SMT: Techniques, Hurdles, Applications

SAT/SMT Summer School, MIT, 2011

Leonardo de Moura and Nikolaj Bjorner
Microsoft Research

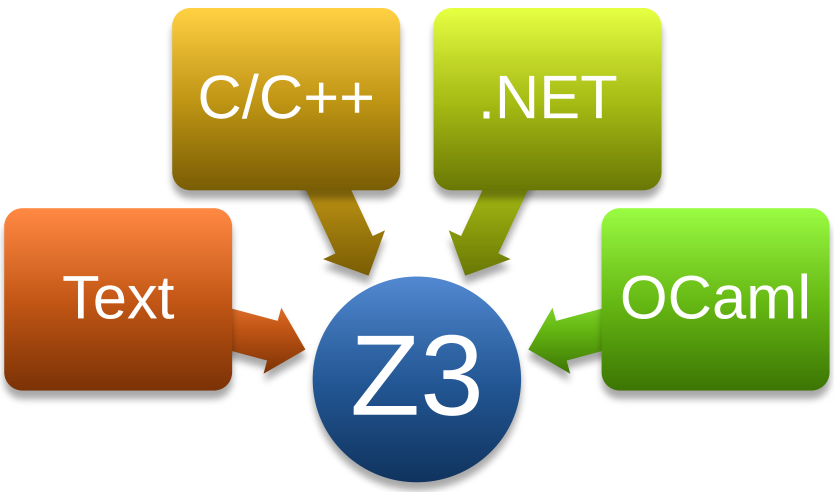
Satisfiability Modulo Theories (SMT)

A Satisfiability Checker

with built-in support for useful theories

SMT@Microsoft: Solver

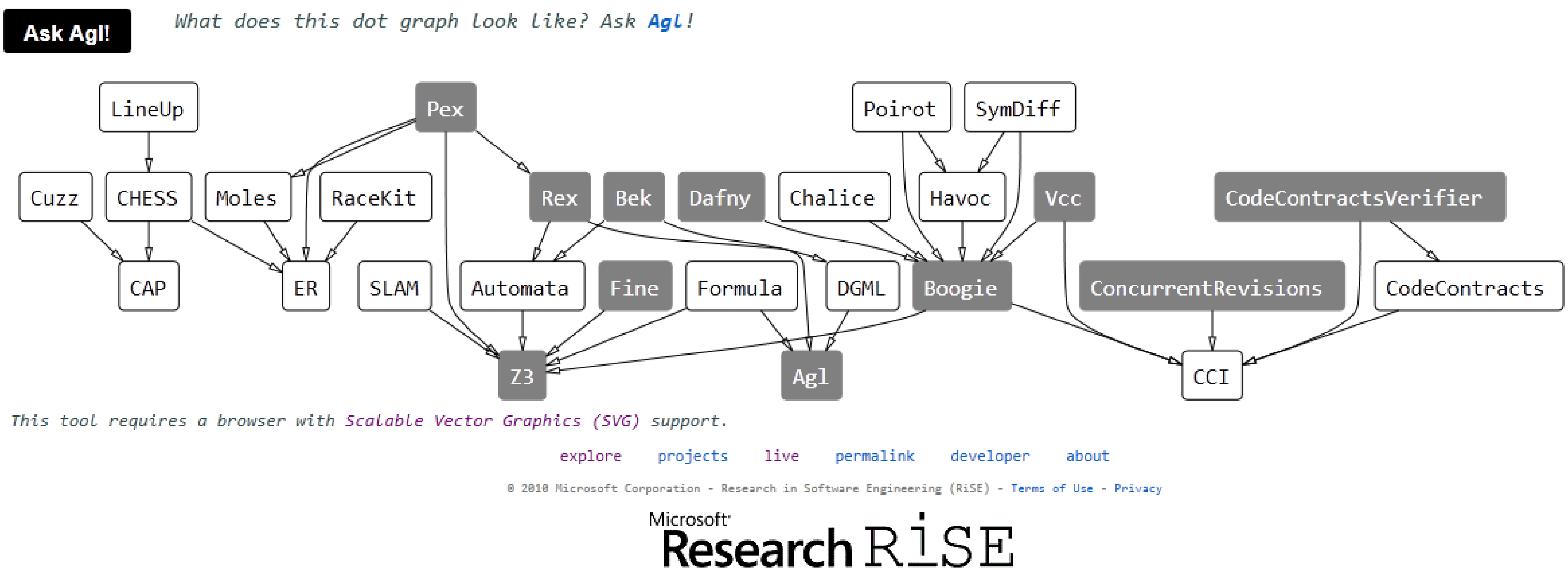
- Z3 is a solver developed at Microsoft Research.
- Development/Research driven by internal customers.
- Free for non-commercial use.
- Interfaces:



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

Microsoft Research

Some Microsoft Tools using Z3



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

Some Microsoft Tools using Z3

Property Driven

Vcc

F7 1.0

Price: FREE

F7 is created to be a simple to use yet ideal typechecker for the F# programming language. F7 supports static checking of properties expressed with refinement types.

