Microsoft Research

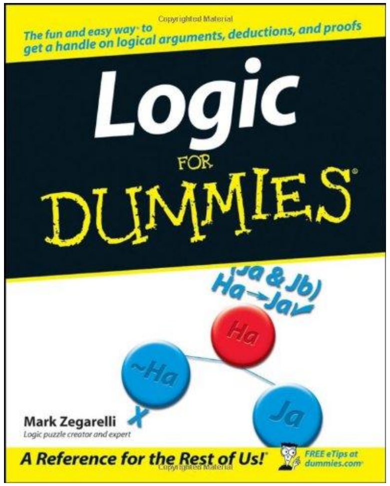
  

Bugs, Moles and Skeletons:
Symbolic Reasoning for Software Development
IJCAR 2010, Edinburgh

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

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(read(write(a,b,3), c-2)) \neq f(c-b+1)$

Satisfiability Modulo Theories (SMT)

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

Arithmetic

Satisfiability Modulo Theories (SMT)

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

Array Theory

Satisfiability Modulo Theories (SMT)

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

Uninterpreted Functions

Satisfiability Modulo Theories (SMT)

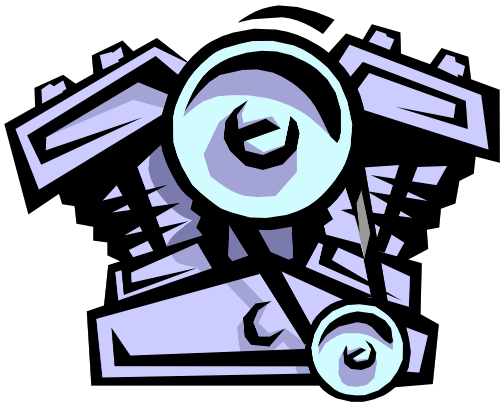
SMT solvers have efficient engines for reasoning modulo theory *T*!

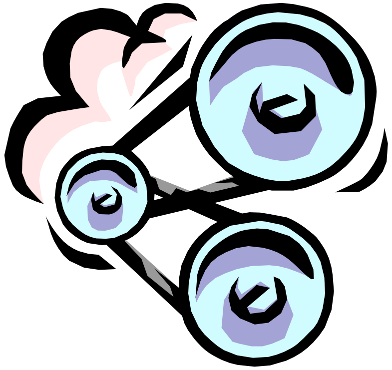
Combining Engines

SMT is also about

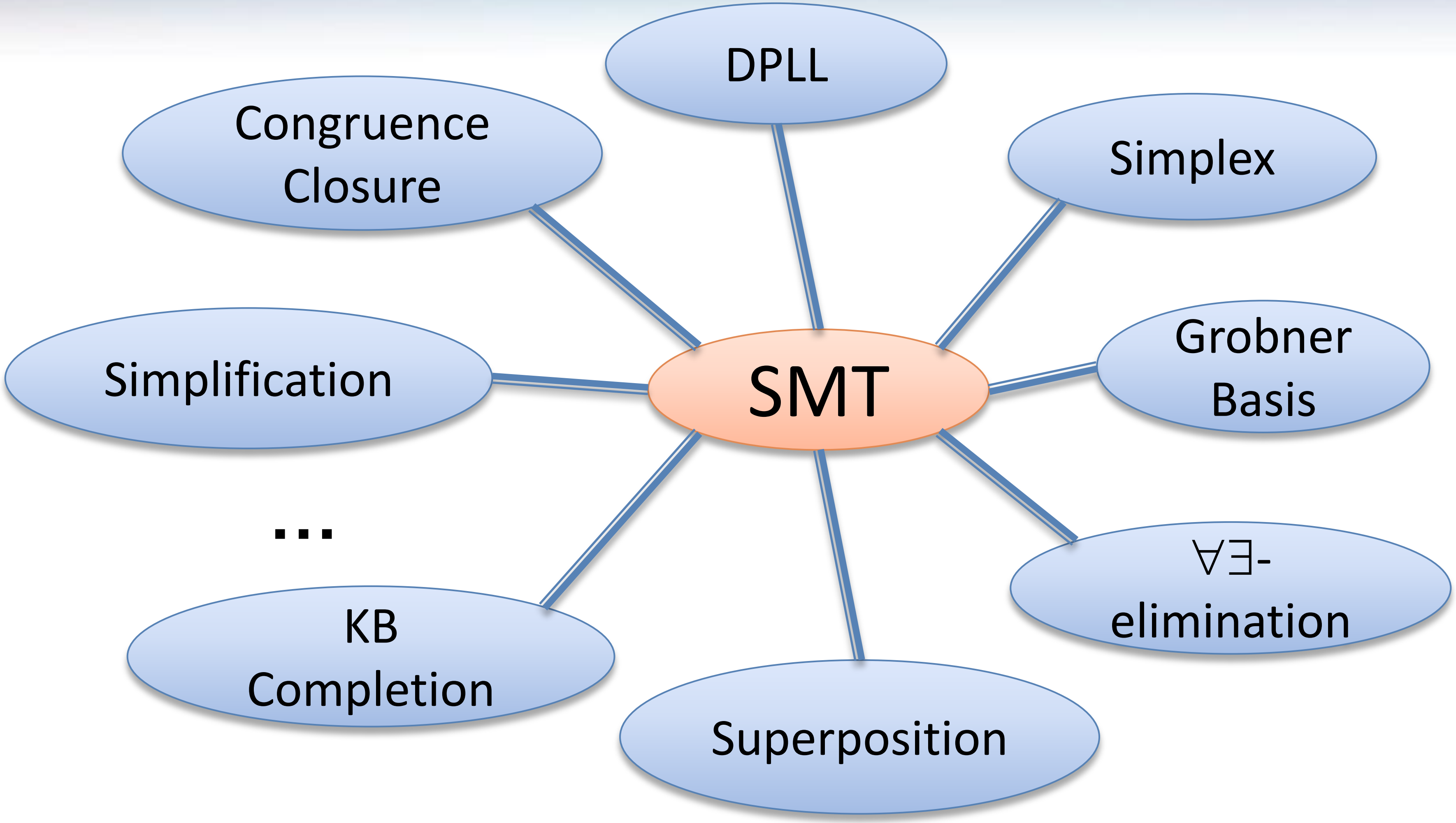
Combining

Different Engines





Combining Engines



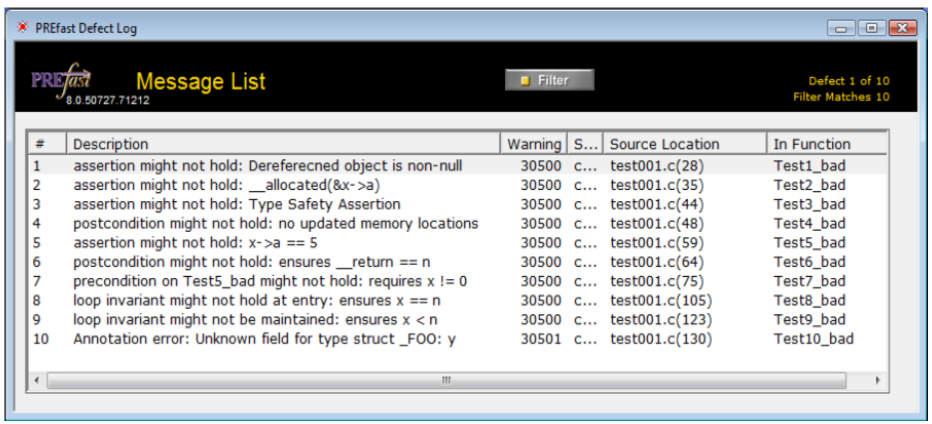
Symbolic Reasoning @ Microsoft

SDV: The Static Driver Verifier

SLAM

Symbolic Reasoning @ Microsoft

PREfix: The Static Analysis Engine for C/C++.



