Spec# Chunker.NextChunk

procedure Chunker.NextChunk(this: ref where $IsNotNil(this, Chunker)) returns ($Result: ref where $IsNotNil($Result, System.String));
// in parameter: target object
free requires $Heap[$this, $Allocated] == $PeerGroupPlaceholder || ($Heap[$this, $OwnerRef] == $Heap[$this, $OwnerFrame])
$Heap[$this, $OwnerFrame] == $BaseClass[$Heap[$this, $OwnerFrame]] && $forall $sp: ref : $sp != null && $Heap[$sp, $Allocated] && $Heap[$sp, $OwnerFrame] == $BaseClass[$Heap[$sp, $OwnerFrame]] && $Heap[$sp, $Allocated] == $Heap[$sp, $Allocated];
// out parameter: return value

free ensures $Heap[$Result, $Allocated] == $PeerGroupPlaceholder || ($Heap[$Result, $OwnerRef] == $Heap[$Result, $OwnerFrame])

Z3 & Program Verification

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms (“what didn’t change”)
- Users provided assertions (e.g., the array is sorted)
- Prototyping decision procedures (e.g., reachability, heaps, ...)
- Solver must be fast in satisfiable instances.
- Trade-off between precision and performance.
- Candidate (Potential) Models
The Static Driver Verifier
SLAM

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

do {
    KeAcquireSpinLock();

    if(*){
        KeReleaseSpinLock();
    }
} while (*);

KeReleaseSpinLock();
Is error path feasible?

\[
\begin{array}{l}
do \{ \\
\quad \text{KeAcquireSpinLock();} \\
\quad \text{nPacketsOld = nPackets;} \\
\quad \text{if(request)} \{ \\
\quad\quad \text{request = request->Next;} \\
\quad\quad \text{KeReleaseSpinLock();} \\
\quad\quad \text{nPackets++;} \\
\quad \} \\
\} \text{ while (nPackets != nPacketsOld);} \\
\text{KeReleaseSpinLock();}
\end{array}
\]

Add new predicate to Boolean program
\[ b: (nPacketsOld == nPackets) \]

\[
\begin{array}{l}
do \{ \\
\quad \text{KeAcquireSpinLock();} \\
\quad b = true; nPackets; \\
\quad \text{if(request)} \{ \\
\quad\quad \text{request = request->Next;} \\
\quad\quad \text{KeReleaseSpinLock();} \\
\quad\quad \text{b = b ? false : *;} \\
\quad \} \\
\} \text{ while (!nPackets != nPacketsOld);} \\
\text{KeReleaseSpinLock();}
\end{array}
\]
do {
    KeAcquireSpinLock();

    b = true;

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

KeReleaseSpinLock();
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
Syntactic Sugar

```c
if (e) {
    S1;
} else {
    S2;
} S3;
```

```
goto L1, L2;
L1: assume(e);
    S1;
    goto L3;
L2: assume(!e);
    S2;
    goto L3;
L3: S3;
```

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 Assignments via WP

- Statement \( y = y + 1 \) and \( F = \{ y < 4, y < 5 \} \)
  - \( \{ y < 4 \}, \{ y < 5 \} = ((!\{ y < 5 \} \mid \mid !\{ y < 4 \}) ? \text{false} : \ast), \{ y < 4 \} \)

- \( WP(x = e, Q) = Q[e/x] \)
- \( WP(y = y + 1, y < 5) = \)
  - \( (y < 5) \quad [y + 1/y] = \)
  - \( (y + 1 < 5) \quad = \)
  - \( (y < 4) \)

WP Problem

- \( WP(s, p_i) \) is not always expressible via \( \{ p_1, \ldots, p_n \} \)
- Example:
  - \( F = \{ x == 0, x == 1, x < 5 \} \)
  - \( WP(x = x + 1, x < 5) = x < 4 \)
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 $e$.
  - $\text{ImpliedBy}_F(e) = \text{not Implies}_F(\text{not } e)$
Computing $\text{Implies}_F(e)$

- minterm $m = l_1 \wedge \ldots \wedge l_n$, 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 $m \Rightarrow e$.
- 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 $y > 1$
- $x < y, x = 2$ implies $y > 1$
- $\neg x < y, x = 2$ implies $y > 1$
- $x < y, x = 2$ implies $y > 1$

$\text{Implies}_F(y > 1) = b_1 y \lor b_2 = 2$
Abstracting Assignments

- if \( \text{Implies}_F(WP(s, p_i)) \) is true before \( s \) then
  - \( p_i \) is true after \( s \)

- if \( \text{Implies}_F(WP(s, \neg p_i)) \) is true before \( s \) then
  - \( p_i \) is false after \( s \)

\[
\{p_i\} = \begin{cases} \text{Implies}_F(WP(s, p_i)) & \text{true} \\ \text{Implies}_F(WP(s, \neg p_i)) & \text{false} \\ * & \end{cases}
\]

Assignment Example

Statement: \( y = y + 1 \)       Predicates: \( \{x == y\} \)

Weakest Precondition:
\( WP(y = y + 1, x==y) = x == y + 1 \)

\( \text{Implies}_F( x==y+1 ) = \text{false} \)
\( \text{Implies}_F( x!=y+1 ) = x==y \)

Abstraction of \( y = y + 1 \)
\( \{x == y\} = \{x == y\} \text{ true} : *; \)
Abstracting Assumes

WP(assume(e), Q) = e implies Q
assume(e) is abstracted to:

\texttt{assume( ImpliedBy}_e(e) )

Example:
F = \{x==2, x<5\}
assume(x < 2) is abstracted to:
\texttt{assume(!}\{x==2\} \&\& \{x<5\})

---

Newton

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

Let $S$ be an unsatisfiable set of formulas.
$S' \subseteq S$ is an unsatisfiable core of $S$ if:
- $S'$ is also unsatisfiable, and
- There is not $S'' \subseteq S'$ that is also unsatisfiable.

Computing $\text{Implies}_F(e)$ with $F = \{p_1, p_2, p_3, p_4\}$
- Assume $p_1, p_2, p_3, p_4 \Rightarrow e$ is valid
- That is $p_1, p_2, p_3, p_4, \neg e$ is unsat
- Now assume $p_1, p_3, \neg e$ is the unsatisfiable core
- Then it is unnecessary to check:
  - $p_1, \neg p_2, p_3, p_4 \Rightarrow e$
  - $p_1, \neg p_2, p_3, \neg p_4 \Rightarrow e$
  - $p_1, p_2, p_3, \neg p_4 \Rightarrow e$
A Verifying C Compiler

Ernie Cohen, Michal Moskal, Herman Venter, Wolfram Schulte
+ Microsoft Aachen + Verisoft Saarbrücken

- **Meta OS**: small layer of software between hardware and OS
- **Mini**: 60K lines of non-trivial concurrent systems C code
- **Critical**: must provide functional resource abstraction
- **Trusted**: a grand verification challenge
What is to be verified?

- Source code
  - C + x64 assembly

- Sample verifiable slices:
  - **Safety**: Basic memory safety
  - **Functionality**: Hypervisor simulates a number of virtual x64 machines.
  - **Utility**: Hypervisor services guest OS with available resources.
HAVOC
Verifying Windows Components

Lahiri & Qadeer, POPL'08,
Also: Ball, Hackett, Lahiri, Qadeer, MSR-TR-08-82.

HAVOC's Architecture

C program      Property      Annotations
              |                  |
               Front End     |
               |                  |
               BoogiePL program

Boogie VC Generator

Verification Conditions

Verified

Warning

Z3
Heaps and Shapes

doubly linked lists in Windows Kernel code

Precise and expressive heap reasoning

pointer arithmetic

transitive closure

Reach(next, u) ≡ {u, u->next, u->next->next, ...}
forall (x, Reach(next,p), CONTAINING_RECORD(x, IRP, link)->state == PENDING)
Annotation Language & Logic

- Procedure contracts
  - requires, ensures, modifies
- Arbitrary C expressions
  - program variables, resources
  - Boolean connectives
  - quantifiers
- Can express a rich set of contracts
  - API usage (e.g. lock acquire/release)
  - Synchronization protocols
  - Memory safety
  - Data structure invariants (linked list)
- Challenge:
  - Retain efficiency
  - Decidable fragments

Logic with Reach, Quantifiers, Arithmetic

- Expressive
- Careful use of quantifiers
- Efficient logic
  - Only NP-complete

Efficient logic for program verification

Logic with Reach, Quantifiers, Arithmetic

- Expressive
- Careful use of quantifiers
- Efficient logic
  - Only NP-complete

Encoding using quantifiers and triggers

© 2007 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries.