Filesize: 3.2 MB

Download now

Details
Screenshot

F7 1.0 simple to use to use Simple To programming language

F

Audit

HAVOC

Type Safety

SLAyer

BOOGIE

Execution Guided

SLAM

Over-Approximation

Yogi

Under-Approximation

SAGE

Pex

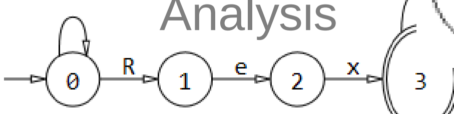
Model Based

The Spec# Programming System

Testing

M3

Analysis



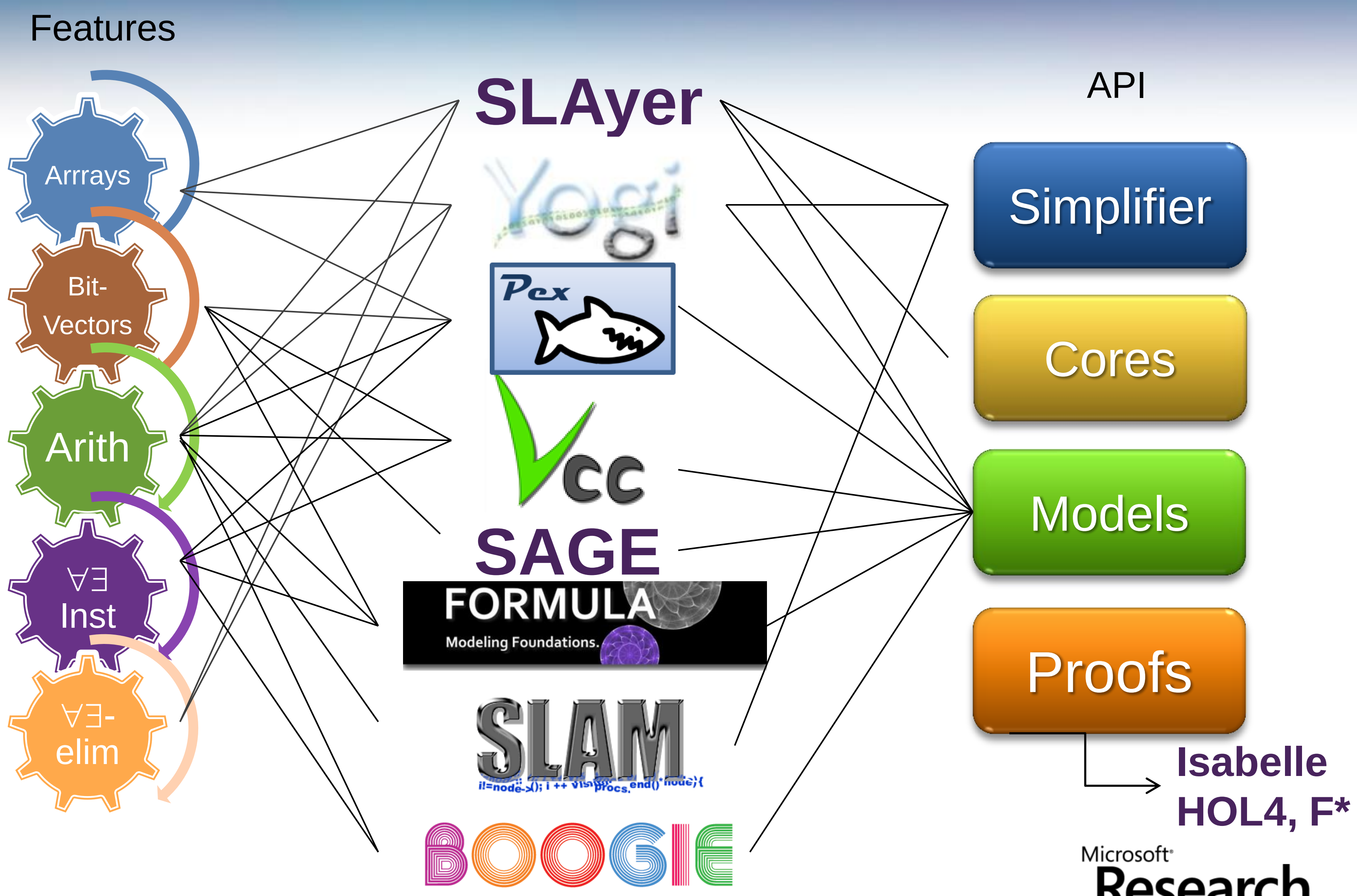
BEK

FORMULA
Modeling Foundations.

