Satisfiability Modulo Theories: An Appetizer

SBMF 2009 - Gramado

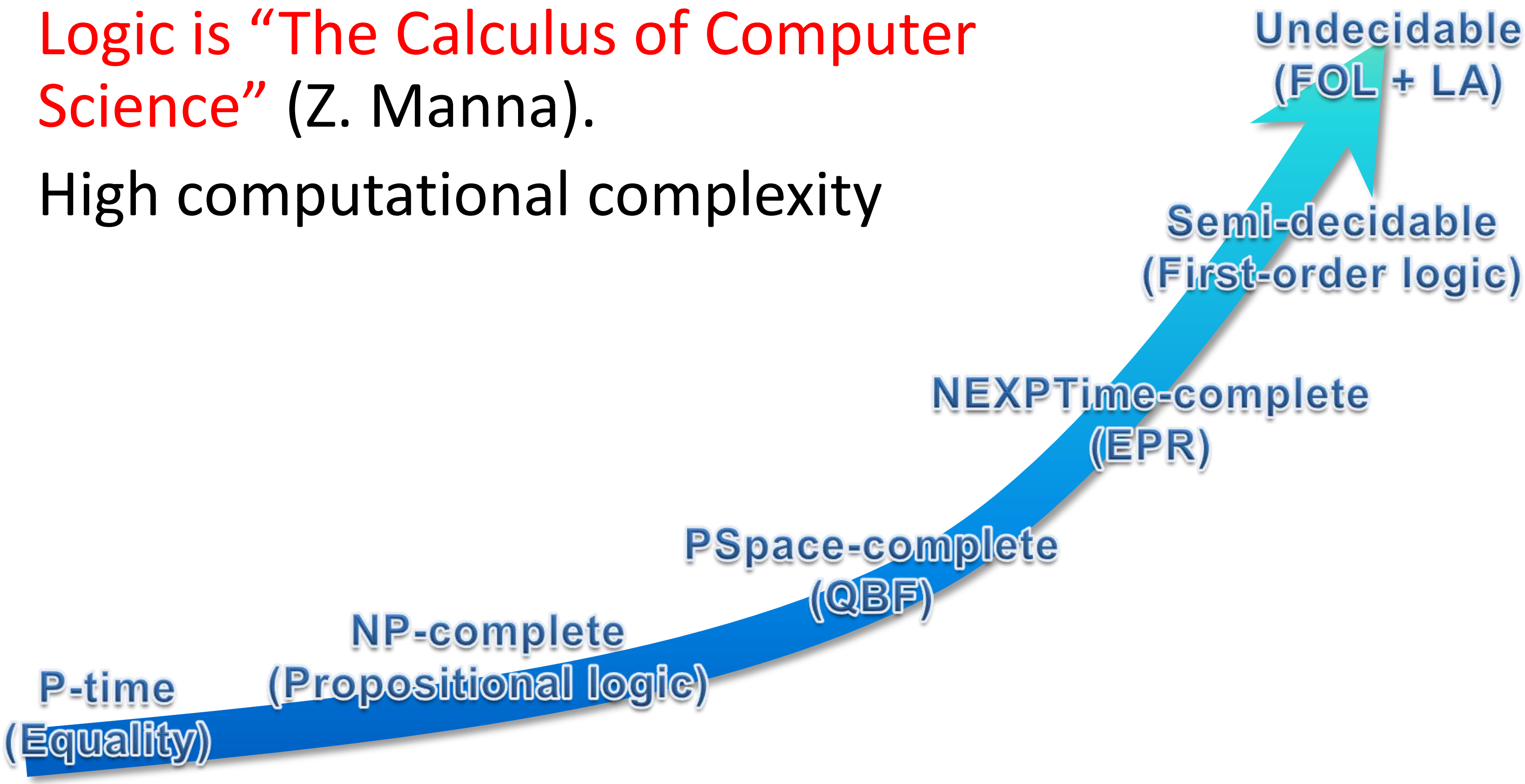
Leonardo de Moura
Microsoft Research

Symbolic Reasoning

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

Symbolic Reasoning

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



Applications

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

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

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.

Type checking

Signature:

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

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)

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

Satisfiability Modulo Theories (SMT)

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

Arithmetic

Satisfiability Modulo Theories (SMT)

