Satisfiability Modulo Theories

Natal 2012

Leonardo de Moura
Microsoft Research

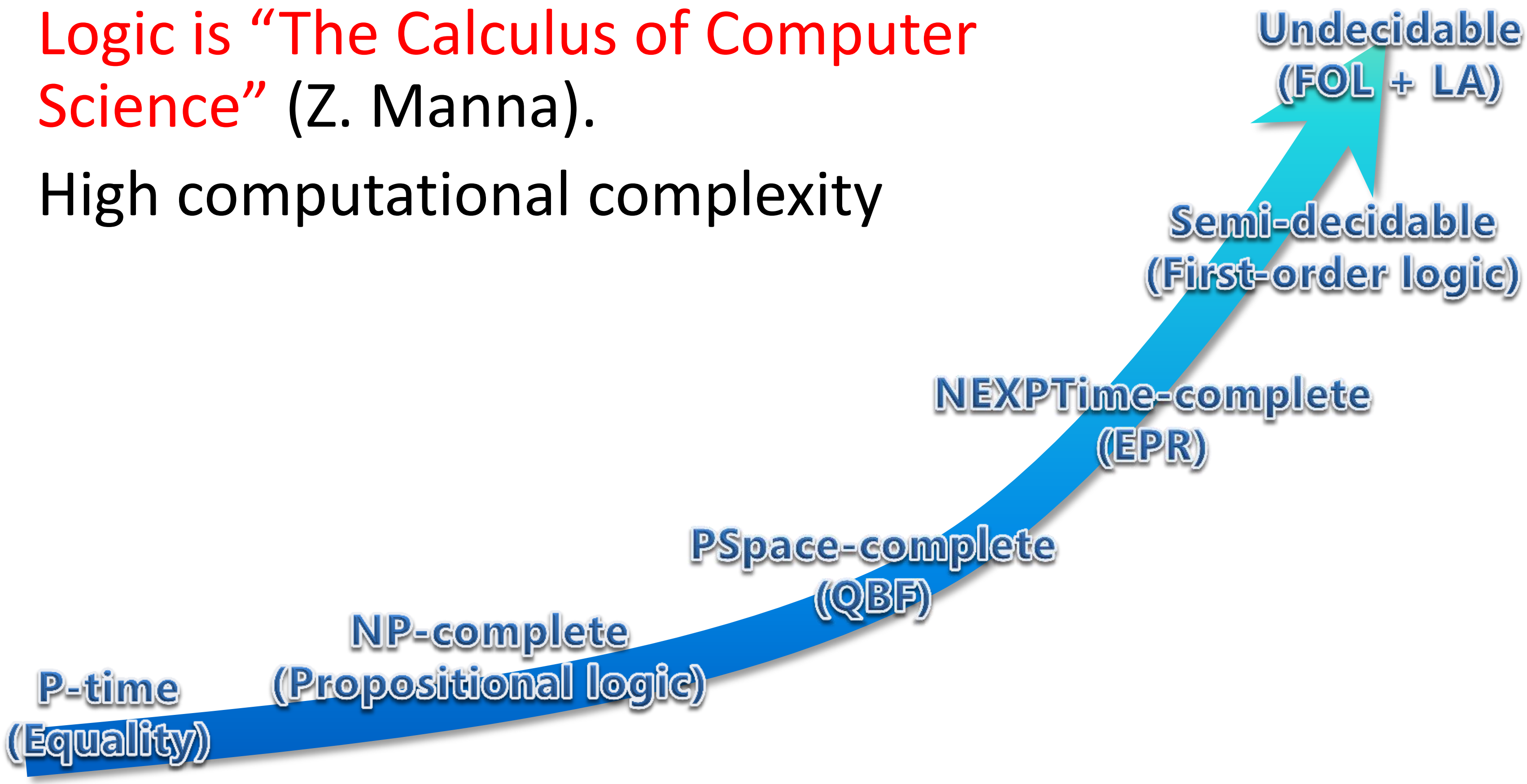
Symbolic Reasoning

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

Microsoft Research

Symbolic Reasoning

- Logic is "The Calculus of Computer Science" (Z. Manna).
- High computational complexity



Microsoft Research

Applications

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

Microsoft® Research

Some Applications @ Microsoft

- The Spec# Programming System
- HAVOC
- ForμLa
- Hyper-V — Microsoft Virtualization
- Terminator T-2
- VCC
- SLAM
- Yogi
- NModel
- Vigilante
- SpecExplorer
- Pex
- F7
- SAGE

Microsoft Research

Test case generation

```
unsigned GCD(x, y) {
  requires(y > 0);
  while (true) {
    unsigned m = x % y;
    if (m == 0) return y;
    x = y;
    y = m;
  }
}
```

SSA

$(y_0 > 0)$ and

$(m_0 = x_0 \% y_0)$ and

not $(m_0 = 0)$ and

$(x_1 = y_0)$ and

$(y_1 = m_0)$ and

$(m_1 = x_1 \% y_1)$ and

$(m_1 = 0)$

Solver

$x_0 = 2$

$y_0 = 4$

$m_0 = 2$

$x_1 = 4$

$y_1 = 2$

$m_1 = 0$

We want a trace where the loop is executed twice.

Microsoft Research

Type checking

Signature:

div : int, $\{ x : \text{int} \mid x \neq 0 \} \rightarrow$ int

Subtype

Call site:

if $a \leq 1$ and $a \leq b$ then

return div(a, b)

Verification condition

$a \leq 1$ and $a \leq b$ implies $b \neq 0$

Satisfiability Modulo Theories (SMT)

Is formula *F* satisfiable modulo theory *T* ?

SMT solvers have specialized algorithms for *T*

Satisfiability Modulo Theories (SMT)

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

Microsoft Research

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

Microsoft® Research

Satisfiability Modulo Theories (SMT)

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

Array Theory

Microsoft Research

Satisfiability Modulo Theories (SMT)

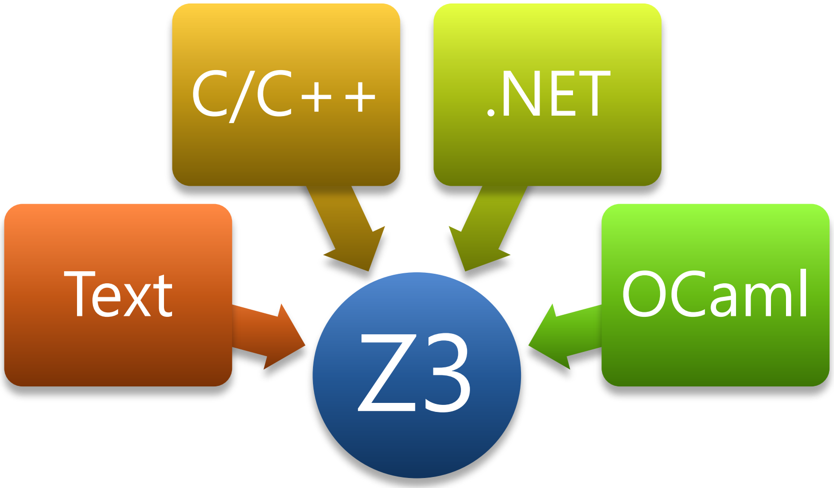
$$b + 2 = c \ \text{ and } \ \boxed{f}(read(write(a,b,3), c\text{-}2) \neq \boxed{f}(c\text{-}b+1)$$

Uninterpreted Functions

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

Microsoft Research

Ground formulas

For most SMT solvers: ***F is a set of ground formulas***

Many Applications

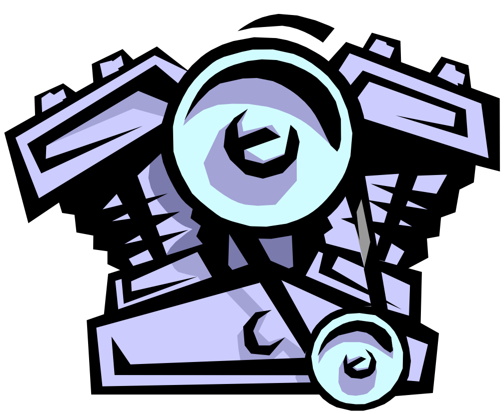
Bounded Model Checking

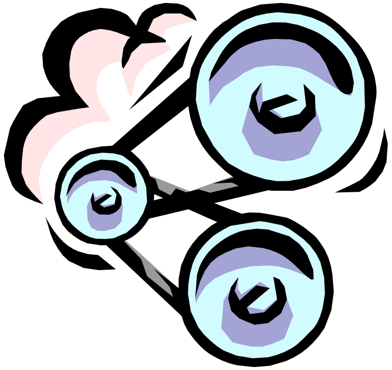
Test-Case Generation

Little Engines of Proof

An SMT Solver is a collection of

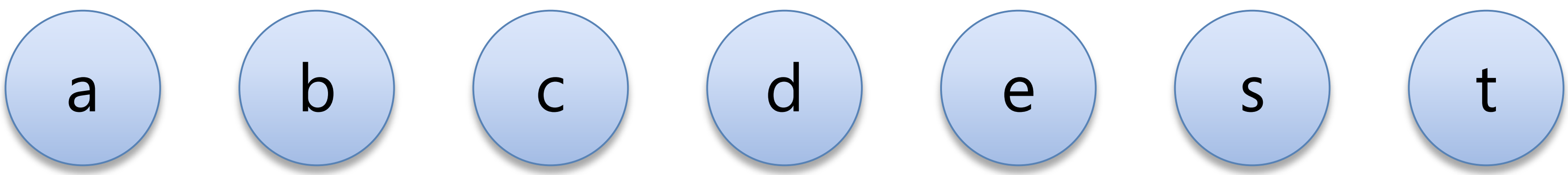
Little Engines of Proof





Deciding Equality

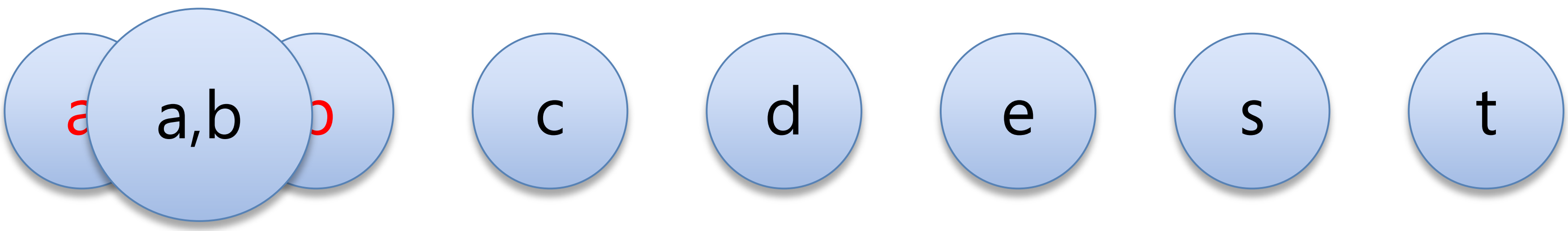
$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$$