Synthesis

Microsoft Research

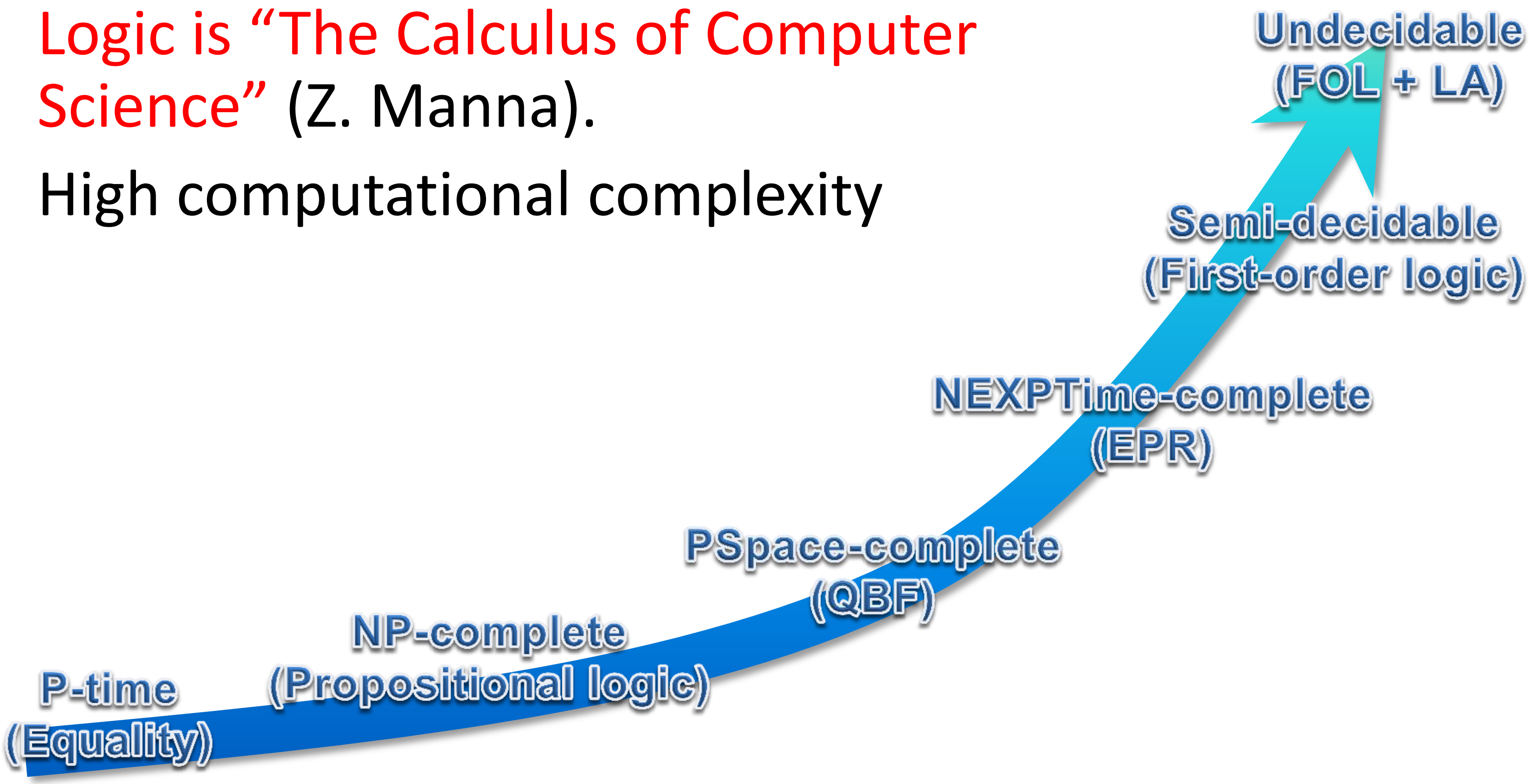
Feature Usage



Microsoft Research

Symbolic Reasoning

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



Microsoft Research

Satisfiability Modulo Theories (SMT)

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

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)

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

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

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Substituting c by b+2

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Simplifying

Microsoft Research

Satisfiability Modulo Theories (SMT)

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

Microsoft Research

Satisfiability Modulo Theories (SMT)

b + 2 = c and f(read(write(a,b,3), b)) ≠ f(3)

Applying array theory axiom

forall a,i,v: read(write(a,i,v), i) = v

Microsoft Research

Satisfiability Modulo Theories (SMT)