#	Description	Warning	S...	Source Location	In Function
1	assertion might not hold: Dereferecned object is non-null	30500	c...	test001.c(28)	Test1_bad
2	assertion might not hold: __allocated(&x->a)	30500	c...	test001.c(35)	Test2_bad
3	assertion might not hold: Type Safety Assertion	30500	c...	test001.c(44)	Test3_bad
4	postcondition might not hold: no updated memory locations	30500	c...	test001.c(48)	Test4_bad
5	assertion might not hold: x->a == 5	30500	c...	test001.c(59)	Test5_bad
6	postcondition might not hold: ensures __return == n	30500	c...	test001.c(64)	Test6_bad
7	precondition on Test5_bad might not hold: requires x != 0	30500	c...	test001.c(75)	Test7_bad
8	loop invariant might not hold at entry: ensures x == n	30500	c...	test001.c(105)	Test8_bad
9	loop invariant might not be maintained: ensures x < n	30500	c...	test001.c(123)	Test9_bad
10	Annotation error: Unknown field for type struct _FOO: y	30501	c...	test001.c(130)	Test10_bad

Symbolic Reasoning @ Microsoft

PEX: Program Exploration for .NET

Symbolic Reasoning @ Microsoft

SAGE: Scalable Automated Guided Execution

Symbolic Reasoning @ Microsoft