The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
Combining Random Testing with Model Checking

Aditya Nori, Sriram Rajamani,
ISSTA08: Proofs from Tests. Nels E. Beckman, Nori, Rajamani, Rob Simmons

DASH Algorithm

- Main workhorse: test case generation
- Use counterexamples from current abstraction to "extend frontier" and generate tests
- When test case generation fails, use this information to "refine" abstraction at the frontier
  - Use only aliases that happen on the tests!
void lock(int *x)
{
23:  if(*x != 0)
24:    error();
25:  *x = 1;
}

void unlock(int *x)
{
26:  if(*x != 1)
27:    error();
28:  *x = 0;
}

Example

struct ProtectedInt
{
  int *lock;
  int *y;
};

void LockUnlock(struct ProtectedInt *pi,
                 int *lock1, int *lock2, int x)
{
  1:  int do_return = 0;
  2:  if(pi->lock == lock1)
  3:    do_return = 1;
  4:    pi->lock = lock2;
  5:  else if(pi->lock == lock2)
  6:    do_return = 1;
  7:    pi->lock = lock1;
  8:    //initialize all locks to be unlocked
  9:    *(pi->lock) = 0;
10:   *lock1 = 0;
11:   *lock2 = 0;
12:   if( do_return ) return;
13:   else {
14:     lock(pi->lock);
15:     if(*lock1 == 1 || *lock2 == 1)
16:       error();
17:     x = *(pi->y);
18:     if ( *NonDet() ) {
19:       (*pi->y)++;
20:       unlock(pi->lock);
21:     } while(x != *(pi->y));
22:     unlock(pi->lock);
}

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!=y);
  8:    unlock();
}
Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)

Symbolic execution + Theorem Proving

Input:
Program P
Property ψ

Construct random tests
Construct initial abstraction

Test succeeded?
yes
no

Abstraction succeeded?
yes
no

Can extend test beyond frontier?
yes
no

Symbolic execution + Theorem proving

Refine abstraction

y = 1

τ = (0,1,2,3,4,7,8,9)

Example

void prove_me(int y)
{
  1:  do {
  2:    lock();
  3:    x = y;
  4:    if (*) {
  5:      unlock();
  6:      y = y + 1;
  7:    } while (x!y);
  8:    unlock();
  }
}

t = (0,1,2,3,4,7,8,9)
Example

void prove_me(int y)
{
    do {
        lock();
        x = y;
        if (i) {
            unlock();
            y = y + 1;
        }
    } while (x!=y);
    unlock();
}

Template-based refinement

ρ = (lock.state != L)
Template-based refinement

\[ \rho = (\text{lock.state} \neq L) \]

Example

```cpp
void prove_me(int y)
{
    do {
        lock();
        x = y;
        if (*) {
            unlock();
            y = y + 1;
        }
    } while (x!=y);
    unlock();
}
```

Input:
Program P
Property \( \psi \)

Construct initial abstraction
Construct random tests

Test succeeded?
Yes
No

Abstraction succeeded?
Yes
No

\( \tau = \text{error path in abstraction} \)
\( f = \text{frontier of error path} \)

Can extend test beyond frontier?
Yes
No

Refine abstraction

\[ \tau = (0,1,2,3,4,7,<8,p>,9) \]

Frontier
Proof!

```c
void prove_me(int y) {
    1:     do {
    2:         lock();
    3:         x = y;
    4:         if (*) { 
    5:             unlock();
    6:             y = y + 1;
    7:         } while (x!=y);
    8:     unlock();
}
```

Correct, the program is

**Yogi's solver interface**

**Representation**
- \( L \)
  - program locations.
- \( R \subseteq L \times L \)
  - Control flow graph
- **State:** \( L \rightarrow \text{Formula } \neg \text{set} \)
  - Symbolic state: each location has set of **disjoint** formulas

**Theorem proving needs**
- Facts about pointers:
  - \(*_x = x\)
- Subsumption checks:
  - \( \varphi \Rightarrow WP(I, \psi) \)
  - \( \varphi \Rightarrow \neg WP(I, \psi) \)
- Structure sharing
  - Similar formulas in different states
- **Simplification**
  - Collapse/Reduce formulas
Better Bug Reporting with Better Privacy

Miguel Castro, Manuel Costa, Jean-Philippe Martin
ASPLOS 08

See also: Vigilante – Internet Worm
Containment Miguel Castro, Manuel Costa, Lintao Zhang

Example program

```c
int ProcessMessage(int sock, char *msg) {
    char url[20];
    char host[20];
    int i = 0;
        return -1;
    }
    msg = msg + 4;
    while (*msg != 'r' && *msg != ' ') {
        url[i++] = *msg++;
    }
    url[i] = 0;
    GetHost(msg, host);
    return ProcessGet(sock, url, host);
}
```

- buffer overflow

Diagram:

- Replay Execution
- Extract Path Condition
- Solve Path Condition (with Z3)
- Compute Bits Revealed

Example:

```
GET /checkout?product=embarrassing&
creditcardnumber=11233445667788...
```

```
GET /checkout?product=teddybear
&creditcardnumber=001022034011100
```
Finding the buffer overflow

```c
int ProcessMessage(int sock, char *msg) {
    char url[20];
    char host[20];
    int i=0;
        return -1;
    msg = msg+4;
    while (*msg != 'n' & *msg != ' ')
        {url[i]=*msg; msg++;
        }
    url[i] = 0;
    GetHost(msg, host);
    return ProcessGet(sock, url, host);
}
```

Privacy: measure distance between original crash input and new input

Program Termination

Byron Cook

http://www.foment.net/byron/fsharp.shtml

A complete method for the synthesis of linear ranking functions. Podelski & Rybalchenkoy; VMCAI 04
Making use of F#'s math libraries together with Z3

A short note by Byron Cook

Recent work on F#'s math libraries, together with the latest release of Z3 make for a pretty powerful mixture. In particular I find it interesting that it's so easy to combine F#s polymorphic meta-code together with the power of Z3. I recently used F#'s new static syntax and the new Z3 release in order to re-implement the rank function synthesis engine used within TERMINATOR. The result turned out to be so concise that I thought it would be interesting to the larger F# community - I expect that, in the future, Z3 will probably pick up this example and use it as an F# example. Thus, if you're looking for an up-to-date version of the example check the F# daily blog.

At the high-level we're going to build a tool that takes as input a set of first-order constraints and outputs a counter-example. For example, the constraint statement that the sum of x and y is always less than 1, which is a relation stating that the new value of x is always one less than the old value of y, is that x is always positive, and that y goes up. We're not able to automatically prove that this relation is well-founded, meaning that if you apply it to any given sequence of pairs (x,y),(x,y), (x,y), then the relation will eventually not hold on a pair. See more on lecture notes (Lec. 5, 6, 7, 8) for more information.

The underlying algorithm that we'll implement is given in a paper by Podelski and Rybalchenko called "A complete method for the synthesis of linear ranking functions." The core of the paper is in Fig. 1:

Input: program \( (A;P)(x) \in \mathbb{Z} \)

Write:

Find positive rational \( \frac{a_1}{a_2} \) and \( \frac{b_1}{b_2} \) such that

\[
\begin{align*}
A_1 & = 0 \\
A_2 & = 0 \\
A_3 & = 0 \\
\end{align*}
\]


Where program is as defined above.

Does this program Terminate?

\[ x > 0 \land y > 0 \land \\
(\text{while } (x > 0 \land y > 0) [ \\
\begin{align*}
x' &= x - 1 \\
y' &= y + 1 + z^2;
\end{align*}
\]

\[ x > 0 \\
x' \geq x - 1 \\
y > 0 \\
y' > y \]

\[
\begin{align*}
0x' + 0y' - 1x + 0y + 1 & \leq 0 \\
1x' + 0y' - 1x + 0y + 1 & \leq 0 \\
-1x' + 0y' + 1x + 0y - 1 & \leq 0 \\
0x' + 0y' + 0x + 1y + 1 & \leq 0 \\
0x' - 1y' + 0x + 1y + 1 & \leq 0
\end{align*}
\]
Can we find \( f, b \), such that the inclusion holds?