$b + 2 = c$ and $f(3) \neq f(3)$

Inconsistent

Microsoft Research

Application Scenarios

"Big" and hard formulas

Thousands of "small" and easy formulas

Short timeout (< 5secs)

Microsoft Research

Application Scenarios

"Big" and hard formulas

The Spec# Programming System

HAVOC

VCC

Thousands of "small" and easy formulas

Yogi

SLAM

Short timeout (< 5secs)

SAGE

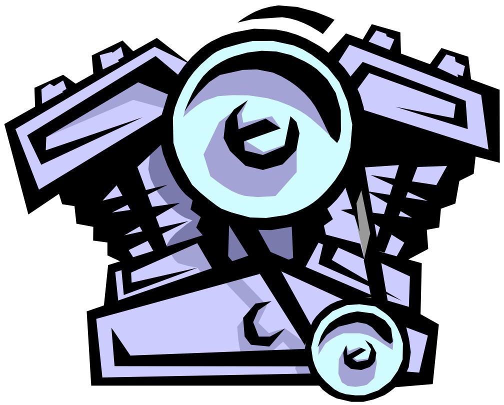
Microsoft Research

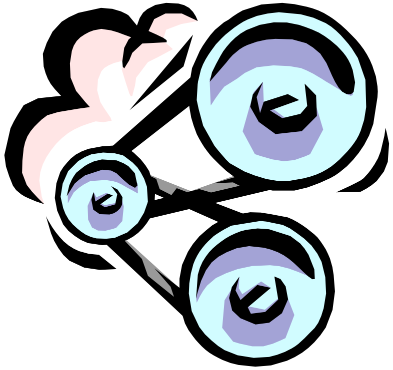
Combining Engines

Current SMT solvers provide

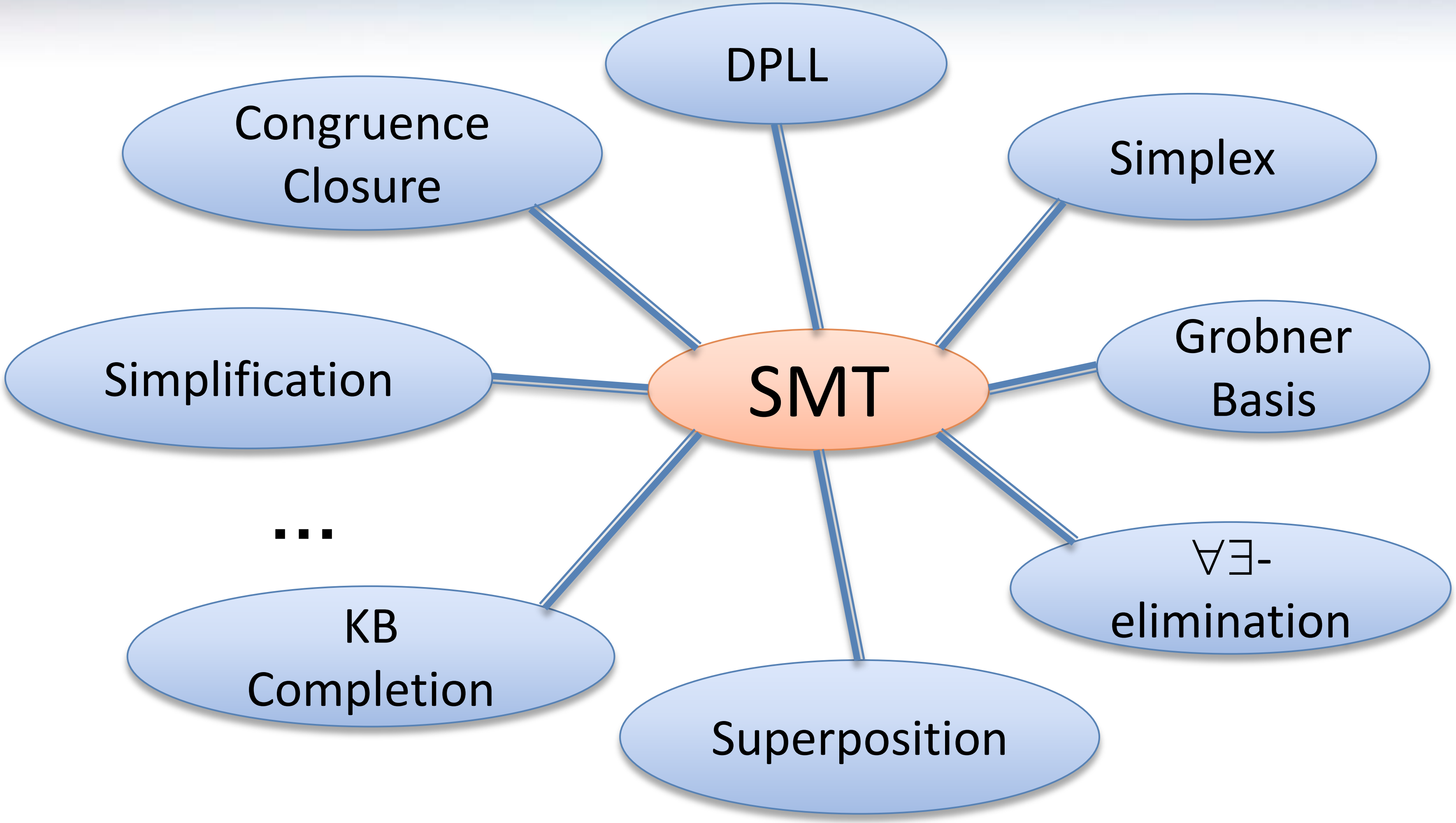
a combination

of different engines





Combining Engines



Main Hurdles in Z3 2.x

Combining Engines

Unfairness

Quadratic behavior

Quantifiers

Abstraction/Relaxation

Linear Integer Arithmetic → Linear Real Arithmetic

SMT → SAT

Arrays → Uninterpreted Functions

Abstraction/Relaxation

Linear Integer Arithmetic → Linear Real Arithmetic

SMT → SAT

Arrays → Uninterpreted Functions

If the relaxation is unsat,
then the original is also unsat

Abstraction/Relaxation

Linear Integer Arithmetic → Linear Real Arithmetic

Refinement: cuts

SMT → SAT

Refinement: theory lemma

Arrays → Uninterpreted Functions

Refinement: array axiom

SMT → SAT Abstraction/Refinement

Basic Idea

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

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

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

SMT → SAT Abstraction/Refinement

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

SMT → SAT Abstraction/Refinement

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

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

SMT → SAT Abstraction/Refinement

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

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

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

SMT → SAT Abstraction/Refinement

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

SMT → SAT Abstraction/Refinement

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

SMT → SAT Abstraction/Refinement

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

Model Guided Approaches

Model Based Theory Combination