Microsoft Research

Deciding Equality

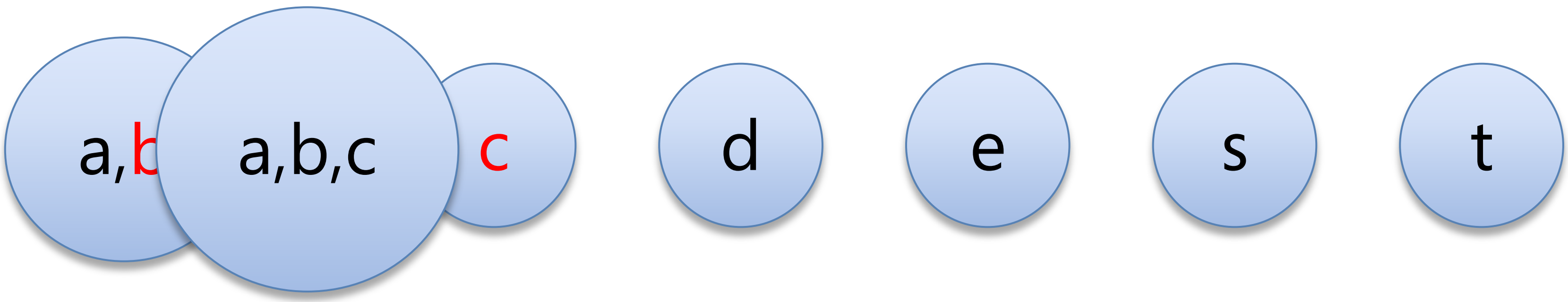
$a = b$, $b = c$, $d = e$, $b = s$, $d = t$, $a \neq e$, $a \neq s$



Microsoft Research

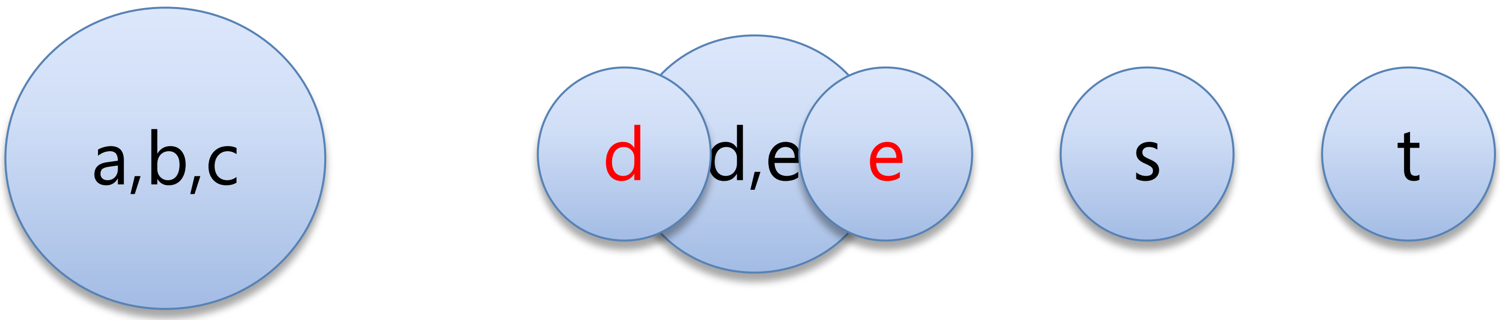
Deciding Equality

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



Deciding Equality

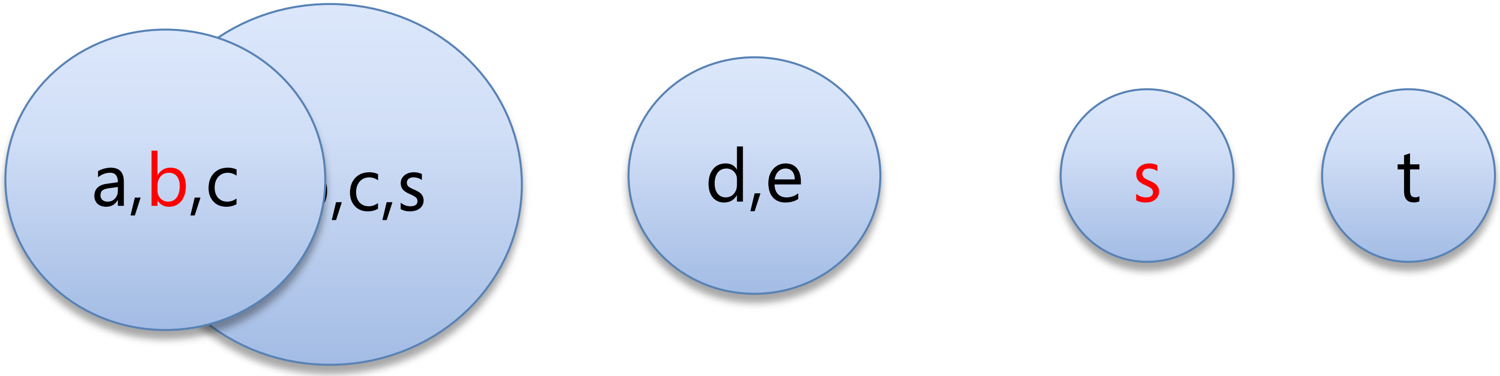
$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$



Microsoft Research

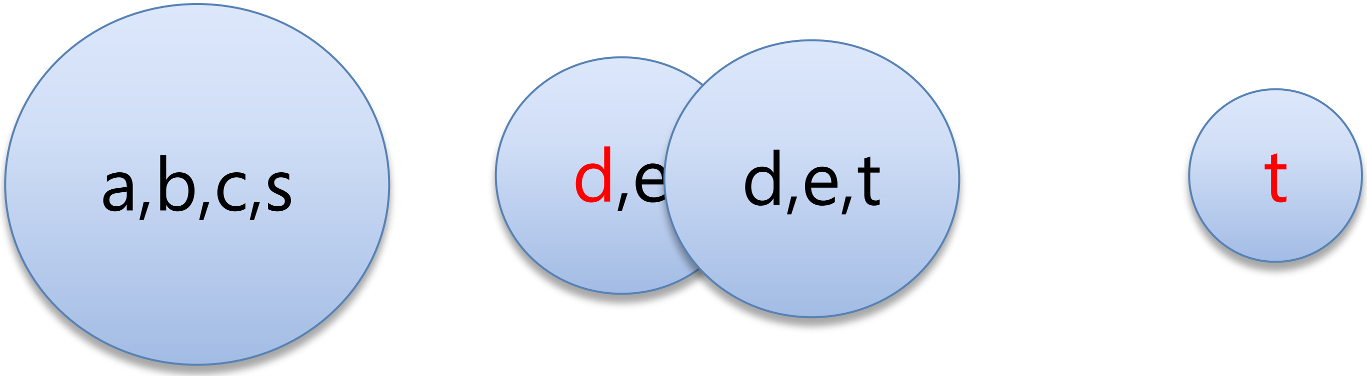
Deciding Equality

$a = b, b = c, d = e, b = s, d = t, a \neq e, a \neq s$



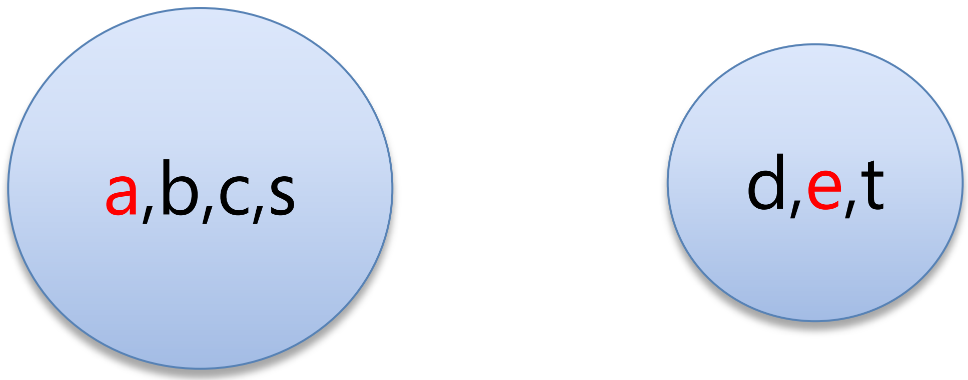
Deciding Equality

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$$



Deciding Equality

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$$



Deciding Equality

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$$

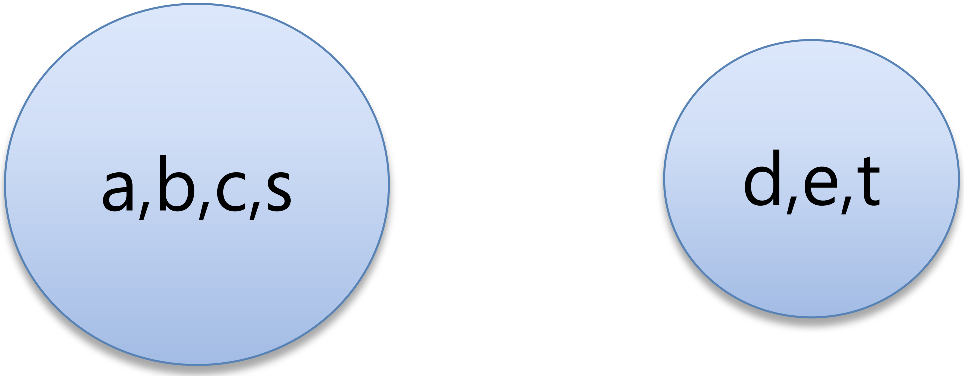
a,b,c,s

d,e,t

Unsatisfiable

Deciding Equality

$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e$



Model

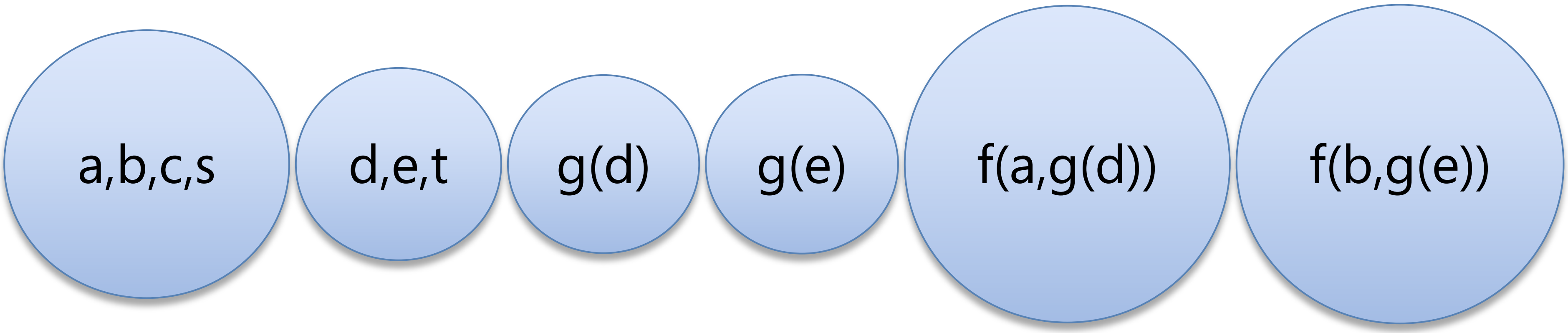
$|M| = \{\ 0, 1\ \}$

$M(a) = M(b) = M(c) = M(s) = 0$

$M(d) = M(e) = M(t) = 1$

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ f(a, g(d)) \neq f(b, g(e))$$

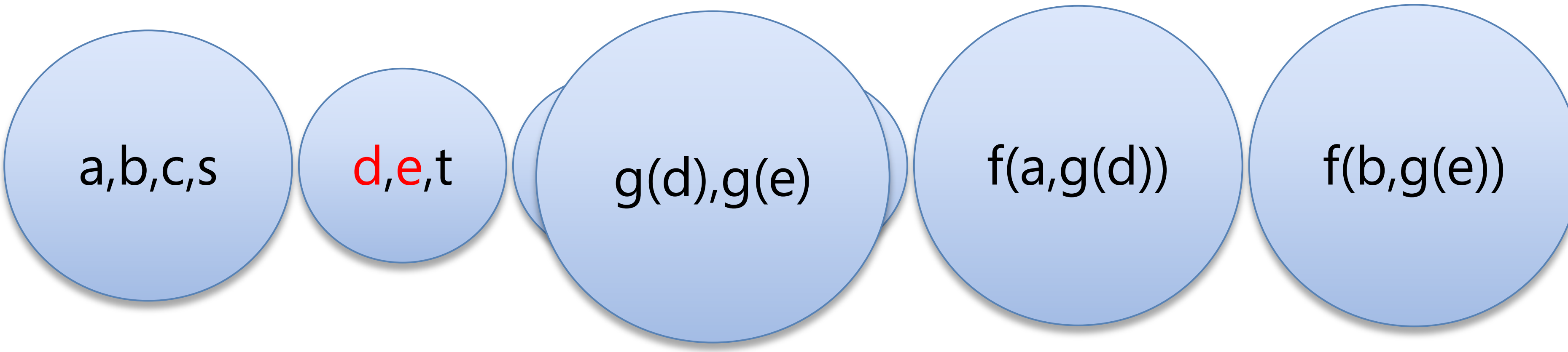


Congruence Rule:

$$x_1 = y_1, \ldots, x_n = y_n \text{ implies } f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ f(a, g(d)) \neq f(b, g(e))$$