That is:

\[
\begin{align*}
0x' + 0y' + -1x + 0y + 1 & \leq 0 \\
1x' + 0y' + -1x + 0y + 1 & \leq 0 \\
-1x' + 0y' + 1x + 0y + -1 & \leq 0 \\
0x' + 0y' + 0x + -1y + 1 & \leq 0 \\
0x' + -1y' + 0x + 1y + 1 & \leq 0
\end{align*}
\]

\[
\begin{align*}
0x' + 0y' + -1x + 0y + 1 & \leq 0 \\
1x' + 0y' + -1x + 0y + 1 & \leq 0 \\
-1x' + 0y' + 1x + 0y + -1 & \leq 0 \\
0x' + 0y' + 0x + -1y + 1 & \leq 0 \\
0x' + -1y' + 0x + 1y + 1 & \leq 0
\end{align*}
\]

Search over linear templates:

\[
\begin{align*}
f(a, b) & \triangleq c_1 a + c_2 b \\
-f(a, b) & \triangleq c_3 a + c_4 b \\
c_1 &= -1c_3 \\
c_2 &= -1c_4
\end{align*}
\]
Rank function synthesis

Find $c_1, c_2, c_3, c_4$

\[
\begin{align*}
0x' + 0y' + -1x + 0y + 1 & \leq 0 \\
1x' + 0y' + -1x + 0y + 1 & \leq 0 \\
-1x' + 0y' + 1x + 0y + -1 & \leq 0 \\
0x' + 0y' + 0x + -1y + 1 & \leq 0 \\
0x' + -1y' + 0x + 1y + 1 & \leq 0
\end{align*}
\]

Search over linear templates:

\[
f(a, b) \triangleq c_1a + c_2b
\]

\[
-f(a, b) \triangleq c_3a + c_4b
\]

\[
c_1 = -1c_3
\]

\[
c_2 = -1c_4
\]

Rank function synthesis

\[
\exists c_1, c_2, c_3, c_4, \forall x, y, x', y'
\]

\[
\begin{align*}
0x' + 0y' + -1x + 0y + 1 & \leq 0 \\
1x' + 0y' + -1x + 0y + 1 & \leq 0 \\
-1x' + 0y' + 1x + 0y + -1 & \leq 0 \\
0x' + 0y' + 0x + -1y + 1 & \leq 0 \\
0x' + -1y' + 0x + 1y + 1 & \leq 0
\end{align*}
\]

Search over linear templates:

\[
f(a, b) \triangleq c_1a + c_2b
\]

\[
-f(a, b) \triangleq c_3a + c_4b
\]

\[
c_1 = -1c_3
\]

\[
c_2 = -1c_4
\]
Rank function synthesis – simplified version

\[ \exists c_1, c_2, c_3, c_4, \forall x, y, x', y'. \quad 0x' + 0y' + -1x + 0y + 1 \leq 0 \\
1x' + 0y' + -1x + 0y + 1 \leq 0 \\
0x' + 0y' + 0x + -1y + 1 \leq 0 \\
0x' + -1y' + 0x + 1y + 1 \leq 0 \]

\[ R \triangleq -1x' + 0y' + 1x + 0y + -1 \leq 0 \]

Search over linear templates:

\[ f(a, b) \triangleq c_1 a + c_2 b \]
\[ -f(a, b) \triangleq c_3 a + c_4 b \]
\[ c_1 = -1c_3 \]
\[ c_2 = -1c_4 \]

Rank function synthesis

\[ \exists c_1, c_2, c_3, c_4, \forall x, y, x', y'. \quad 0x' + 0y' + -1x + 0y + 1 \leq 0 \\
1x' + 0y' + -1x + 0y + 1 \leq 0 \\
0x' + 0y' + 0x + -1y + 1 \leq 0 \\
0x' + -1y' + 0x + 1y + 1 \leq 0 \]

\[ R \triangleq -1x' + 0y' + 1x + 0y + -1 \leq 0 \]

\[ \psi \triangleq c_1 x' + c_2 y' + c_3 x + c_4 y + 1 \leq 0 \]

Farkas’ lemma. \( R \Rightarrow \psi \) iff there exist real multipliers \( \lambda_1, \ldots, \lambda_5 \geq 0 \) such that

\[ c_1 = \sum_{i=1}^{5} \lambda_i a_{i,1} \quad \cdots \quad c_4 = \sum_{i=1}^{5} \lambda_i a_{i,4} \quad \land \quad 1 \leq (\sum_{i=0}^{5} \lambda_i b_i) \]
Rank function synthesis

Instead solve: $\exists c_1, c_2, c_3, c_4, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$

$c_1 = 0\lambda_1 + 1\lambda_2 + -1\lambda_3 + 0\lambda_4 + 0\lambda_5$
$c_2 = 0\lambda_1 + 0\lambda_2 + 0\lambda_3 + 0\lambda_4 + -1\lambda_5$
$c_3 = -1\lambda_1 + -1\lambda_2 + 1\lambda_3 + 0\lambda_4 + 0\lambda_5$
$c_4 = 0\lambda_1 + 0\lambda_2 + 0\lambda_3 + -1\lambda_4 + 1\lambda_5$
$1 \leq 1\lambda_1 + 1\lambda_2 + -1\lambda_3 + 1\lambda_4 + 1\lambda_5$
$c_1 = -1c_5 \land \lambda_1 \geq 0 \land \lambda_2 \geq 0 \land \lambda_3 \geq 0$
$c_2 = -1c_4 \land \lambda_4 \geq 0 \land \lambda_5 \geq 0$

Farkas’ lemma. $R \Rightarrow \psi$ iff there exist real multipliers $\lambda_1, \ldots, \lambda_5 \geq 0$ such that

$c_1 = \sum_{i=1}^{5} \lambda_i a_{i,1} \land \ldots \land c_4 = \sum_{i=1}^{5} \lambda_i a_{i,4} \land 1 \leq (\sum_{i=0}^{5} \lambda_i b_i)$

Rank function synthesis

Instead solve: $\exists c_1, c_2, c_3, c_4, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$

$c_1 = 0\lambda_1 + 1\lambda_2 + -1\lambda_3 + 0\lambda_4 + 0\lambda_5$
$c_2 = 0\lambda_1 + 0\lambda_2 + 0\lambda_3 + 0\lambda_4 + -1\lambda_5$
$c_3 = -1\lambda_1 + -1\lambda_2 + 1\lambda_3 + 0\lambda_4 + 0\lambda_5$
$c_4 = 0\lambda_1 + 0\lambda_2 + 0\lambda_3 + -1\lambda_4 + 1\lambda_5$
$1 \leq 1\lambda_1 + 1\lambda_2 + -1\lambda_3 + 1\lambda_4 + 1\lambda_5$
$c_1 = -1c_5 \land \lambda_1 \geq 0 \land \lambda_2 \geq 0 \land \lambda_3 \geq 0$
$c_2 = -1c_4 \land \lambda_4 \geq 0 \land \lambda_5 \geq 0$

Solver: Dual Simplex for Th(LRA).

See Byron Cook’s blog for an F# program that produces input to Z3
Program Analysis as Constraint Solving

Sumit Gulwani, Saurabh Srivastava, Ramarathnam Venkatesan,
PLDI 2008

Loop invariants

How to find loop invariant $I$?
**Loop invariants**

\[ \Theta(x) \Rightarrow I(x) \]
\[ \exists I \forall x \left( I(x) \land c(x) \land S(x, x') \Rightarrow I(x') \right) \]
\[ \neg c(x) \land I(x) \Rightarrow Post(x) \]

- Assume \( I \) is of the form \( \sum_j a_j x_j \leq b \)

- Simplified problem: \( \exists A, b \forall x \phi_1(\lambda x. Ax \leq b, x) \)

---

**Loop invariants \( \Rightarrow \) Existential**

- Original: \( \exists I \forall x \phi_1(I, x) \)

- Relaxed: \( \exists A, b \forall x \phi_1(\lambda x. Ax \leq b, x) \)

- Farkas':
  \[ \forall x (Ax \leq 0 \Rightarrow bx \leq 0) \]
  \[ \Leftrightarrow \exists \lambda, \lambda_1, ..., \lambda_m (b = \lambda + \sum \lambda_k a_k) \]

- Existential:
  Problem: contains multiplication
  \( \exists A, b, \lambda \phi_2(A, b, \lambda) \)
Loop invariants $\Rightarrow$ SMT solving

- Original: $\exists I \forall x \varphi_1(I, x)$
- Existential: $\exists A, b \exists \lambda \varphi_2(A, b, \lambda)$
- Bounded: $\exists A, b, p_1, p_2, p_3 \varphi_2(A, b, \left[\begin{array}{c}
ite(p_1, 4, 0) + \\
ite(p_2, 2, 0) + \\
\ite(p_3, 1, 0)
\end{array}\right])$
- Or: Bit-vectors: $\exists A, b, \lambda : \text{BitVec}[8].\varphi_2(A, b, \lambda)$