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

Array Theory

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Uninterpreted Functions

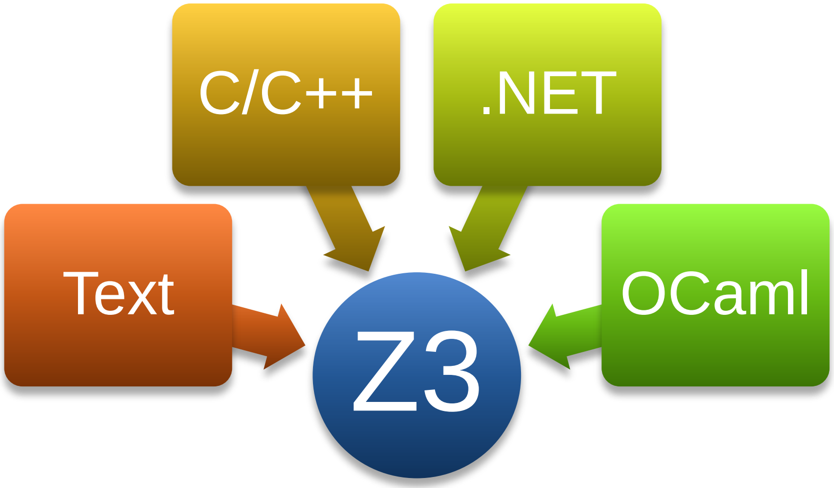
Theories

- ***A Theory is a set of sentences***

- Alternative definition:
  A Theory is a class of structures

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***

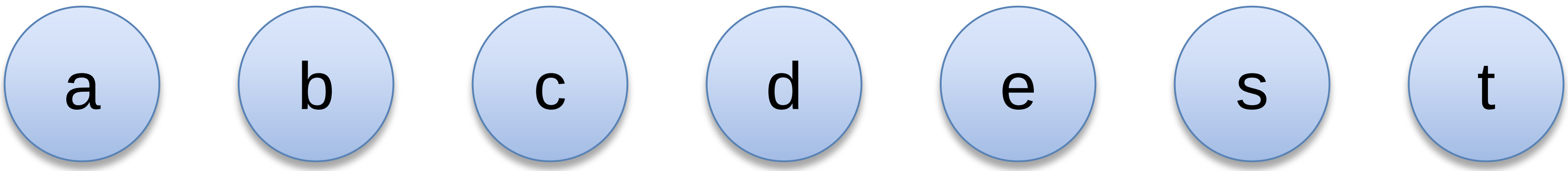
Many Applications

Bounded Model Checking

Test-Case Generation

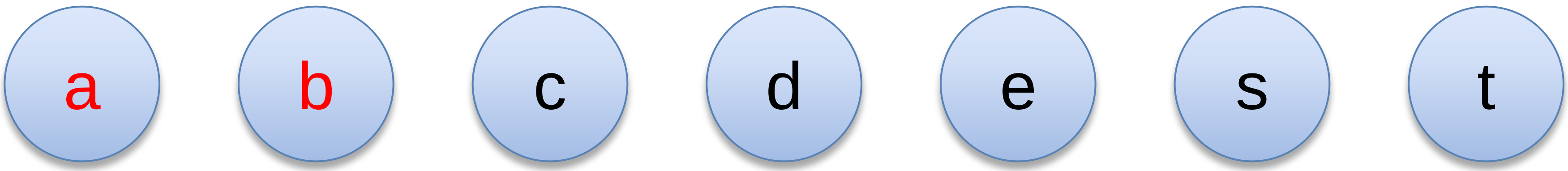
Deciding Equality

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



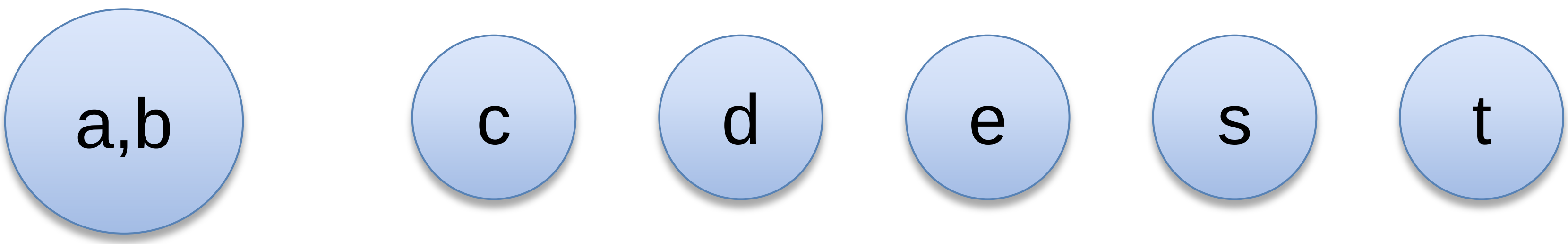
Deciding Equality

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



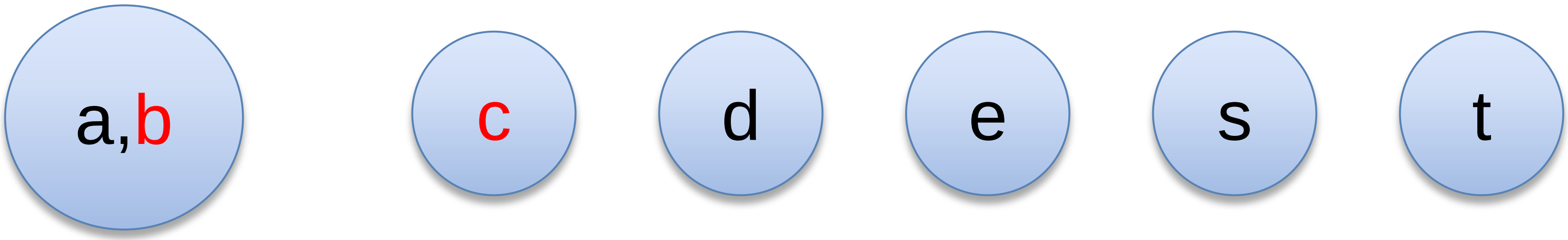
Deciding Equality

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



Deciding Equality

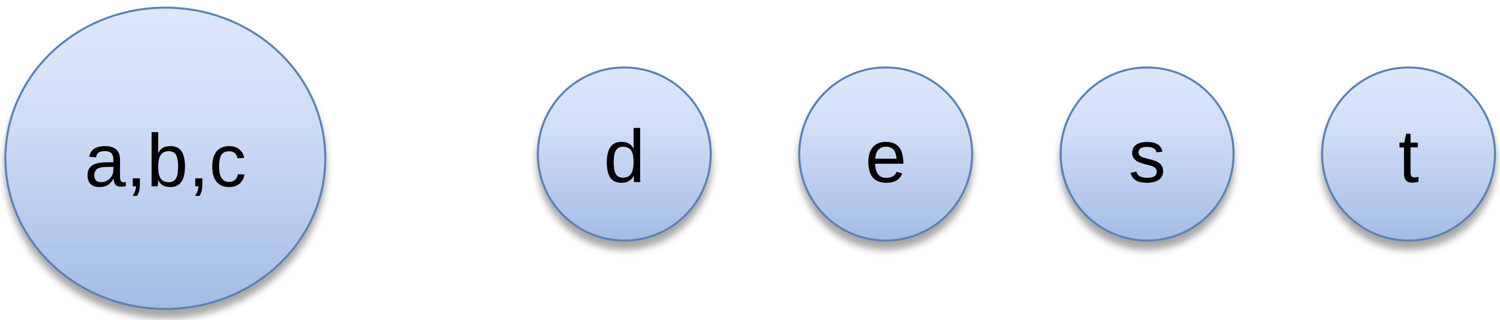
$$a = b,\ b = c,\ d = e,\ b = s,\ d = t,\ a \neq e,\ a \neq s$$



Microsoft Research

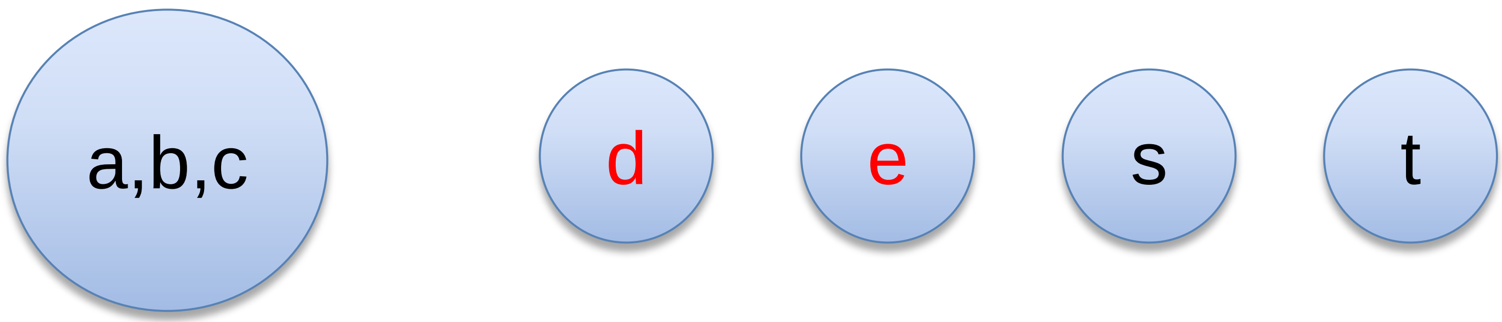
Deciding Equality

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



Deciding Equality

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



Deciding Equality

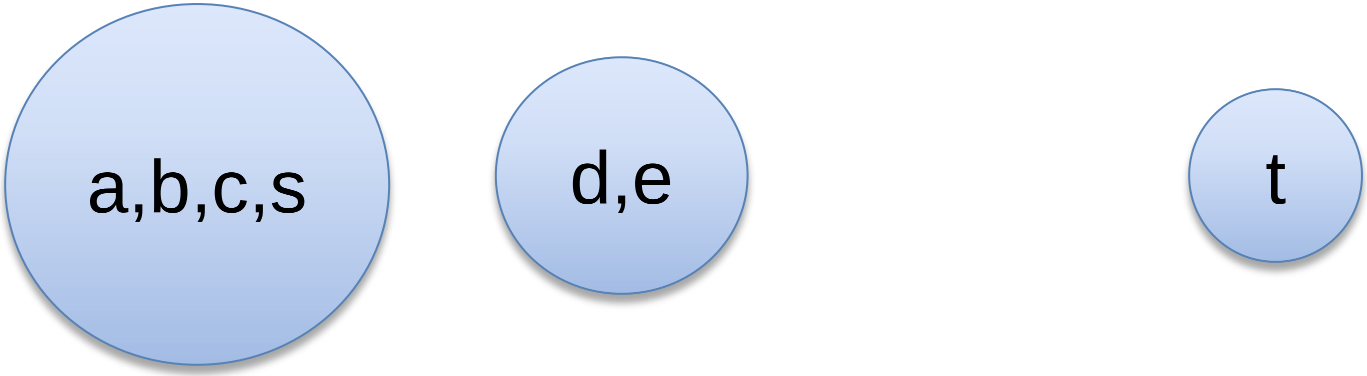
$$a = b,\ b = c,\ \color{red}{d = e},\ b = s,\ d = t,\ a \neq e,\ a \neq s$$