Yogi: Dynamic Symbolic Execution + Abstraction

Symbolic Reasoning @ Microsoft

SPEC#: C# + Contracts

The Spec# Programming System

Symbolic Reasoning @ Microsoft

VCC: Verifying C Compiler

Hyper-V

Microsoft | Virtualization

Symbolic Reasoning @ Microsoft

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

…

Symbolic Reasoning @ Microsoft

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

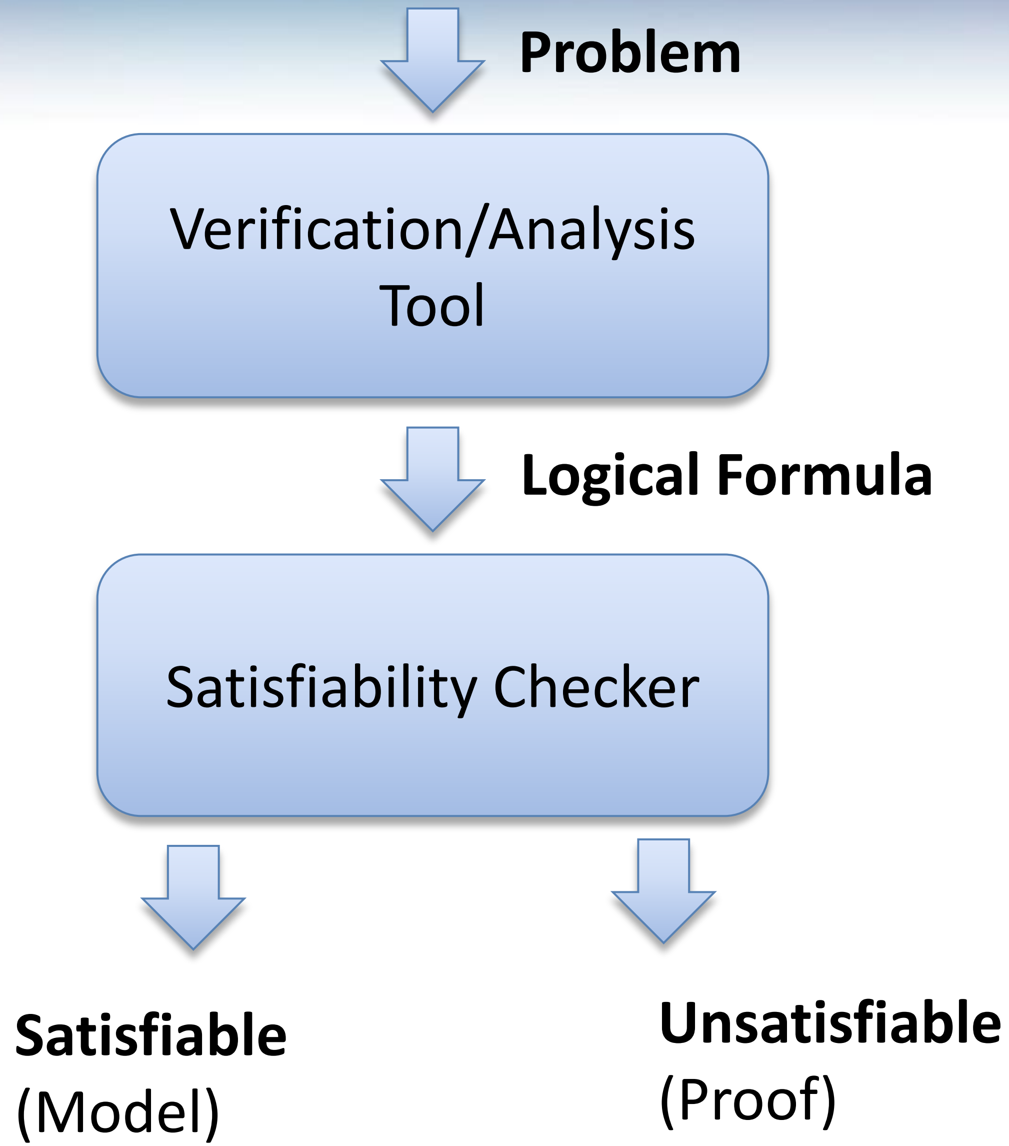
M3: Model Program Modeling

VS3: Abstract interpretation and Synthesis

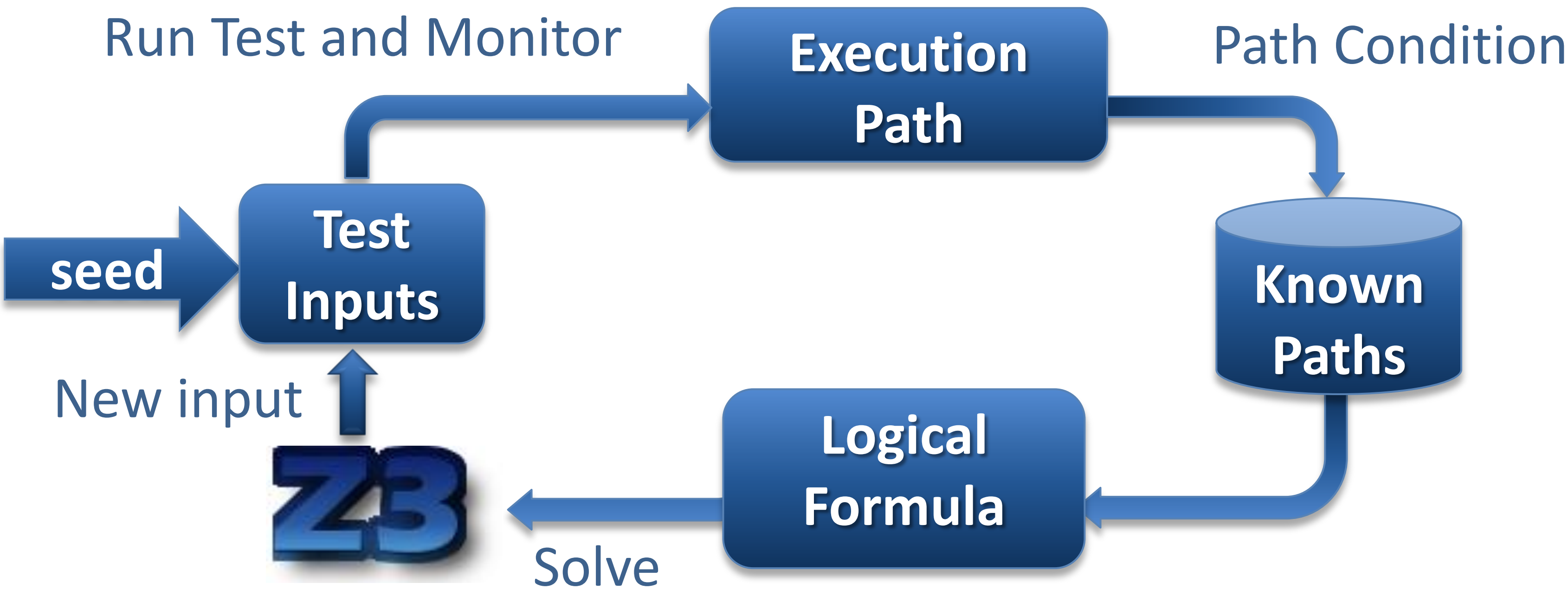
…

They all use Z3

Software Analysis Tool: "Template"



Directed Automated Random Testing (DART)



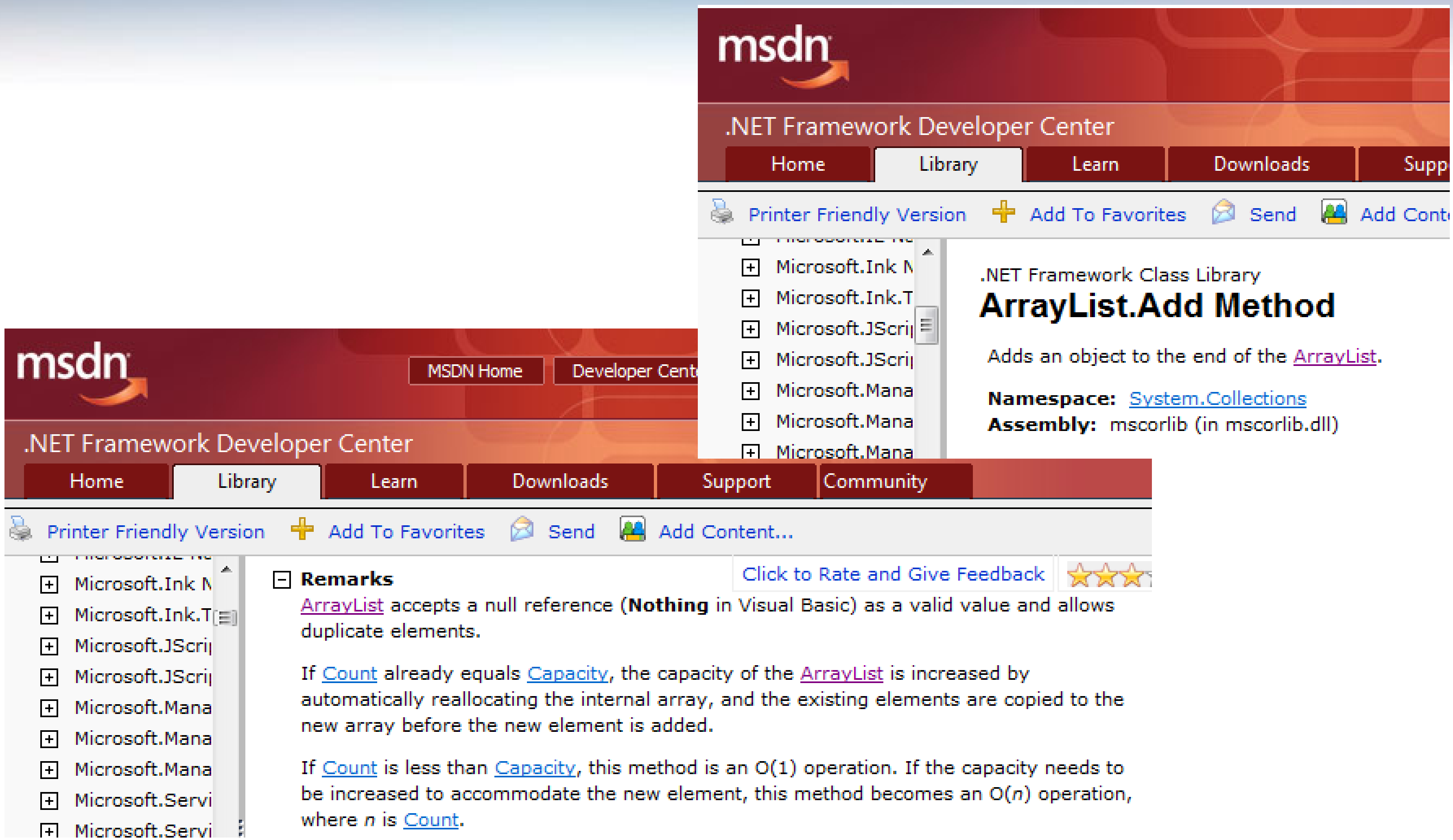
DARTish projects at Microsoft

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

What is Pex?

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

ArrayList: The Spec



ArrayList: AddItem Test

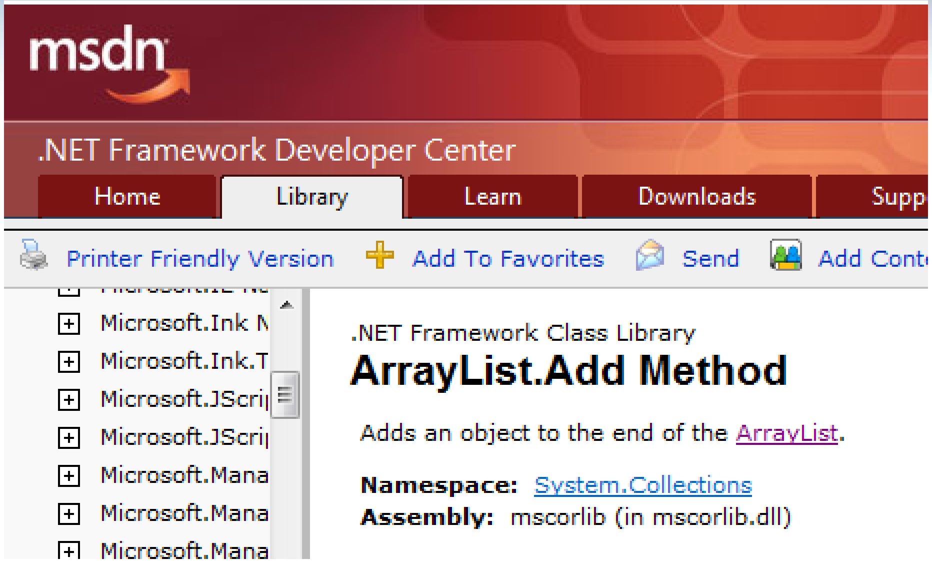
```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

```
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; }
...
```



ArrayList: Starting Pex…