Program Verification: Example

(n=1 \land m=1) \quad x := 0; y := 0; \quad \text{while} (x < 100) \quad \{y \geq 100\}

\text{Invariant Template} \quad \text{Satisfying Solution} \quad \text{Loop Invariant}

\begin{align*}
a_0 + a_1 x + a_2 y + a_3 n + a_4 m &\geq 0 \\
b_0 + b_1 x + b_2 y + b_3 n + b_4 m &\geq 0 \\
c_0 + c_1 x + c_2 y + c_3 n + c_4 m &\geq 0
\end{align*}

\begin{align*}
a_2 = b_3 = c_4 = 1, a_1 = b_2 = c_0 = -1 &\quad \Rightarrow \quad y \geq x \\
m \geq 1 \\
n \geq 1
\end{align*}

\begin{align*}
a_0 + a_1 x + a_2 y + a_3 n + a_4 m &\geq 0 \\
b_0 + b_1 x + b_2 y + b_3 n + b_4 m &\geq 0
\end{align*}

\begin{align*}
a_2 = b_2 = 1, a_1 = b_1 = -1 &\quad \Rightarrow \quad y \geq x \\
m \geq n
\end{align*}

Invalid triple or imprecise template
Bit-vector multiplication

- For each sub-term A*B
  - Replace by fresh vector OUT
  - Create circuit for:
    OUT = A*B
  - Convert circuit into clauses:
    For each internal gate
    - Create fresh propositional variable
    - Represent gate as clause

{Out[0], ~A[0], ~B[0]}, {A[0], ~Out[0]}, {B[0], ~Out[0]}, ...

Tableau + DPLL = Relevancy Propagation

- Tableau goes outside in, DPLL inside out
- Relevancy propagation: If DPLL sets $\theta \land \psi \lor \phi$ to true, $\theta$ is marked as relevant, then first of $\psi, \phi$ to be set to true gets marked as relevant.
- Used for circuit gates and for quantifier matching
Abstract Interpretation and modular arithmetic

Material based on:
King & Søndergård, CAV 08
Muller-Olm & Seidl, ESOP 2005


Programs as transition systems

Transition system:

\[\langle L, V, S = [V \rightarrow Val], R \subseteq L \times S \times S \times L, \Theta \subseteq S, \ell_{\text{init}} \in L \rangle\]

- \(L\): locations
- \(V\): variables
- \(S\): states
- \(R\): transitions
- \(\Theta\): initial states
- \(\ell_{\text{init}}\): initial location
Concrete reachable states:

\[ \text{CR} : L \to \wp(S) \]

Abstract reachable states:

\[ \text{AR} : L \to A \]

Connections:

\[
\begin{align*}
\sqcup : & \ A \times A \to A \\
\gamma : & \ A \to \wp(S) \\
\alpha : & \ S \to A \\
\alpha : & \ \wp(S) \to A \quad \text{where } \alpha(S) = \sqcup \{ \alpha(s) \mid s \in S \} 
\end{align*}
\]

Concrete reachable states:

\[
\begin{align*}
\text{CR } \ell x & \leftarrow \Theta x \land \ell = \ell_{\text{init}} \\
\text{CR } \ell x & \leftarrow \text{CR } \ell_0 x_0 \land R \ell_0 x_0 \ell
\end{align*}
\]

Abstract reachable states:

\[
\begin{align*}
\text{AR } \ell x & \leftarrow \alpha(\Theta(x)) \land \ell = \ell_{\text{init}} \\
\text{AR } \ell x & \leftarrow \alpha(\gamma(\text{AR } \ell_0 x_0) \land R \ell_0 x_0 \ell)
\end{align*}
\]

Why? fewer (finite) abstract states
Abstraction using SMT

Abstract reachable states:

$$\text{AR } l_{init} \leftarrow \alpha(\Theta)$$

Find interpretation $$M$$:

$$M \models \gamma(AR \ l_0 x_0) \land R \ l_0 x_0 \times l \land \neg \gamma(AR \ l x)$$

Then:

$$\text{AR } l \leftarrow \text{AR } l \sqcup \alpha(x^M)$$

Abstraction: Linear congruences

- States are linear congruences:
  $$A \ V = b \mod 2^m$$

- $$V$$ is set of program variables.
- $$A$$ matrix, $$b$$ vector of coefficients [0.. $$2^m-1$$]
Example

\[ \ell_0: y \leftarrow x; \ c \leftarrow 0; \]
\[ \ell_1: \text{while } y \neq 0 \text{ do } [ \ y \leftarrow y \& (y-1); \ c \leftarrow c+1 \ ] \]
\[ \ell_2: \]

- When at \( \ell_2 \):
  - \( y \) is 0.
  - \( c \) contains number of bits in \( x \).

Abstraction: Linear congruences

- States are linear congruences:
  \[
  \gamma \begin{bmatrix} 2 & 3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \mod 2^3 \Leftrightarrow \\
  2x_0 + 3x_1 = 1 \mod 2^3 \land x_0 + x_1 = 3 \mod 2^3 \Leftrightarrow \\
  \]

As Bit-vector constraints (SMTish syntax):

\[
\begin{align*}
\text{(and} & (\text{bvadd (bvmul 010 } x_0) (\text{bvmul 011 } x_1)) \text{ 001}) \\
\text{(and} & (\text{bvadd } x_0 \ x_1) \text{ 011})
\end{align*}
\]
**Abstraction: Linear congruences**

\[ \alpha(x=1, y=2) \Delta \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \]

\((A \lor V = b \mod 2^m) \sqcup (A' \lor V = b' \mod 2^m)\)

Combine:

\[
\begin{bmatrix}
1 & 1 & 0 & 0 & 0 \\
-b & 0 & A & 0 & 0 \\
0 & -b' & 0 & A' & 0 \\
0 & 0 & -I & -I & I
\end{bmatrix}
\begin{bmatrix}
 s_1 \\
 s_2 \\
 x_1 \\
 x_2 \\
x
\end{bmatrix}
= \begin{bmatrix}
1 \\
0 \\
0 \\
0
\end{bmatrix}
\]

Triangulate (Muller-Olm & Seidl)
Project on \(x\)

---

**Bounded Model Checking of Model Programs**

Margus Veanes

FORTE 08
Integration with symbolic analysis techniques at design time – smart model debugging

- Theorem proving
- Model checking
- Compositional reasoning
- Domain specific front ends
- Different subareas require different adaptations
- Model programs provide the common framework

**Motivating example**

- SMB2 Protocol Specification
- Sweet spot for model-based testing and verification.

---

**Symbolic Reachability**

**Model Program**

```plaintext
val shorthand = List of Integer = [1];
val mand = List of Integer = [0];
val required as List of Integer to Integer = [0-0]

<table>
<thead>
<tr>
<th>Endline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success as List of Integer, as List of Integer</td>
</tr>
<tr>
<td>returns in succeeds and i &lt;= 0</td>
</tr>
<tr>
<td>requires i requests. Add y to y</td>
</tr>
<tr>
<td>window = window + 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUCCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success as List of Integer, as List of Integer</td>
</tr>
<tr>
<td>succeeds in succeeds</td>
</tr>
<tr>
<td>requires i requests() + y</td>
</tr>
<tr>
<td>returns i &lt;= 3</td>
</tr>
<tr>
<td>window = window + 0</td>
</tr>
</tbody>
</table>

(max) 0

Compress(0,0)/Compress(1,0)

- require succeeds = 1 |
- initials window = 0 |
- `z3`

**Invariant Checker**

**Counter example**

---

© 2007 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
Bounded-reachability formula

Given a model program $P$ step bound $k$ and reachability condition $\varphi$

$$\text{Reach}(P, \varphi, k) \overset{\text{def}}{=} I_P \land (\bigwedge_{0 \leq i < k} P[i]) \land (\bigvee_{0 \leq i \leq k} \varphi[i])$$

$$P[i] \overset{\text{def}}{=} \bigvee_{f \in A_P} \left( \text{action}[i] = f(f_1[i], \ldots, f_n[i]) \land G^f_P[i] \land \bigwedge_{v \in V_f^P} v[i + 1] = t_v^f[i] \land v[i + 1] = v[i] \right)$$

Array model programs and quantifier elimination

- **Array model programs** use only maps with integer domain sort.
- For normalizable comprehensions universal quantifiers can be eliminated using a decision procedure for the **array property fragment** [Bradley et. al, VMCAI 06]
Implementation using the SMT solver Z3

- Set comprehensions are introduced through skolem constant definitions using support for quantifiers in Z3
- Elimination of quantifiers is partial.
- Model is refined if a spurious model is found by Z3.
- A spurious model may be generated by Z3 if an incomplete heuristic is used during quantifier elimination.

A different example: Adaptive Planning with Finite Horizon Lookahead

Model program:

