Satisfiability Modulo Theories (SMT): ideas and applications

Università Degli Studi Di Milano
Scuola di Dottorato in Informatica, 2010

Leonardo de Moura
Microsoft Research

Symbolic Reasoning

Verification/Analysis tools need some form of **Symbolic Reasoning**

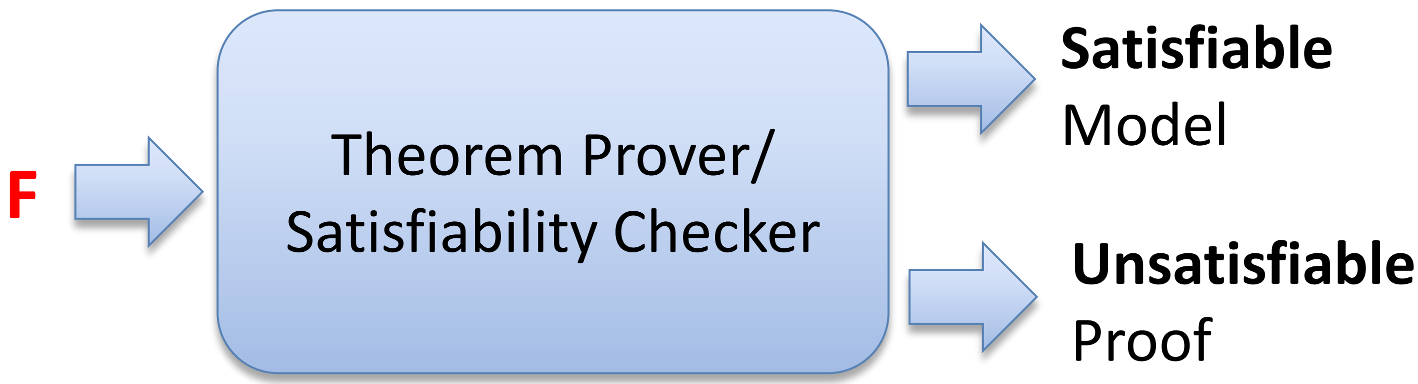
Microsoft Research

Theorem Provers/Satisfiability Checkers

A formula F is valid

Iff

$\neg$F is unsatisfiable



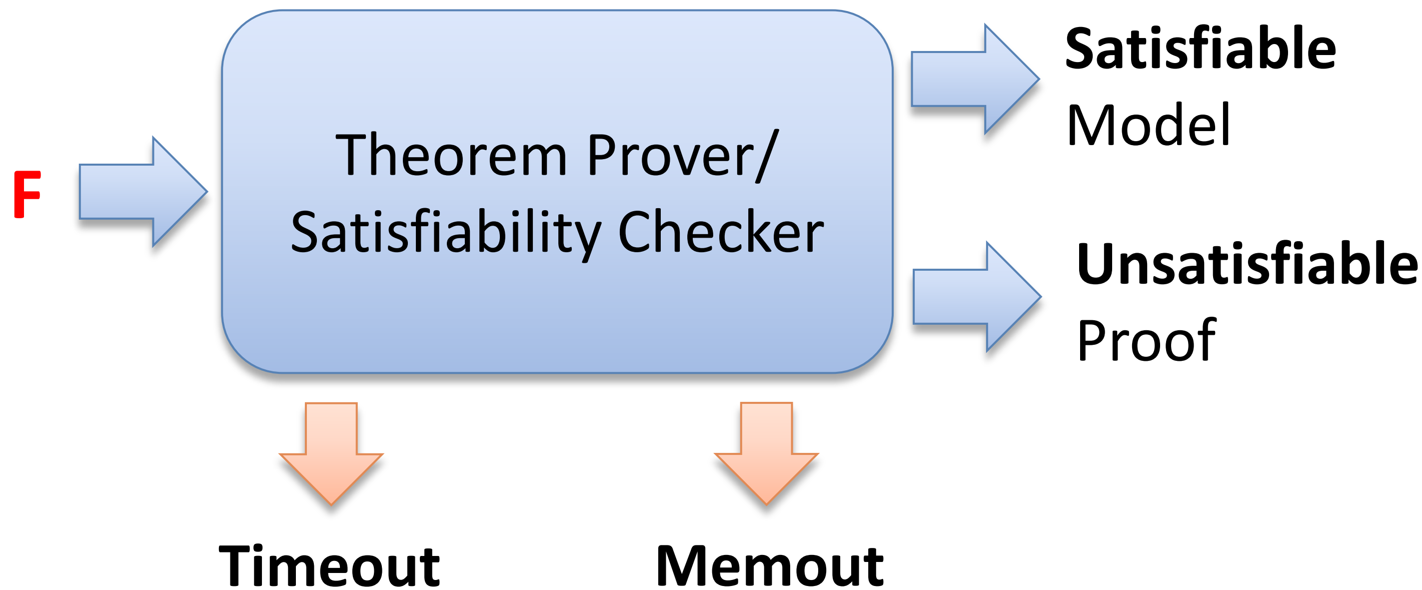
Microsoft Research

Theorem Provers/Satisfiability Checkers

A formula F is valid

Iff

¬F is unsatisfiable



Microsoft Research

Verification/Analysis Tool: "Template"

Problem

Verification/Analysis Tool

Logical Formula

Theorem Prover/ Satisfiability Checker

Satisfiable
(Counter-example)

Unsatisfiable

Microsoft Research

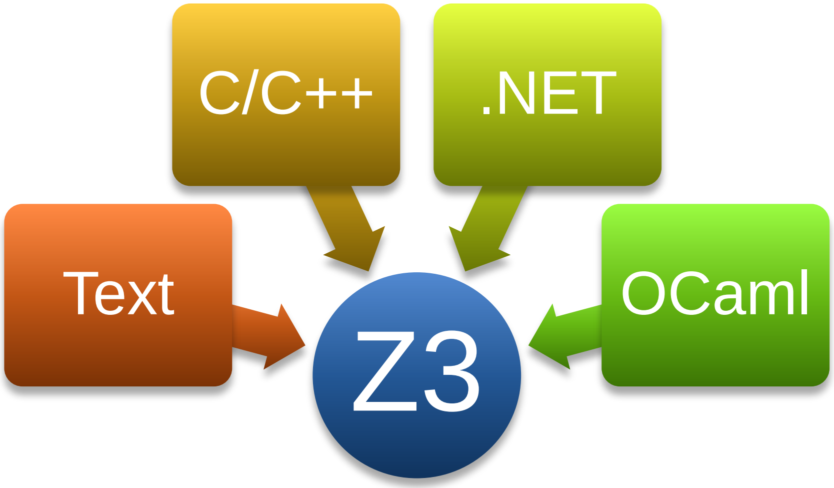
Applications

- Test case generation
- Verifying Compilers
- Predicate Abstraction
- Invariant Generation
- Type Checking
- Model Based Testing

Microsoft Research

SMT@Microsoft: Solver

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



- http://research.microsoft.com/projects/z3

Test case generation

Test case generation

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

Security is critical

- Security bugs can be very expensive:
  - Cost of each MS Security Bulletin: $600k to $Millions.
  - Cost due to worms: $Billions.
  - The real victim is the customer.
- Most security exploits are initiated via files or packets.
  - Ex: Internet Explorer parses dozens of file formats.
- Security testing: hunting for million dollar bugs
  - Write A/V
  - Read A/V
  - Null pointer dereference
  - Division by zero

Microsoft Research

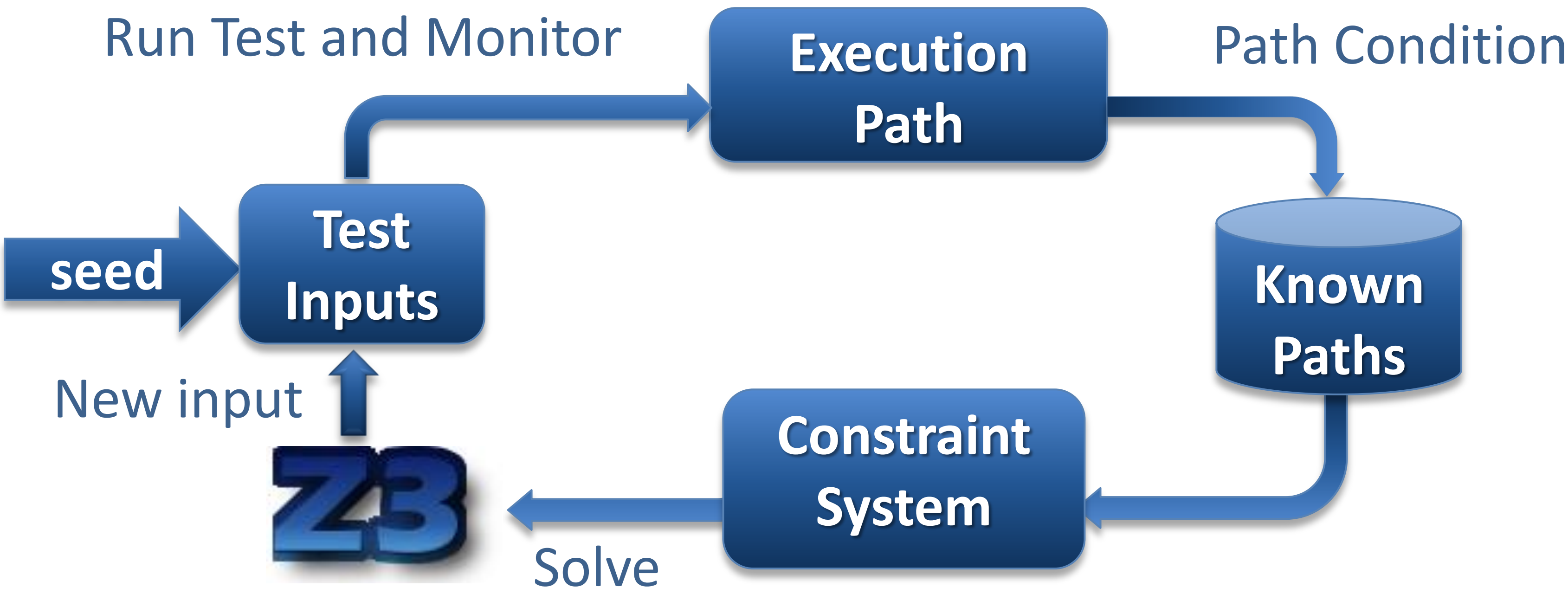
Hunting for Security Bugs.