Model Based Quantifier Instantiation

Simplex (Linear Real Arithmetic)

Boolector: Extensional Array Theory

CutSat (Linear Integer Arithmetic)

Model Based Theory Combination

$$x = f(y-1), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1$$

Purifying

Microsoft Research

Model Based Theory Combination

$$x = f(z), f(x) \neq f(y), 0 \leq x \leq 1, 0 \leq y \leq 1, z = y - 1$$

Microsoft Research

Model Based Theory Combination

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{y\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 0$
	$\{z\}$	$E(z) = *_3$	$z = y - 1$	$A(z) = -1$
	$\{f(x)\}$	$E(f) = \{*_1 \mapsto *_4,$		
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Assume x = y

Model Based Theory Combination

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, y, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{z\}$	$E(y) = *_1$	$0 \leq y \leq 1$	$A(y) = 0$
$x = y$	$\{f(x), f(y)\}$	$E(z) = *_2$	$z = y - 1$	$A(z) = -1$
		$E(f) = \{*_1 \mapsto *_3,$	$x = y$	
		$*_2 \mapsto *_1,$		
		else $\mapsto *_4\}$		

Unsatisfiable

Model Based Theory Combination

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{y\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 0$
$x \neq y$	$\{z\}$	$E(z) = *_3$	$z = y - 1$	$A(z) = -1$
	$\{f(x)\}$	$E(f) = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Backtrack, and assert $x \neq y$.

$\mathcal{T}_A$ model need to be fixed.

Model Based Theory Combination

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, f(z)\}$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$\{y\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$\{z\}$	$E(z) = *_3$	$z = y - 1$	$A(z) = 0$
	$\{f(x)\}$	$E(f) = \{*_1 \mapsto *_4,$	$x \neq y$	
	$\{f(y)\}$	$*_2 \mapsto *_5,$		
		$*_3 \mapsto *_1,$		
		else $\mapsto *_6\}$		

Assume x = z

Model Based Theory Combination

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$	$\{x, z,$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$f(x), f(z)\}$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$\{y\}$	$E(z) = *_1$	$z = y - 1$	$A(z) = 0$
$x = z$	$\{f(y)\}$	$E(f) = \{*_1 \mapsto *_1,$	$x \neq y$	
		$*_2 \mapsto *_3,$	$x = z$	
		else $\mapsto *_4\}$		