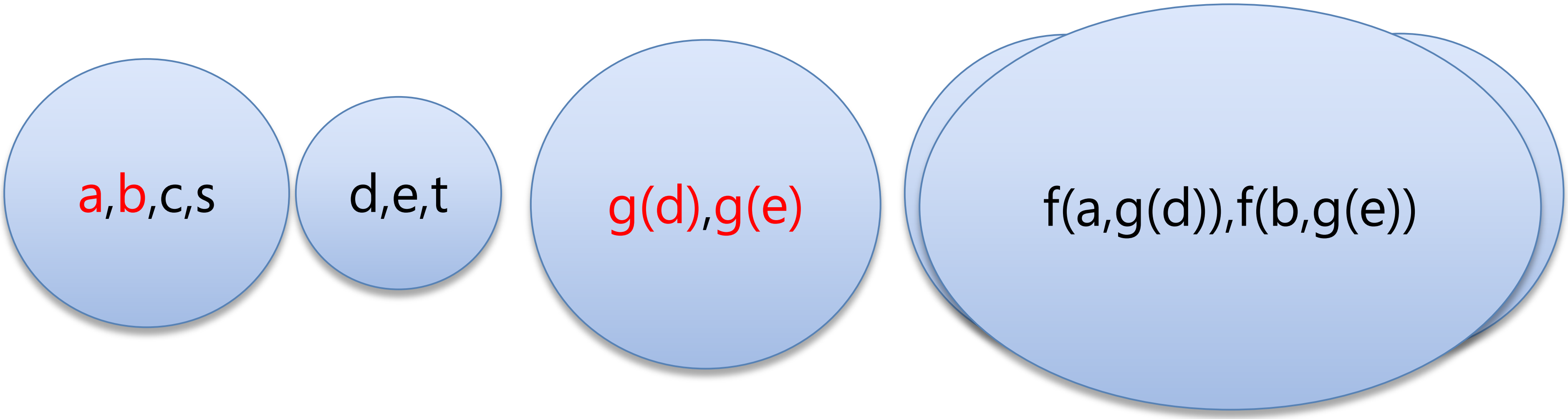
Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Microsoft Research

Deciding Equality + (uninterpreted) Functions

$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ f(a, g(d)) \neq f(b, g(e))$$

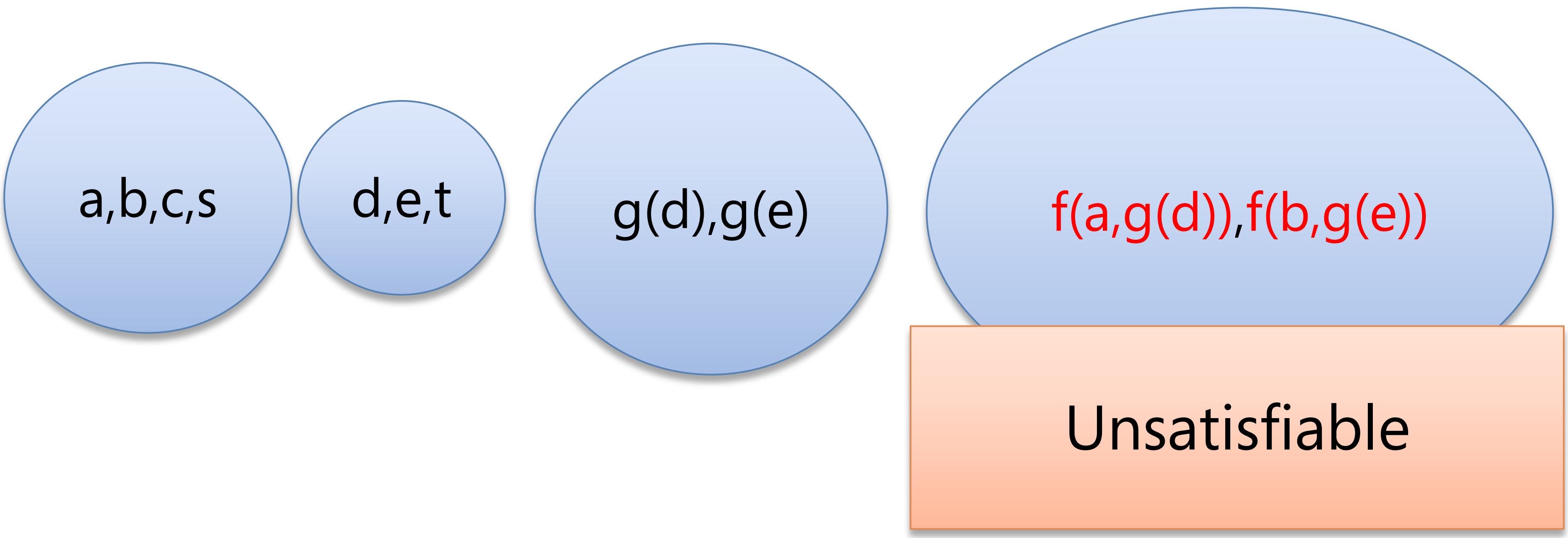


Congruence Rule:

$x_1 = y_1, \ldots, x_n = y_n$ implies $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$

Deciding Equality + (uninterpreted) Functions

$a = b, b = c, d = e, b = s, d = t, f(a, g(d)) \neq f(b, g(e))$



Deciding Equality + (uninterpreted) Functions

(fully shared) DAGs for representing terms

Union-find data-structure + Congruence Closure

$O(n \log n)$

Difference Logic: $a - b \leq 5$

Very useful in practice!

Most arithmetical constraints in software verification/analysis are in this fragment.

$x := x + 1$

$\Downarrow$

$x_1 = x_0 + 1$

$\Downarrow$

$x_1 - x_0 \leq 1,\ x_0 - x_1 \leq -1$

Microsoft Research

Job shop scheduling

$d_{i,j}$	Machine 1	Machine 2
Job 1	2	1
Job 2	3	1
Job 3	2	3

$max = 8$

Solution

$t_{1,1} = 5,\ t_{1,2} = 7,\ t_{2,1} = 2,$
$t_{2,2} = 6,\ t_{3,1} = 0,\ t_{3,2} = 3$

Encoding

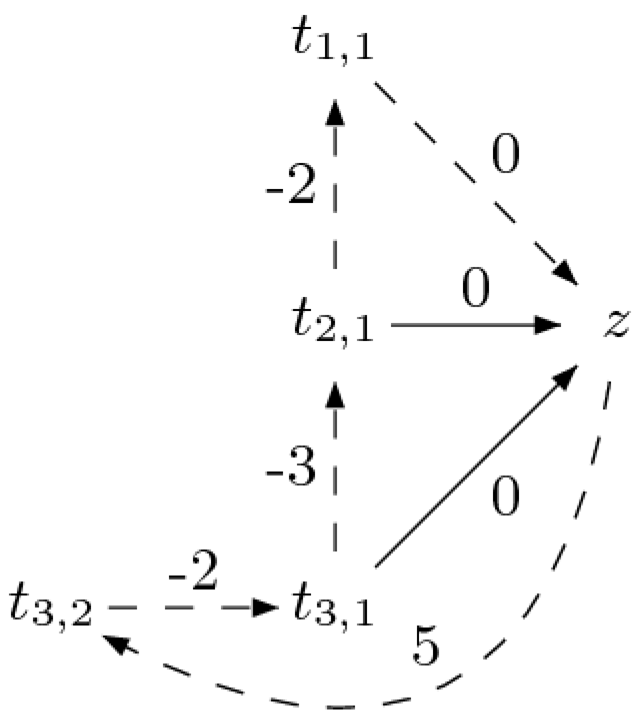
$$
\begin{array}{l}
(t_{1,1} \geq 0) \wedge (t_{1,2} \geq t_{1,1} + 2) \wedge (t_{1,2} + 1 \leq 8) \wedge \\
(t_{2,1} \geq 0) \wedge (t_{2,2} \geq t_{2,1} + 3) \wedge (t_{2,2} + 1 \leq 8) \wedge \\
(t_{3,1} \geq 0) \wedge (t_{3,2} \geq t_{3,1} + 2) \wedge (t_{3,2} + 3 \leq 8) \wedge \\
((t_{1,1} \geq t_{2,1} + 3) \vee (t_{2,1} \geq t_{1,1} + 2)) \wedge \\
((t_{1,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{1,1} + 2)) \wedge \\
((t_{2,1} \geq t_{3,1} + 2) \vee (t_{3,1} \geq t_{2,1} + 3)) \wedge \\
((t_{1,2} \geq t_{2,2} + 1) \vee (t_{2,2} \geq t_{1,2} + 1)) \wedge \\
((t_{1,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{1,2} + 1)) \wedge \\
((t_{2,2} \geq t_{3,2} + 3) \vee (t_{3,2} \geq t_{2,2} + 1))
\end{array}
$$

Difference Logic

Chasing negative cycles!

Algorithms based on Bellman-Ford (O(mn)).

$$
\begin{aligned}
z &- t_{1,1} \leq 0 \\
z &- t_{2,1} \leq 0 \\
z &- t_{3,1} \leq 0 \\
t_{3,2} &- z \leq 5 \\
t_{3,1} &- t_{3,2} \leq -2 \\
t_{2,1} &- t_{3,1} \leq -3 \\
t_{1,1} &- t_{2,1} \leq -2
\end{aligned}
$$



Microsoft Research

Combining Solvers

In practice, we need a combination of theory solvers.

Nelson-Oppen combination method.

Reduction techniques.

Model-based theory combination.