```plaintext
// Model program of walking in a grid until reaching goal
var x as Integer
var y as Integer
var xGoal as Integer
var yGoal as Integer
var xmax as Integer
var ymax as Integer
var yBlocks as Map of Integer to Set of Integer
var xBlocks as Map of Integer to Set of Integer

[Action] (up)
    require y < ymax and not (y in yBlocks(x))
    and not (x = xGoal and y = yGoal)
    y = y + 1

[Action] (down)
    require y > 0 and not (y-1 in yBlocks(x))
    and not (x = xGoal and y = yGoal)
    y = y - 1

[Action] (right)
    require x < xmax and not (x in xBlocks(y))
    and not (x = xGoal and y = yGoal)
    x = x + 1

[Action] (left)
    require x > 0 and not (x-1 in xBlocks(y))
    and not (x = xGoal and y = yGoal)
    x = x - 1
```
Verifying Garbage Collectors  
- Automatically and fast


**Context**

**Singularity**
- Safe micro-kernel  
  - 95% written in C#  
  - all services and drivers in processes  
- Software isolated processes (SIPs)  
  - all user code is verifiably safe  
  - some unsafe code in trusted runtime  
  - processes and kernel sealed at execution  
- Communication via channels  
  - channel behavior is specified and checked  
  - fast and efficient communication  
- Working research prototype  
  - not Windows replacement  
  - shared source download

**Bartok**
- MSIL → X86 Compiler

**BoogiePL**
- Procedural low-level language  
- Contracts  
- Verification condition generator

**Garbage Collectors**
- Mark&Sweep  
- Copying GC  
- Verify small garbage collectors  
  - more automated than interactive provers  
  - borrow ideas from type systems for regions

---

8/19/2008
Goal: safely run untrusted code

- MSIL
  - compiler
  - typed x86
  - I/O
  - exception handling
  - garbage collector
  - linker, loader
  - safety verifier

Mark-sweep and copying collectors

- abstract graph
  - A (root)
  - B
  - C

- mark-sweep
  - A
  - B
  - C

- copying from
  - A
  - B
  - C

- copying to
  - A
  - B
  - C

MSIL: MSFT Intermediary Language

trusted computing base
(minimize this!)

untrusted code

8/19/2008
Garbage collector properties

- safety: gc does no harm
  - type safety
    - gc turns well-typed heap into well-typed heap
  - graph isomorphism
    - concrete graph represents abstract graph
- effectiveness
  - after gc, unreachable objects reclaimed
- termination
- efficiency

Proving safety

procedure GarbageCollectMs()
requires MsMutatorInv(root, Color, $toAbs, $AbsMem, Mem);
modifies Mem, Color, $toAbs;
ensures
  function MsMutatorInv(...) returns (bool) {
    WellFormed($toAbs) && memAddr(root) && $toAbs[root] != NO_ABS
    && (forall i:int {memAddr(i)} memAddr(i) ==> ObjInv(i, $toAbs, $AbsMem, Mem))
    && (forall i:int {memAddr(i)} memAddr(i) ==> White(Color[i]))
    && (forall i:int {memAddr(i)} memAddr(i) ==> ($toAbs[i] == NO_ABS <==>
      UnAlloc(Color[i])))
  }

function ObjInv(...) returns (bool) { memAddr(i) && $toAbs[i] != NO_ABS ==>
  ... $toAbs[Mem[i, field1]] != NO_ABS ... $toAbs[Mem[i, field1]] == $AbsMem[$toAbs[i], field1] ... }
Controlling quantifier instantiation

- Idea: use marker

```plaintext
function T(i:int) returns (bool) { true }
```

- Relativize quantifiers using marker

```plaintext
function GcInv(Color:[int]int, $stoAbs:[int]int, $AbsMem:[int,int]int, Mem:[int,int]int) returns (bool) {
  WellFormed($stoAbs)
  && (forall i:int::(T(i)) T(i) ==> memAddr(i) ==> ObjInv(i, $stoAbs, $AbsMem, Mem)
  && 0 <= Color[i] && Color[i] < 4
  && (Black(Color[i]) ==> !White(Color[absMem[i,0]]) && !White(Color[absMem[i,1]]))
  && ($stoAbs[i] == NO_ABS <= Unalloc(Color[i]))
}
```

Controlling quantifier instantiation

- Insert markers to enable triggers

```plaintext
procedure Mark(ptr:int)
  requires GcInv(Color, $stoAbs, $AbsMem, Mem);
  requires memAddr(ptr) && T(ptr);
  requires $stoAbs[ptr] != NO_ABS;
  modifies Color;
  ensures GcInv(Color, $stoAbs, $AbsMem, Mem);
  ensures (forall i:int::(T(i)) T(i) ==> !Black(Color[i]) ==> Color[i] == old(Color[i]));
  ensures !White(Color[ptr]);
  {
    if (White(Color[ptr])) {
      Color[ptr] := 2; // make gray
      call Mark(Mem[ptr,0]);
      call Mark(Mem[ptr,1]);
      Color[ptr] := 3; // make black
    }
  }
```
Refinement Types for Secure Implementations

http://research.microsoft.com/F7

Jesper Bengtson,
Karthikeyan Bhargavan,
Cédric Fournet,
Andrew D. Gordon,
Sergio Maffeis
CSF 2008

Executable code has more details than models

Executable code has better tool support: types, compilers, testing, debuggers, libraries, verification

Using dependent types: integrate cryptographic protocol verification as a part of program verification

Such predicates can also represent security-related concepts like roles, permissions, events, compromises, access rights,...
Example: access control for files

- **Un-trusted code** may call a trusted library
- **Trusted code** expresses security policy with assumes and asserts
- Each policy violation causes an assertion failure
- F7 statically prevents any assertion failures by typing

```hs
type facts = CanRead of string | CanWrite of string

let read file = assert(CanRead(file)); ...
let delete file = assert(CanWrite(file)); ...

let pwd = “C:/etc/passwd"
let tmp = “C:/temp/temp"

assume CanWrite(tmp)
assume \forall x . CanWrite(x) \rightarrow CanRead(x)

let untrusted() =
    let v1 = read tmp in // ok
    let v2 = read pwd in //CanRead(pwd)
    // assertion fails
```

Access control with refinement types