Satisfiable

Model Based Theory Combination

$\mathcal{T}_E$			$\mathcal{T}_A$	
Literals	*Eq. Classes*	*Model*	*Literals*	*Model*
$x = f(z)$ $f(x) \neq f(y)$ $x \neq y$ $x = z$	$\{x, z,$ $f(x), f(z)\}$ $\{y\}$ $\{f(y)\}$	$E(x) = *_1$ $E(y) = *_2$ $E(z) = *_1$ $E(f) = \{*_1 \mapsto *_1,$ $*_2 \mapsto *_3,$ *else* $\mapsto *_4\}$	$0 \leq x \leq 1$ $0 \leq y \leq 1$ $z = y - 1$ $x \neq y$ $x = z$	$A(x) = 0$ $A(y) = 1$ $A(z) = 0$

Let h be the bijection between $|E|$ and $|A|$.

$$h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}$$

Model Based Theory Combination

$\mathcal{T}_E$		$\mathcal{T}_A$	
Literals	*Model*	*Literals*	*Model*
$x = f(z)$	$E(x) = *_1$	$0 \leq x \leq 1$	$A(x) = 0$
$f(x) \neq f(y)$	$E(y) = *_2$	$0 \leq y \leq 1$	$A(y) = 1$
$x \neq y$	$E(z) = *_1$	$z = y - 1$	$A(z) = 0$
$x = z$	$E(f) = \{*_1 \mapsto *_1,$	$x \neq y$	$A(f) = \{0 \mapsto 0$
	$*_2 \mapsto *_3,$	$x = z$	$1 \mapsto -1$
	else $\mapsto *_4\}$		*else* $\mapsto 2\}$

Extending A using h.

$$h = \{*_1 \mapsto 0, *_2 \mapsto 1, *_3 \mapsto -1, *_4 \mapsto 2, \ldots\}$$

Laziness

Delay “Expensive” Engines

Main problem: unfairness

Laziness

Delay "Expensive" Engines

Main problem: unfairness

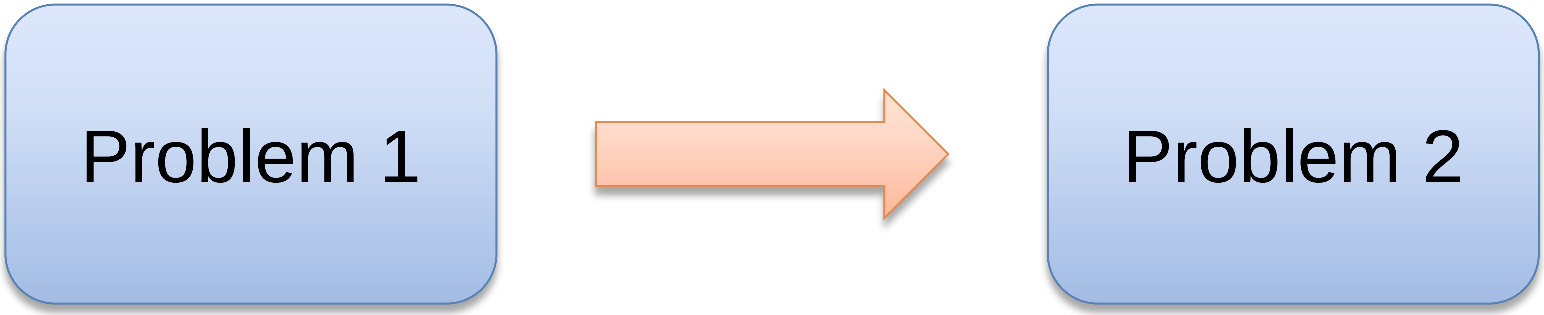
Solutions:

Give a budget to each engine

A single engine should not take control of the whole solver

Allow user to specify their own strategies

Reduction



Reduction: Bit-Vector to SAT

x : bit-vector[2]

y : bit-vector[2]

x + y = 2

Reduction: Bit-Vector to SAT

x : bit-vector[2]

y : bit-vector[2]

x + y = 2

$\Rightarrow$

x_0 : bool

x_1 : bool

y_0 : bool

y_1 : bool

not (x_0 xor y_0)

x_1 xor y_1 xor (x_0 and y_0)

Reduction: Ackermann

Eliminate uninterpreted function symbols

$f(a) \neq f(b)$,

$b = f(c)$,

$a = f(d)$,

$c = d$

Reduction: Ackermann

Eliminate uninterpreted function symbols

$f(a) \neq f(b)$,
$b = f(c)$,
$a = f(d)$,
$c = d$

$\Rightarrow$

$k_{f(a)} \neq k_{f(b)}$,
$b = k_{f(c)}$,
$a = k_{f(d)}$,
$c = d$

Fresh constants

Reduction: Ackermann

Eliminate uninterpreted function symbols

$f(a) \neq f(b),$
$b = f(c),$
$a = f(d),$
$c = d$

$\Rightarrow$

$k_{f(a)} \neq k_{f(b)},$
$b = k_{f(c)},$
$a = k_{f(d)},$
$c = d,$
$a = b \Rightarrow k_{f(a)} = k_{f(b)},$
$a = c \Rightarrow k_{f(a)} = k_{f(c)},$
…
$c = d \Rightarrow k_{f(c)} = k_{f(d)}$