- Two main techniques used by *“black hats”*:
  - Code inspection (of binaries).
  - *Black box fuzz testing.*

- **Black box** fuzz testing:
  - A form of black box random testing.
  - Randomly *fuzz* (=modify) a well formed input.
  - Grammar-based fuzzing: rules to encode how to fuzz.
- **Heavily** used in security testing
  - At MS: several internal tools.
  - Conceptually simple yet effective in practice

Microsoft Research

Directed Automated Random Testing (DART)



Microsoft Research

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.

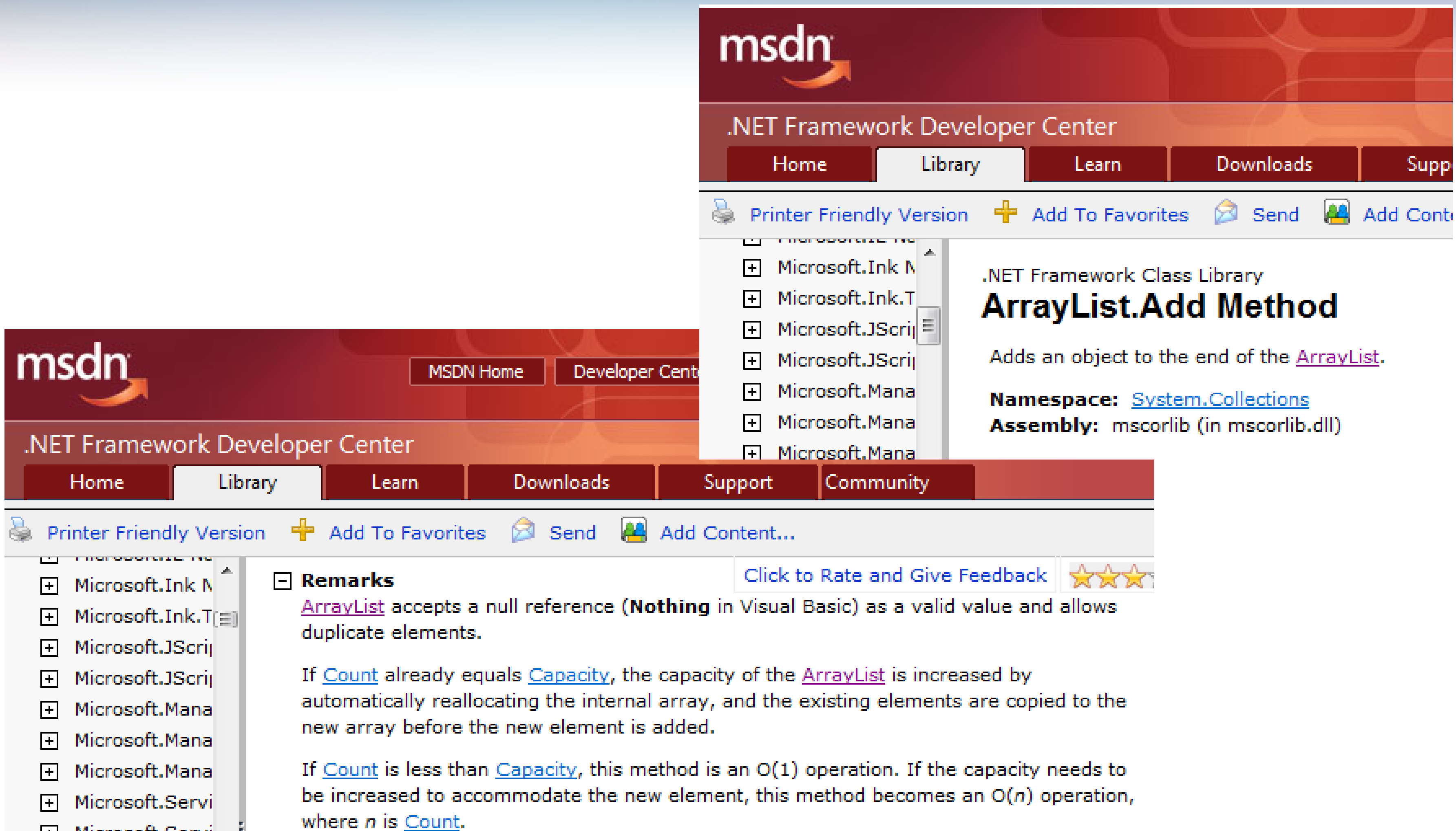
Microsoft Research

What is Pex?

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

Microsoft Research

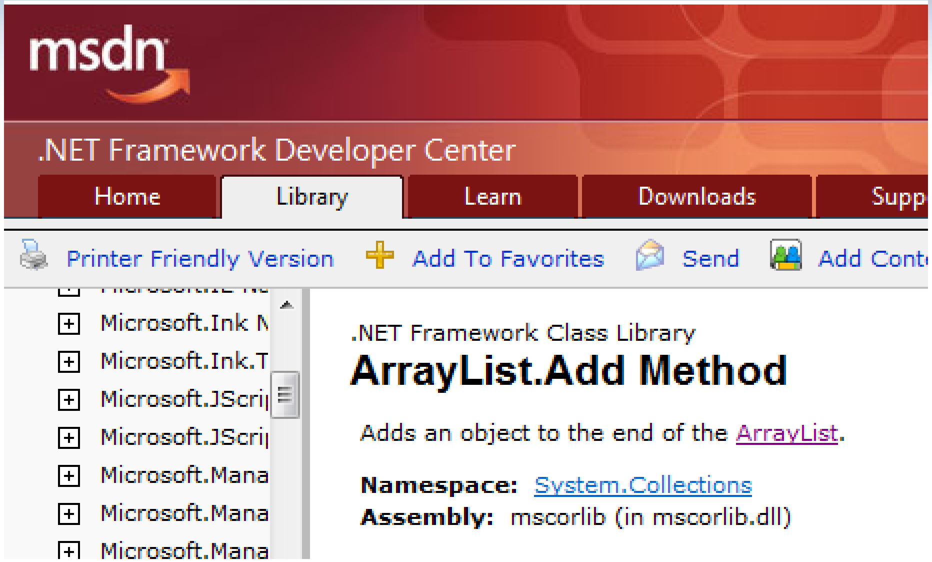
ArrayList: The Spec



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; }
...
```



Microsoft Research

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

Microsoft Research

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

Microsoft Research

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

```
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

Microsoft Research

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; }
...
```

Microsoft Research

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**		

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)`**	

Microsoft Research

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

Microsoft Research

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; }
...
```