```hs
val read: file:string{CanRead(file)} \rightarrow string
val delete: file:string{CanDelete(file)} \rightarrow unit
val publish: file:string \rightarrow unit{Public(file)}
```

- Pre-conditions express access control requirements
- Post-conditions express results of validation
- F7 type checks partially trusted code to guarantee that all preconditions (and hence all asserts) hold at runtime
Models for Domain Specific Languages with FORMULA & BAM

Ethan Jackson

FORTE 08

Designing Complex Systems Requires Multiple Abstractions

Automotive system is just processors and their communication buses

Forget about the network; think about the software components

Functional architecture taken from AUTOSAR: http://www.autosar.org

More Abstract

Product lines abstract across families of implementations

Screenshot of "Build Your Scion": http://www.scion.org

BMW architecture:

Taken from A General Synthesis Approach for Embedded Systems Design with Applications to Multi-media and Automotive Designs, Sangiovanni-Vincetelli et al., 2007.
Many Modeling Styles are Used to Build Abstractions

Abstraction: ECU/Bus
Style: Domain-specific Language

Abstraction: Scheduling Problem
Style: Platform-based design

Abstraction: Automotive Product-line
Style: Feature Diagram

A Notorious Problem: How Do We Compose Abstractions?
We view each abstraction as providing (among other things) a constraint system representing the legal "models" of the abstraction. Composition occurs via these constraint systems:

For example, this instance must satisfy the constraints of each abstraction used in its construction.
FORMULA is a CLP Language for Specifying, Composing, and Analyzing Abstractions

A domain encapsulates a reusable, composable constraint system

Special function symbols (malform, wellform) capture legal instances in a domain-independent way.

FORMULA can construct satisfying instances to logic program queries using Z3.

Search for satisfying instances are Reduced to Z3

This model finding procedure allows us to:

1. Determine if a composition of abstractions contains inconsistencies
2. Construct (partial) architectures that satisfy many domain constraints.
3. Generate design spaces of architectural invariants.

Reduction to Z3 works as follows:

Symbolic backwards chaining yields a set of candidate terms $S$ with the following property:

A finite instance exists that satisfies the query $Q$ iff some subset of $S$ satisfies the query $Q$.

Once the finite set $S$ is calculated, then $S + Q$ is reduced to SMT and evaluated by Z3.
Selected Background on SMT

Basics

Pre-requisites and notation
Language of logic - summary

- Functions, Variables, Predicates
  - \( f, g, \ x, y, z, \ P, Q, = \)
- Atomic formulas, Literals
  - \( P(x,f(y)), \neg Q(y,z) \)
- Quantifier free formulas
  - \( P(f(a), b) \land c = g(d) \)
- Formulas, sentences
  - \( \forall x . \forall y . [ P(x, f(x)) \lor g(y,x) = h(y) ] \)

Language: Signatures

- A signature \( \Sigma \) is a finite set of:
  - Function symbols:
    \( \Sigma_F = \{ f, g, \ldots \} \)
  - Predicate symbols:
    \( \Sigma_P = \{ P, Q, =, \text{true, false, } \ldots \} \)
  - And an arity function:
    \( \Sigma \rightarrow \mathbb{N} \)
- Function symbols with arity 0 are constants
- A countable set \( V \) of variables
- disjoint from \( \Sigma \)
Language: Terms

The set of terms $T(\Sigma_F, V)$ is the smallest set formed by the syntax rules:

- $t \in T ::= v$ \quad $v \in V$
  - $f(t_1, ..., t_n)$ \quad $f \in \Sigma_F$ \quad $t_1, ..., t_n \in T$

- Ground terms are given by $T(\Sigma_F, \emptyset)$

Language: Atomic Formulas

- $a \in Atoms ::= P(t_1, ..., t_n)$
  - $P \in \Sigma_P$ \quad $t_1, ..., t_n \in T$

An atom is ground if $t_1, ..., t_n \in T(\Sigma_F, \emptyset)$

Literals are (negated) atoms:

- $l \in Literals ::= a \mid \neg a$ \quad $a \in Atoms$
Language: Quantifier free formulas

The set $QFF(\Sigma, V)$ of quantifier free formulas is the smallest set such that:

$$
\phi \in QFF \quad ::= \quad a \in Atoms \quad \text{atoms} \\
\quad | \quad \neg \phi \quad \text{negations} \\
\quad | \quad \phi \leftrightarrow \phi' \quad \text{bi-implications} \\
\quad | \quad \phi \land \phi' \quad \text{conjunction} \\
\quad | \quad \phi \lor \phi' \quad \text{disjunction} \\
\quad | \quad \phi \rightarrow \phi' \quad \text{implication}
$$

Language: Formulas

The set of first-order formulas are obtained by adding the formation rules:

$$
\phi ::= \ldots \\
\quad | \quad \forall x . \phi \quad \text{universal quant.} \\
\quad | \quad \exists x . \phi \quad \text{existential quant.}
$$

- Free (occurrences) of variables in a formula are those not bound by a quantifier.
- A sentence is a first-order formula with no free variables.
A (first-order) theory $T$ (over signature $\Sigma$) is a set of (deductively closed) sentences (over $\Sigma$ and $V$)

Let $DC(\Gamma)$ be the deductive closure of a set of sentences $\Gamma$.
- For every theory $T$, $DC(T) = T$

A theory $T$ is consistent if $\text{false} \notin T$

We can view a (first-order) theory $T$ as the class of all models of $T$ (due to completeness of first-order logic).

A model $M$ is defined as:
- Domain $S$; set of elements.
- Interpretation, $f^M : S^n \rightarrow S$ for each $f \in \Sigma_F$ with $\text{arity}(f) = n$
- Interpretation $P^M \subseteq S^n$ for each $P \in \Sigma_P$ with $\text{arity}(P) = n$
- Assignment $x^M \in S$ for every variable $x \in V$

A formula $\varphi$ is true in a model $M$ if it evaluates to true under the given interpretations over the domain $S$.

$M$ is a model for the theory $T$ if all sentences of $T$ are true in $M$. 
T-Satisfiability

A formula \( \varphi(x) \) is T-satisfiable in a theory \( T \) if there is a model of \( DC(T \cup \exists x \, \varphi(x)) \). That is, there is a model \( M \) for \( T \) in which \( \varphi(x) \) evaluates to true.

Notation:

\[ M \vDash_T \varphi(x) \]

T-Validity

A formula \( \varphi(x) \) is T-valid in a theory \( T \) if \( \forall x \, \varphi(x) \in T \). That is, \( \varphi(x) \) evaluates to true in every model \( M \) of \( T \).

T-validity:

\[ \models_T \varphi(x) \]
Checking the validity of $\varphi$ in a theory $T$:

$\varphi$ is $T$-valid

$\equiv T$-unsat: $\neg \varphi$

$\equiv T$-unsat: $\forall x \exists y \forall z \exists u. \phi$ (prenex of $\neg \varphi$)

$\equiv T$-unsat: $\forall x \forall z. \phi[f(x),g(x,z)]$ (skolemize)

$\Leftarrow T$-unsat: $\phi[f(a_1),g(a_1,b_1)] \land ...$ (instantiate)

$\land \phi[f(a_n),g(a_n,b_n)]$ (⇒ if compactness)

$\equiv T$-unsat: $\phi_1 \lor ... \lor \phi_m$ (DNF)

where each $\phi_i$ is a conjunction.

Checking Validity – the morale

Theory solvers must minimally be able to

check unsatisfiability of conjunctions of literals.
We want to only work with formulas in Conjunctive Normal Form CNF.

\[ \varphi : x = 5 \iff (y < 3 \lor z = x) \] is not in CNF.

\[ \varphi' : (\neg p \lor x = 5) \land (p \lor \neg x = 5) \land \\
(\neg p \lor y < 3 \lor z = x) \land \\
(p \lor \neg y < 3) \land (p \lor \neg z = x) \]

Equi-satisfiable CNF formula
Clauses – CNF conversion

\[
\text{cnf}(\varphi) = \text{let } (q,F) = \text{cnf'}(\varphi) \text{ in } q \land F
\]

\[
\text{cnf}'(a) = (a, \text{true})
\]

\[
\text{cnf}'(\varphi \land \varphi') = \text{let } (q,F_1) = \text{cnf}'(\varphi)
\]
\[
(r, F_2) = \text{cnf}'(\varphi')
\]
\[
p = \text{fresh Boolean variable}
\]
\[
in
\]
\[
(p, F_1 \land F_2 \land (\neg p \lor q) \land (\neg p \lor r) \land (\neg p \lor \neg q \lor \neg r))
\]

Exercise: \(\text{cnf}'(\varphi \lor \varphi'), \text{cnf}'(\varphi \leftrightarrow \varphi'), \text{cnf}'(\neg \varphi)\)