Deciding Equality

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

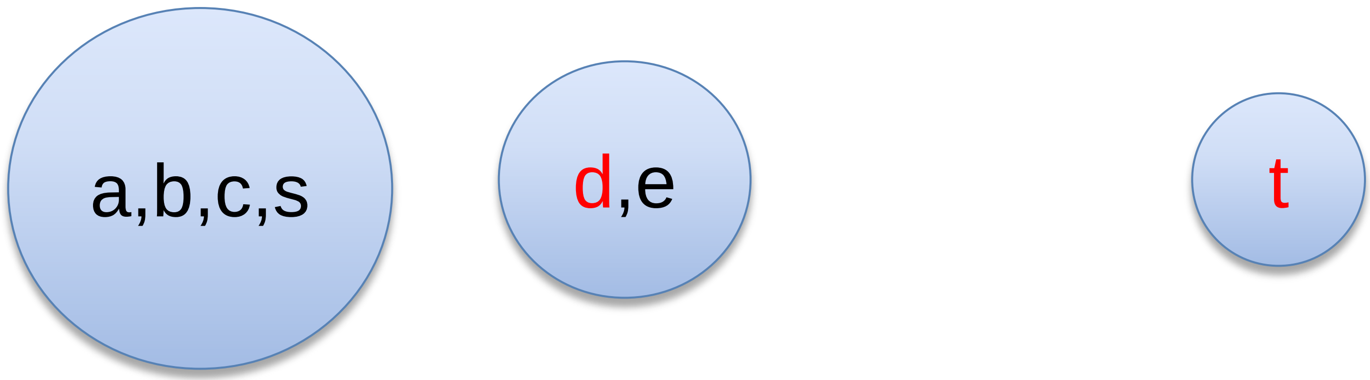
Deciding Equality

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



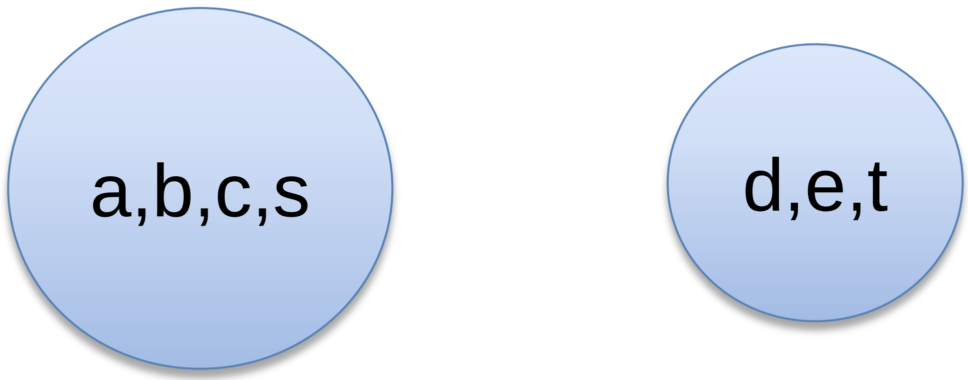
Deciding Equality

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



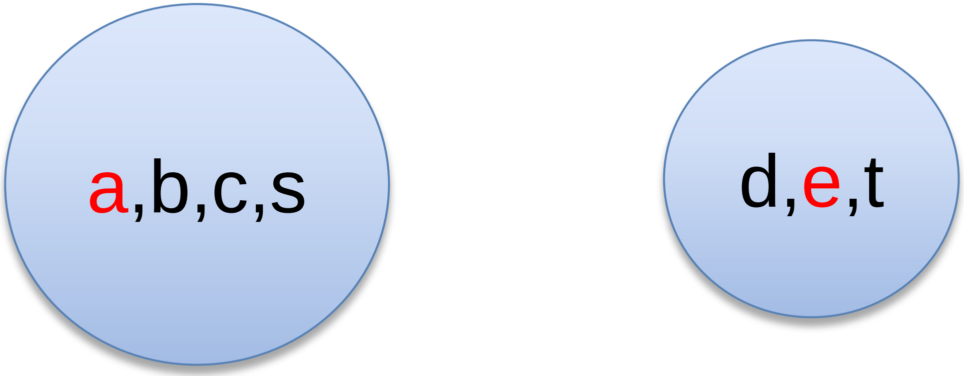
Deciding Equality

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



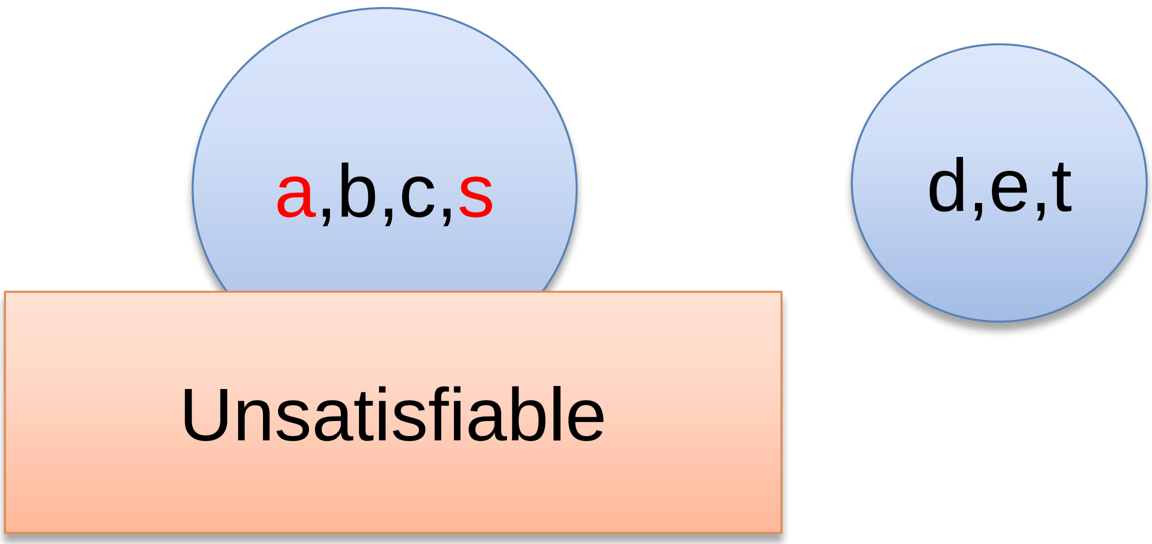
Deciding Equality

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



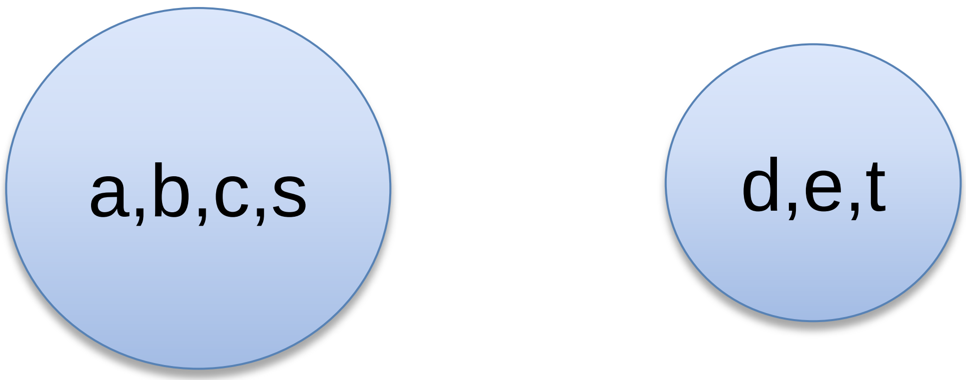
Deciding Equality

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



Deciding Equality

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