Microsoft Research

SAT (propositional checkers): Case Analysis

$$p \vee q,$$
$$p \vee \neg q,$$
$$\neg p \vee q,$$
$$\neg p \vee \neg q$$

Microsoft Research

SAT (propositional checkers): Case Analysis

$$\begin{array}{r} p \vee q, \\ p \vee \neg q, \\ \neg p \vee q, \\ \neg p \vee \neg q \end{array}$$

Assignment:
p = false,
q = false

SAT (propositional checkers): Case Analysis

$$
\begin{aligned}
p &\lor q, \\
p &\lor \neg q, \\
\neg p &\lor q, \\
\neg p &\lor \neg q
\end{aligned}
$$

Assignment:
p = false,
q = true

Microsoft Research

SAT (propositional checkers): Case Analysis

$$p \vee q,$$
$$p \vee \neg q,$$
$$\neg p \vee q,$$
$$\neg p \vee \neg q$$

Assignment:
p = true,
q = false

Microsoft Research

SAT (propositional checkers): Case Analysis

$$
\begin{array}{r}
p \lor q, \\
p \lor \neg q, \\
\neg p \lor q, \\
\neg p \lor \neg q
\end{array}
$$

Assignment:
p = true,
q = true

Microsoft Research

DPLL

M | F

Partial model

Set of clauses

DPLL

- Guessing

$$p \mid p \vee q, \neg q \vee r$$

$$\Downarrow$$

$$p, \neg q \mid p \vee q, \neg q \vee r$$

Microsoft Research

DPLL

- Deducing

$$p \mid p \vee q, \neg p \vee s$$

$$\Downarrow$$

$$p, s \mid p \vee q, \neg p \vee s$$

Microsoft Research

DPLL

- Backtracking

$$p, \neg s, q \;|\; p \vee q, s \vee q, \neg p \vee \neg q$$

$$\Downarrow$$

$$p, s \;|\; p \vee q, s \vee q, \neg p \vee \neg q$$

Microsoft Research

Modern DPLL

- Efficient indexing (two-watch literal)
- Non-chronological backtracking (backjumping)
- Lemma learning
- …

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

$\Downarrow$ Abstract (aka "naming" atoms)

$p_1,\ \ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$

$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \lor y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \lor p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT
Solver

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT
Solver

Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment

$p_1,\ p_2,\ \neg p_3,\ p_4$

$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka "naming" atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

Theory Solver

Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

SAT + Theory solvers

Basic Idea

$x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka “naming” atoms)

$p_1,\ p_2,\ (p_3 \vee p_4)$

$p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT Solver

Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

$x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

Theory Solver

Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

New Lemma
$\neg p_1 \vee \neg p_2 \vee \neg p_4$

SAT + Theory solvers

New Lemma

$\neg p_1 \vee \neg p_2 \vee \neg p_4$

Unsatisfiable

$x \geq 0,\ y = x + 1,\ y < 1$

Theory Solver

AKA
Theory conflict

SAT + Theory solvers: Main loop

procedure SmtSolver(F)

  (F_p, M) := Abstract(F)

  loop

    (R, A) := SAT_solver(F_p)

    if R = UNSAT **then return** UNSAT

    S := Concretize(A, M)

    (R, S') := Theory_solver(S)

    if R = SAT **then return** SAT

    L := New_Lemma(S', M)

    Add L to F_p

SAT + Theory solvers

Basic Idea

F: $x \geq 0,\ y = x + 1,\ (y > 2 \vee y < 1)$

Abstract (aka “naming” atoms)

F_p : $p_1,\ p_2,\ (p_3 \vee p_4)$

M: $p_1 \equiv (x \geq 0),\ p_2 \equiv (y = x + 1),$
$p_3 \equiv (y > 2),\ p_4 \equiv (y < 1)$

SAT
Solver

A: Assignment
$p_1,\ p_2,\ \neg p_3,\ p_4$

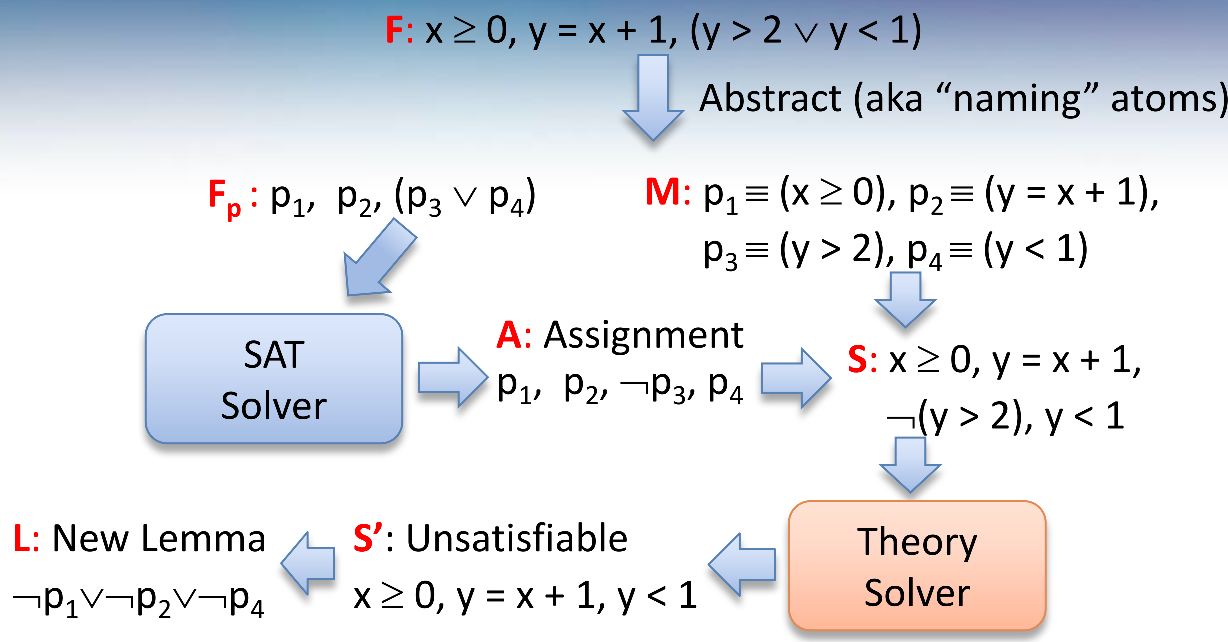
S: $x \geq 0,\ y = x + 1,$
$\neg(y > 2),\ y < 1$

Theory
Solver

S’: Unsatisfiable
$x \geq 0,\ y = x + 1,\ y < 1$

L: New Lemma
$\neg p_1 \vee \neg p_2 \vee \neg p_4$

SAT + Theory solvers



procedure SMT_Solver(F)
  (F_p, M) := Abstract(F)
  loop
    (R, A) := SAT_solver(F_p)
    if R = UNSAT **then return** UNSAT
    S = Concretize(A, M)
    (R, S') := Theory_solver(S)
    if R = SAT **then return** SAT
    L := New_Lemma(S, M)
    Add L to F_p

"Lazy translation"
to
DNF

SAT + Theory solvers

State-of-the-art SMT solvers implement many improvements.

SAT + Theory solvers

Incrementality

Send the literals to the Theory solver as they are assigned by the SAT solver

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1, \; p_2, \; p_4 \;\; | \;\; p_1, \; p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

Partial assignment is already Theory inconsistent.

SAT + Theory solvers

Efficient Backtracking

We don't want to restart from scratch after each backtracking operation.

SAT + Theory solvers

Efficient Lemma Generation (computing a small S')

Avoid lemmas containing redundant literals.

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1, \; p_2, p_3, p_4 \;\; | \;\; p_1, \; p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

$\neg p_1 \vee \neg p_2 \vee \neg p_3 \vee \neg p_4$

Imprecise Lemma

SAT + Theory solvers

Theory Propagation

It is the SMT equivalent of unit propagation.

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1, \; p_2 \mid \; p_1, \; p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

$\Downarrow$ p_1, p_2 imply $\neg p_4$ by theory propagation

$p_1, \; p_2, \neg p_4 \mid \; p_1, \; p_2, (p_3 \vee p_4), (p_5 \vee \neg p_4)$

SAT + Theory solvers

Theory Propagation

It is the SMT equivalent of unit propagation.

$p_1 \equiv (x \geq 0)$, $p_2 \equiv (y = x + 1)$,

$p_3 \equiv (y > 2)$, $p_4 \equiv (y < 1)$, $p_5 \equiv (x < 2)$,

$p_1,\ p_2 \mid\ p_1,\ p_2, (p_3 \lor p_4), (p_5 \lor \neg p_4)$

$\Downarrow$ p_1, p_2 imply $\neg p_4$ by theory propagation

$p_1,\ p_2, \neg p_4 \mid\ p_1,\ p_2, (p_3 \lor p_4), (p_5 \lor \neg p_4)$

Tradeoff between precision × performance.