Reduction: Commutative Functions

Commutative Functions to Uninterpreted Functions

$f(a,b) \neq c,$

$c = f(d, a),$

$b = d$

Reduction: Commutative Functions

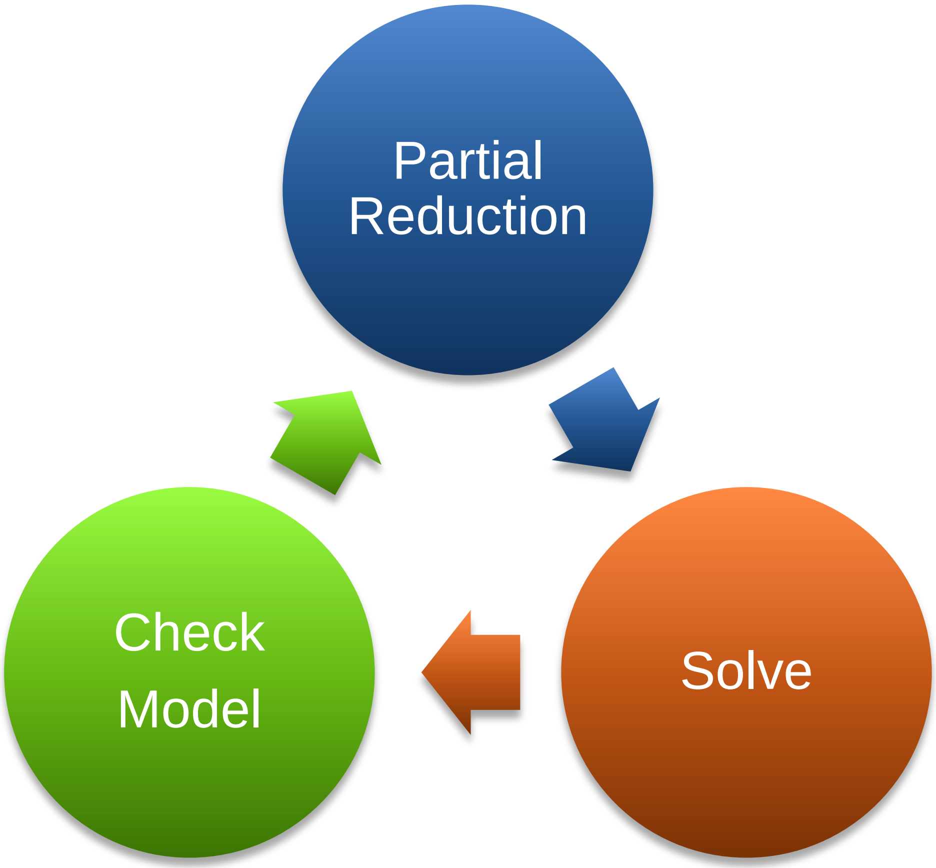
Commutative Functions to Uninterpreted Functions