Microsoft Research

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

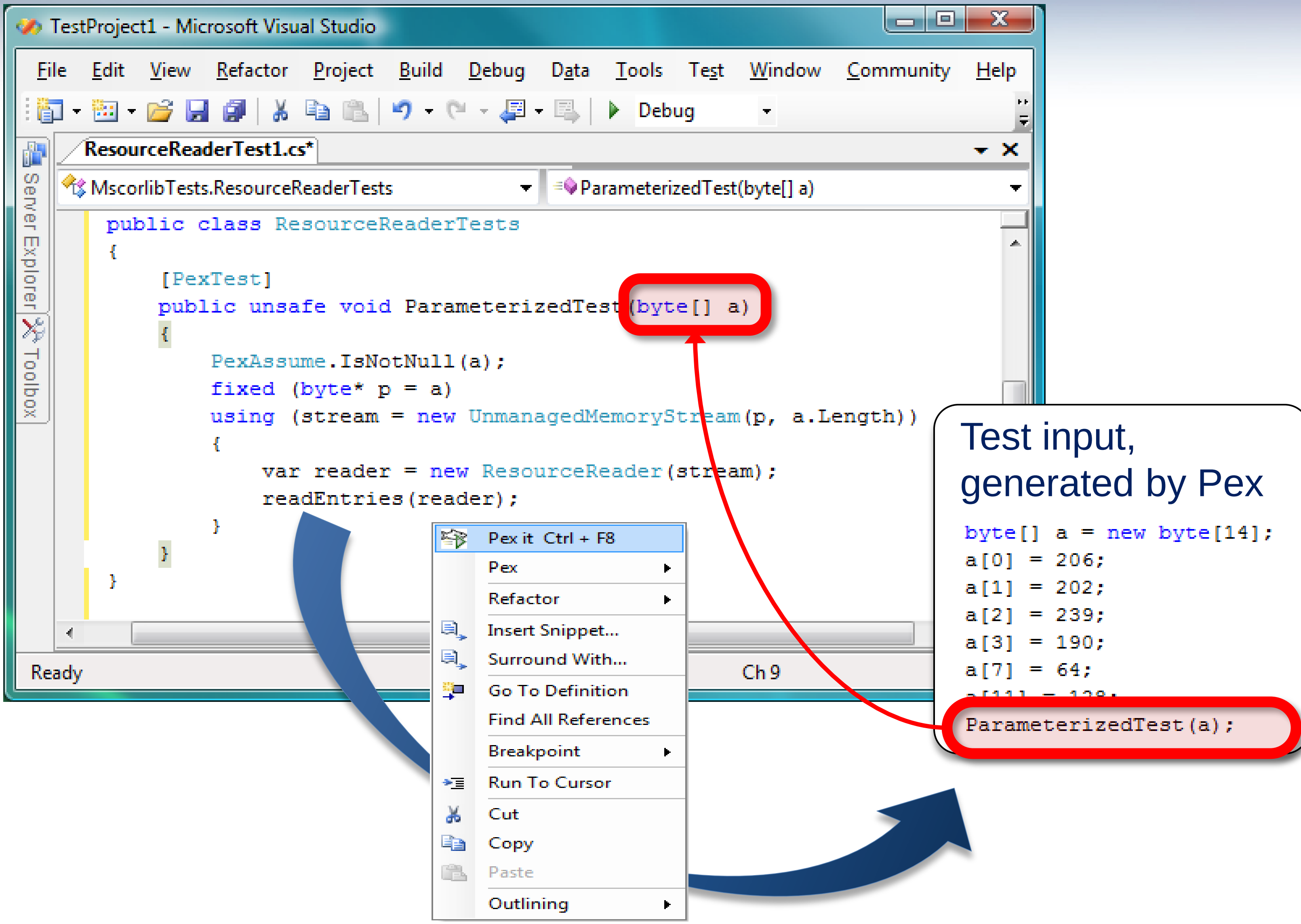
```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader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

White box testing in practice

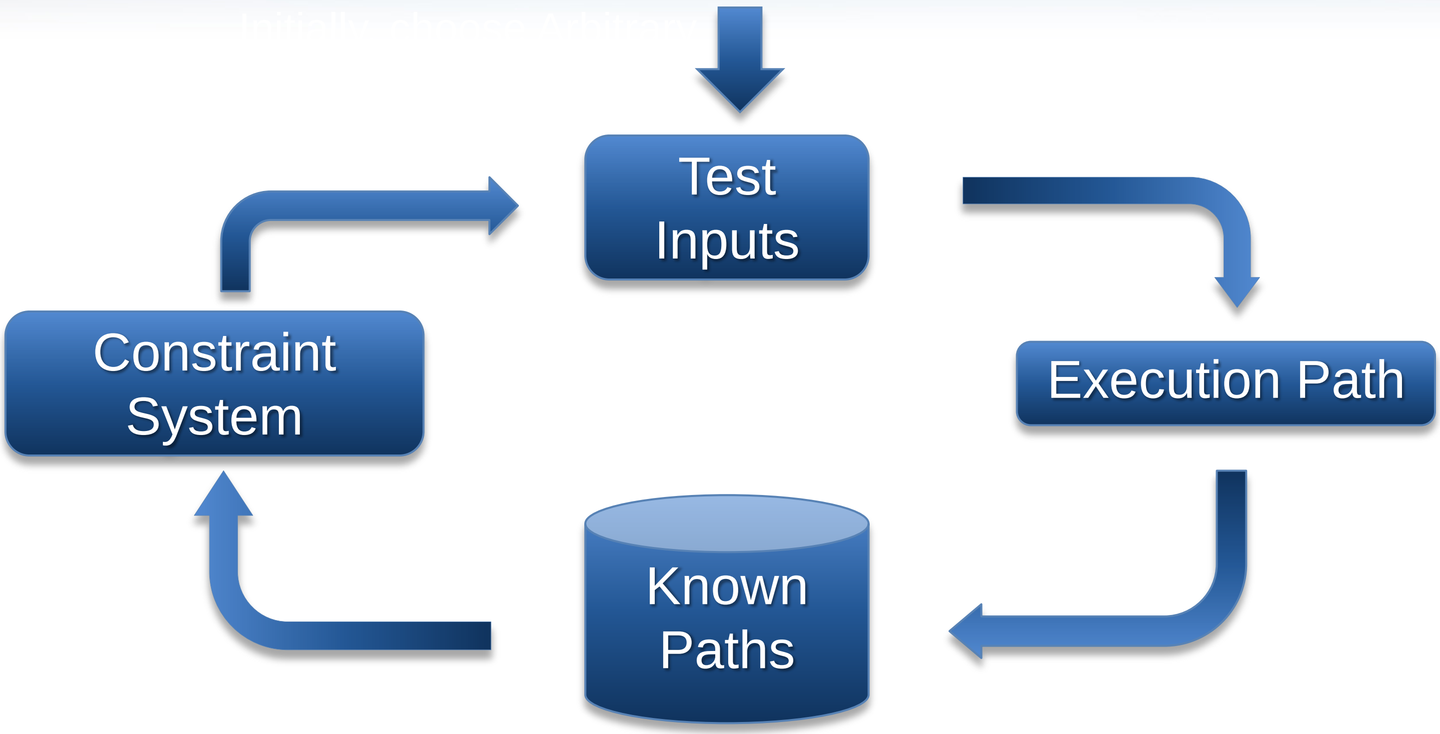
```
        // Reads in the header information for a .resources file.  Verifies some
        // of the assumptions about this resource set, and builds the class table
        // for the default resource file format.
        private void ReadResources() {
            BCLDebug.Assert(_store != null, "ResourceReader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

```
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 | Free Functions
- **Models**: Model used as test inputs
- **∀-Quantifier**: Used to model custom theories (e.g., .NET type system)
- **API**: Huge number of small problems. Textual interface is too inefficient.

PEX ⟷ Z3

- Rich Combination
- Linear arithmetic
- Bitvector
- Arrays
- Free Functions

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

Undecidable (in general)

PEX ↔ Z3

- Rich Combination
- Linear arithmetic
- Bitvector
- Arrays
- Free Functions

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

Undecidable (in general)

Solution:

Return “Candidate” Model

Check if trace is valid by executing it

PEX ↔ Z3

Rich Combination | Linear arithmetic | Bitvector | Arrays | Free Functions

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

Undecidable (in general)

Refined solution:

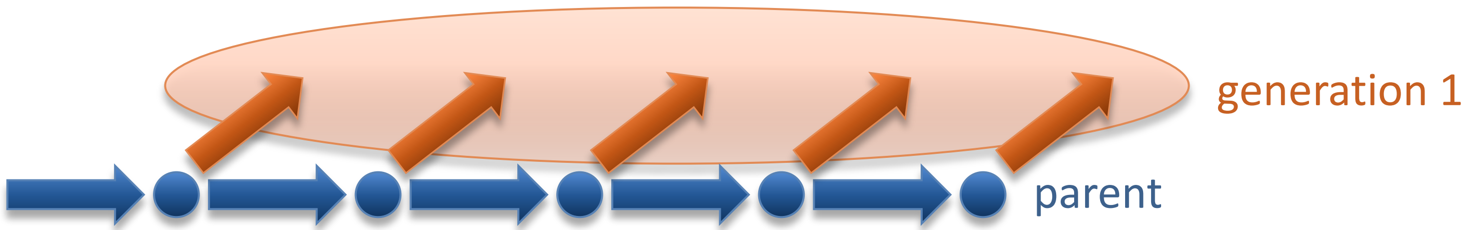
Support for decidable fragments.

SAGE

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

SAGE

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



Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 0 – seed file

Microsoft Research

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 10 – CRASH

SAGE (cont.)

- SAGE is very effective at finding bugs.
- Works on large applications.
- Fully automated
- Easy to deploy (x86 analysis – any language)
- Used in various groups inside Microsoft
- Powered by Z3.

SAGE ⟷ Z3

- Formulas are usually big conjunctions.
- SAGE uses only the bitvector and array theories.
- Pre-processing step has a huge performance impact.
  - Eliminate variables.
  - Simplify formulas.
- Early unsat detection.

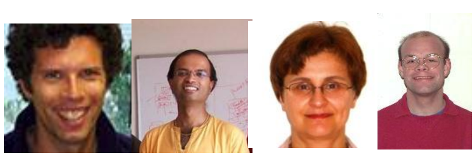
Microsoft Research

Static Driver Verifier

SLAM

Static Driver Verifier

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



Ella Bounimova, Vlad Levin, Jakob Lichtenberg,
Tom Ball, Sriram Rajamani, Byron Cook

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.

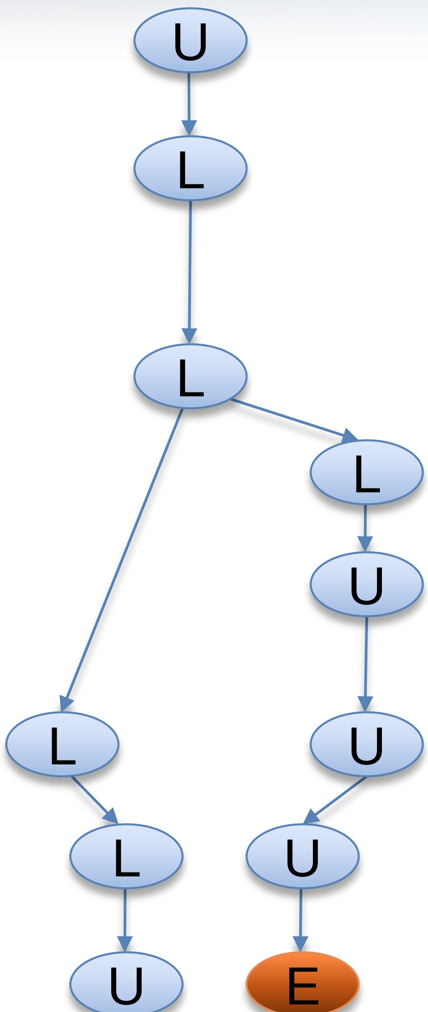
  newton Model refinement (check for path feasibility)
- SMT solvers are used to perform predicate abstraction and to check path feasibility.
- c2bp makes several calls to the SMT solver. The formulas are relatively small.

Example

Do this code obey the looking rule?

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

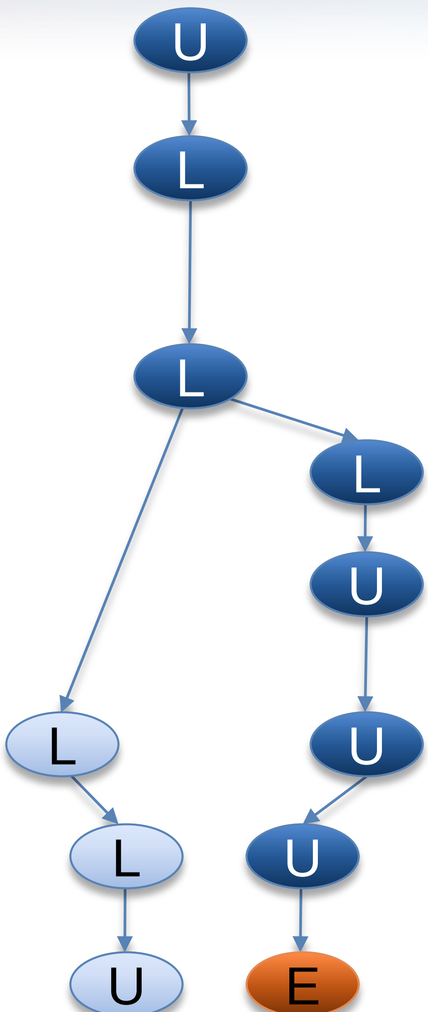


Model checking
Boolean program