---

Clauses – CNF

**Main properties of basic CNF**

- Result \(F\) is a set of *clauses*.

- \(\varphi\) is \(T\)-satisfiable iff \(\text{cnf}(\varphi)\) is.

- \(
\text{size}(\text{cnf}(\varphi)) \leq 4(\text{size}(\varphi))
\)

- \(\varphi \leftrightarrow \exists p_{aux} \text{cnf}(\varphi)\)
Incrementally build a model $M$ for a CNF formula $F$ (set of clauses).

Initially $M$ is the empty assignment.

**Propagate**: $M$: $M(r) \leftarrow \text{false}$
- if $(p \lor \neg q \lor \neg r) \in F$, $M(p) = \text{false}$, $M(q) = \text{true}$

**Decide** $M(p) \leftarrow \text{true}$ or $M(p) \leftarrow \text{false}$,
- if $p$ is not assigned.

**Backtrack**:
- if $(p \lor \neg q \lor \neg r) \in F$, $M(p) = \text{false}$, $M(q) = M(r) = \text{true}$, (e.g. $M \models T \rightarrow \neg \square$)
Modern DPLL – as transitions

- **Maintain states of the form:**
  - $M || F$ - during search
  - $M || F || C$ - for backjumping
  - $M$ a partial model, $F$ are clauses, $C$ is a clause.

- **Decide** $M || F \Rightarrow M^d || F$ if $l \in F \setminus M$
  - $d$ is a decision marker

- **Propagate** $M || F \Rightarrow M^C || F$
  - if $l \in C \in F$, $C = (C' \lor l)$, $M \models_T \neg C'$

- **Conflict** $M || F \Rightarrow M || F || C$ if $C \in F$, $M \models_T \neg C$

- **Learn** $M || F || C \Rightarrow M || F, C || C$ i.e., add $C$ to $F$

- **Resolve** $M p^{(C' \lor p)} || F || C \lor \neg p \Rightarrow M || F || C \lor C'$

- **Skip** $M p || F || C \Rightarrow M || F || C$ if $\neg l \not\in C$

- **Backjump** $M M' || F || C \Rightarrow M \neg l^C || F$
  - if $\neg l \not\in C$ and $M'$ does not intersect with $\neg C$
Congruence closure just checks satisfiability of conjunction of literals.

How does this fit together with Boolean search DPLL?

DPLL builds partial model $M$ incrementally
- Use $M$ to build $C^*$
  - After every Decision or Propagate, or
  - When $F$ is propositionally satisfied by $M$.
- Check that disequalities are satisfied.
Recall **Conflict**: 

- **Conflict** \( M \parallel F \implies M \parallel F \parallel C \text{ if } C \in F, M \models T \neg C \)

A version more useful for theories:

- **Conflict** \( M \parallel F \implies M \parallel F \parallel C \text{ if } C \subseteq \neg M, \models T C \)

Example

- \( M = fff(a) = a, g(b) = c, fffff(a) = a, a \neq f(a) \)
- \( \rightarrow C = fff(a) = a, fffff(a) = a, a \neq f(a) \)
- \( \models_E fff(a) \neq a \lor fffff(a) \neq a \lor a = f(a) \)

- Use \( C \) as a conflict clause.
Fourier-Motzkin:
- Quantifier elimination procedure
  \[ \exists x \ (t \leq ax \land t' \leq bx \land cx \leq t'') \Leftrightarrow ct \leq at' \land ct' \leq bt'' \]
- Polynomial for difference logic.
- Generally: exponential space, doubly exponential time.

Simplex:
- Worst-case exponential, but
- Time-tried practical efficiency.
- Linear space
Combining Theory Solvers

Nelson-Oppen procedure

**Initial state:** \( L \) is set of literals over \( \Sigma_1 \cup \Sigma_2 \)

**Purify:** Preserving satisfiability, convert \( L \) into \( L' = L_1 \cup L_2 \) such that
\[
L_1 \in T(\Sigma_1, V), \quad L_2 \in T(\Sigma_2, V)
\]
So \( L_1 \cap L_2 = V_{\text{shared}} \subseteq V \)

**Interaction:**
Guess a partition of \( V_{\text{shared}} \)
Express the partition as a conjunction of equalities.
Example, \{ \( x_1 \) \}, \{ \( x_2, x_3 \) \}, \{ \( x_4 \) \} is represented as:
\[
\psi: x_1 \neq x_2 \wedge x_1 \neq x_4 \wedge x_2 \neq x_4 \wedge x_2 = x_3
\]

**Component Procedures:**
Use solver 1 to check satisfiability of \( L_1 \wedge \psi \)
Use solver 2 to check satisfiability of \( L_2 \wedge \psi \)
NO – reduced guessing

- Instead of guessing, we can often deduce the equalities to be shared.

- **Interaction:** \( T_1 \land L_1 \models x = y \) then add equality to \( \psi \).

- If theories are convex, then we can:
  - Deduce all equalities.
  - Assume every thing not deduced is distinct.
  - Complexity: \( O(n^4 \times T_1(n) \times T_2(n)) \).

---

Model-based combination

- Reduced guessing is only complete for convex theories.

- Deducing all implied equalities may be expensive.
  - Example: Simplex – no direct way to extract from just bounds and \( \beta \)

- **But:** backtracking is pretty cheap nowadays:
  - If \( \beta(x) = \beta(y) \), then \( x, y \) are equal in arithmetical component.
Model-based combination

- Backjumping is cheap with modern DPLL:
  - If $\beta(x) = \beta(y)$, then $x$, $y$ are equal in arithmetical model.
  - So let's add $x = y$ to $\psi$, but allow to backtrack from guess.

- In general: if $M_1$ is the current model
  - $M_1 \models x = y$ then add literal $(x = y)^d$
Theory of arrays

- Functions: $\Sigma_F = \{ \text{read}, \text{write} \}$
- Predicates: $\Sigma_P = \{ = \}$
- Convention $a[i]$ means: $\text{read}(a,i)$

Non-extensional arrays $T_A$:
- $\forall a, i, v . \text{write}(a,i,v)[i] = v$
- $\forall a, i, j, v . i \neq j \Rightarrow \text{write}(a,i,v)[j] = a[j]$

Extensional arrays: $T_{EA} = T_A +$
- $\forall a, b. ((\forall i. a[i] = b[i]) \Rightarrow a = b)$

Decision procedures for arrays

- Let $L$ be literals over $\Sigma_F = \{ \text{read}, \text{write} \}$
- Find $M$ such that: $M \models_{T_A} L$

Basic algorithm, reduce to $E$:
- for every sub-term $\text{read}(a,i), \text{write}(b,j,v)$ in $L$
  - $i \neq j \land a = b \Rightarrow \text{read}(\text{write}(b,j,v),i) = \text{read}(a,i)$
  - $\text{read}(\text{write}(b,j,v),j) = v$
- Find $M_E$, such that $M_E \models_E L \land \text{AssertedAxioms}$

© 2007 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
We can use DPLL(T) for $\varphi$ with quantifiers.

- Treat quantified sub-formulas as atomic predicates.

- In other words, if $\forall x.\psi(x)$ is a sub-formula if $\varphi$, then introduce fresh $p$. Solve instead

$$\varphi[\forall x.\psi(x) \leftarrow p]$$
Suppose DPLL(T) sets \( p \) to false

- \( \Rightarrow \) any model \( M \) for \( \varphi \) must satisfy:
  \[
  M \vDash \neg \forall x.\psi(x)
  \]

- \( \Rightarrow \) for some \( sk_x \):
  \[
  M \vDash \neg \psi(sk_x)
  \]

- In general:
  \[
  \vDash \neg p \rightarrow \neg \psi(sk_x)
  \]

Suppose DPLL(T) sets \( p \) to true

- \( \Rightarrow \) any model \( M \) for \( \varphi \) must satisfy:
  \[
  M \vDash \forall x.\psi(x)
  \]

- \( \Rightarrow \) for every term \( t \):
  \[
  M \vDash \psi(t)
  \]

- In general:
  \[
  \vDash p \rightarrow \psi(t)
  \]

For every term \( t \).
Summary of auxiliary axioms:

- \( \models \neg p \rightarrow \neg \psi(\text{sk}_x) \)  
  For fixed, fresh \( \text{sk}_x \)

- \( \models p \rightarrow \psi(t) \)  
  For every term \( t \).

Which terms \( t \) to use for auxiliary axioms of the second kind?

\[ \models p \rightarrow \psi(t) \]  For every term \( t \).

Approach:
- Add patterns to quantifiers
- Search for instantiations in \( E \)-graph.

\[ \forall a,i,v \ (\text{write}(a,i,v)) . \ \text{read}(\text{write}(a,i,v),i) = v \]
DPLL(QT) with E-matching

- \( p \rightarrow \psi(t) \) For every term \( t \).

**Approach:**
- Add patterns to quantifiers
- Search for pattern matches in \( E \)-graph.

\[ \forall a,i,v \{ \text{write}(a,i,v) \} . \text{read}(\text{write}(a,i,v),i) = v \]

- Add equality every time there is a write\((b,j,w)\) term in \( E \).

---

**Z3 - An Efficient SMT Solver**
Main features

- Linear real and integer arithmetic.
- Fixed-size bit-vectors
- Uninterpreted functions
- Extensional arrays
- Quantifiers
- Model generation
- Several input formats (Simplify, SMT-LIB, Z3, Dimacs)
- Extensive API (C/C++, .Net, OCaml)
Example: C API