SMT x SAT

For some theories, SMT can be reduced to SAT

Higher level of abstraction

$$\text{bvmul}_{32}(a,b) = \text{bvmul}_{32}(b,a)$$

Microsoft Research

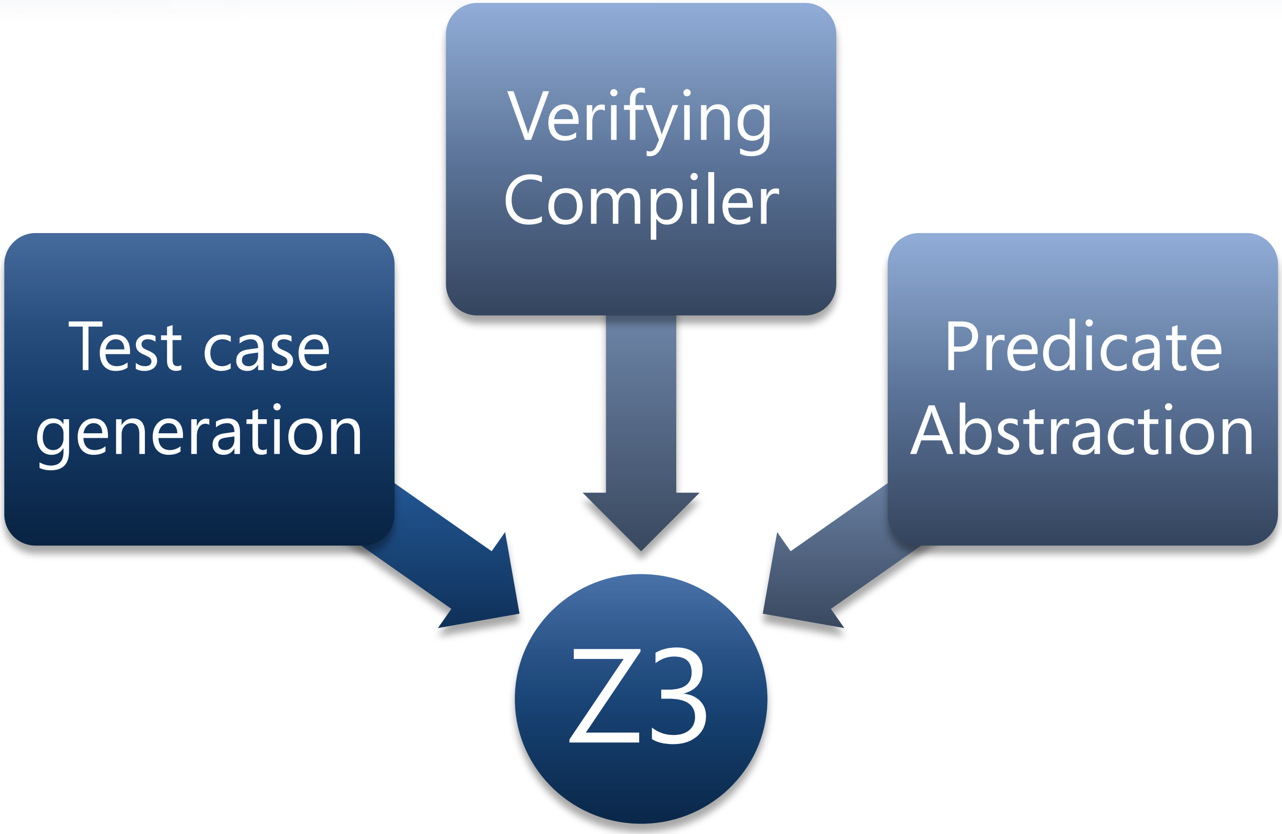
SMT x First-order provers

$F \cup T$ → First-order Theorem Prover

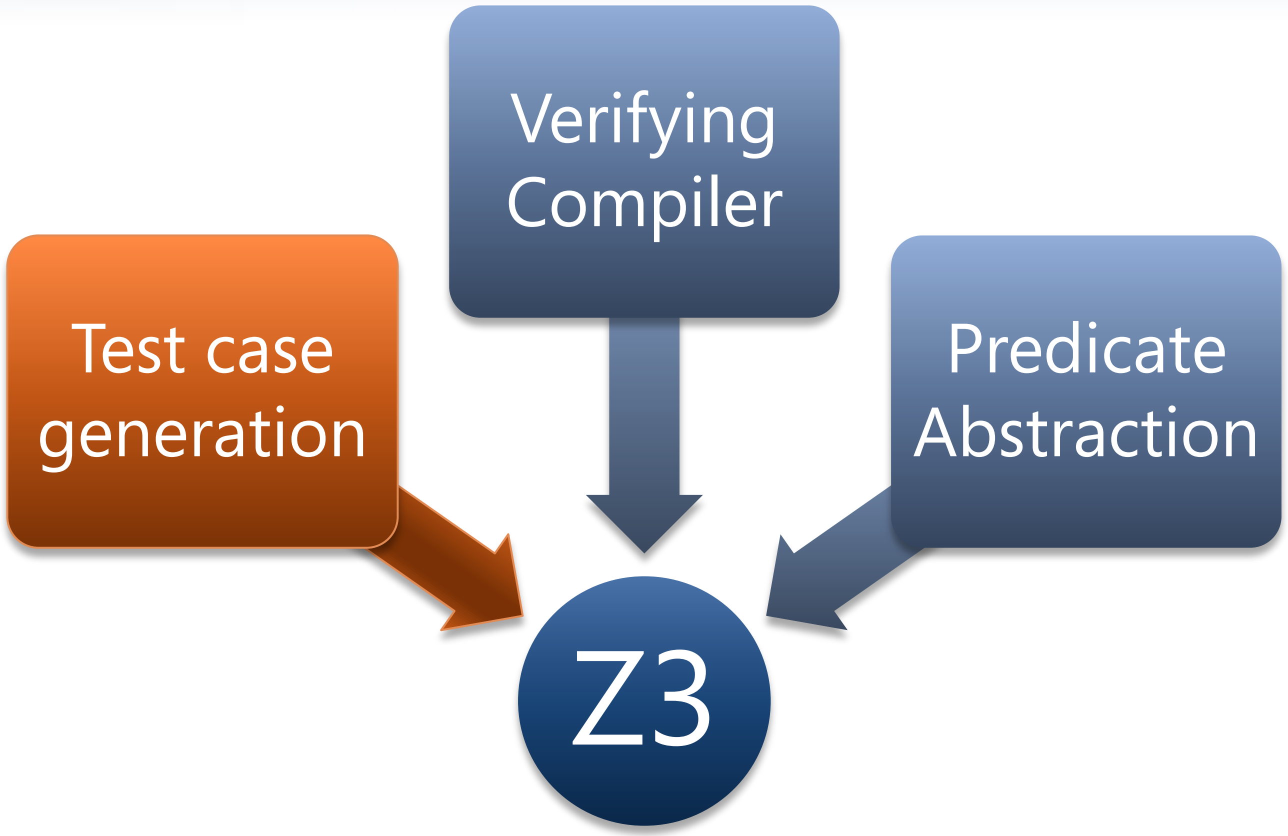
T may not have a finite axiomatization

Microsoft Research

SMT: Some Applications



SMT: Some Applications



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

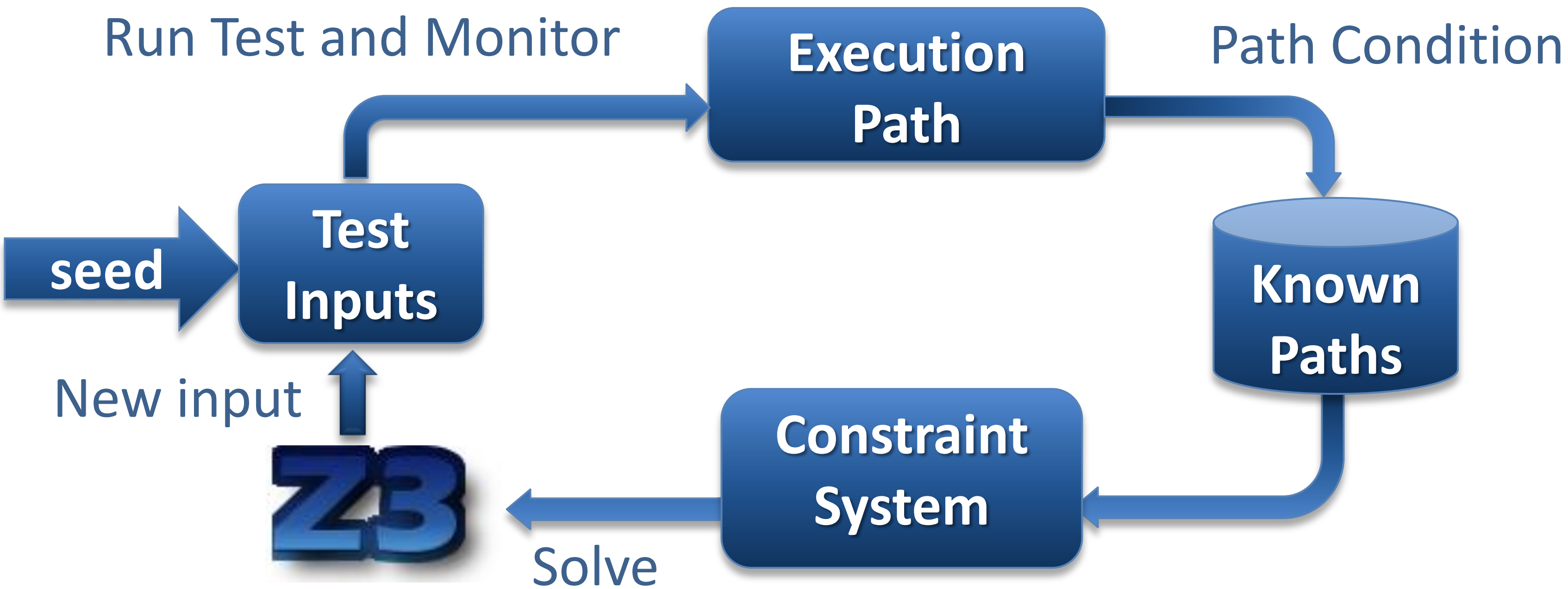
Hunting for Security Bugs.

- 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)



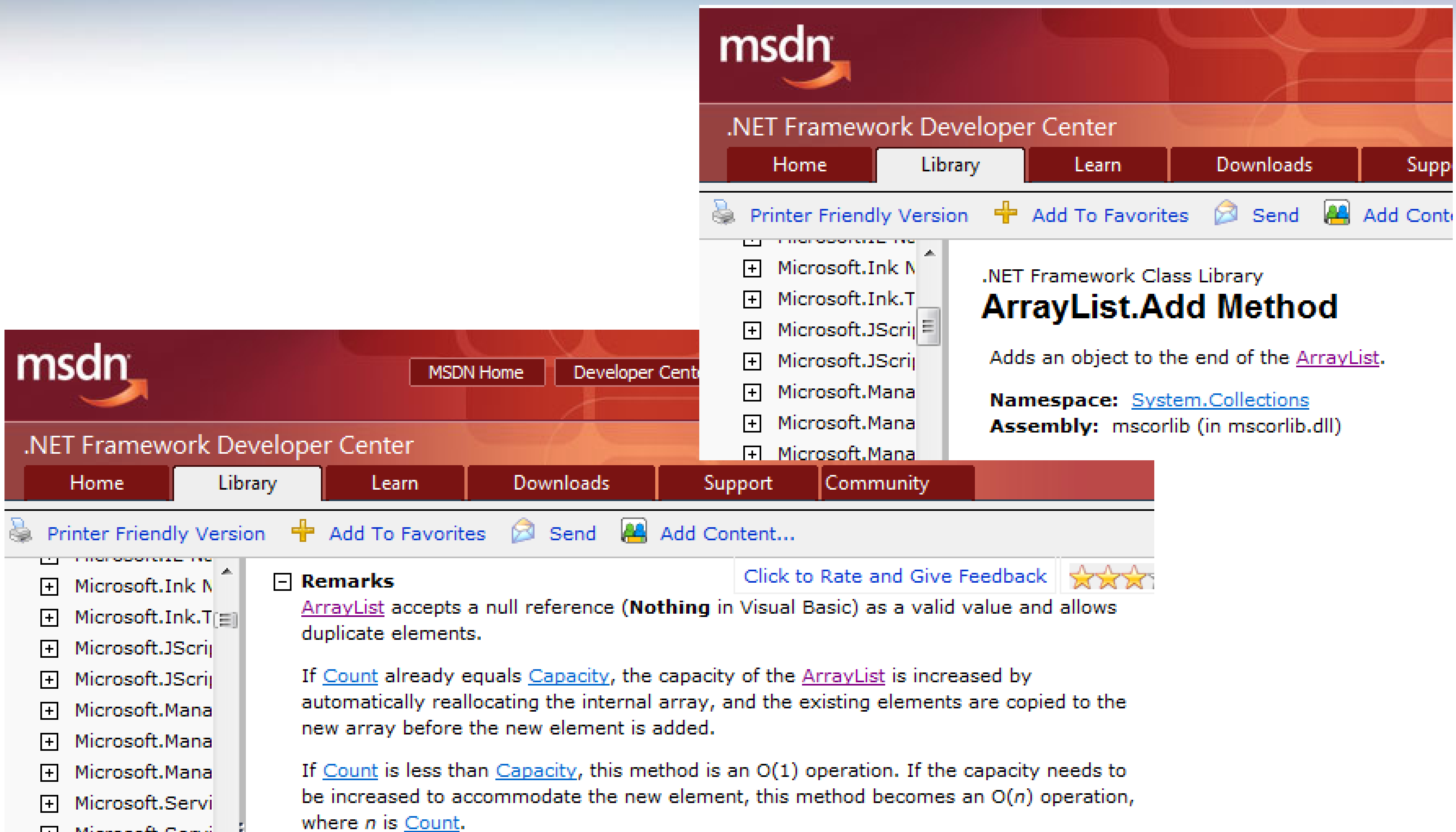
DARTish projects at Microsoft

PEX	Implements DART for .NET.
SAGE	Implements DART for x86 binaries.
YOGI	Implements DART to check the feasibility of program paths generated statically using a SLAM-like tool.
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



msdn

.NET Framework Developer Center

Home | Library | Learn | Downloads | Supp

Printer Friendly Version | Add To Favorites | Send | Add Cont

.NET Framework Class Library

ArrayList.Add Method

Adds an object to the end of the ArrayList.

Namespace: System.Collections
Assembly: mscorlib (in mscorlib.dll)

msdn

MSDN Home | Developer Cent

.NET Framework Developer Center

Home | Library | Learn | Downloads | Support | Community

Printer Friendly Version | Add To Favorites | Send | Add Content...

Click to Rate and Give Feedback

Remarks

ArrayList accepts a null reference (**Nothing** in Visual Basic) as a valid value and allows duplicate elements.

If Count already equals Capacity, the capacity of the ArrayList is increased by automatically reallocating the internal array, and the existing elements are copied to the new array before the new element is added.

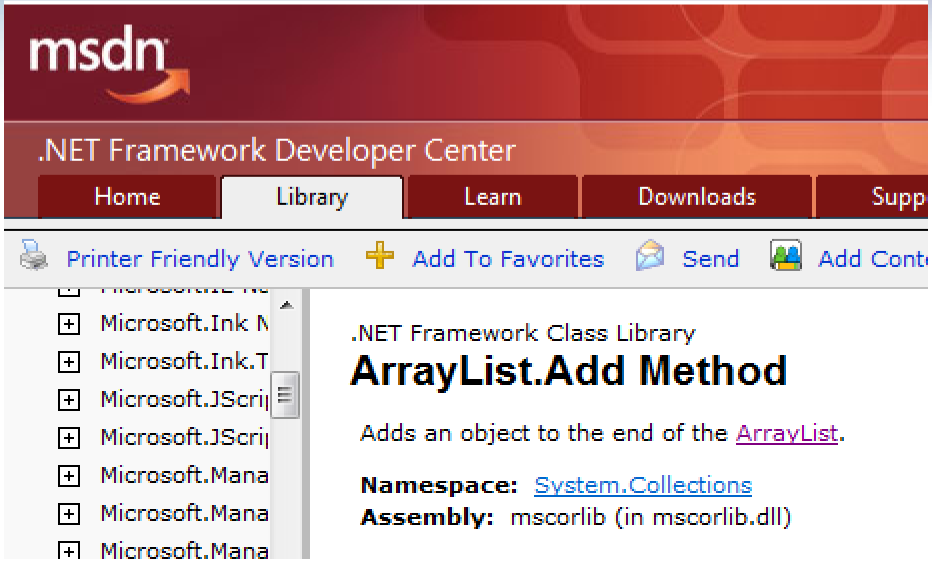
If Count is less than Capacity, this method is an O(1) operation. If the capacity needs to be increased to accommodate the new element, this method becomes an O(*n*) operation, where *n* is Count.