```
do {
   KeAcquireSpinLock();



   if(*){

      KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

Example

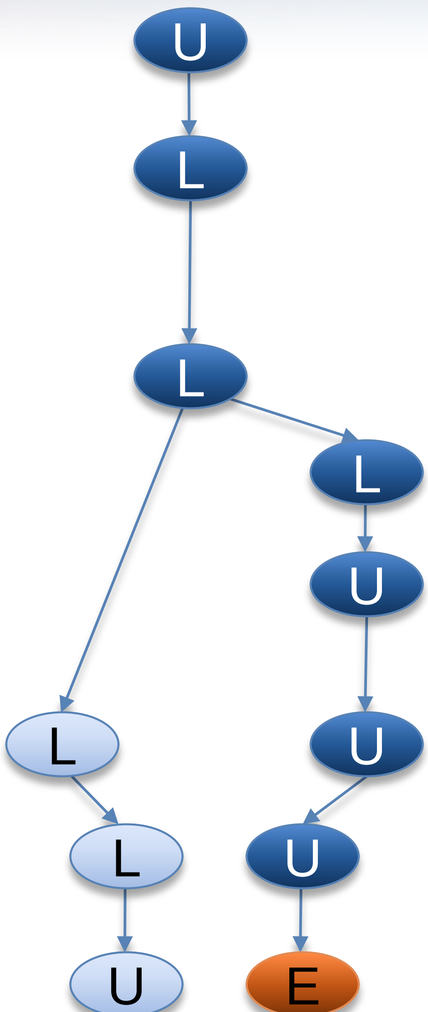
Is error path feasible?




```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

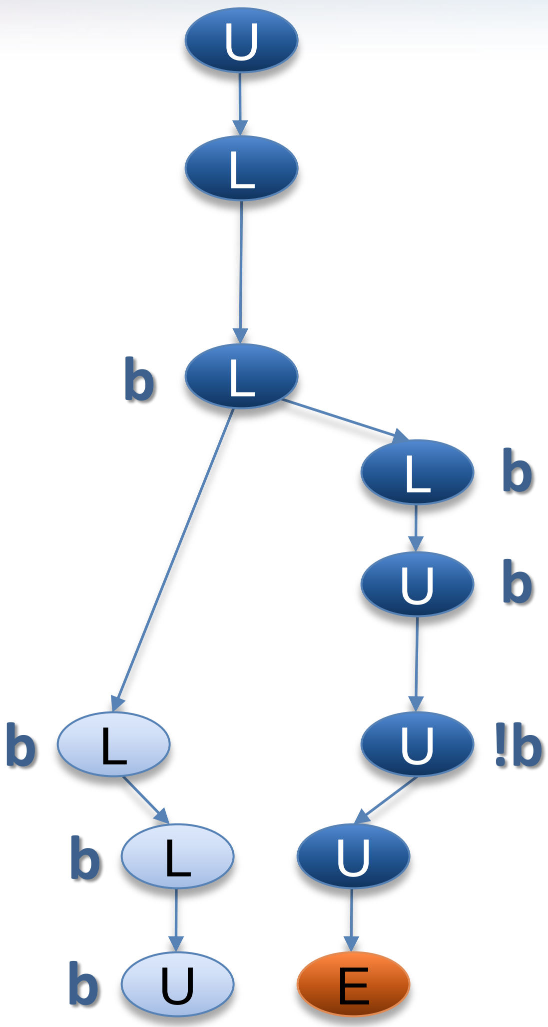
Add new predicate to
Boolean program
b: (nPacketsOld == nPackets)




```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;
     b = true;
   if(request){
        request = request->Next;
        KeReleaseSpinLock();
        nPackets++;
   }         b = b ? false : *;
} while (nPackets != nPacketsOld);
               !b
KeReleaseSpinLock();
```

Example

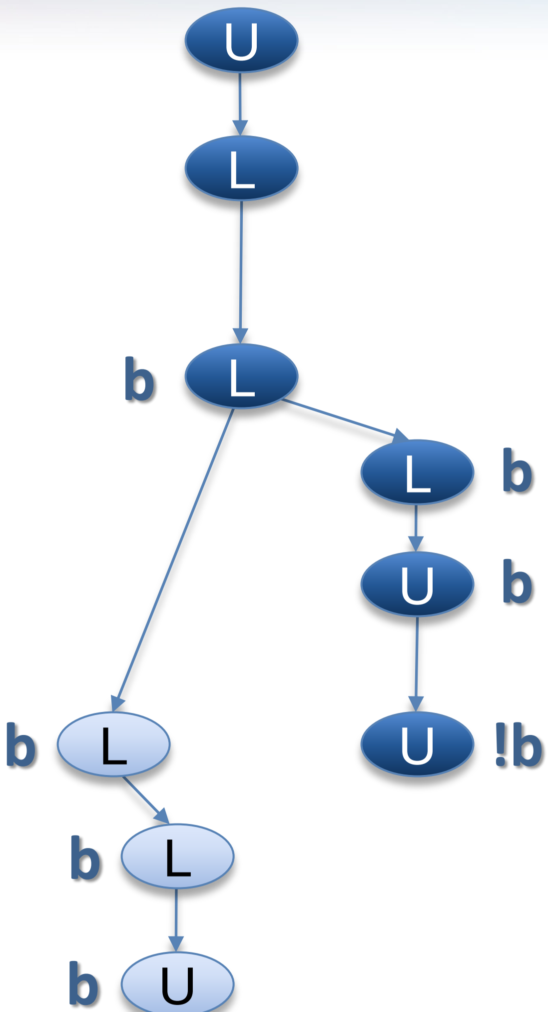
Model Checking
Refined Program
b: (nPacketsOld == nPackets)




```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

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

KeReleaseSpinLock();
```

Example

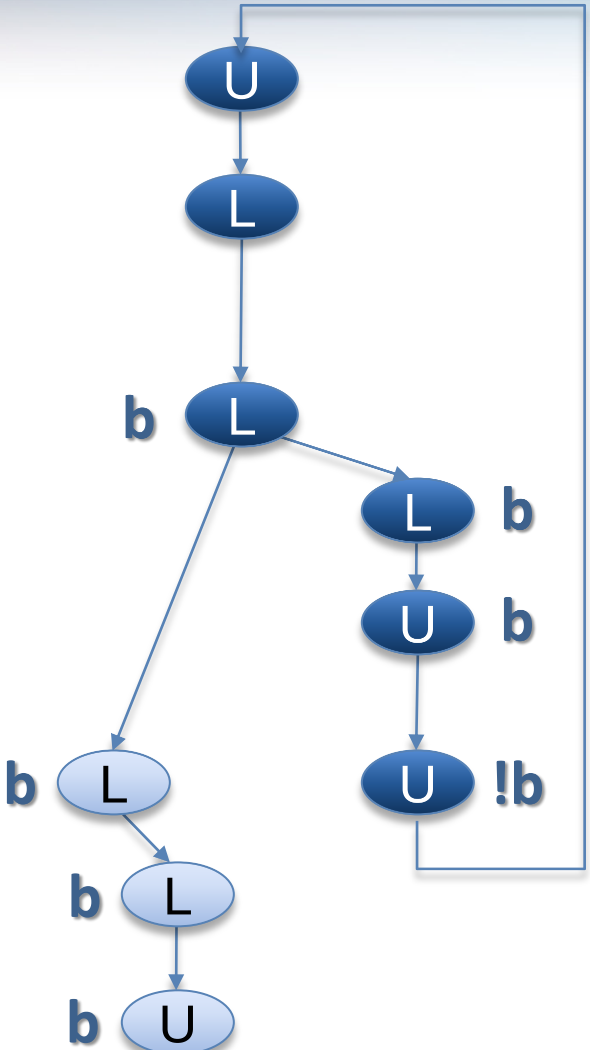
Model Checking
Refined Program
b: (nPacketsOld == nPackets)




```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

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

KeReleaseSpinLock();
```

Example

Model Checking
Refined Program
b: (nPacketsOld == nPackets)




```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

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

KeReleaseSpinLock();
```

Observations about SLAM

- Automatic discovery of invariants
  - driven by property and a finite set of (false) execution paths
  - predicates are ***not*** invariants, but *observations*
  - abstraction + model checking computes inductive invariants (Boolean combinations of observations)

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

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

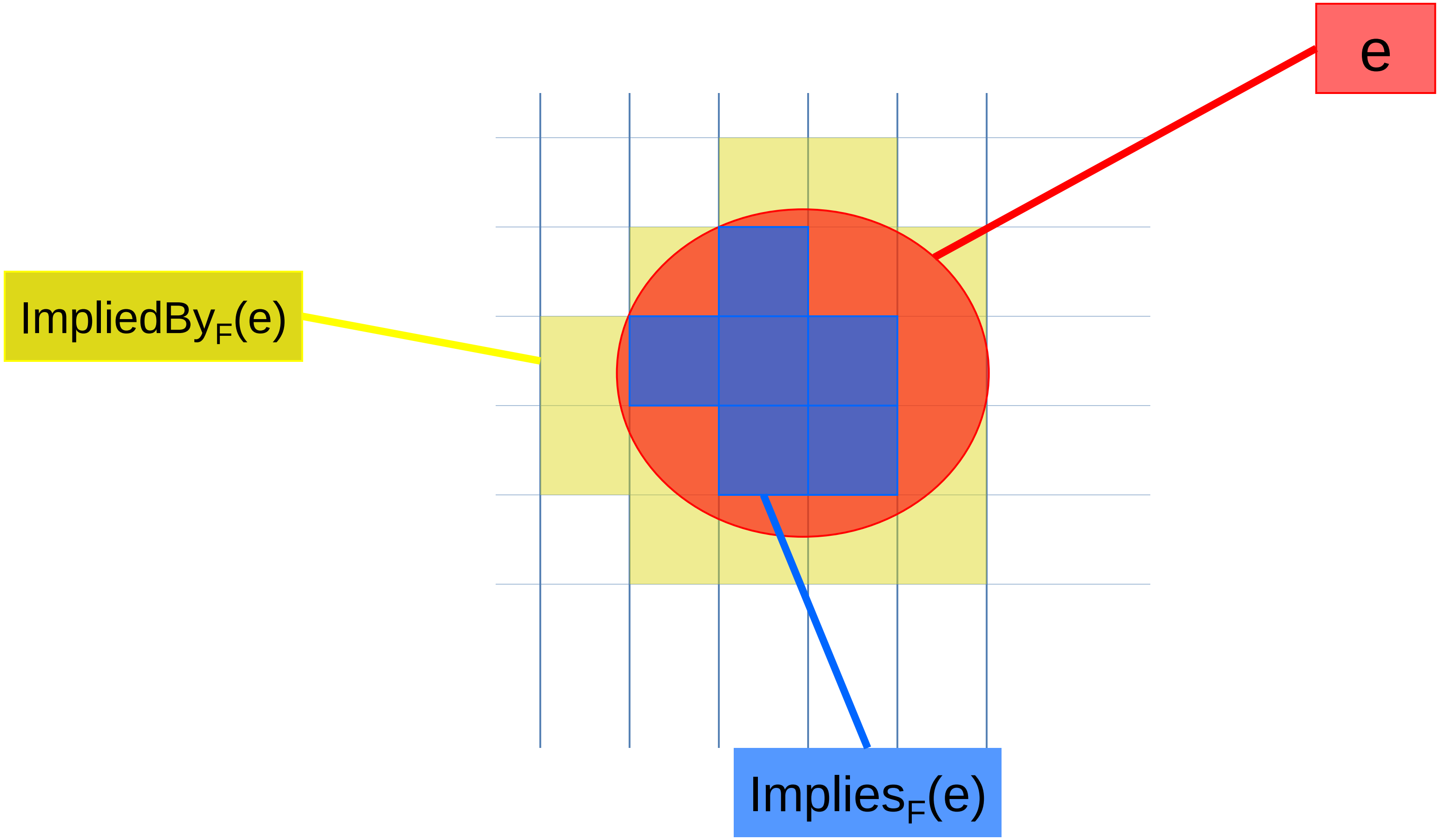
Predicate Abstraction: *c2bp*

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

Abstracting Expressions via *F*

- *$Implies_F$ (e)*
  - Best Boolean function over *F* that implies *e.*
- *$ImpliedBy_F$ (e)*
  - Best Boolean function over *F* that is implied by *e.*
  - *$ImpliedBy_F$ (e) = not $Implies_F$ (not e)*

$Implies_F(e)$ and $ImpliedBy_F(e)$



Computing *Implies$_F$(e)*

- minterm m = l_1 and ... and l_n, where $l_i = p_i$, or l_i = *not* p_i.
- *Implies$_F$(e)*: disjunction of all minterms that imply *e.*
- Naive approach
  - Generate all 2^n possible minterms.
  - For each minterm *m*, use SMT solver to check validity of *m* implies *e*.
- Many possible optimizations

Computing $Implies_F(e)$

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1
  - x<y, !x=2 implies y>1
  - !x<y, x=2 implies y>1
  - x<y, x=2 implies y>1

Computing *Implies$_F$(e)*

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1
  - x<y, !x=2 implies y>1
  - !x<y, x=2 implies y>1
  - x<y, x=2 implies y>1

Computing *Implies*$_F$*(e)*

- F = { x < y, x = 2}
- *e* : y > 1
- Minterms over F
  - !x<y, !x=2 implies y>1
  - x<y, !x=2 implies y>1
  - !x<y, x=2 implies y>1
  - x<y, x=2 implies y>1 ✓

Implies$_F$(y>1) = $x<y \land x=2$

Computing $Implies_F(e)$

- $F = \{ x < y, x = 2\}$
- $e : y > 1$
- Minterms over F
  - $!x<y, !x=2$ implies $y>1$ ⃠
  - $x<y, !x=2$ implies $y>1$ ⃠
  - $!x<y, x=2$ implies $y>1$ ⃠
  - $x<y, x=2$ implies $y>1$ ✓

$Implies_F(y>1) = b_1 \wedge b_2$

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.

Z3 & Static Driver Verifier

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

Bit-precise Scalable Static Analysis

PREfix [Moy, Bjorner, Sielaff 2009]

What is wrong here?

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

Package: java.util.Arrays
Function: binary_search