```
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; }
...
```

Inputs

ArrayList: Run 1, (0,null)

```
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; }
...
```

Inputs
(0,null)

ArrayList: Run 1, (0,null)

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

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

```
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; }
...
```

c < 0 → false

ArrayList: Run 1, (0,null)

```
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; }
...
```

`0 == c → true`

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

ArrayList: Run 1, (0,null)

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

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

```
item == item  →  true
```

```
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: Picking the next branch to cover

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

```
class ArrayList {
  object[] items;
  int count;

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

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

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

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

Z3

ArrayList: Solve constraints using SMT solver

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

```
class ArrayList {
  object[] items;
  int count;

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

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

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

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

ArrayList: Run 2, (1, null)

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

```
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; }
...
```

0 == c → false

ArrayList: Pick new branch

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

```
class ArrayList {
  object[] items;
  int count;

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

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

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

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

ArrayList: Run 3, (-1, null)

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

```
class ArrayList {
  object[] items;
  int count;

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

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

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

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

ArrayList: Run 3, (-1, null)

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

```
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; }
...
```

c < 0 → true

ArrayList: Run 3, (-1, null)

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

```
class ArrayList {
  object[] items;
  int count;

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

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

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

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

White box testing in practice

How to test this code?

(Real code from .NET base class libraries.)

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

    _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

```
        // 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 (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.Log("RESMGRFILEFORMAT", LogLevel.Status, "ReadResources: Unexpected ResMgr header
                    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
```

White box testing in practice

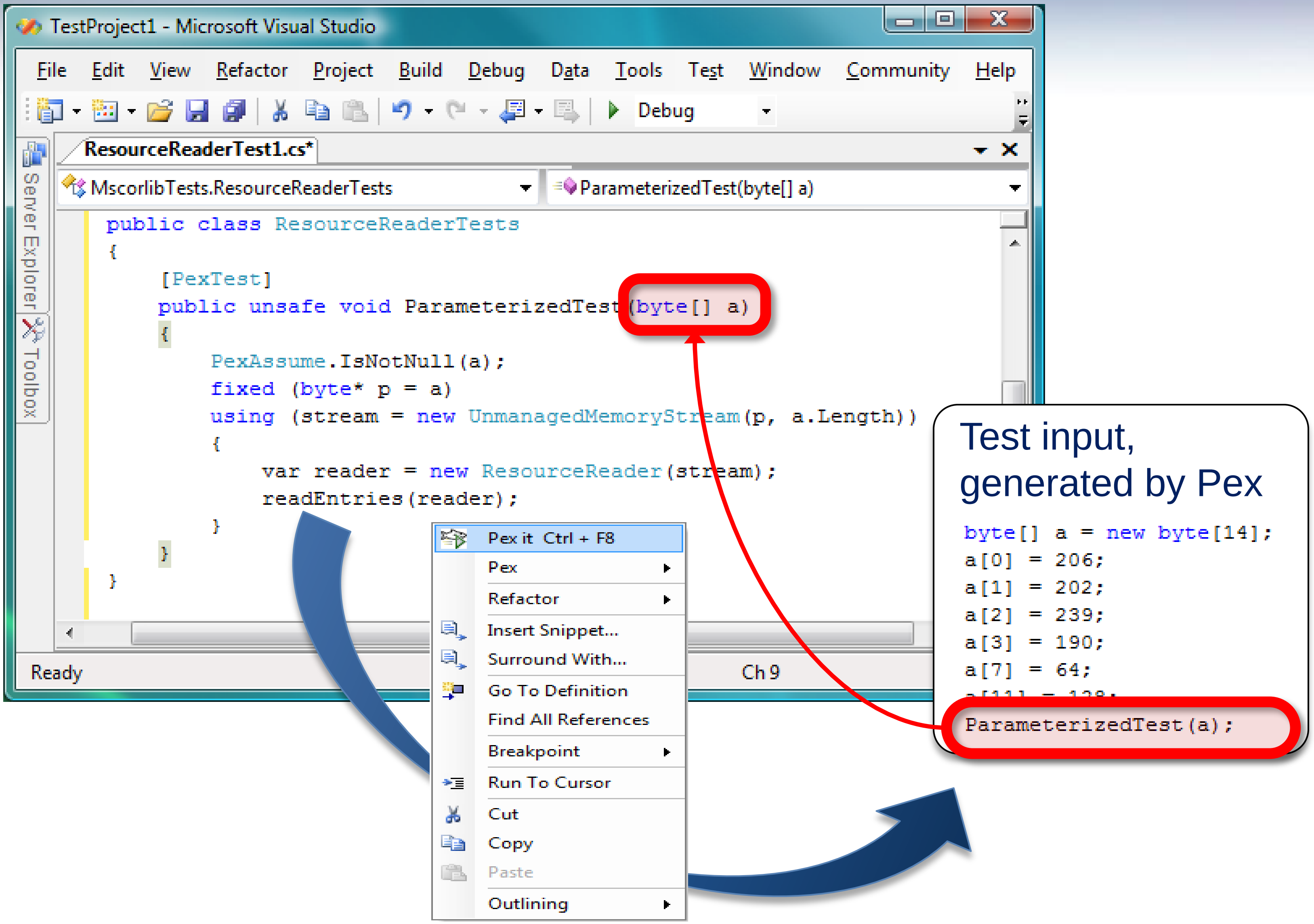
```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader 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
                //
                //
                //
                int
                if

                } e
                    // Read in type name for a suitable ResourceReader
```

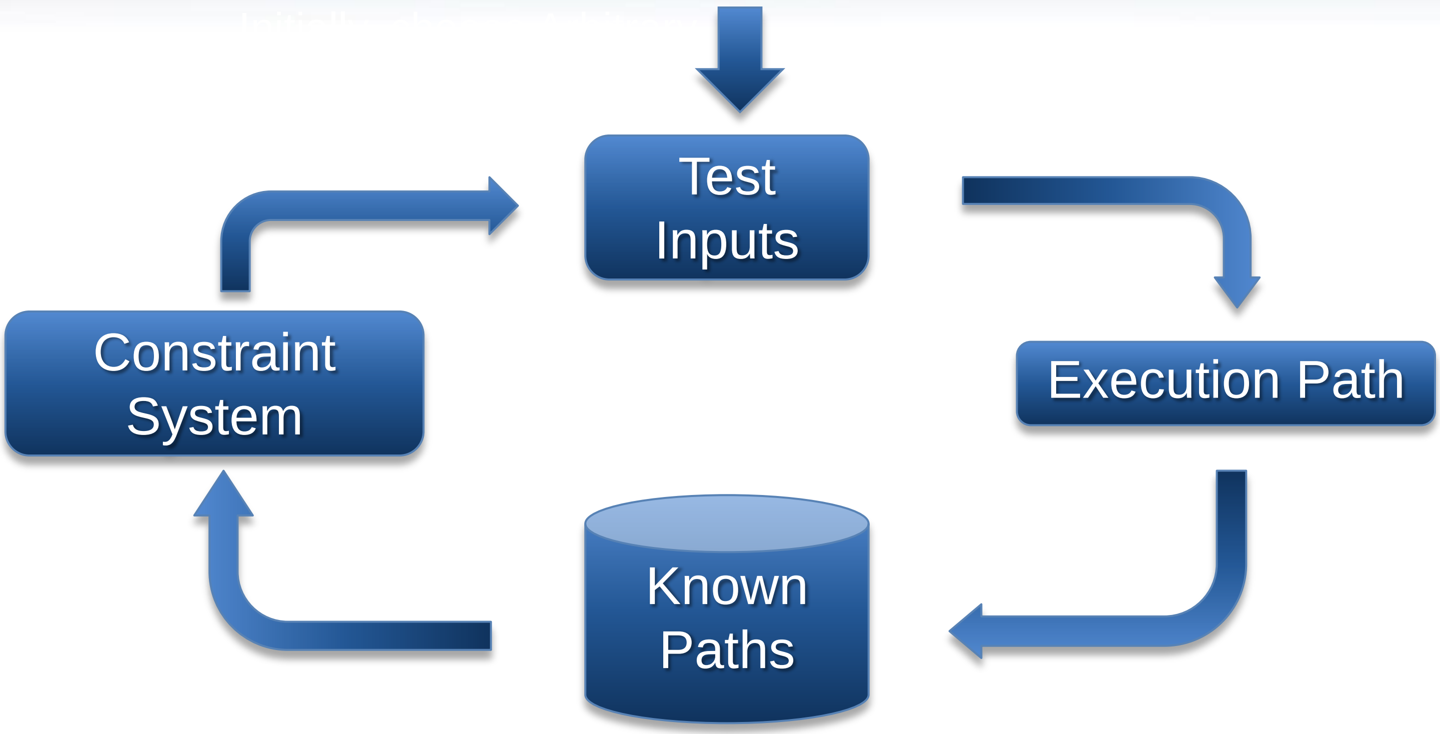
```
public virtual int ReadInt32() {
    if (m_isMemoryStream) {
        // read directly from MemoryStream buffer
        MemoryStream mStream = m_stream as MemoryStream;
        BCLDebug.Assert(mStream != null, "m_stream as MemoryStream != null");