Microsoft Research

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

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)

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)

```
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); }
}
```

```
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; }
...
```

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		

Z3

Microsoft Research

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

Microsoft Research

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)`**	

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; }
...
```

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; }
...
```

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: Incrementality

- Pex "sends" several similar formulas to Z3.
- Plus: backtracking primitives in the Z3 API.
  - **push**
  - **pop**
- Reuse (some) lemmas.

PEX ⟷ Z3: Small models

- **Given** a set of constraints *C*, find a model *M* that minimizes the interpretation for $x_0, \ldots, x_n$.
- In the ArrayList example:
  - Why is the model where *c* = 2147483648 less desirable than the model with *c* = 1?

```
!(c<0) && 0!=c
```

- Simple solution:

  Assert *C*

  while satisfiable

  Peek x_i such that $M[x_i]$ is big

  Assert $x_i < n$, where n is a small constant

  Return last found model

Microsoft Research

PEX ⟷ Z3: Small models

- **Given** a set of constraints *C*, find a model *M* that minimizes the interpretation for $x_0, \ldots, x_n$.
- In the ArrayList example:
  - Why is the model where *c* = 2147483648 less desirable than the model with *c* = 1?

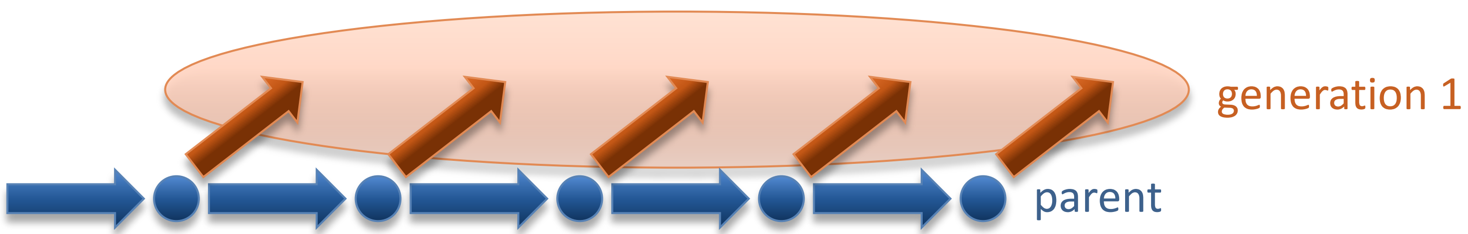
```
!(c<0) && 0!=c
```

- **Refinement**:
  - Eager solution stops as soon as the system becomes unsatisfiable.
  - A “bad” choice (peek x_i) may prevent us from finding a good solution.
  - Use **push** and **pop** to retract “bad” choices.

Microsoft Research

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 00 00 00 00 00 00 00 00 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 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 1

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 00 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 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 2

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 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 3

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 00 00 00 00 ; ....strh........
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 4

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 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 5

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 00 00 00 00 00 00 00 00 ; ....strf........
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 6

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 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 7