```
void itoa(int n, char* s) {
    if (n < 0) {
       *s++ = '-';
       n = -n;
    }
    // Add digits to s
    ....
```

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

What is wrong here?

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

```
int binary_se
while (low <= hig
  {
      // Find middle value
      int mid = (low + high) / 2;
      int val = arr[mid];
      if (val == key) return mid;
      if (val < key) low = mid+1;
      else high = mid-1;
   }
   return -1;
```

Package: java.util.Arrays
Function: binary_search

```
id itoa(int n, char* s) {
  if (n < 0) {
     *s++ = '-';
     n = -n;
  }
  // Add digits to s
  ....
```

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

What is wrong here?

-INT_MIN=
INT_MIN

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

```
int binary_se

while (low <= hig
   {
      // Find middle value
      int mid = (low + high) / 2;
      int val = arr[mid];
      if (val == key) return mid;
      if (val < key) low = mid+1;
      else high = mid-1;
   }
   return -1;
```

Package: java.util.Arrays
Function: binary_search

```
id itoa(int n,  ar* s) {
  if (n < 0) {
     *s++ = '-';
     n = -n;
  }
  // Add digits to s
  ….
```

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

The PREfix Static Analysis Engine

```
int init_name(char **outname, uint n)
{
   if (n == 0) return 0;
   else if (n > UINT16_MAX) exit(1);
   else if ((*outname = malloc(n)) == NULL) {
      return 0xC0000095; // NT_STATUS_NO_MEM;
   }
   return 0;
}

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

C/C++ functions

The PREfix Static Analysis Engine

```
int init_name(char **outname, uint n)
{
   if (n == 0) return 0;
   else if (n > UINT16_MAX) exit(1);
   else if ((*outname = malloc(n)) == NULL) {
      return 0xC0000095; // NT_STATUS_NO_MEM;
   }
   return 0;
}

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

C/C++ functions

```
model for function init_name
outcome init_name_0:
   guards: n == 0
   results: result == 0
outcome init_name_1:
    guards: n > 0; n <= 65535
    results: result == 0xC0000095
outcome init_name_2:
    guards: n > 0|; n <= 65535
    constraints: valid(outname)
    results: result == 0; init(*outname)
```

models

The PREfix Static Analysis Engine

```
int init_name(char **outname, uint n)
{
   if (n == 0) return 0;
   else if (n > UINT16_MAX) exit(1);
   else if ((*outname = malloc(n)) == NULL) {
      return 0xC0000095; // NT_STATUS_NO_MEM;
   }
   return 0;
}

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

C/C++ functions

model for function init_name
outcome init_name_0:
guards: n == 0
results: result == 0
outcome init_name_1:
guards: n > 0; n <= 65535
results: result == 0xC0000095
outcome init_name_2:
guards: n > 0|; n <= 65535
constraints: valid(outname)
results: result == 0; init(*outname)

models

path for function get_name
guards: size == 0
constraints:
facts: init(dst); init(size); status == 0

paths

pre-condition for function strcpy
init(dst) and valid(name)

warnings

Overflow on unsigned addition

m_nSize == m_nMaxSize == UINT_MAX

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

iElement + 1 == 0

Write in unallocated memory

Code was written for address space < 4GB

Using an overflown value as allocation size

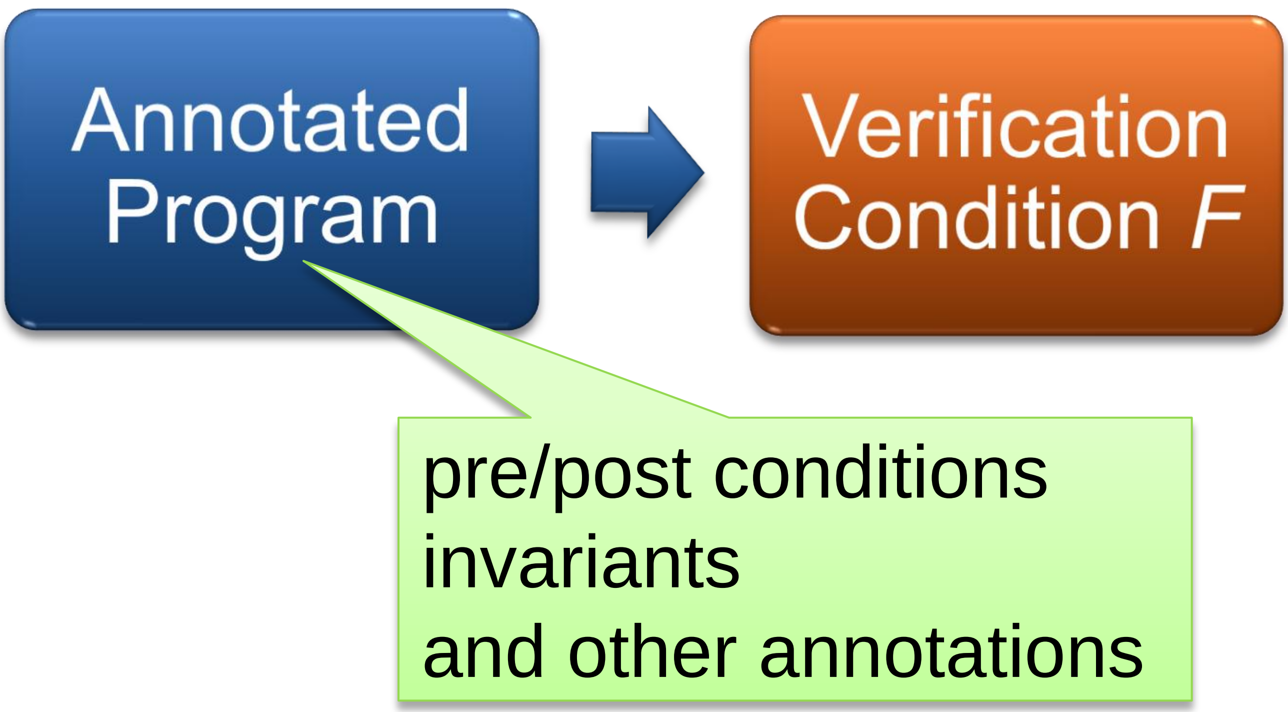
Overflow check

```
ULONG AllocationSize;
while (CurrentBuffer != NULL) {
    if (NumberOfBuffers > MAX_ULONG / sizeof(MYBUFFER)) {
        return NULL;
    }
    NumberOfBuffers++;
    CurrentBuffer = CurrentBuffer->NextBuffer;
}
AllocationSize = sizeof(MYBUFFER)*NumberOfBuffers;
UserBuffersHead = malloc(AllocationSize);
```

Increment and exit from loop

Possible overflow

Verifying Compilers



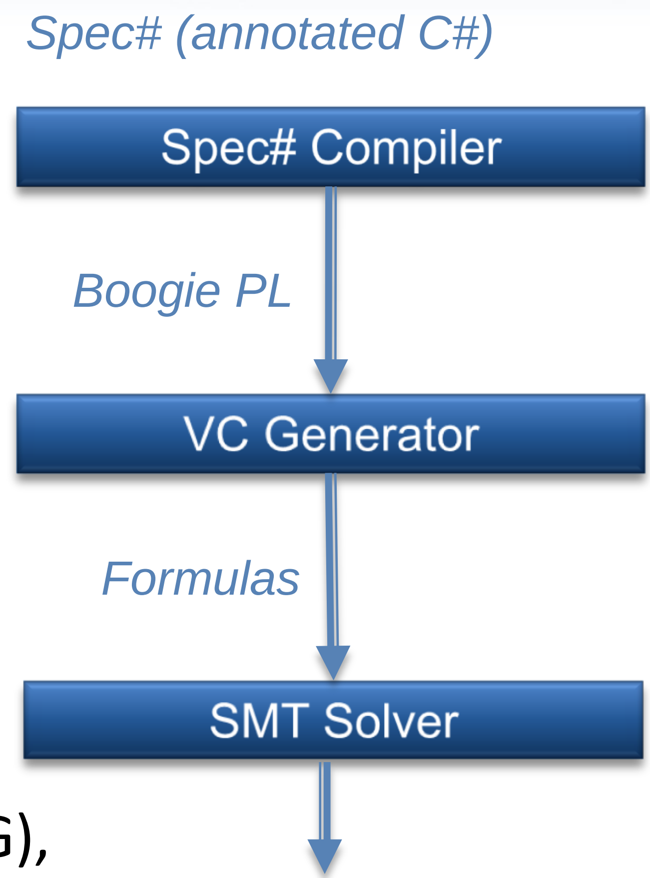
Annotations: Example

```
class C {
    private int a, z;
    invariant z > 0

    public void M()
        requires a != 0
        {
            z = 100/a;
        }
}
```