        return mStream.InternalReadInt32();
    }
    else
    {
        FillBuffer(4);
        return (int)(m_buffer[0] | m_buffer[1] << 8 | m_buffer[2] << 16 | m_buffer[3] << 24);
    }
}
```

Pex – Test Input Generation



Test Input Generation by Dynamic Symbolic Execution



Result: small test suite, high code coverage

Finds only real bugs
No false warnings

PEX ⟷ Z3

- **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

“Conjecture”

Bug finding and Test-case generation tools

are successful

because

Bugs and moles can be easily digested

Evidence is easy to understand and check.

The Ultimate Oracle

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

How do I trust the verifier?

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…”

Engineer's Perspective

Verification tools are bug-finding tools!

- When they return "Proved", it just means they cannot find more bugs.

We add Loop invariants to speedup the process.

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

They are successful if they expose bugs not exposed by regular testing.

"It is not about proving."

Symbolic Reasoning Support for Models

Bugs and Moles are extracted from models produced by SMT solvers.

“In symbolic reasoning we need attractive models”

“Conjecture”

SAT/SMT Solvers are successful

because

They produce models

Quantifiers & SMT

Quantifier Reasoning in SMT
is a long-standing challenge

Heuristics

Incomplete

Candidate Models

Quantifiers in Practice

Modeling the runtime

$$\forall\ h,o,f:$$
$$\quad IsHeap(h) \wedge o \neq null \wedge read(h, o, alloc) = t$$
$$\quad \Rightarrow$$
$$\quad read(h,o, f) = null \vee read(h, read(h,o,f),alloc) = t$$

Quantifiers in Practice

Frame axioms

$$\forall\, o, f: \quad o \neq null \wedge read(h_0, o, alloc) = t \Rightarrow read(h_1,o,f) = read(h_0,o,f) \vee (o,f) \in M$$

Quantifiers in Practice

User provided assertions

$$\forall\ i,j:\ i \leq j \Rightarrow read(a,i) \leq read(b,j)$$

Quantifiers in Practice

Axiomatizing Theories

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

$\forall$ x,y,z: p(x,y), p(y,z) $\Rightarrow$ p(x,z)

$\forall$ x,y: p(x,y), p(y,x) $\Rightarrow$ x = y

Candidate Models

$$\overbrace{\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)}^{F} \wedge$$

$$\underbrace{w \geq v + 2 \ \wedge\ f(v) \leq 1 \ \wedge\ f(w) \leq 3}_{G}$$

Quantifier Free

Candidate Models

$$\underbrace{w \geq v + 2 \;\wedge\; f(v) \leq 1 \;\wedge\; f(w) \leq 3}_{G}$$

with, over the first line:

$$\overbrace{\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)}^{F} \wedge$$

$$v \mapsto 0,\ w \mapsto 2,\ f \mapsto [0 \mapsto 1,\ 2 \mapsto 3,\ else \mapsto 4]$$

Candidate Models in Practice

can be extracted from candidate models

Candidate Models in Practice

can be extracted from candidate models

Ultimate Oracle

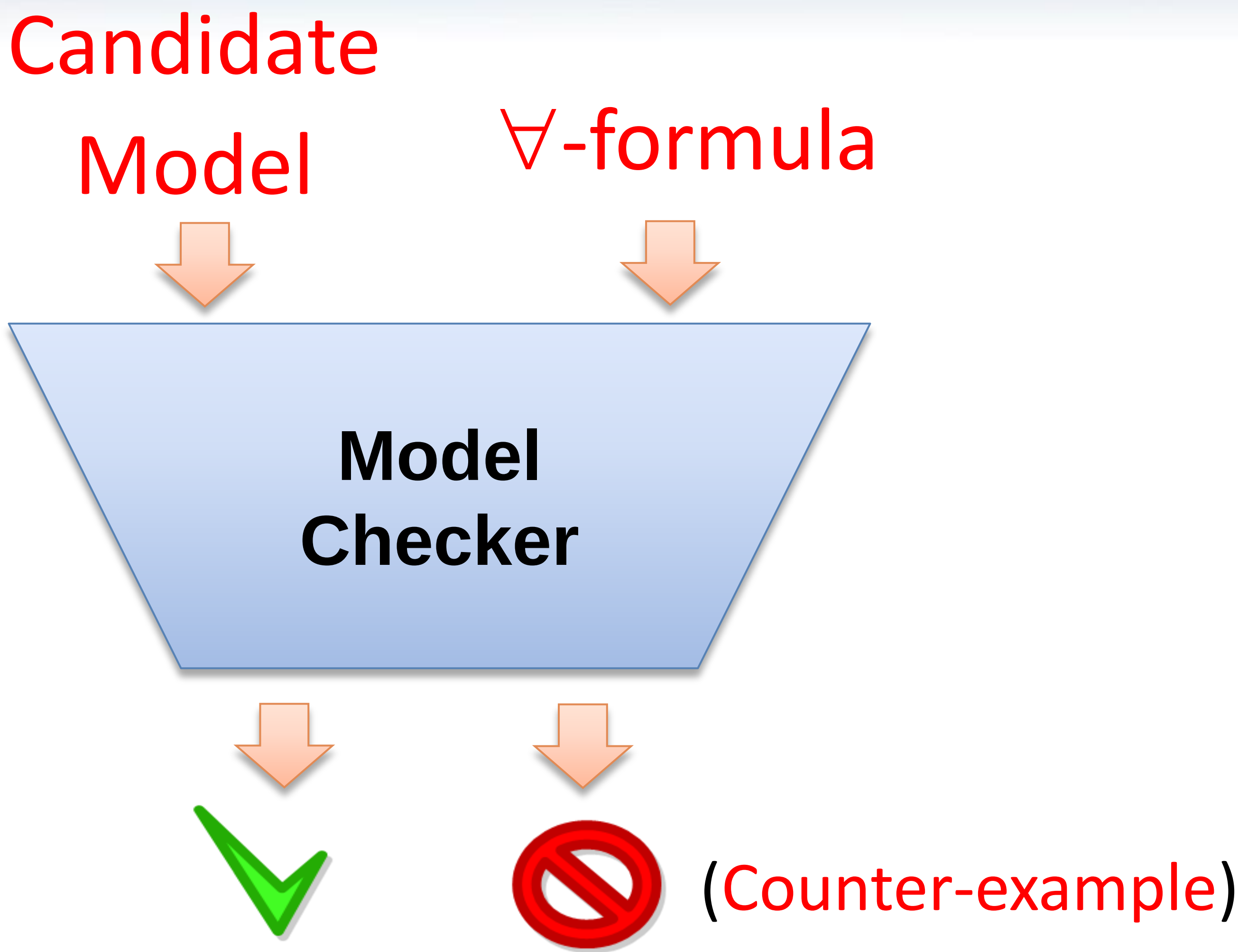
(the actual program)

Streams of Candidate Models

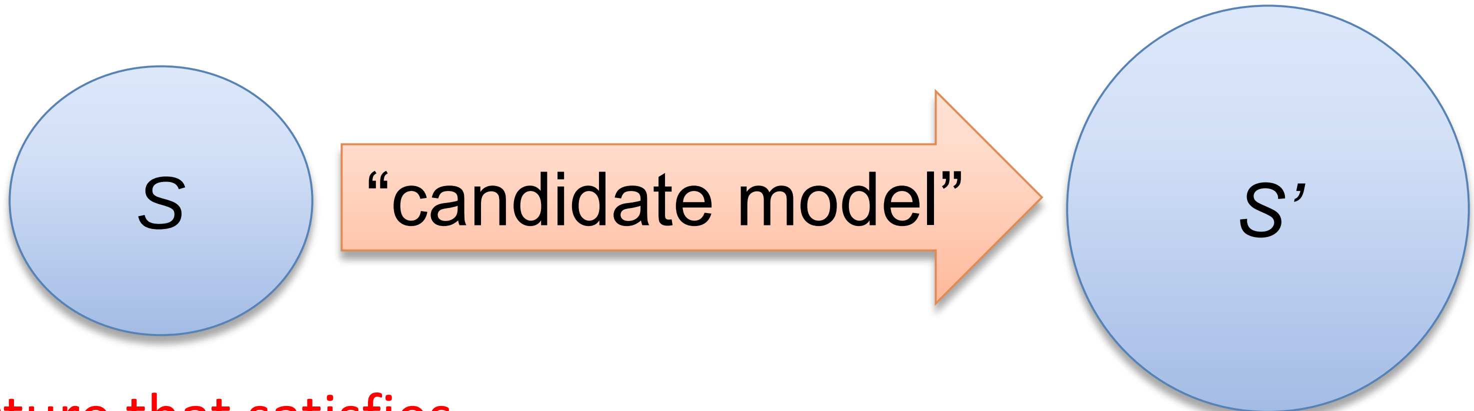
M_1, M_2, M_3, M_4, …

Sequence of more and more refined candidate models

Model Checking Candidate Models



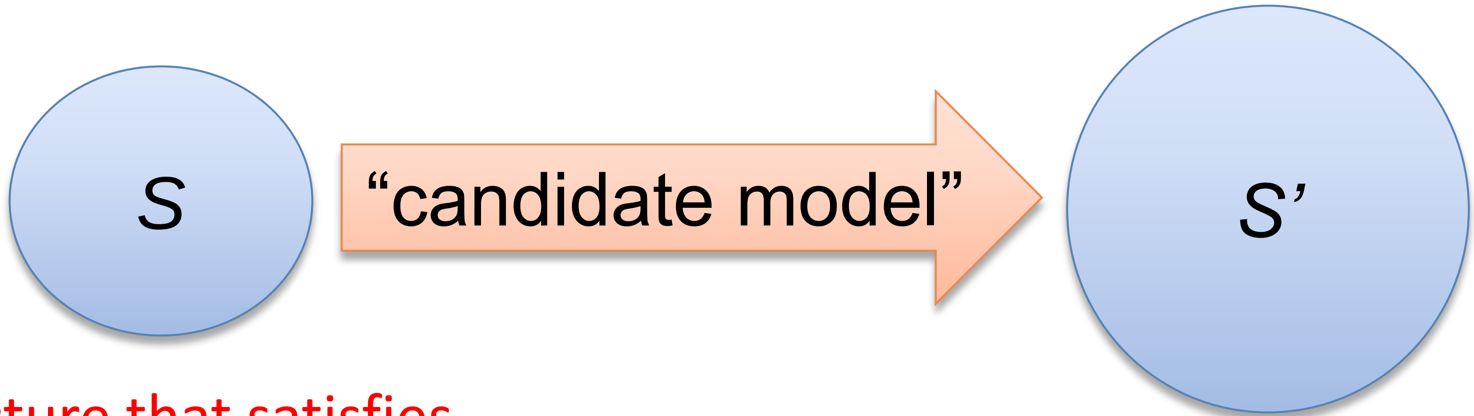
Representing Candidate Models



Structure that satisfies Background theory T

Structure that satisfies $T \cup F$

Representing Candidate Models



Structure that satisfies Background theory T

Structure that satisfies $T \cup F$

Interpretation of uninterpreted symbol $\boldsymbol{f}$ is an expression $\boldsymbol{I_f[x]}$

Representing Candidate Models

$$\overbrace{\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)}^{F} \wedge$$

$$\underbrace{w \geq v + 2 \ \wedge \ f(v) \leq 1 \ \wedge \ f(w) \leq 3}_{G}$$

$$\Downarrow$$

$$v \mapsto 0,\ w \mapsto 2,\ f \mapsto [0 \mapsto 1,\ 2 \mapsto 3,\ else \mapsto 4]$$

Representing Candidate Models

$$\overbrace{\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)}^{F} \wedge$$
$$\underbrace{w \geq v + 2 \ \wedge \ f(v) \leq 1 \ \wedge \ f(w) \leq 3}_{G}$$

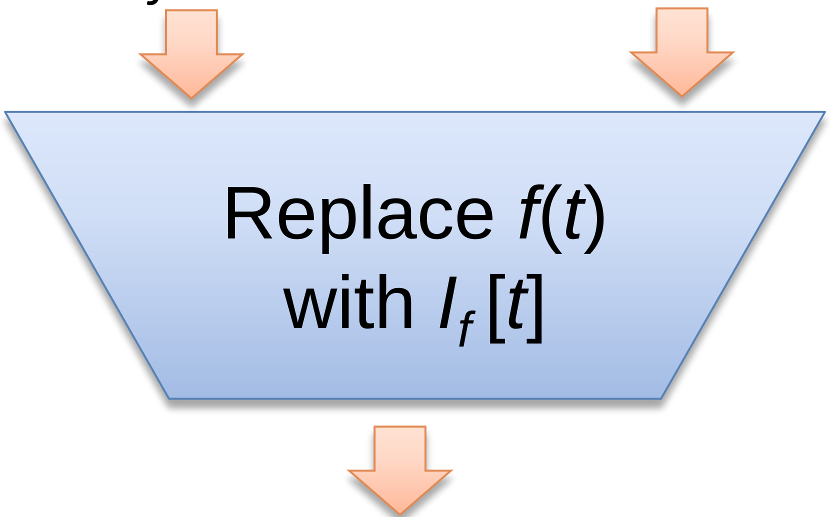
$$v \mapsto 0, \ \ w \mapsto 2, \ \ f(x) \mapsto ite(x = 0, 1, ite(x = 2, 3, 4))$$

Model Checking Candidate Models

Candidate Model *M*

$f \rightarrow I_f[x]$

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



$\forall \bar{x}.\ F'[\bar{x}]$

Model Checking Candidate Models

Candidate Model *M*

$f \rightarrow I_f[x]$

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

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

$\forall \bar{x}.\ F'[\bar{x}]$

Contains only interpreted symbols and variables

Model Checking Candidate Models

Candidate Model *M*

$f \rightarrow I_f[x]$

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

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

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

$\neg F^{I}[\bar{s}]$

Is satisfied by *M* if negation is unsatisfiable.

Contains only interpreted symbols and variables

Model Checking Candidate Models

$$\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)$$

$$v \mapsto 0,\ \ w \mapsto 2,\ \ f(x) \mapsto ite(x = 0, 1, ite(x = 2, 3, 4))$$