Model

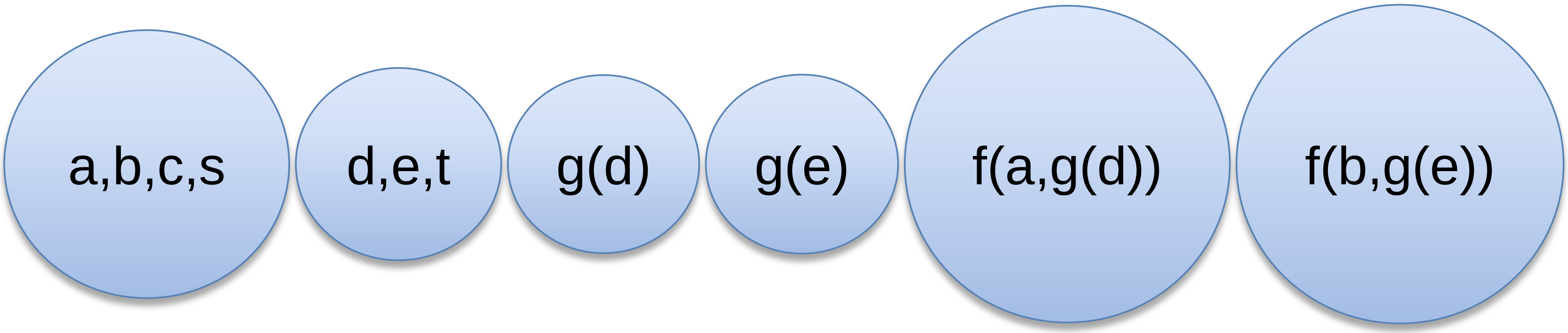
$|M| = \{\ 0, 1\ \}$

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

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

Deciding Equality + (uninterpreted) Functions

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

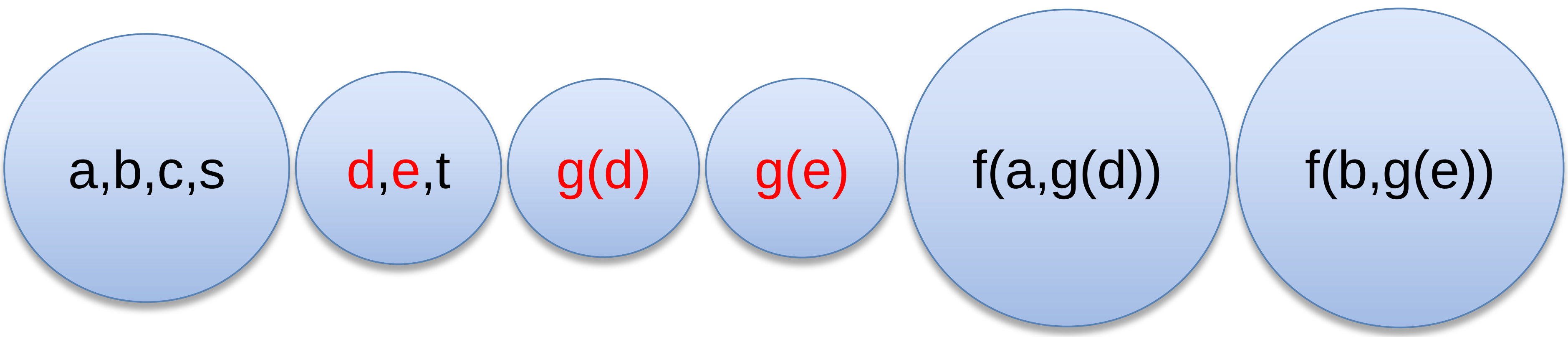


Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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

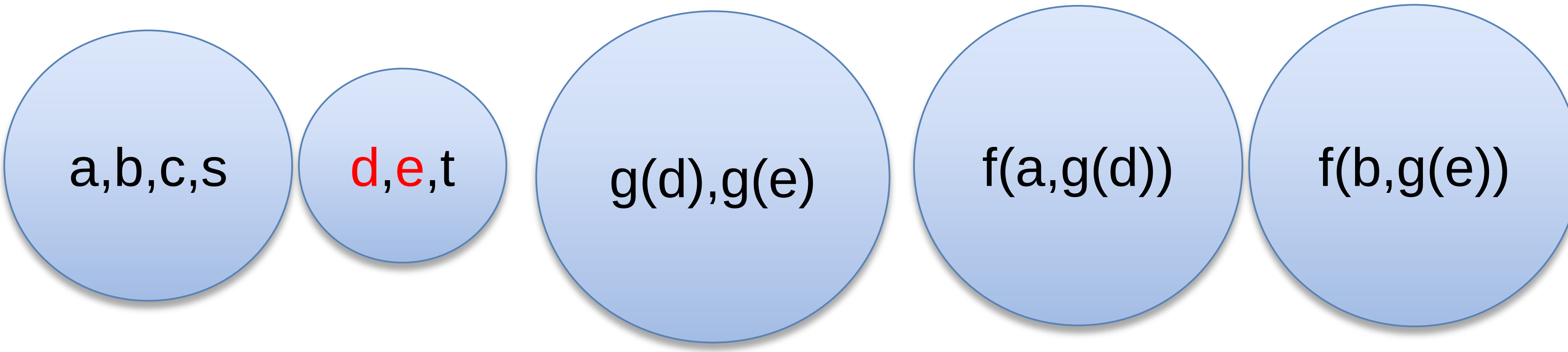


Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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

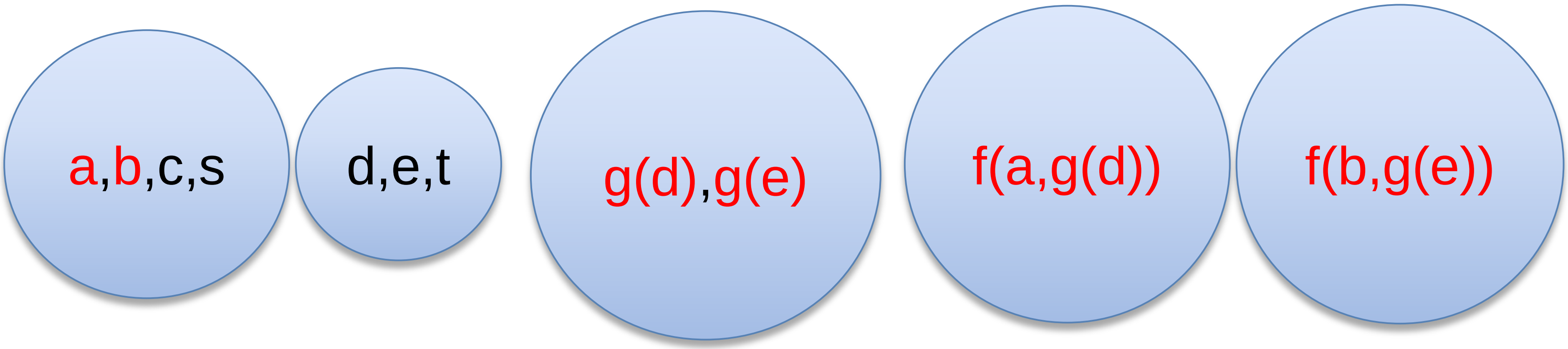


Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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