Microsoft Research

Spec# Approach for a Verifying Compiler

- *Source Language*
  - C# + goodies = Spec#
- *Specifications*
  - method contracts,
  - invariants,
  - field and type annotations.
- *Program Logic:*
  - *Dijkstra's weakest preconditions.*
- *Automatic Verification*
  - type checking,
  - verification condition generation (VCG),
  - **SMT**



Microsoft Research

Command language

- x := E
  - x := x + 1
  - x := 10
- havoc x
- S ; T
- assert P
- assume P
- S □ T

Reasoning about execution traces

- Hoare triple { P } S { Q } says that

  every terminating execution trace of S that starts in a state satisfying P

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

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 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) =	$P \wedge Q$
wp(assume P, Q) =	$P \Rightarrow Q$
wp(S ; T, Q) =	wp(S, wp(T, Q))
wp($S \square T$, Q) =	wp(S, Q) $\wedge$ wp(T, Q)

Structured if statement

if E then S else T end =

assume E; S

□

assume ¬E; T

While loop with loop invariant

```
while E
    invariant J
do
    S
end
```

where x denotes the assignment targets of S

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

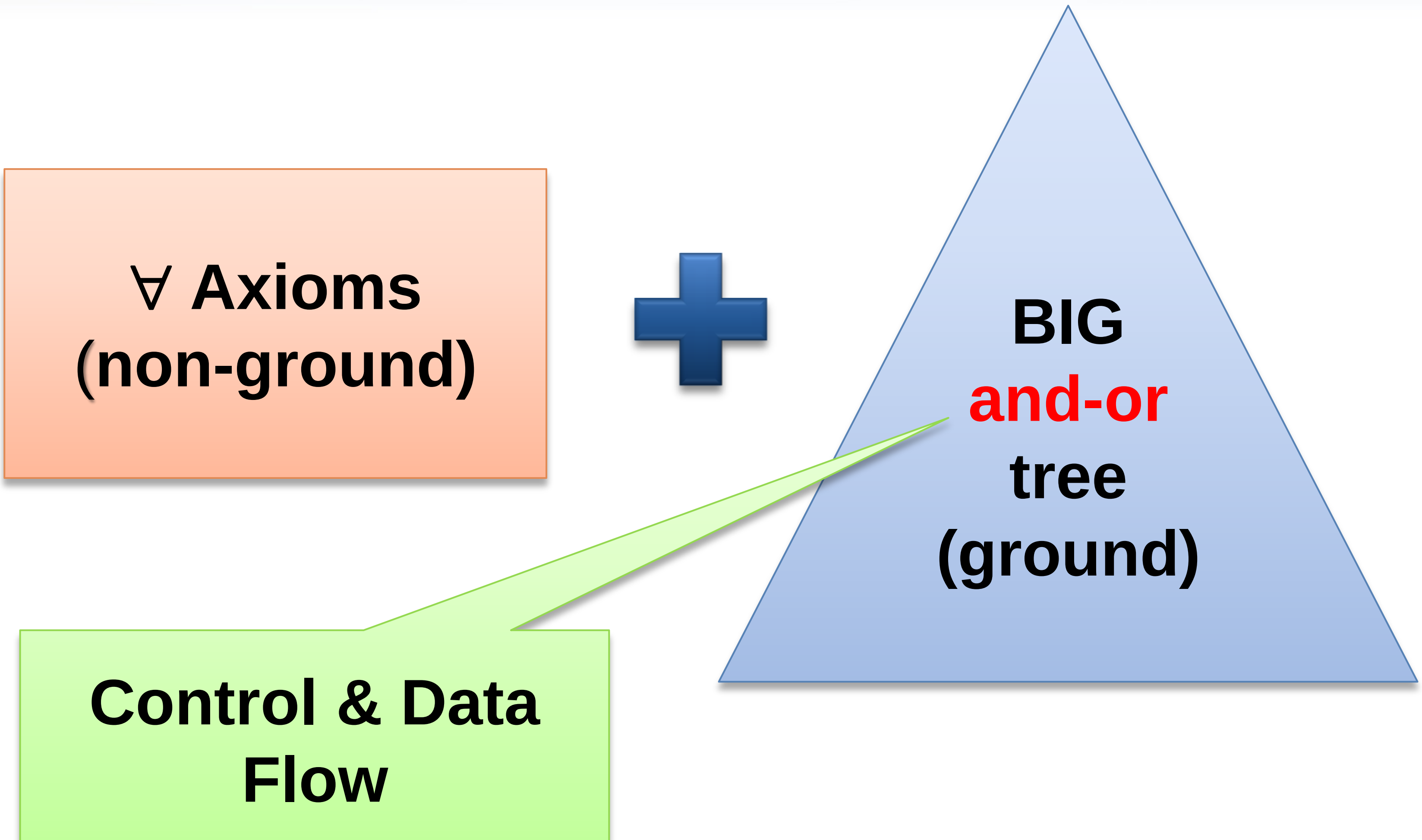
- check that the loop invariant holds initially (→ assert J)
- "fast forward" to an arbitrary iteration of the loop (→ havoc x; assume J;)
- check that the loop invariant is maintained by the loop body (→ assume false)

Microsoft Research

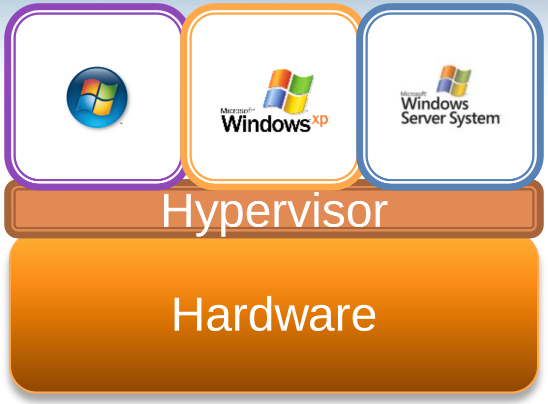
Spec# Chunker.NextChunk translation

```
procedure Chunker.NextChunk(this: ref where $IsNotNull(this, Chunker)) returns ($result: ref where $IsNotNull($result, System.String));
  // in-parameter:  target object
  free requires $Heap[this, $allocated];
  requires ($Heap[this, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[this, $ownerRef], $inv] <: $Heap[this, $ownerFrame]) ||
    $Heap[$Heap[this, $ownerRef], $localinv] == $BaseClass($Heap[this, $ownerFrame])) && (forall $pc: ref :: $pc != null && $Heap[$pc, $allocated]
    && $Heap[$pc, $ownerRef] == $Heap[this, $ownerRef] && $Heap[$pc, $ownerFrame] == $Heap[this, $ownerFrame] ==> $Heap[$pc, $inv] ==
    $typeof($pc) && $Heap[$pc, $localinv] == $typeof($pc));
  // out-parameter:  return value
  free ensures $Heap[$result, $allocated];
  ensures ($Heap[$result, $ownerFrame] == $PeerGroupPlaceholder || !($Heap[$Heap[$result, $ownerRef], $inv] <: $Heap[$result, $ownerFrame]) ||
    $Heap[$Heap[$result, $ownerRef], $localinv] == $BaseClass($Heap[$result, $ownerFrame])) && (forall $pc: ref :: $pc != null && $Heap[$pc,
    $allocated] && $Heap[$pc, $ownerRef] == $Heap[$result, $ownerRef] && $Heap[$pc, $ownerFrame] == $Heap[$result, $ownerFrame] ==>
    $Heap[$pc, $inv] == $typeof($pc) && $Heap[$pc, $localinv] == $typeof($pc));
  // user-declared postconditions
  ensures $StringLength($result) <= $Heap[this, Chunker.ChunkSize];
  // frame condition
  modifies $Heap;
  free ensures (forall $o: ref, $f: name :: { $Heap[$o, $f] } $f != $inv && $f != $localinv && $f != $FirstConsistentOwner && (!IsStaticField($f) ||
    !IsDirectlyModifiableField($f)) && $o != null && old($Heap)[$o, $allocated] && (old($Heap)[$o, $ownerFrame] == $PeerGroupPlaceholder ||
    !(old($Heap)[old($Heap)[$o, $ownerRef], $inv] <: old($Heap)[$o, $ownerFrame]) || old($Heap)[old($Heap)[$o, $ownerRef], $localinv] ==
    $BaseClass(old($Heap)[$o, $ownerFrame])) && old($o != this || !(Chunker <: DeclType($f)) || !$IncludedInModifiesStar($f)) && old($o != this || $f
    != $exposeVersion) ==> old($Heap)[$o, $f] == $Heap[$o, $f]);
  // boilerplate
  free requires $BeingConstructed == null;
  free ensures (forall $o: ref :: { $Heap[$o, $localinv] } { $Heap[$o, $inv] } $o != null && !old($Heap)[$o, $allocated] && $Heap[$o, $allocated] ==>
    $Heap[$o, $inv] == $typeof($o) && $Heap[$o, $localinv] == $typeof($o));
  free ensures (forall $o: ref :: { $Heap[$o, $FirstConsistentOwner] } old($Heap)[old($Heap)[$o, $FirstConsistentOwner], $exposeVersion] ==
    $Heap[old($Heap)[$o, $FirstConsistentOwner], $exposeVersion] ==> old($Heap)[$o, $FirstConsistentOwner] == $Heap[$o,
    $FirstConsistentOwner]);
  free ensures (forall $o: ref :: { $Heap[$o, $localinv] } { $Heap[$o, $inv] } old($Heap)[$o, $allocated] ==> old($Heap)[$o, $inv] == $Heap[$o, $inv] &&
    old($Heap)[$o, $localinv] == $Heap[$o, $localinv]);
  free ensures (forall $o: ref :: { $Heap[$o, $allocated] } old($Heap)[$o, $allocated] ==> $Heap[$o, $allocated]) && (forall $ot: ref :: { $Heap[$ot,
    $ownerFrame] } { $Heap[$ot, $ownerRef] } old($Heap)[$ot, $allocated] && old($Heap)[$ot, $ownerFrame] != $PeerGroupPlaceholder ==>
    old($Heap)[$ot, $ownerRef] == $Heap[$ot, $ownerRef] && old($Heap)[$ot, $ownerFrame] == $Heap[$ot, $ownerFrame]) &&
    old($Heap)[$BeingConstructed, $NonNullFieldsAreInitialized] == $Heap[$BeingConstructed, $NonNullFieldsAreInitialized];
```