$$s_1 \leq s_2 \wedge$$
$$\neg(ite(s_1 = 0, 1, ite(s_1 = 2, 3, 4)) \leq ite(s_2 = 0, 1, ite(s_2 = 2, 3, 4)))$$

Model Checking Candidate Models

$$\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)$$

$$v \mapsto 0,\ \ w \mapsto 2,\ \ f(x) \mapsto ite(x = 0, 1, ite(x = 2, 3, 4))$$

$$s_1 \leq s_2 \wedge$$
$$\neg(ite(s_1 = 0, 1, ite(s_1 = 2, 3, 4)) \leq ite(s_2 = 0, 1, ite(s_2 = 2, 3, 4)))$$

is satisfied by

$$s_1 \mapsto 1, s_2 \mapsto 2$$

Model Checking Candidate Models

$$\forall i, j.\ i \leq j \rightarrow f(i) \leq f(j)$$

$$v \mapsto 0,\ \ w \mapsto 2,\ \ f(x) \mapsto ite(x \leq 0, 1, ite(x \leq 2, 3, 4))$$

$$s_1 \leq s_2 \wedge$$
$$\neg(ite(s_1 \leq 0, 1, ite(s_1 \leq 2, 3, 4)) \leq ite(s_2 \leq 0, 1, ite(s_2 \leq 2, 3, 4)))$$

is unsatisfiable

Synthesis a "new" application

Bug finding

Synthesis

Verification

Synthesis

Loop Invariants

Ranking Functions (Termination)

Code

Synthesis: Loop Invariants

$$
\begin{array}{l}
pre \\
\textbf{while} \;\; (c) \;\; \{ \\
\quad T \\
\} \\
post
\end{array}
$$

$$\forall s.\ pre[s] \rightarrow I(s)$$
$$\forall s, s'.\ I(s) \wedge c[s] \wedge T[s, s'] \rightarrow I(s')$$
$$\forall s.\ I(s) \wedge \neg c[s] \rightarrow post[s]$$

Synthesis: Ranking Functions

$$
\begin{array}{l}
pre \\
\textbf{while}\ \ (c)\ \ \{ \\
\quad T \\
\} \\
post
\end{array}
$$

$$\forall s.\ rank(s) \geq 0$$
$$\forall s, s'.\ c[s] \wedge T[s, s'] \rightarrow rank(s') < rank(s)$$

Synthesis: Loop Invariants

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

$$\forall x, y, n.\ n \geq 0 \land x = 0 \land y = 0 \rightarrow I(x, y, n)$$

$$\forall x, y, n, x', y', n'.\ I(x, y, n) \land x < n \land x' = x + 1 \land y' = y + 1 \land n' = n \rightarrow I(x', y', n')$$

$$\forall x, y, n.\ I(x, y, n) \land \neg(x < n) \rightarrow y = n$$

$$I(x,y,n) \mapsto x = y \land x \leq n$$

Synthesis and SMT

$$\forall x, y, n.\ n \geq 0 \wedge x = 0 \wedge y = 0 \rightarrow I(x, y, n)$$

$$\forall x, y, n, x', y', n'.\ I(x, y, n) \wedge x < n \wedge x' = x + 1 \wedge y' = y + 1 \wedge n' = n \rightarrow I(x', y', n')$$

$$\forall x, y, n.\ I(x, y, n) \wedge \neg(x < n) \rightarrow y = n$$

SMT formula (UFLIA)

Most SMT Solvers will return "unknown" for this formula.

Idea from Synthesis: Templates (Skeletons)

Basic Idea for synthesising a function:

Use a (set of) skeletons.

Reduce the search to the space of instances.

Skeleton Based Model Finding

An Skeleton is just an expression $t[\overline{x}, \overline{c}]$