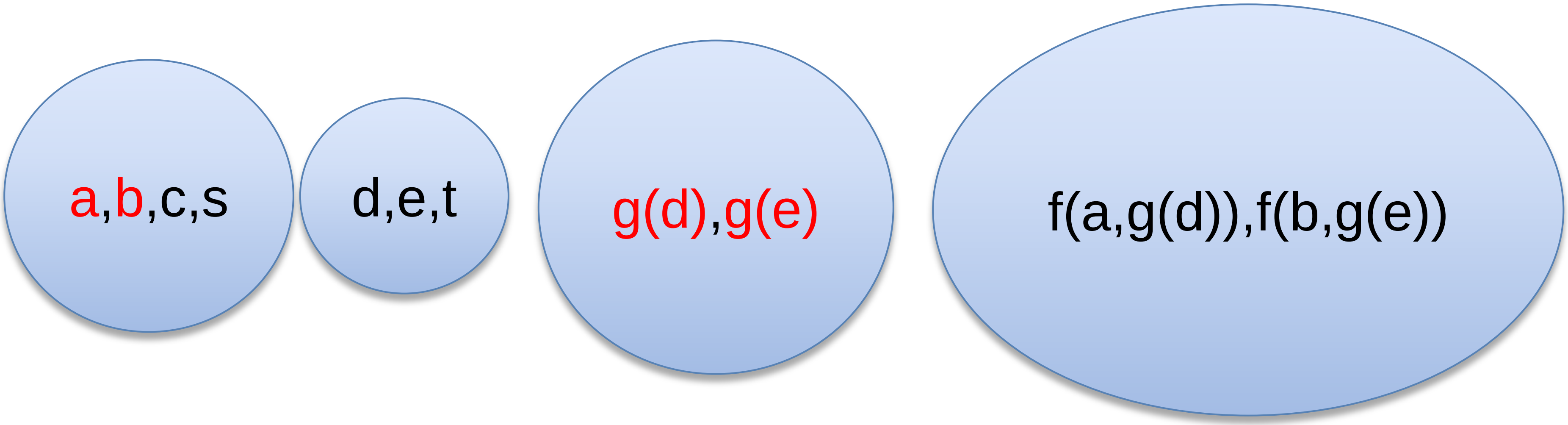


Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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

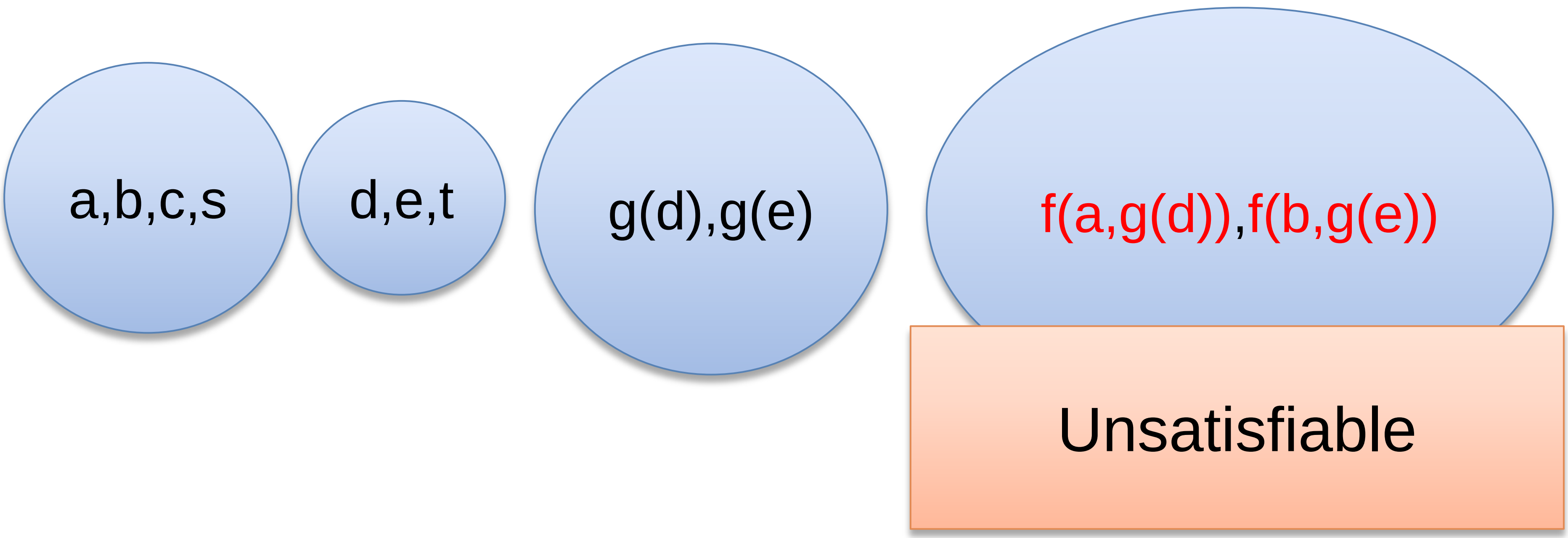


Congruence Rule:

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

Deciding Equality + (uninterpreted) Functions

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



Deciding Equality + (uninterpreted) Functions

(fully shared) DAGs for representing terms

Union-find data-structure + Congruence Closure

$O(n \log n)$

Deciding Polynomial Equations (over C)

$$x^2y - 1 = 0, \quad xy^2 - y = 0, \quad xz - z + 1 = 0$$

Tool: Gröbner Basis

Deciding Polynomial Equations (over C)

Polynomial Ideals:

Algebraic generalization of zeroness

$0 \in I$

$p \in I, q \in I$ implies $p + q \in I$

$p \in I$ implies $pq \in I$

Deciding Polynomial Equations (over C)

The ideal generated by a finite collection of polynomials $P = \{ p_1, \ldots, p_n \}$ is defined as:

$$I(P) = \{p_1 q_1 + \ldots + p_n q_n \mid q_1, \ldots, q_n \text{ are polynomials}\}$$

P is called a basis for $I(P)$.