Verification conditions: Structure



Hypervisor: A Manhattan Project

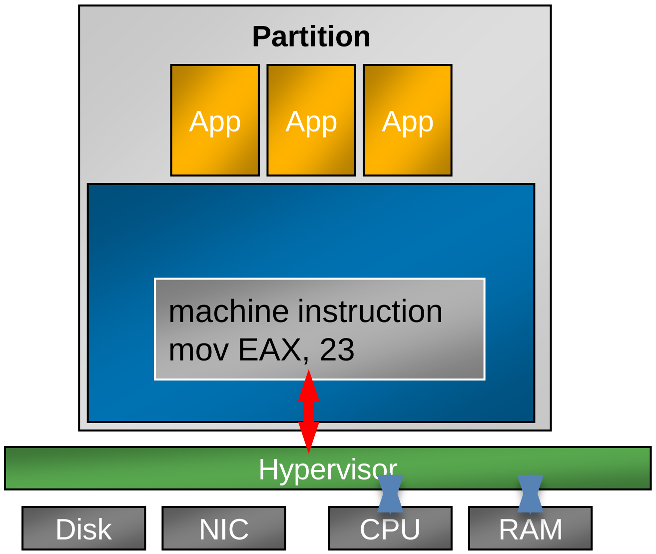


- **Meta OS**: small layer of software between hardware and OS
- **Mini**: 100K lines of non-trivial concurrent systems C code
- **Critical:** must provide functional resource abstraction
- **Trusted**: a verification grand challenge

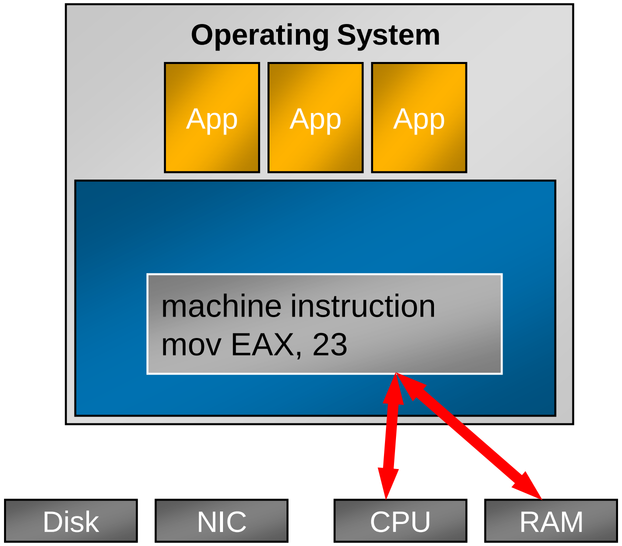
HV Correctness: Simulation

A partition cannot distinguish (with some exceptions)
whether a machine instruction is executed

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



≈



Hypervisor Implementation

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

Hypervisor Verification (2007 – 2010)

Partners:

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

co-funded by the German Ministry of Education and Research

http://www.verisoftxt.de

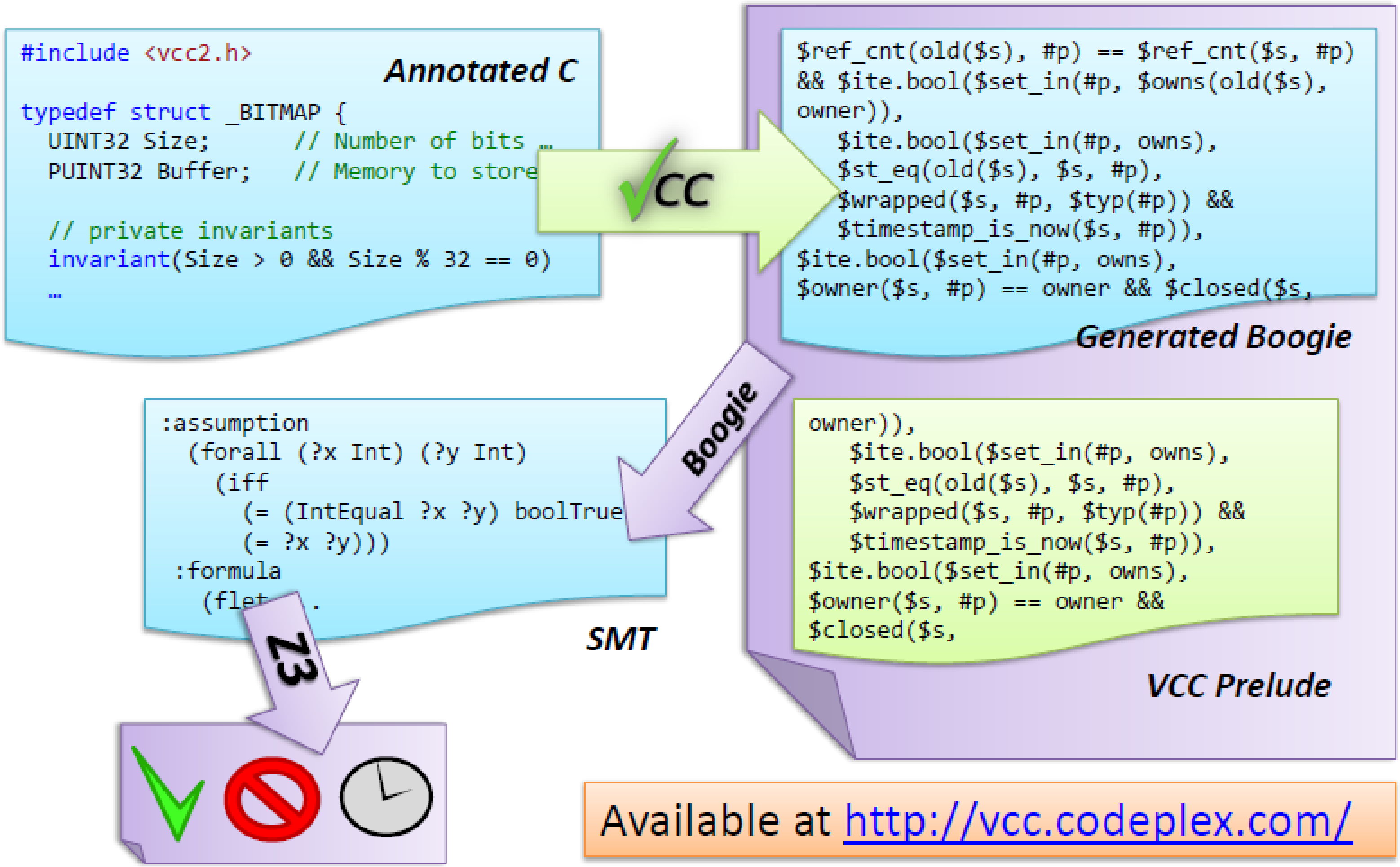
Challenges for Verification of Concurrent C

1. **Memory model** that is adequate and efficient to reason about
2. **Modular reasoning** about concurrent code
3. **Invariants** for (large and complex) C data structures
4. Huge verification conditions to be proven **automatically**
5. “Live” specifications that **evolve with the code**

The Microsoft Verifying C Compiler (VCC)

- Source Language
  - ANSI C +
  - Design-by-Contract Annotations +
  - Ghost state +
  - Theories +
  - Metadata Annotations
- Program Logic
  - Dijkstra's weakest preconditions
- Automatic Verification
  - verification condition generation (VCG)
  - automatic theorem proving (SMT)

VCC Architecture



Available at http://vcc.codeplex.com/

Contracts / Modular Verification

```
int foo(int x)
  requires(x > 5)        // precond
  ensures(result > x)    // postcond
{
…
}
```

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

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

advantages:

- modular verification: one function at a time
- no unfolding of code: scales to large applications