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 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 C9 9D E4 4E ; ............ÉäN
00000060h: 00 00 00 00                                     ; ....
```

Generation 8

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 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 9

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

Microsoft Research

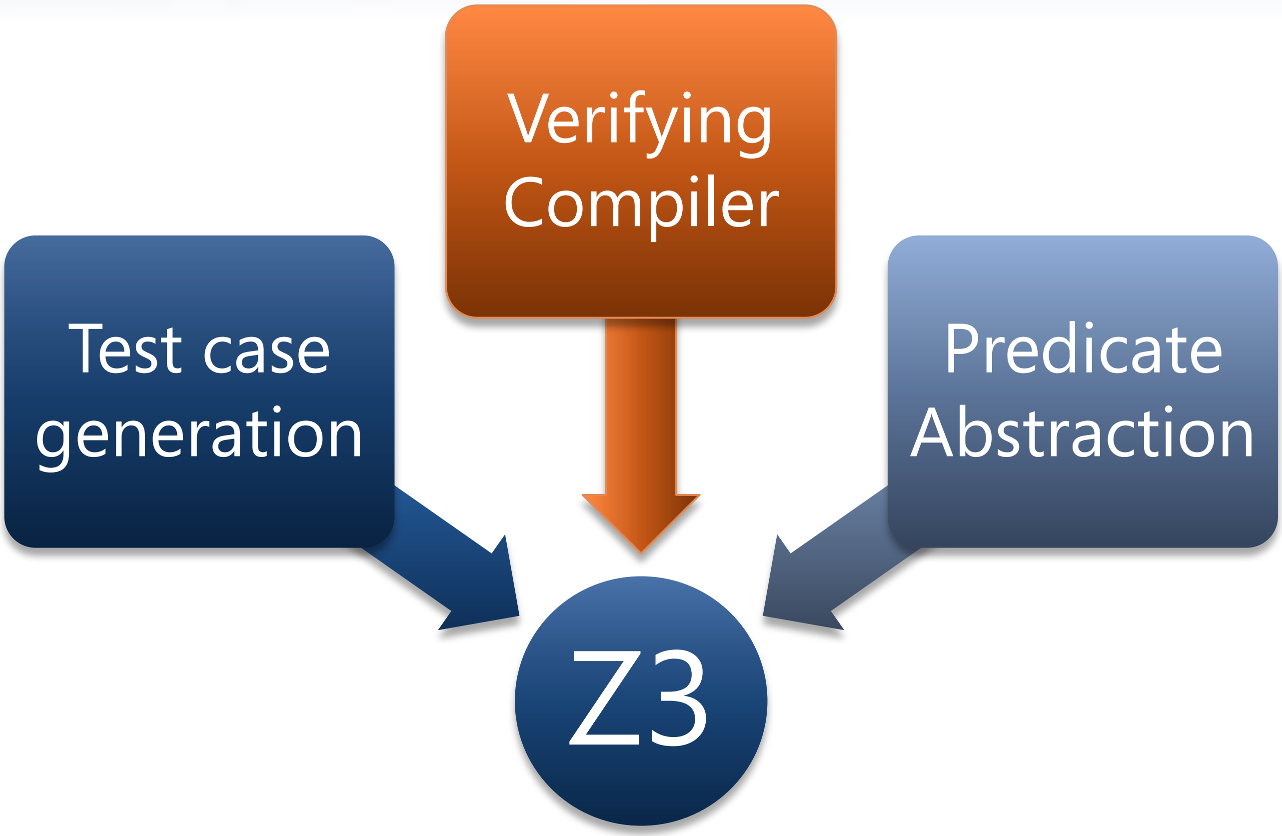
SAGE (cont.)

- 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

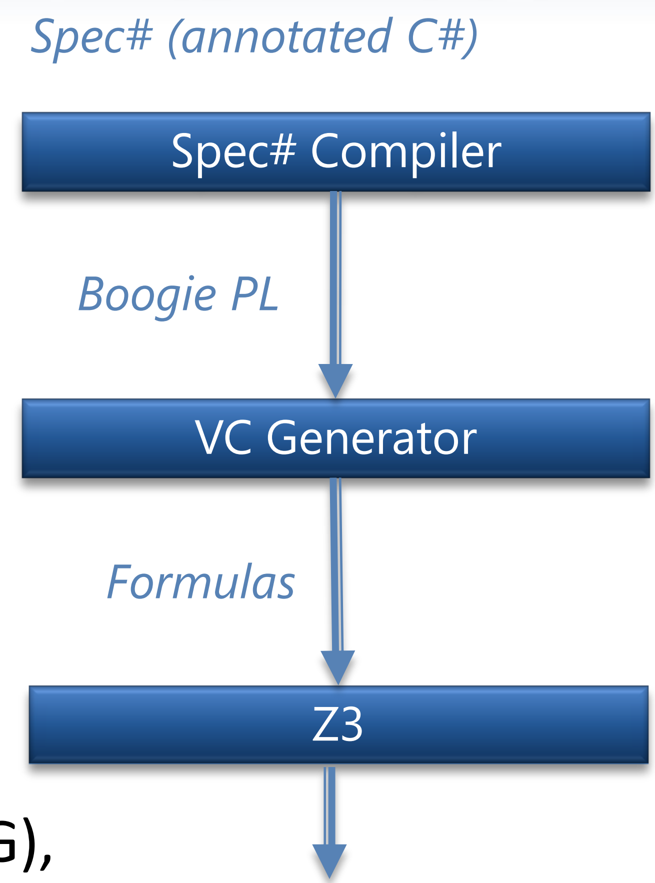
- Formulas are usually big conjunctions.
- SAGE uses only the bitvector and array theories.
- Pre-processing step has a huge performance impact.
  - Eliminate variables.
  - Simplify formulas.
- Early unsat detection.

SMT: Some Applications



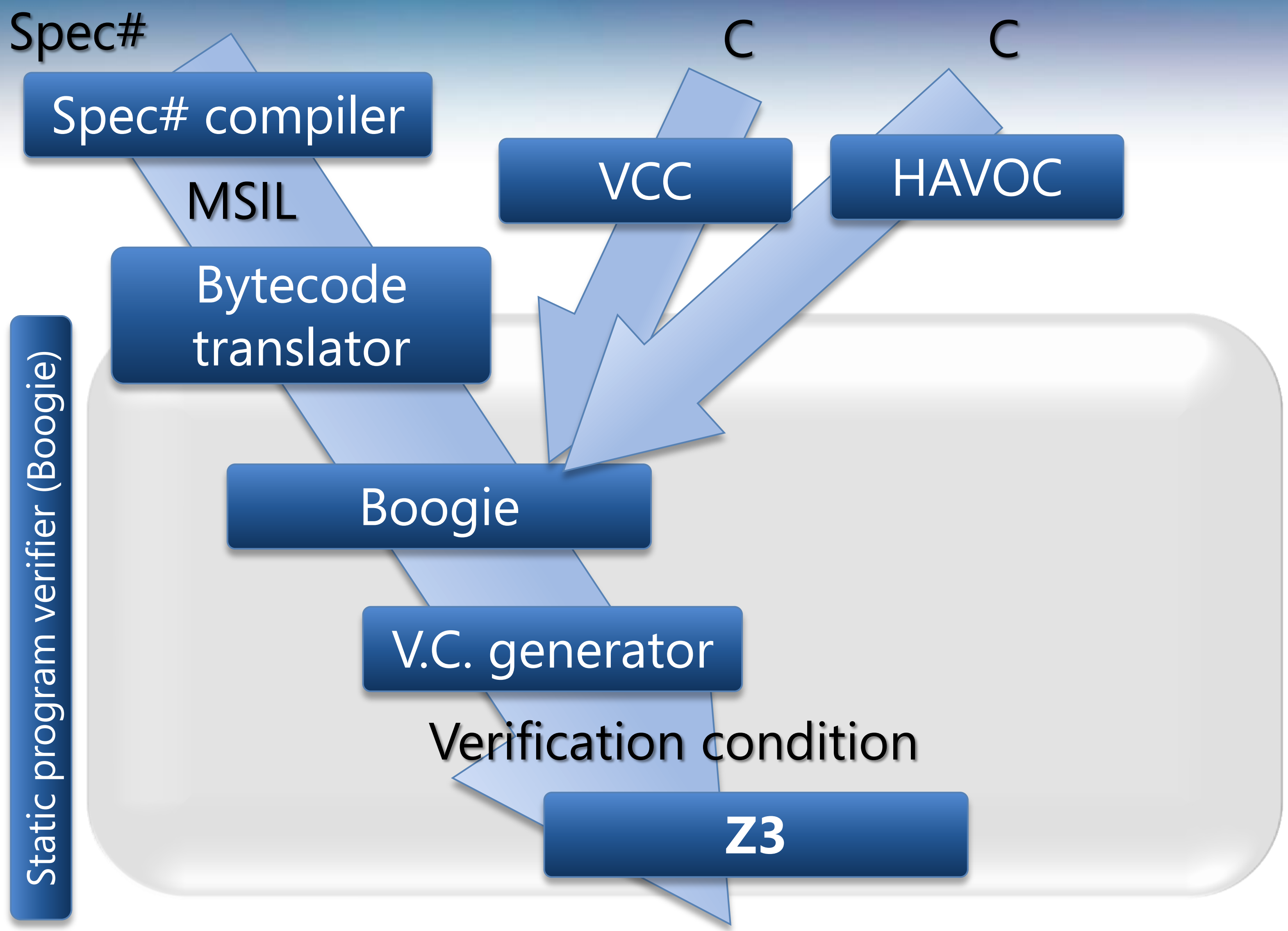
Spec# Approach for a Verifying Compiler

- *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



Microsoft Research

Verification architecture



HAVOC

- A tool for specifying and checking properties of systems software written in C.
- It also translates annotated C into Boogie PL.
- It allows the expression of *richer properties about the program heap and data structures* such as linked lists and arrays.
- HAVOC is being used to specify and check:
  - Complex locking protocols over heap-allocated data structures in Windows.
  - Properties of collections such as IRP queues in device drivers.
  - Correctness properties of custom storage allocators.

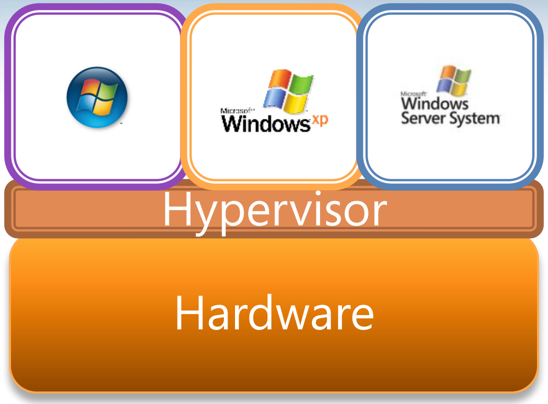
Microsoft Research

A Verifying C Compiler

- VCC translates an *annotated C program* into a *Boogie PL* program.
- A C-ish memory model
  - Abstract heaps
  - Bit-level precision
- Microsoft Hypervisor: verification grand challenge.

Microsoft Research

Hypervisor: A Manhattan Project



- **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 verification grand challenge

Hypervisor: Some Statistics

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

Main Challenge

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

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

Main Challenge

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

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

Microsoft Research

Main 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

Main 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

Main 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!

Microsoft Research

Bad news

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

Many Approaches

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

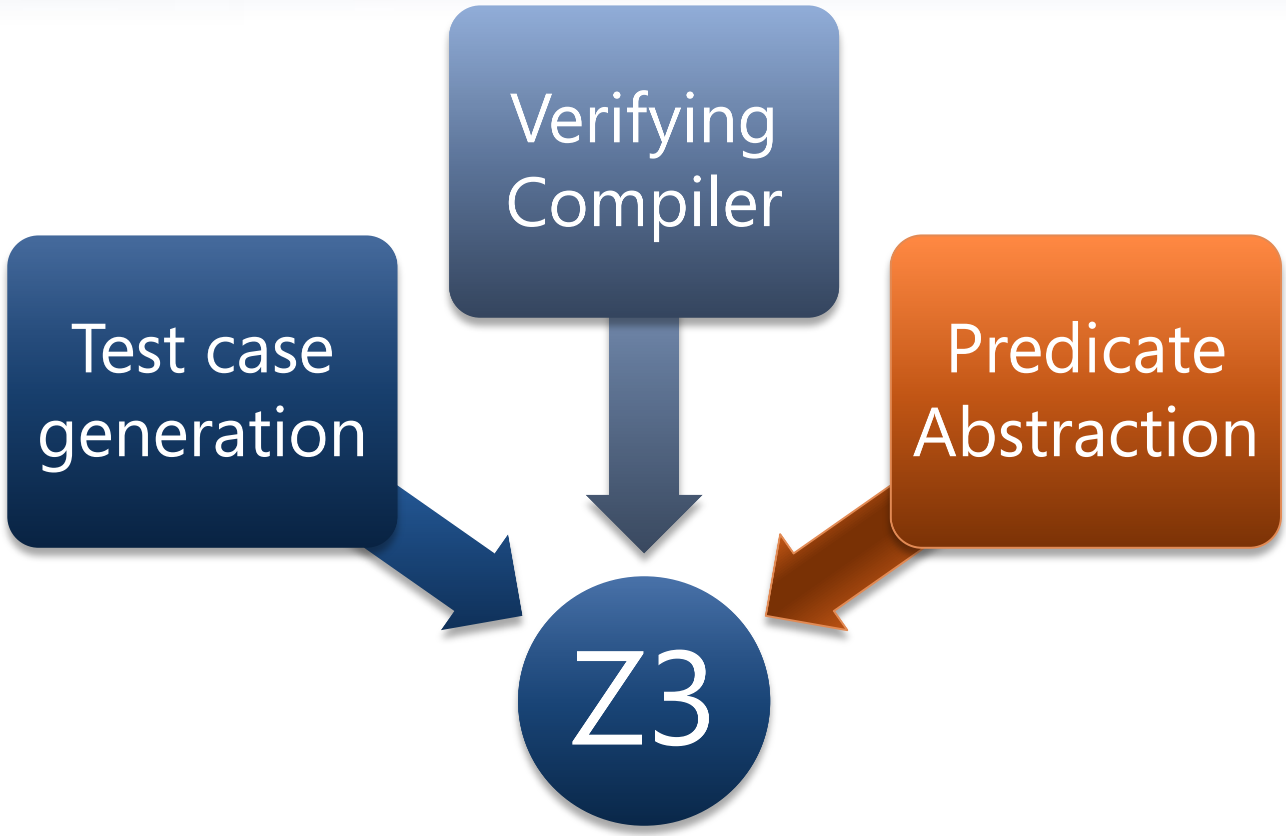
Challenge: Robustness

- 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

Microsoft Research

SMT: Some Applications



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.

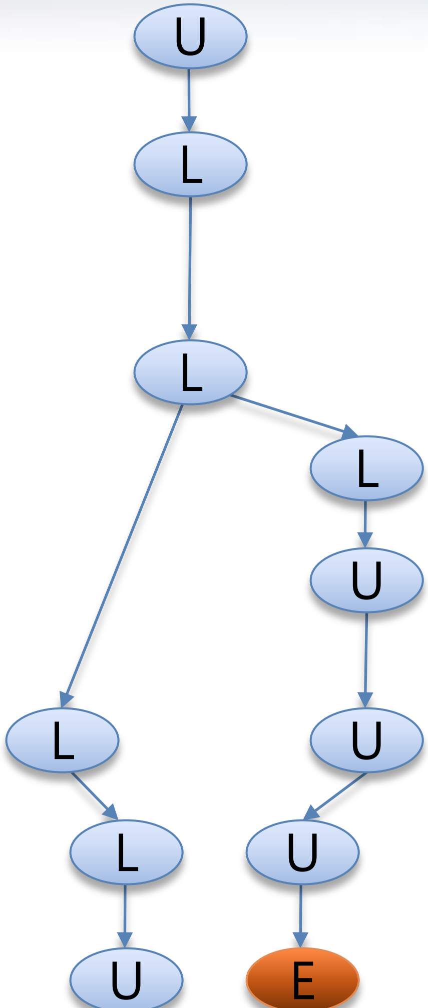
Example

Do this code obey the looking rule?

```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

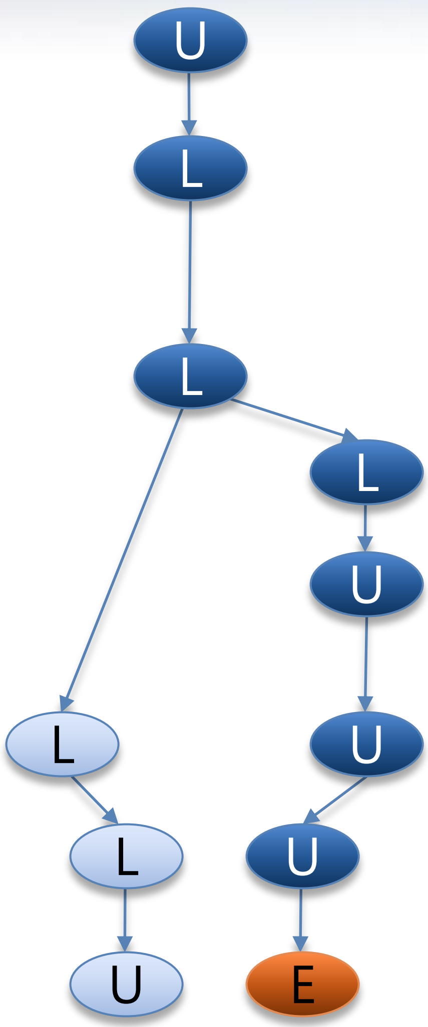
Model checking
Boolean program




```
do {
   KeAcquireSpinLock();

   if(*){

      KeReleaseSpinLock();

   }
} while (*);

KeReleaseSpinLock();
```

Example

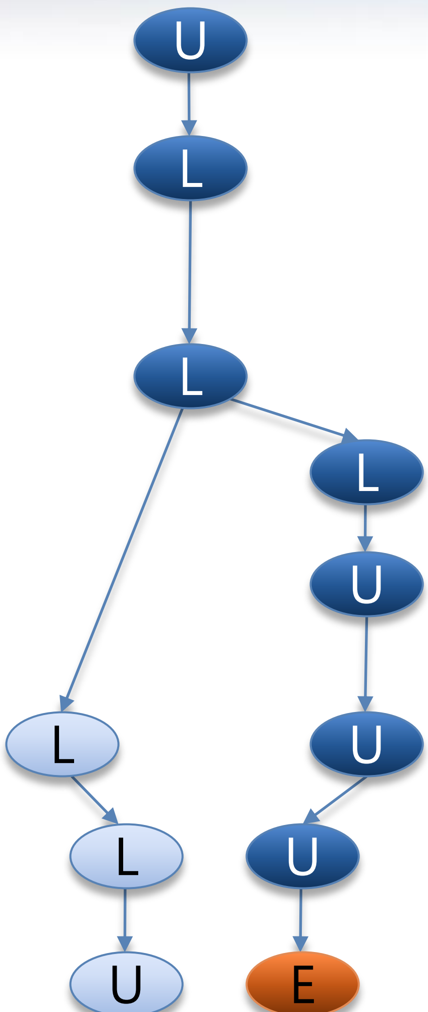
Is error path feasible?




```
do {
   KeAcquireSpinLock();

   nPacketsOld = nPackets;

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

KeReleaseSpinLock();
```