Intuition:

For all $s \in I(P)$,

$p_1 = 0, \ldots, p_n = 0$ implies $s = 0$

Deciding Polynomial Equations (over C)

Hilbert's Weak Nullstellensatz

$p_1 = 0, \ldots, p_n = 0$ is unsatisfiable over C

iff

$I(\{p_1, \ldots, p_n\})$ contains all polynomials

$1 \in I(\{p_1, \ldots, p_n\})$

Deciding Polynomial Equations (over C)

1st Key Idea: polynomials as rewrite rules.

$$xy^2 - y = 0$$

Becomes

$$xy^2 \rightarrow y$$

The rewriting system is terminating but it is not **confluent**.

$$xy^2 \rightarrow y, \qquad x^2y \rightarrow 1$$

$$\begin{tikzcd}[row sep=small] & xy \\ x^2y^2 \arrow[ur] \arrow[dr] & \\ & y \end{tikzcd}$$

Deciding Polynomial Equations (over C)

2nd Key Idea: Completion.

$$xy^2 \to y, \qquad x^2y \to 1$$

$$\begin{tikzcd} & xy \\ x^2y^2 \arrow[ur] \arrow[dr] & \\ & y \end{tikzcd}$$

Add polynomial:

$$xy - y = 0$$

$$xy \to y$$

Deciding Polynomial Equations (over C)

$x^2y - 1 = 0, \quad xy^2 - y = 0, \quad xz - z + 1 = 0$

$x^2y \to 1, \quad xy^2 \to y, \quad xz \to z - 1$

$x^2y \to 1, \quad xy^2 \to y, \quad xz \to z - 1, \quad xy \to y$

$x^2y \to 1, \quad xy^2 \to y, \quad xz \to z - 1, \quad xy \to y$

$xy \to 1, \quad xy^2 \to y, \quad xz \to z - 1, \quad xy \to y$

$y \to 1, \quad xy^2 \to y, \quad xz \to z - 1, \quad xy \to y$

$y \to 1, \quad x \to 1, \quad xz \to z - 1, \quad xy \to y$

$y \to 1, \quad x \to 1, \quad 1 = 0, \quad xy \to y$

Combining Solvers

In practice, we need a combination of theory solvers.

Nelson-Oppen combination method.

Reduction techniques.

Model-based theory combination.

SAT (propositional checkers): DPLL

M | F

Partial model

Set of clauses

DPLL

- Guessing (case-splitting)

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

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

DPLL

- Deducing

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

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

DPLL

- Backtracking

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

$$\Downarrow$$

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

Modern DPLL

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

Solvers = DPLL + Decision Procedures

- Efficient decision procedures for conjunctions of ground literals.

$$a=b,\ a<5 \mid \neg a=b \lor f(a)=f(b),\quad a < 5 \lor a > 10$$

Theory Conflicts

$a=b$, $a > 0$, $c > 0$, $a + c < 0$ | F

backtrack

Naïve recipe?

SMT Solver = DPLL + Decision Procedure

Standard question:

Why don't you use CPLEX for handling linear arithmetic?

Efficient SMT solvers

Decision Procedures must be:

Incremental & Backtracking

Theory Propagation

$$a=b,\ a<5 \mid \ldots\ a<6 \lor f(a) = a$$

$$\Downarrow$$

$$a=b,\ a<5,\ a<6 \mid \ldots\ a<6 \lor f(a) = a$$

Efficient SMT solvers

Decision Procedures must be:

- Incremental & Backtracking
- Theory Propagation
- Precise (theory) lemma learning
  - $a=b,\ a > 0,\ c > 0,\ a + c < 0 \mid F$
  - Learn clause:
  - $\neg(a=b) \vee \neg(a > 0) \vee \neg(c > 0) \vee \neg(a + c < 0)$
  - Imprecise!
  - Precise clause:
  - $\neg a > 0 \vee \neg c > 0 \vee \neg a + c < 0$

SMT x SAT

For some theories, SMT can be reduced to SAT

Higher level of abstraction

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

SMT x First-order provers

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

T may not have a finite axiomatization

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

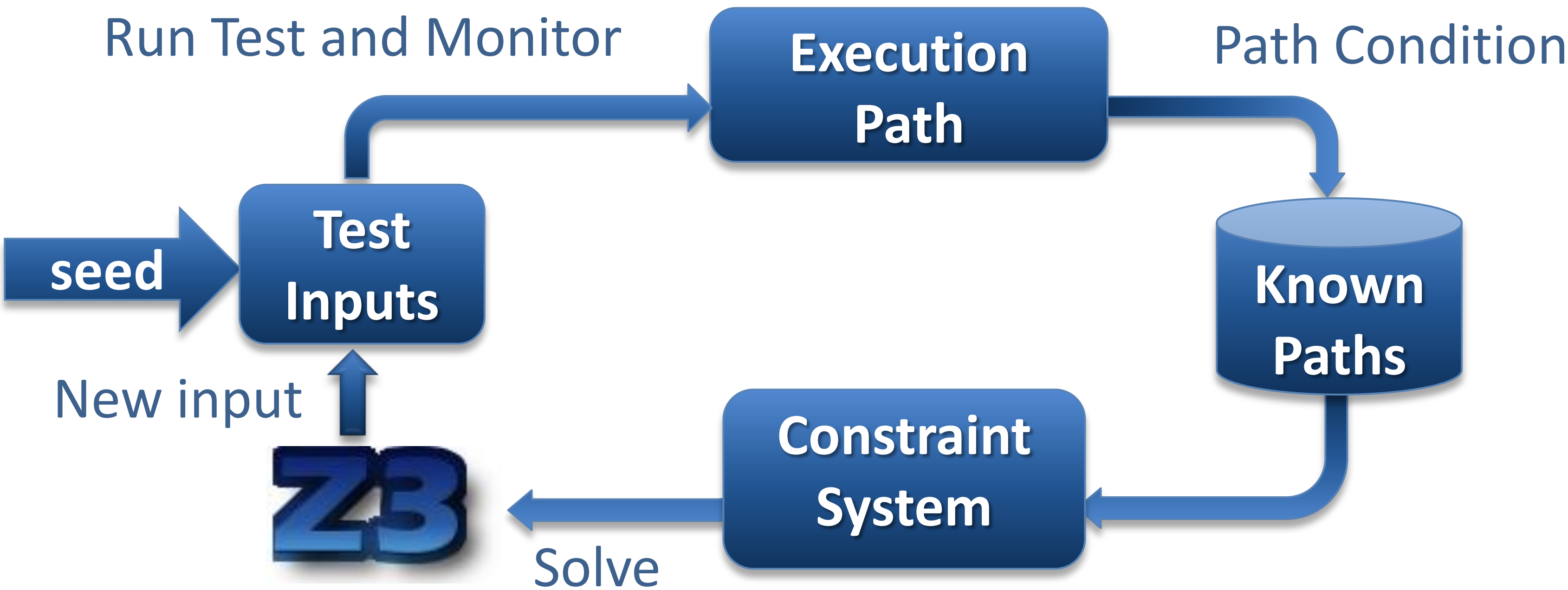
Microsoft Research

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

Directed Automated Random Testing (DART)