fresh constants

Example:

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

Instances:

$x+1\ (a \mapsto 1, b \mapsto 1)$ and $2x\ (a \mapsto 2, b \mapsto 0)$

Skeleton Based Model Finding

Template Binding (TB):
associate a template with each uninteprеted function
$f \mapsto t[\overline{x}, \overline{c}]$

SMF procedure for instantiating templates.
Given quantifier-free formula φ and TB

$$\varphi \wedge \bigwedge_{f(\overline{r}) \in \varphi} f(\overline{r}) = t[\overline{r}, \overline{c}]$$

Skeleton Based Model Finding

$$f(a_1) \geq 10 \wedge f(a_2) \geq 100 \wedge f(a_3) \geq 1000 \wedge$$
$$a_1 = 0 \wedge a_2 = 1 \wedge a_3 = 2$$

$$f \mapsto c_1 x + c_2$$

$$f(a_1) \geq 10 \wedge f(a_2) \geq 100 \wedge f(a_3) \geq 1000 \wedge$$
$$a_1 = 0 \wedge a_2 = 1 \wedge a_3 = 2 \wedge$$
$$f(a_1) = c_1 a_1 + c_2 \wedge f(a_2) = c_1 a_2 + c_2 \wedge$$
$$f(a_3) = c_1 a_3 + c_2$$

$$c_1 \rightarrow 1, \; c_2 \rightarrow 1000$$

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)

Arrays in QBVF

Example: array updates

$f' = write(f,\ a+1, 0)$

$$f'(a+1) = 0 \ \wedge\ (\forall x : 8.\ x = a+1 \vee f'(x) = f(x))$$

Quantified Bit-Vector Logic (QBVF)

New (prototype) Solver:

- First-order techniques:
  - Mini-scoping, Skolemization, DER,
  - Rewriting, …
- Theory rewriting
- Quasi-Macro detection
- Skeleton Based Model Finding

Putting everything together

solver(φ, $\texttt{TB}$)

φ := $\texttt{Simplify}(\varphi)$

w.l.o.g. assume φ is of the form $\forall \overline{x}.\ \phi[\overline{x}]$

ρ := $\texttt{HeuristicInst}(\phi[\overline{x}])$

loop

if $\texttt{SMT}(\rho)$ = *unsat* **return** *unsat*

M := SMF(ρ, $\texttt{TB}$)

if $M = \bot$ **return** *unsat modulo* $\texttt{TB}$

V := $\texttt{MC}(\varphi, M)$

if $V = \top$ **return** (*sat*, M)

$\rho := \rho \wedge \bigwedge_{\overline{v} \in V} \phi[\overline{v}]$

Quantified Bit-Vector Logic (QBVF)

If we replace every $f(s)$ with $t[s,c]$, we are essentially reducing QBVF to QBF.

NEXTPTIME $\rightarrow$ PSPACE

Quantified Bit-Vector Logic (QBVF)

Very good experimental results.

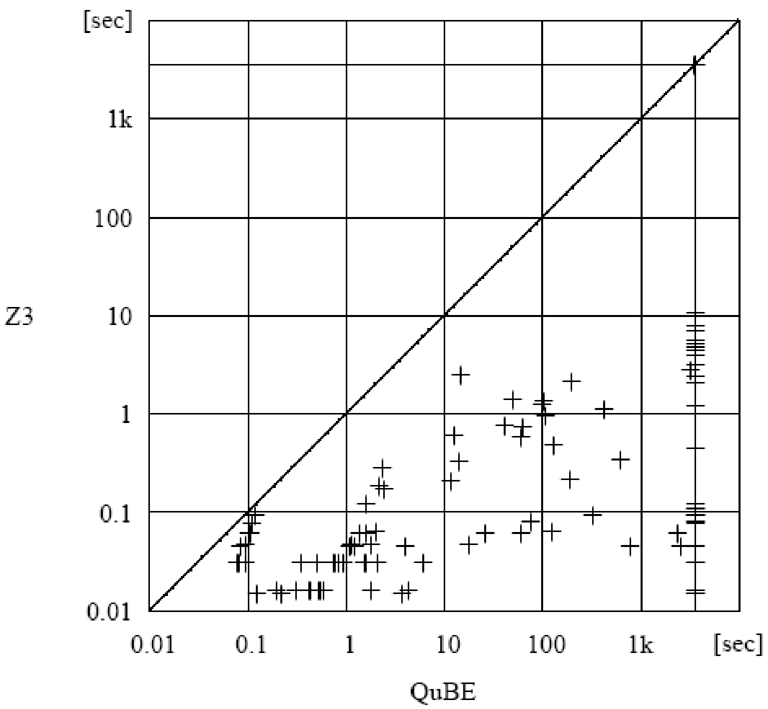
Hardware Fixpoint Checks.

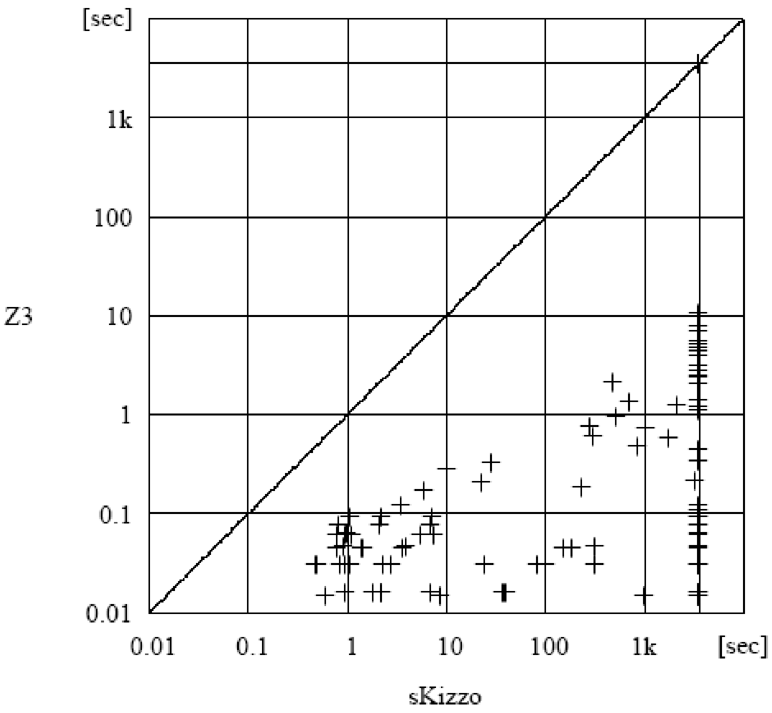
Given: $I[x]$ and $T[x, x']$

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

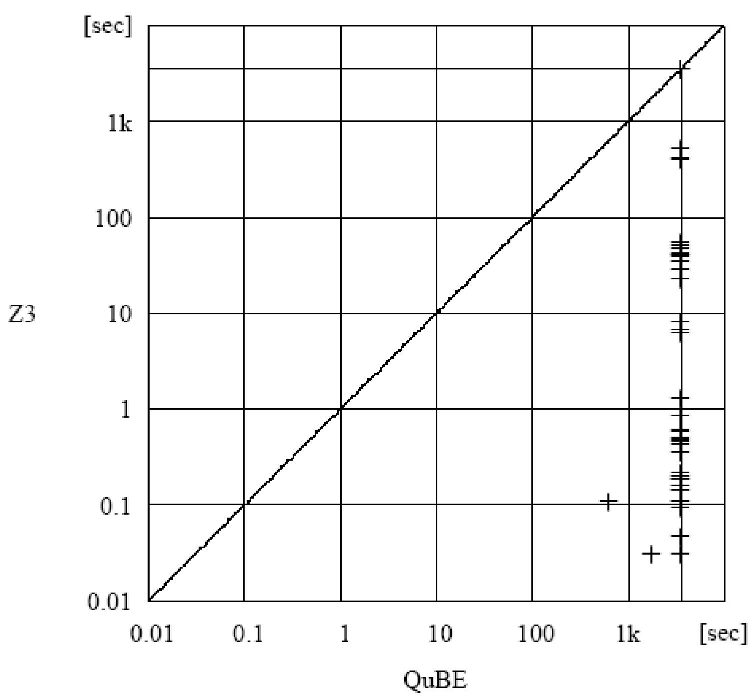
Ranking function synthesis.

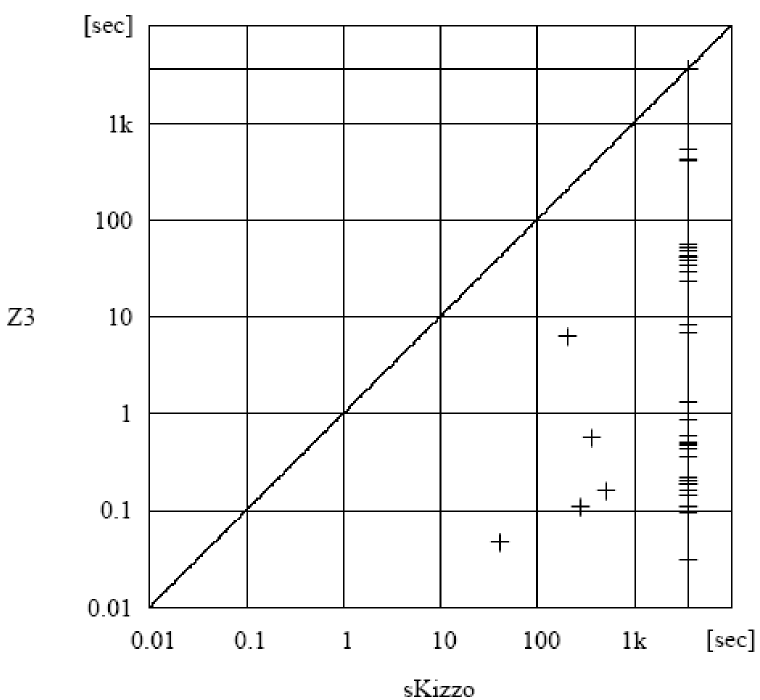
Hardware Fixpoint Checks





Ranking Function Synthesis





Conclusion

- 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.