```
for (n = 2; n <= 5; n++) {
    printf("n = %d\n", n);
    ctx = z3_mk_context(cfg);

    bool_type = z3_mk_bool_type(ctx);
    array_type = z3_mk_array_type(ctx, bool_type, bool_type);

    /* create arrays */
    for (i = 0; i < n; i++) {
        z3_symbol s = z3_mk_int_symbol(ctx, i);
        a[i] = z3_mk_const(ctx, s, array_type);
    }

    /* assert distinct (a[0], ..., a[n]) */
    d = z3_mk_distinct(ctx, n, s);
    printf("\n", z3_sat_to_string(ctx, d));
    z3_assert_constr(ctx, d);

    /* context is satisfiable if n < 5 */
    if (z3_check(ctx) == 1_false)
        printf("unsatisfiable, n: %d\n", n);
    z3_del_context(ctx);
}
```

Given arrays:

```c
bool a1[bool];
bool a2[bool];
bool a3[bool];
bool a4[bool];
```

All can be distinct.

Add:

```c
bool a5[bool];
```

Two of a1,...,a5 must be equal.

Example: SMT-LIB

```smtlib
(benchmark integer-linear-arithmetic)
:status sat
:logic QF_LIA
:extrafuns ((x1 Int) (x2 Int) (x3 Int) (x4 Int) (x5 Int))
:formula (and (+ (* -1 x1) x2) 1) (+ (* -1 x1) x2) 3) (+ (* 2 x3) x5)) (+ (* 6 x4))))
```

```smtlib
(benchmark array)
:logic QF_AUFLIA
:status unsat
:extrafuns ((a Array) (b Array) (c Array))
:extrafuns ((i Int) (j Int))
:formula (and (= (store a i v) b) (= (store a j w) c) (= (select b j) w) (= (select c i) v) (not (= b c)))
```
SMT-LIB syntax – basics

- **benchmark**: (benchmark name
  - [status (sat | unsat | unknown)]
  - logic logic-name
  - declaration*)

- **declaration**: :extrafuns (func-decl*)
  - :extrapreds (pred-decl*)
  - :extrasorts (sort-decl*)
  - :assumption fmla
  - :formula fmla

- **sort-decl**: ::= id - identifier
- **func-decl**: ::= id sort-decl* sort-decl - name of function, domain, range
- **pred-decl**: ::= id sort-decl* - name of predicate, domain
- **fmla**: ::= (and fmla* | (or fmla*) | (not fmla)
  - (if_then_else fmla fmla fmla) | (= term term)
  - (implies fmla fmla) (iff fmla fmla) | (predicate term*)

- **Term**: ::= (ite fmla term term)
  - (id term*) - function application
  - id - constant

SMT-LIB syntax - basics

- **Logics:**
  - QF_UF – Un-interpreted functions. Built-in sort U
  - QF_AUFLIA – Arrays and Integer linear arithmetic.
  - Built-in Sorts:
    - Int, Array (of Int to Int)
  - Built-in Predicates:
    - <=, >=, <, >,
  - Built-in Functions:
    - +, *, -, select, store.
  - Constants: 0, 1, 2, ...
Q: There is no built-in function for \textit{max} or \textit{min}. How do I encode it?

\begin{itemize}
  \item \((\max x y)\) is the same as \((\ite (> x y) x y)\)
  \item Also: replace \((\max x y)\) by fresh constant \(\max_x_y\) add assumptions:
    \begin{itemize}
      \item \textbf{assumption} \((\implies (> x y) (= \max_x_y x))\)
      \item \textbf{assumption} \((\implies (\leq x y) (= \max_x_y y))\)
    \end{itemize}
\end{itemize}

Q: Encode the predicate \((\text{even } n)\), that is true when \(n\) is even.

Quantifiers

Quantified formulas in SMT-LIB:

\begin{itemize}
  \item \textbf{fmla} ::= ... |
    \begin{itemize}
      \item \((\forall \text{bound} \ast \text{fmla})\)
      \item \((\exists \text{bound} \ast \text{fmla})\)
    \end{itemize}
  \item \textbf{Bound} ::= \((\text{id sort-id})\)
\end{itemize}

Q: I want \(f\) to be an injective function. Write an axiom that forces \(f\) to be injective.

Patterns: guiding the instantiation of quantifiers (Lecture 5)

\begin{itemize}
  \item \textbf{fmla} ::= ... |
    \begin{itemize}
      \item \((\forall (?x A) (?y B) \text{fmla} : \text{pat} \{ \text{term} \})\)
      \item \((\exists (?x A) (?y B) \text{fmla} : \text{pat} \{ \text{term} \})\)
    \end{itemize}
  \end{itemize}

Q: what are the patterns for the injectivity axiom?
Using the Z3 (managed) API

Create a context z3:

```
let par = new Config()
do par.SetParamValue("MODEL", "true")
let z3 = new TypeSafeContext(par)
```

Check a formula:

```
let check (fmla) =
z3.Push();
z3.AssertCnstr(fmla);
(match z3.Check() with
| LBool.False -> Printf.printf "unsat\n"
| LBool.True -> Printf.printf "sat\n"
| LBool.Undef -> Printf.printf "unknown\n"
| _ -> assert false);
z3.Pop(1ul)
```

Declaring z3 shortcuts, constants and functions:

```
let (==) x y = z3.MkEq(x,y)
let (==>) x y = z3.MkImplies(x,y)
let (&&) x y = z3.MkAnd(x,y)
let neg x = z3.MkNot(x)
let a = z3.MkType("a")
let f_decl = z3.MkFuncDecl("f",a,a)
let x = z3.MkConst("x",a)
let f x = z3.MkApp(f_decl,x)
```

Proving a theorem:

```
let fmla1 = ((x == f(f(f(f x)))) && (x == f(f x))) ==> (x == f x))
do check (neg fmla1)
```

(benchmark euf
:logic QF_UF
:extrafuns ((f U U) (x U))
:formula (not (implies (and (= x (f(f(f(f x)))))) (= x (f(f(f x)))))) (= x (f(f x))))

compared to

(benchmark euf.
:logic QF_UF
:extrafuns ((f U U) (x U))
:formula (not (implies (and (= x (f(f(f(f x)))))) (= x (f(f(f x)))))) (= x (f(f x))))

compared to

(benchmark euf
:logic QF_UF
:extrafuns ((f U U) (x U))
:formula (not (implies (and (= x (f(f(f(f x)))))) (= x (f(f(f x)))))) (= x (f(f x))))
We want to find models for

\[\begin{align*}
2 &< i_1 \leq 5 \land 1 &< i_2 \leq 7 \land -1 &< i_3 \leq 17 \land \\
0 &\leq i_1 + i_2 + i_3 \land i_2 + i_3 &= i_1
\end{align*}\]

But we only care about different \( i_1 \)

---

Representing the problem

```csharp
void Test() {
    Config par = new Config();
    par.SetParamValue("MODEL", "true");
    z3 = new TypeSafeContext(par);
    intT = z3.MkIntType();
    i1 = z3.MkConst("i1", intT);  i2 = z3.MkConst("i2", intT);
    i3 = z3.MkConst("i3", intT);
    z3.AssertCnstr(Num(2) < i1 & i1 <= Num(5));
    z3.AssertCnstr(Num(1) < i2 & i2 <= Num(7));
    z3.AssertCnstr(Num(-1) < i3 & i3 <= Num(17));
    z3.AssertCnstr(Num(0) <= i1 + i2 + i3 & Eq(i2 + i3, i1));
    Enumerate();
    par.Dispose();
    z3.Dispose();
}
```
**Enumerating models**

**Enumeration:**

```csharp
void Enumerate() {
    TypeSafeModel model = null;
    while (LBool.True == z3.CheckAndGetModel(ref model)) {
        model.Display(Console.Out);
        int v1 = model.GetNumeralValueInt(model.Eval(i1));
        TermAst block = Eq(Num(v1),i1);
        Console.WriteLine("Block {0}", block);
        z3.AssertCnstr(!block);
        model.Dispose();
    }
}

TermAst Eq(TermAst t1, TermAst t2) { return z3.MkEq(t1,t2); }

TermAst Num(int i) { return z3.MkNumeral(i, intT); }
```

**Push, Pop**

```csharp
int Maximize(TermAst a, int lo, int hi) {
    while (lo < hi) {
        int mid = (lo+hi)/2;
        Console.WriteLine("lo: {0}, hi: {1}, mid: {2}, lo, hi, mid");
        z3.Push();
        z3.AssertCnstr(Num(mid+1) <= a & a <= Num(hi));
        TypeSafeModel model = null;
        if (LBool.True == z3.CheckAndGetModel(ref model)) {
            lo = model.GetNumeralValueInt(model.Eval(a));
            model.Dispose();
        } else hi = mid;
        z3.Pop();
    }
    return hi;
}
```

Maximize(i3, -1, 17):

```csharp
lo: -1, hi: 17, mid: 8
lo: -1, hi: 8, mid: 3
lo: -1, hi: 3, mid: 1
lo: 2, hi: 3, mid: 2
Optimum: 3
```
int Maximize(TermAst a, int lo, int hi) {
    while (lo < hi) {
        int mid = (lo+hi)/2;
        Console.WriteLine("lo: {0}, hi: {1}, mid: {2},
            lo, hi, mid);
        z3.Push();
        z3.AssertCnstr(Num(mid+1) <= a & a <= Num(hi));
        TypeSafeModel model = null;
        if (LBool.True == z3.CheckAndGetModel(ref model)) {
            lo = model.GetNumeralValueInt(model.Eval(a));
            model.Dispose();
            lo = Maximize(a, lo, hi);
        }
        else hi = mid;
        z3.Pop();
    }
    return hi;
}