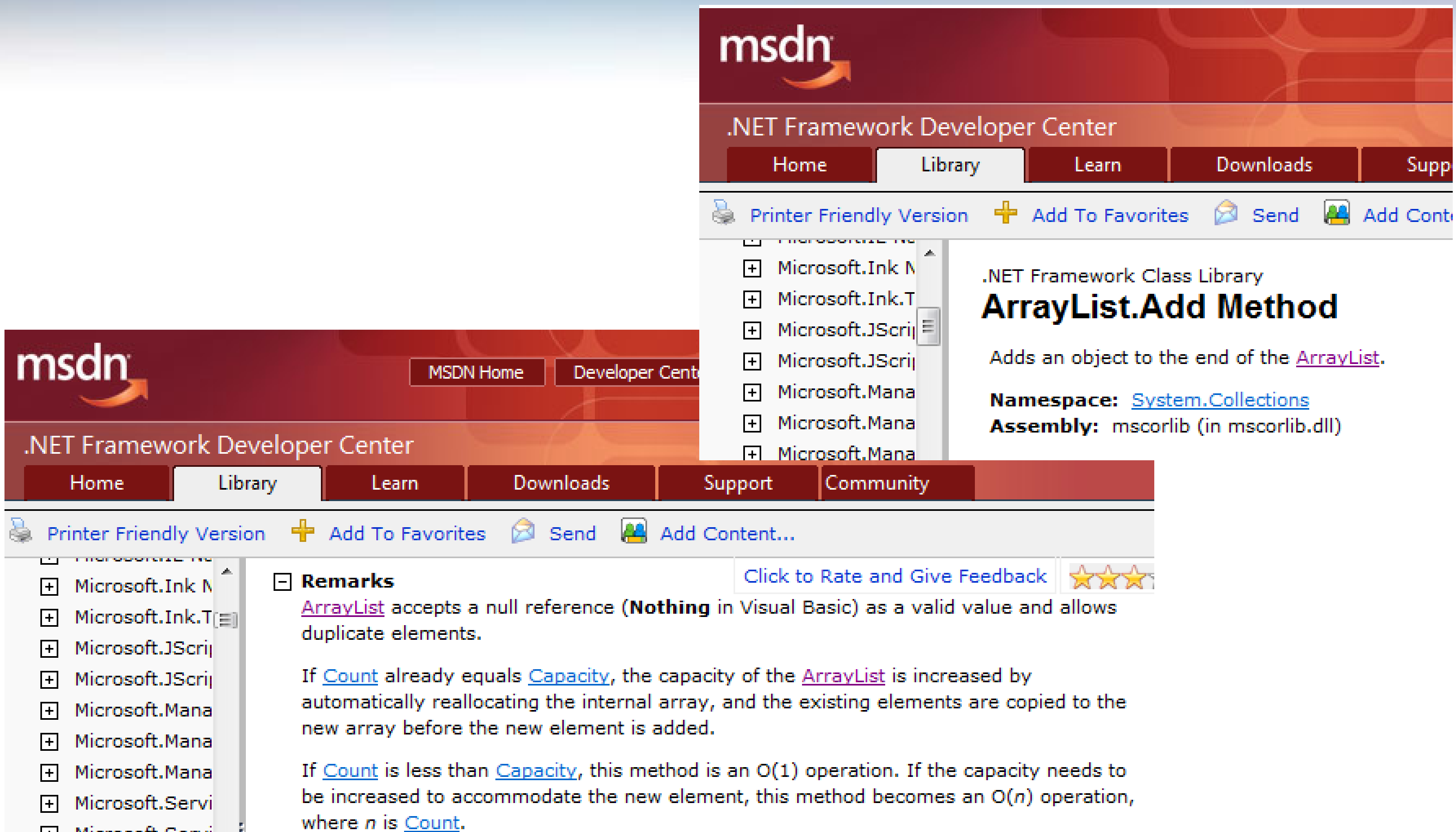
DARTish projects at Microsoft

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

What is *Pex*?

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

ArrayList: The Spec



ArrayList: AddItem Test

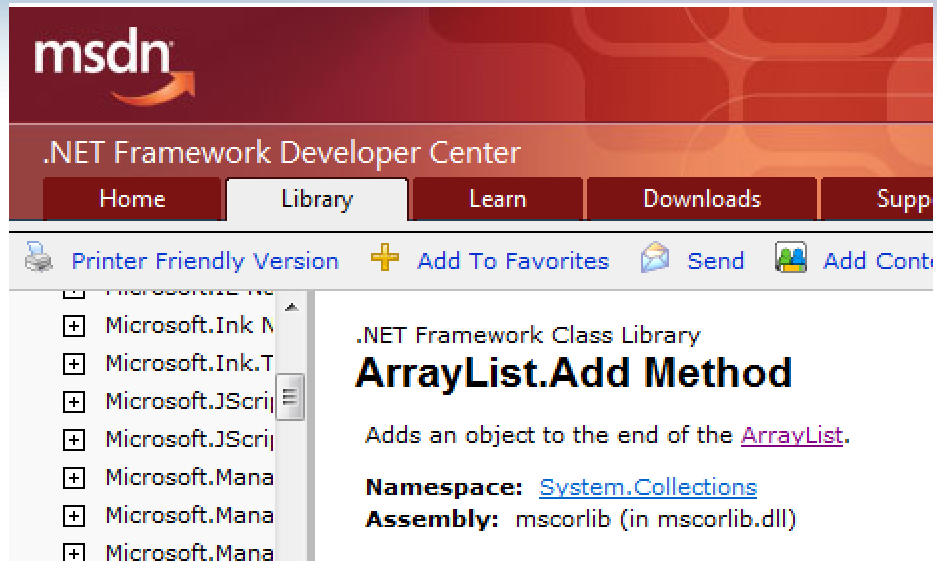
```
class ArrayListTest {
  [PexMethod]
  void AddItem(int c, object item) {
      var list = new ArrayList(c);
      list.Add(item);
      Assert(list[0] == item); }
}
```

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

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

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

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



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

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

ArrayList: Solve constraints using SMT solver

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

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

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

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

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

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

ArrayList: Run 2, (1, null)

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

```
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

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

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`		

Z3

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); }
}
```

```
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); }
}
```

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

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

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

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

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

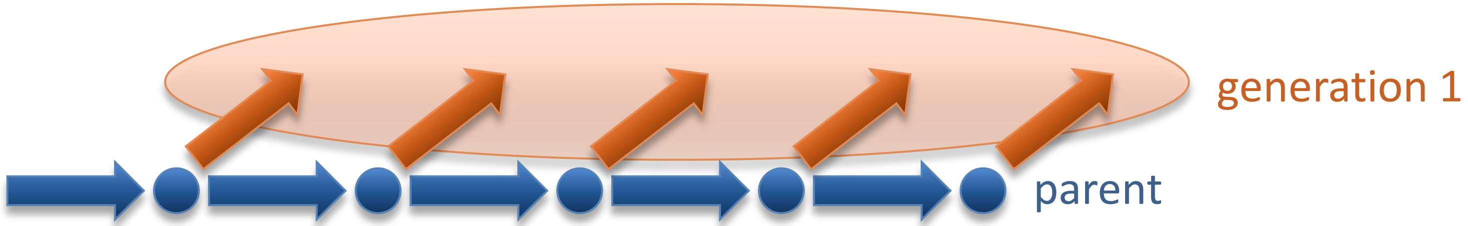
Microsoft Research

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.

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.



Microsoft Research

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

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

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

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

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

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

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

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 $\leftrightarrow$ 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.