Hypervisor: Some Statistics

- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

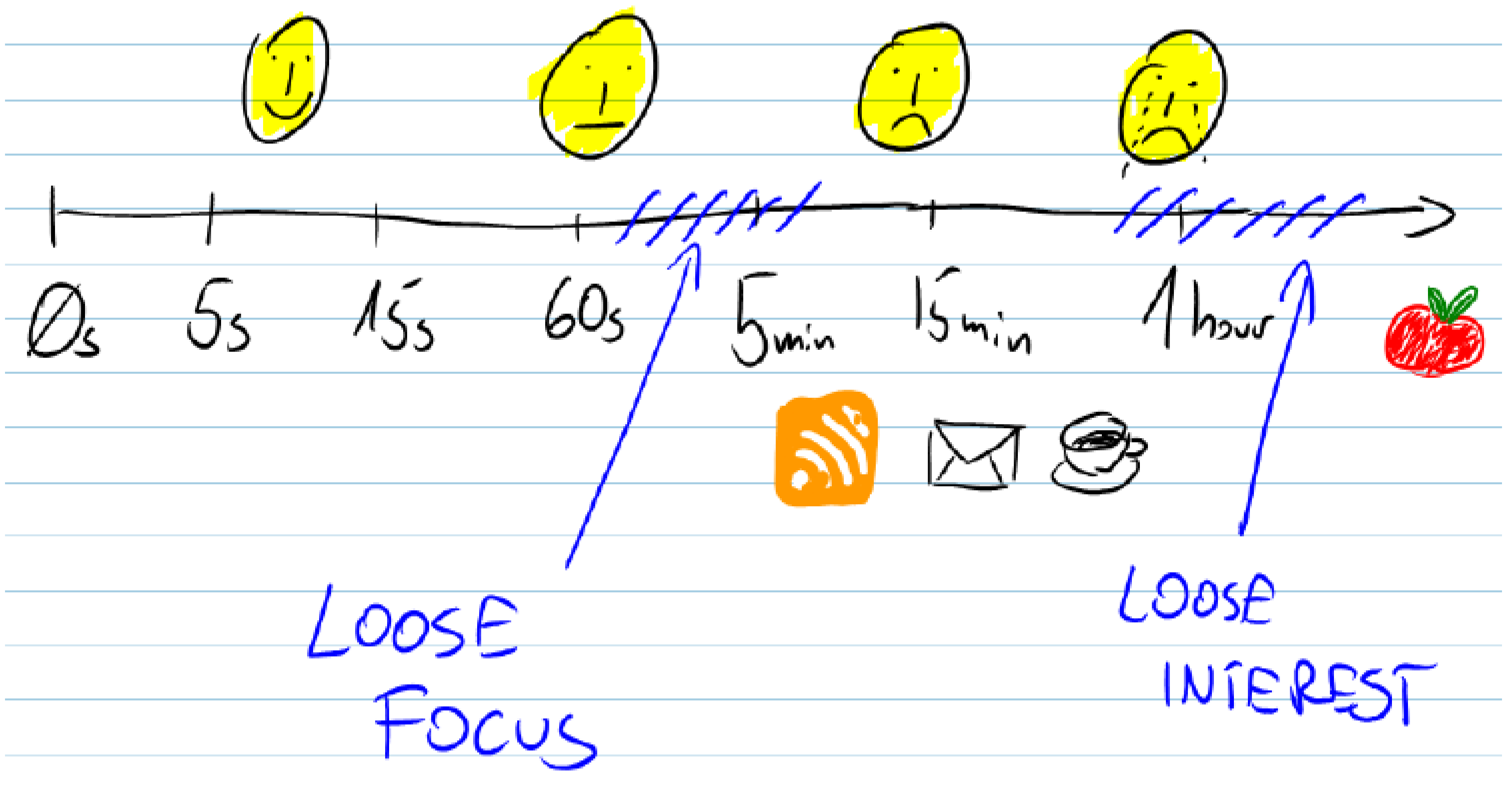
Microsoft Research

Hypervisor: Some Statistics

- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

Are you willing to wait more than
5 min for your compiler?

Verification Attempt Time vs. Satisfaction and Productivity



By Michal Moskal (VCC Designer and Software Verification Expert)

Why did my proof attempt fail?

1. My annotations are not strong enough!

weak loop invariants and/or contracts

2. My theorem prover is not strong (or fast) enough.

Send “angry” email to Nikolaj and Leo.

Microsoft Research

Challenge

- Quantifiers, quantifiers, quantifiers, …
- Modeling the runtime

$\forall$ h,o,f:

IsHeap(h) $\wedge$ o $\neq$ null $\wedge$ read(h, o, alloc) = t

$\Rightarrow$

read(h,o, f) = null $\vee$ read(h, read(h,o,f),alloc) = t

Microsoft Research

Challenge

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

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

Challenge

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

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

Microsoft Research

Challenge

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

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

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

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

Challenge

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

We want to find bugs!

Bad news

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

Microsoft Research

Many Approaches

- Heuristic quantifier instantiation
- Combining SMT with Saturation provers
- Complete quantifier instantiation
- Decidable fragments
- Model based quantifier instantiation

Challenge: Modeling Runtime

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

Microsoft Research

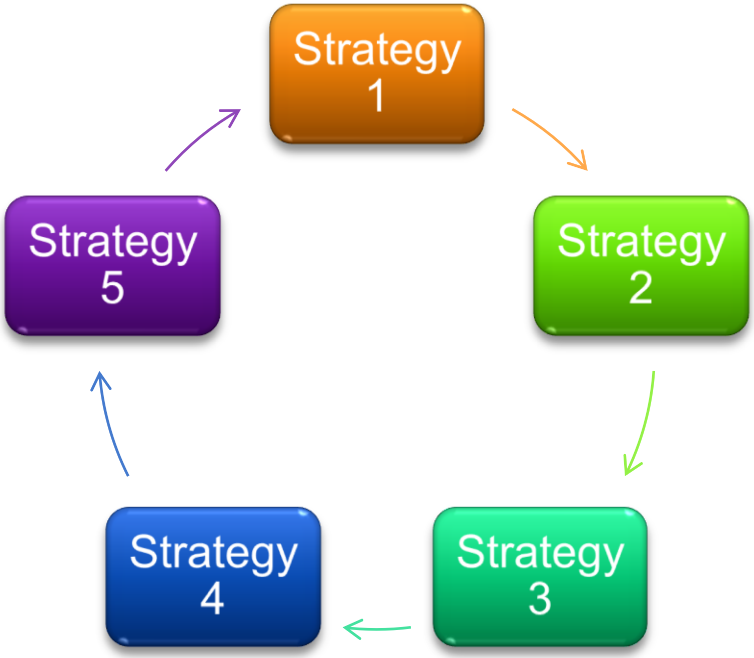
Challenge: Robustness

- Standard complain

  "I made a small modification in my Spec, and Z3 is timingout"
- This also happens with SAT solvers (NP-complete)
- In our case, the problems are undecidable
- Partial solution: parallelization

Parallel Z3

- Joint work with Y. Hamadi (MSRC) and C. Wintersteiger
- Multi-core & Multi-node (HPC)
- Different strategies in parallel
- Collaborate exchanging lemmas



Microsoft Research

Hey, I don't trust these proofs

Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

Microsoft Research

Hey, I don't trust these proofs

Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.

Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)

Microsoft Research

Hey, I don't trust these proofs

Z3 may be buggy.

Solution: proof/certificate generation.

Engineering problem: these certificates are too big.

The Axiomatization of the runtime may be buggy or inconsistent.

Yes, this is true. We are working on new techniques for proving satisfiability (building a model for these axioms)

The VCG generator is buggy (i.e., it makes the wrong assumptions)

Do you trust your compiler?

Microsoft Research

Engineer Perspective

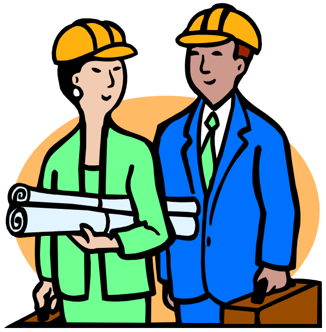
These are bug-finding tools!

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

I add Loop invariants to speedup the process.

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

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



Microsoft Research

Conclusion

Powerful, mature, and versatile tools like SMT solvers can now be exploited in very useful ways.

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

SMT is hot at Microsoft.

Z3 is a new SMT solver.

Main applications:

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

Books

- Bradley & Manna: The Calculus of Computation
- Kroening & Strichman: Decision Procedures, An Algorithmic Point of View
- Chapter in the Handbook of Satisfiability

Web Links

Z3:

http://research.microsoft.com/projects/z3

http://research.microsoft.com/~leonardo

- Slides & Papers

http://www.smtlib.org

http://www.smtcomp.org

References

[Ack54] W. Ackermann. Solvable cases of the decision problem. *Studies in Logic and the Foundation of Mathematics*, 1954

[ABC$^+$02] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In *Proc. of CADE'02*, 2002

[BDS00] C. Barrett, D. Dill, and A. Stump. A framework for cooperating decision procedures. In *17th International Conference on Computer-Aided Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 79–97. Springer-Verlag, 2000

[BdMS05] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability Modulo Theories Competition. In *Int. Conference on Computer Aided Verification (CAV'05)*, pages 20–23. Springer, 2005

[BDS02] C. Barrett, D. Dill, and A. Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer-Verlag, July 2002. Copenhagen, Denmark

[BBC$^+$05] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In *Int. Conf. on Computer-Aided Verification (CAV)*, volume 3576 of *LNCS*. Springer, 2005

[Chv83] V. Chvatal. *Linear Programming*. W. H. Freeman, 1983

References

[CG96] B. Cherkassky and A. Goldberg. Negative-cycle detection algorithms. In *European Symposium on Algorithms*, pages 349–363, 1996

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962

[DNS03] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A theorem prover for program checking. Technical Report HPL-2003-148, HP Labs, 2003

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan. Variations on the Common Subexpression Problem. *Journal of the Association for Computing Machinery*, 27(4):758–771, 1980

[dMR02] L. de Moura and H. Rueß. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*. Cincinnati, Ohio, 2002

[DdM06] B. Dutertre and L. de Moura. Integrating simplex with DPLL(T). Technical report, CSL, SRI International, 2006

[dMB07b] L. de Moura and N. Bjørner. Efficient E-Matching for SMT solvers. In *CADE-21*, pages 183–198, 2007

References

[dMB07c] L. de Moura and N. Bjørner. Model Based Theory Combination. In *SMT'07*, 2007

[dMB07a] L. de Moura and N. Bjørner. Relevancy Propagation . Technical Report MSR-TR-2007-140, Microsoft Research, 2007

[dMB08a] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS 08*, 2008

[dMB08c] L. de Moura and N. Bjørner. Engineering DPLL(T) + Saturation. In *IJCAR'08*, 2008

[dMB08b] L. de Moura and N. Bjørner. Deciding Effectively Propositional Logic using DPLL and substitution sets. In *IJCAR'08*, 2008

[GHN$^+$04] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In R. Alur and D. Peled, editors, *Int. Conference on Computer Aided Verification (CAV 04)*, volume 3114 of *LNCS*, pages 175–188. Springer, 2004

[MSS96] J. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proc. of ICCAD'96*, 1996

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, 1979

[NO05] R. Nieuwenhuis and A. Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Int. Conference on Computer Aided Verification (CAV'05)*, pages 321–334. Springer, 2005

References

[Opp80] D. Oppen. Reasoning about recursively defined data structures. *J. ACM*, 27(3):403–411, 1980

[PRSS99] A. Pnueli, Y. Rodeh, O. Shtrichman, and M. Siegel. Deciding equality formulas by small domains instantiations. *Lecture Notes in Computer Science*, 1633:455–469, 1999

[Pug92] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Communications of the ACM*, volume 8, pages 102–114, August 1992

[RT03] S. Ranise and C. Tinelli. The smt-lib format: An initial proposal. In *Proceedings of the 1st International Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR'03), Miami, Florida*, pages 94–111, 2003

[RS01] H. Ruess and N. Shankar. Deconstructing shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, June 2001

[SLB03] S. Seshia, S. Lahiri, and R. Bryant. A hybrid SAT-based decision procedure for separation logic with uninterpreted functions. In *Proc. 40th Design Automation Conference*, pages 425–430. ACM Press, 2003

[Sho81] R. Shostak. Deciding linear inequalities by computing loop residues. *Journal of the ACM*, 28(4):769–779, October 1981

References

[dMB09] L. de Moura and N. Bjørner. Generalized and Efficient Array Decision Procedures. FMCAD, 2009.

[GdM09] Y. Ge and L. de Moura. Complete Quantifier Instantiation for quantified SMT formulas, CAV, 2009.