Example

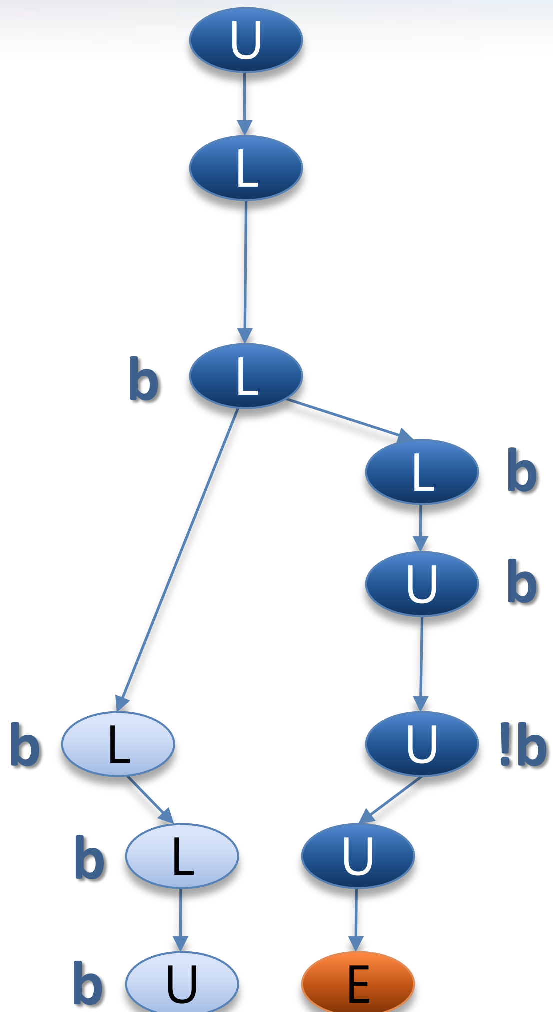
Add new predicate to
Boolean program
b: (nPacketsOld == nPackets)




```
do {
   KeAcquireSpinLock();

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

Example

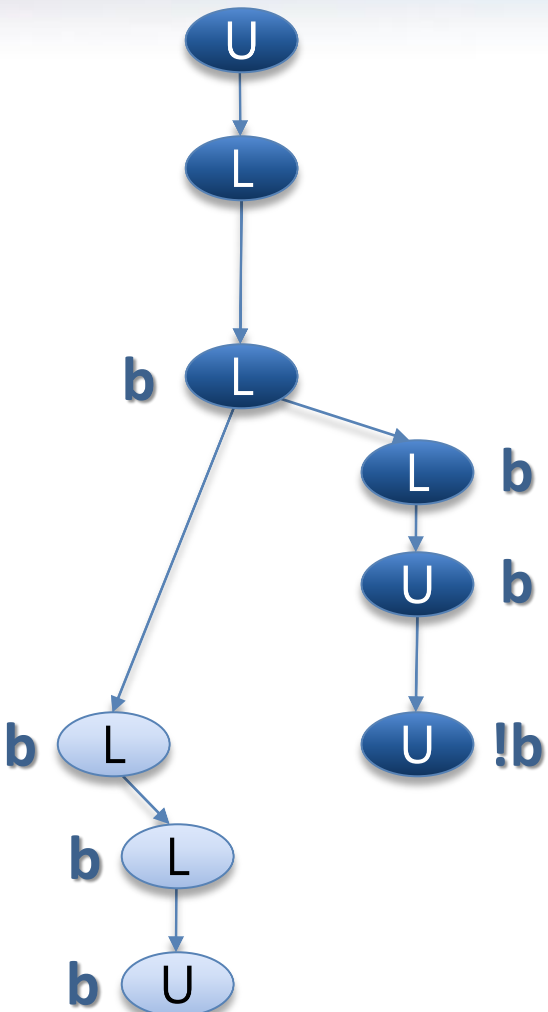
Model Checking
Refined Program
b: (nPacketsOld == nPackets)




```
do {
   KeAcquireSpinLock();

   b = true;

   if(*){

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

KeReleaseSpinLock();
```

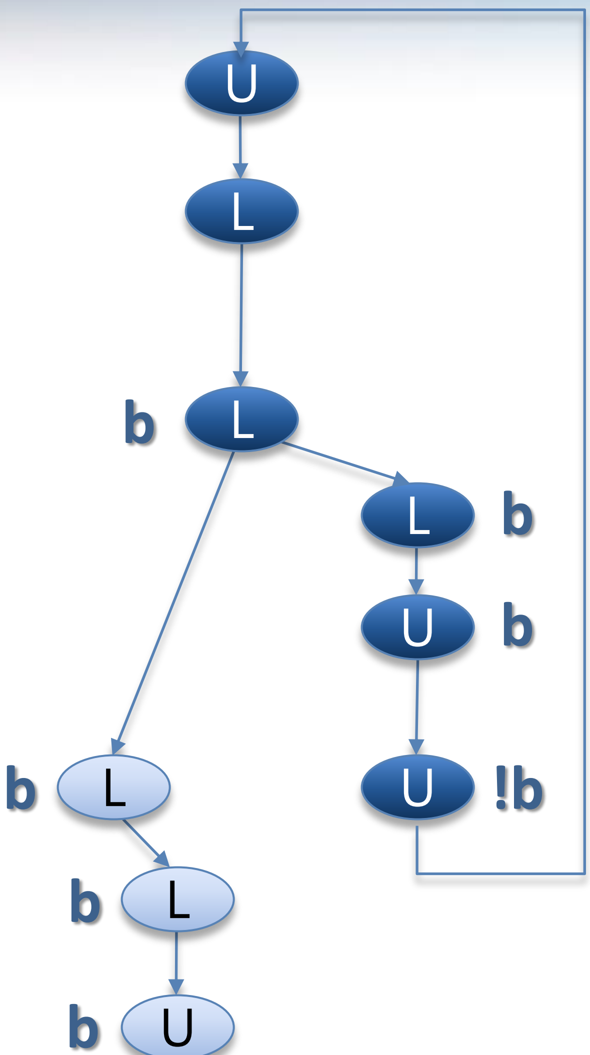
Example

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

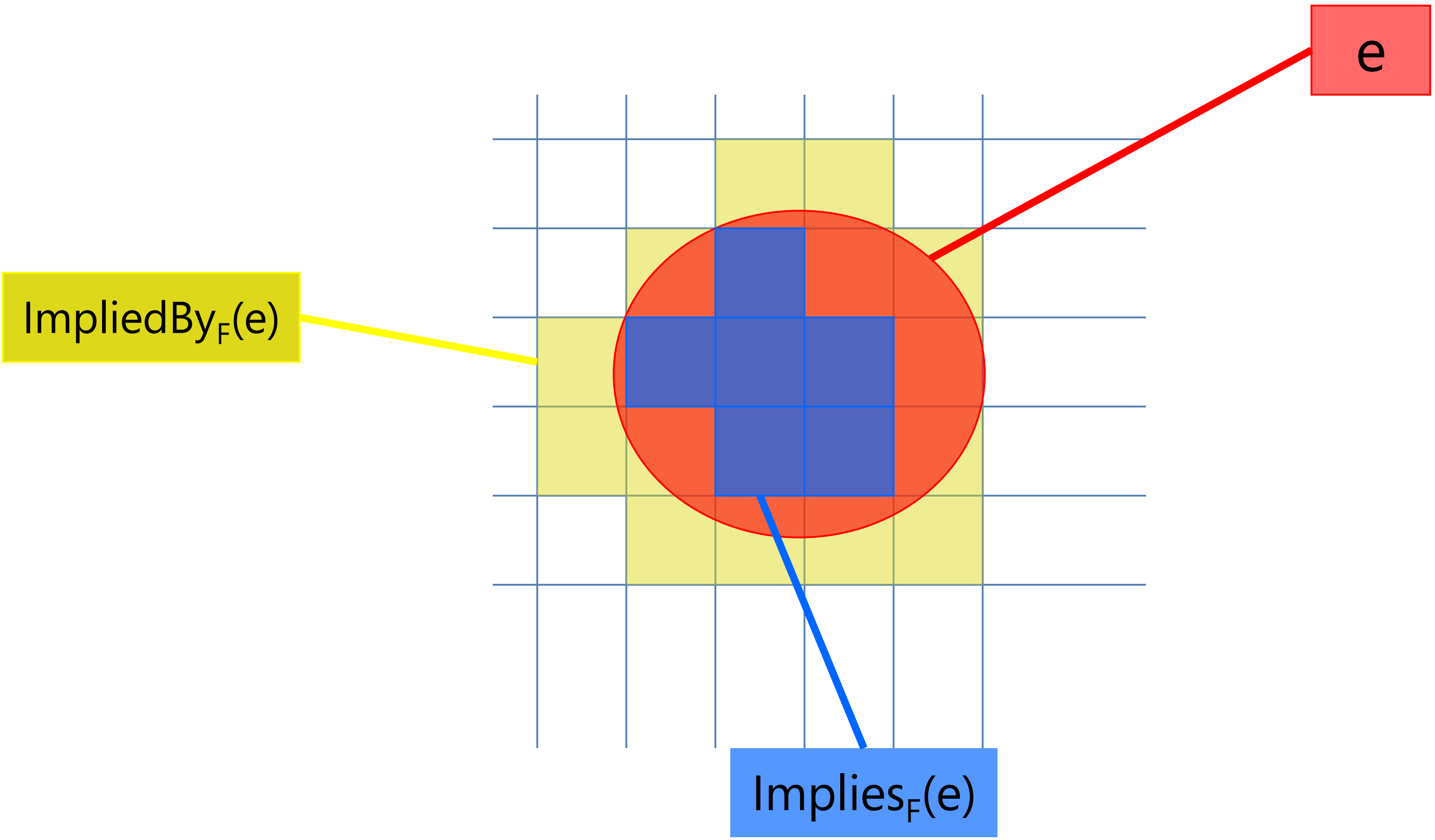
Predicate Abstraction: *c2bp*

- **Given** a C program P and $F = \{p_1, \dots, p_n\}$.
- **Produce** a Boolean program $B(P, F)$
  - Same control flow structure as P.
  - Boolean variables $\{b_1, \dots, b_n\}$ to match $\{p_1, \dots, p_n\}$.
  - 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.
  - $ImpliedBy_F\ (e) = not\ Implies_F\ (not\ e)$

$\text{Implies}_F(e)$ and $\text{ImpliedBy}_F(e)$



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 $\mathit{Implies}_F(e)$

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

$\mathit{Implies}_F(y>1) = x<y \wedge x=2$

Computing $\mathit{Implies}_F(e)$

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

$\mathit{Implies}_F(y>1) = b_1 \wedge b_2$

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.

SLAM $\leftrightarrow$ Z3: Unsatisfiable cores

- 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 $S'' \subset S'$ 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:
    - $p_1, \neg p_2, p_3, p_4 \Rightarrow e$
    - $p_1, \neg p_2, p_3, \neg p_4 \Rightarrow e$
    - $p_1, p_2, p_3, \neg p_4 \Rightarrow e$

Microsoft Research

SMT: Some Applications (Extra)

Bit-Precise
Static Analysis

Microsoft Research

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?

```
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;
```

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

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;
}
```

```
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

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

```
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

Other Microsoft clients

- Model programs (M. Veanes – MSRR)
- Termination (B. Cook – MSRC)
- Security protocols (A. Gordon and C. Fournet - MSRC)
- Business Application Modeling (E. Jackson - MSRR)
- Cryptography (R. Venki – MSRR)
- Verifying Garbage Collectors (C. Hawblitzel – MSRR)
- Model Based Testing (L. Bruck – SQL)
- Semantic type checking for D models (G. Bierman – MSRC)
- More coming soon…

Microsoft Research

Conclusion

- SMT is hot at Microsoft.
- Many applications.
- Z3 is an efficient SMT solver.
- http://research.microsoft.com/projects/z3
- http://research.microsoft.com/~leonardo

Thank You!

Microsoft Research