Verifying Compilers

Annotated Program → Verification Condition *F*

pre/post conditions
invariants
and other annotations

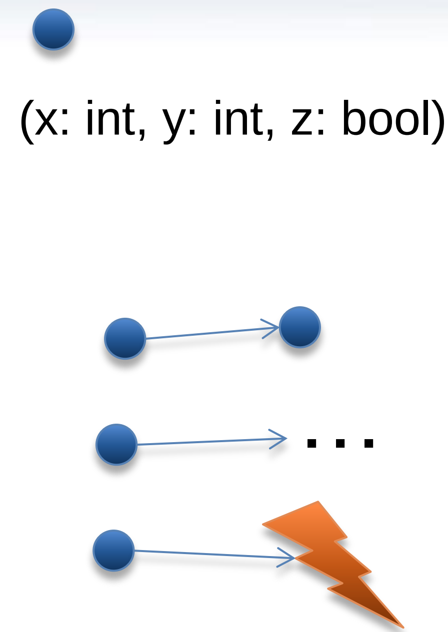
Annotations: Example

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

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

States and execution traces

- State
  - Cartesian product of variables
- Execution trace
  - Nonempty finite sequence of states
  - Infinite sequence of states
  - Nonempty finite sequence of states followed by special error state



Command language

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

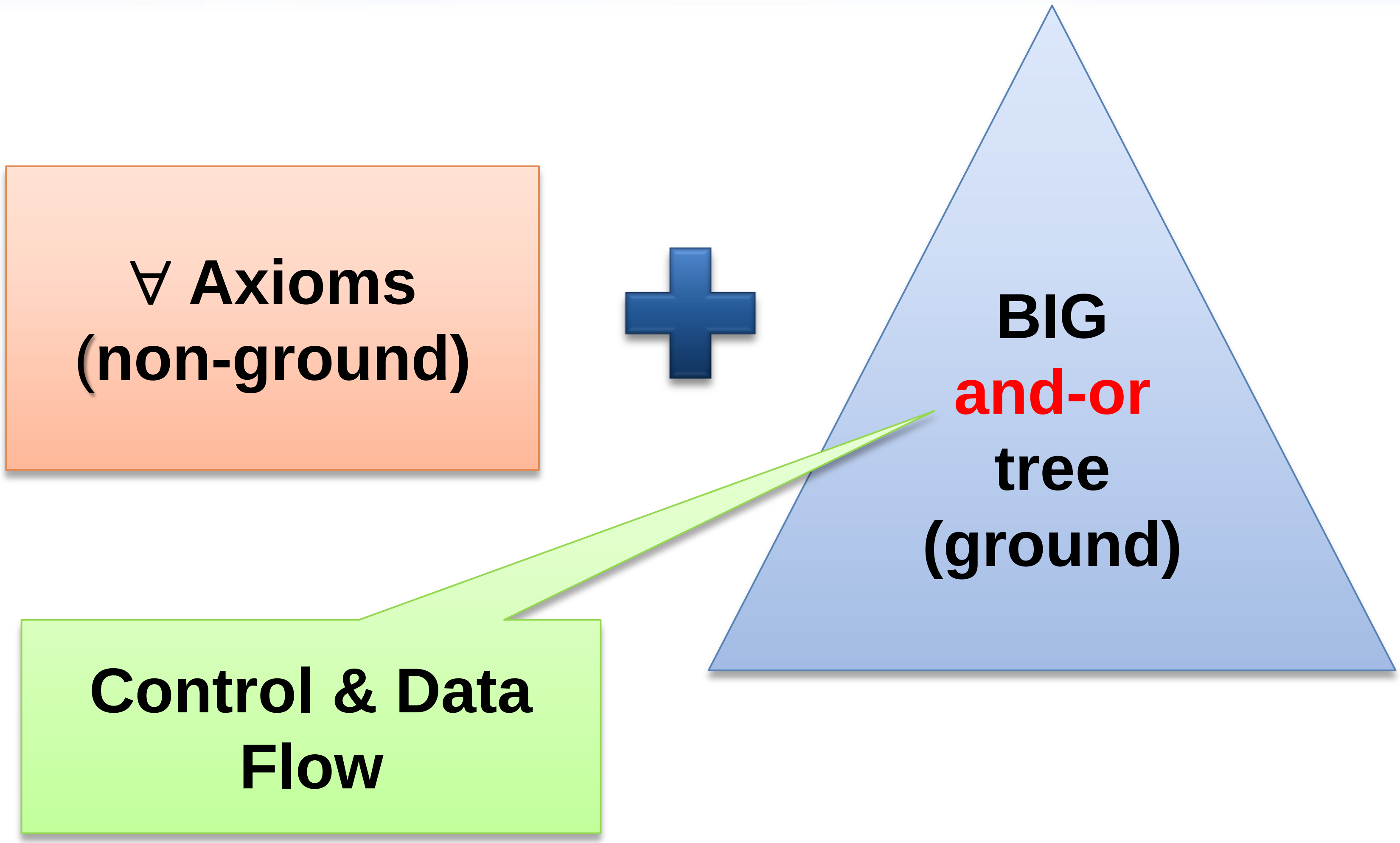
check that the loop invariant holds initially

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

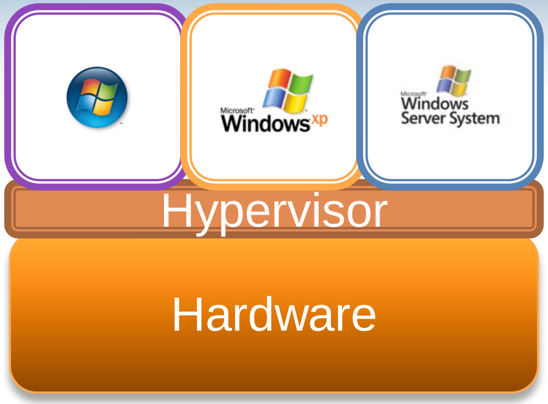
"fast forward" to an arbitrary iteration of the loop

check that the loop invariant is maintained by the loop body

Verification conditions: Structure



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

Challenge: annotation burden

- Partial solutions
  - Automatic generation of: Loop Invariants
  - Houdini-style automatic annotation generation

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

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)

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

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

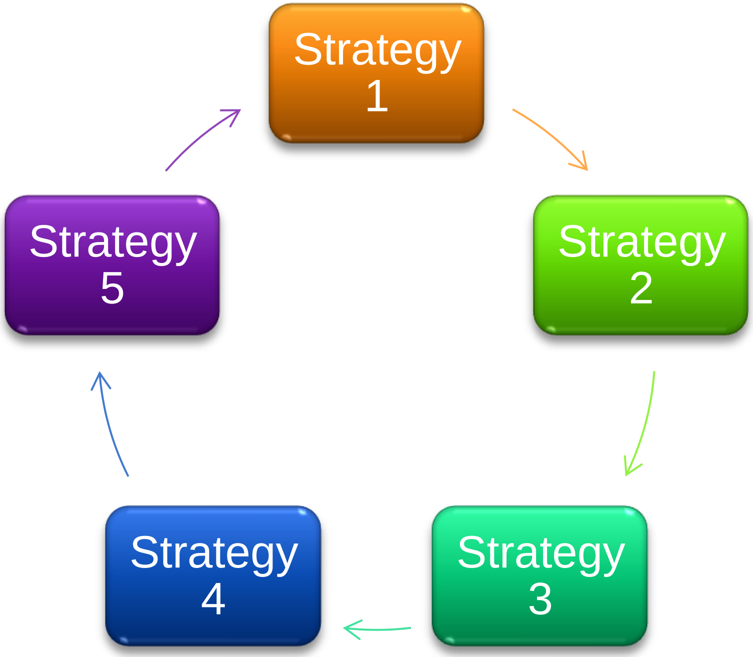
Challenge: Robustness

- 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



Conclusion

- Logic as a platform
- Most verification/analysis tools need symbolic reasoning
- SMT is a hot area
- Many applications & challenges
- http://research.microsoft.com/projects/z3

Thank You!