$f(a,b) \neq c,$
$c = f(d, a),$
$b = d$

$\Rightarrow$

$f(a,b) \neq c,$
$c = f(d, a),$
$b = d,$
$f(a,b) = f(b, a),$
$f(d, a) = f(a, d)$

Lazy Reduction: Model guided + Reduction



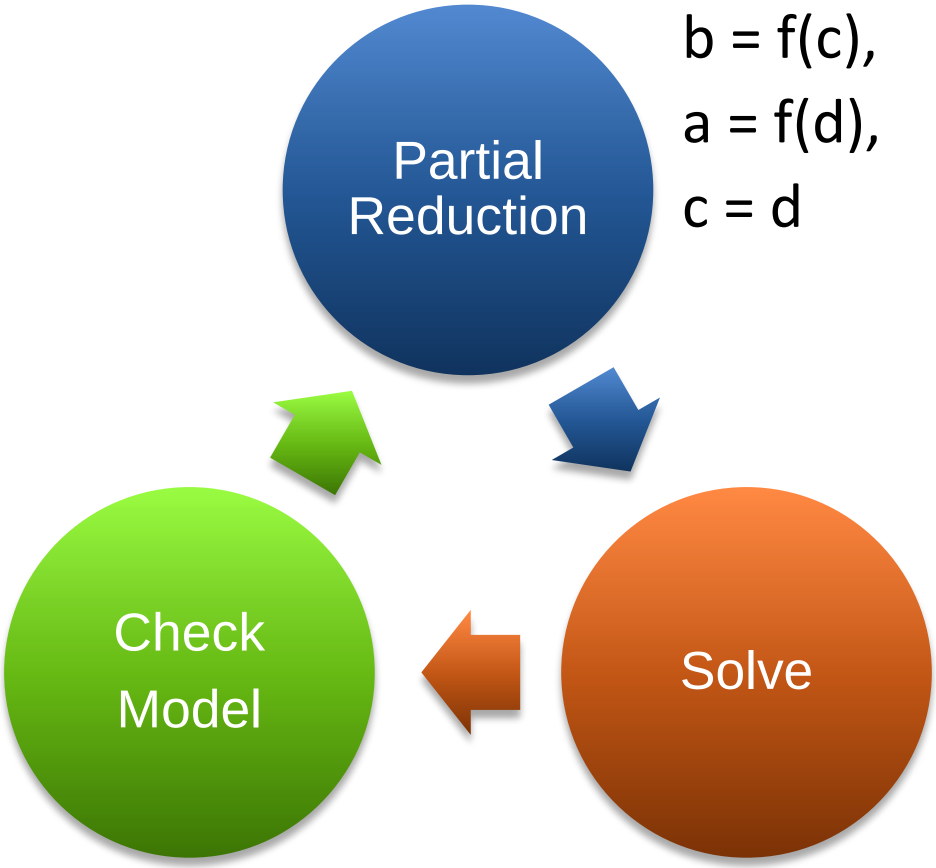
Lazy Reduction: Model guided + Reduction

$f(a) \neq f(b)$,

$b = f(c)$,

$a = f(d)$,

$c = d$



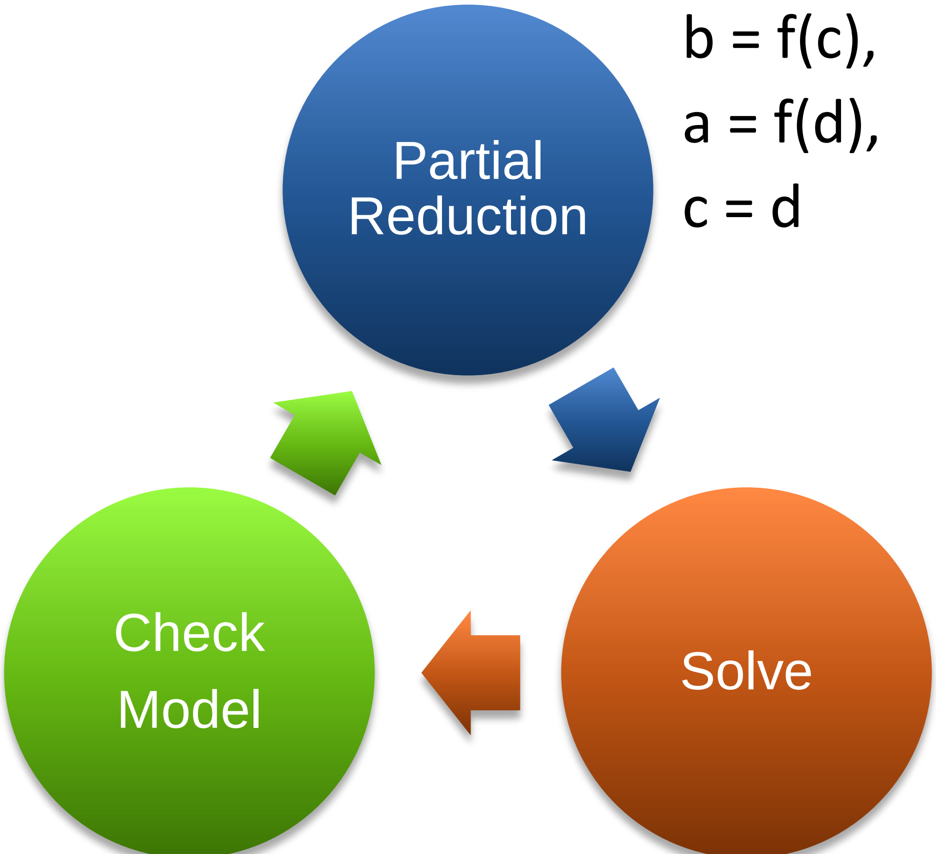
Lazy Reduction: Model guided + Reduction

$f(a) \neq f(b)$,

$b = f(c)$,

$a = f(d)$,

$c = d$



$k_{f(a)} \neq k_{f(b)}$,

$b = k_{f(c)}$,

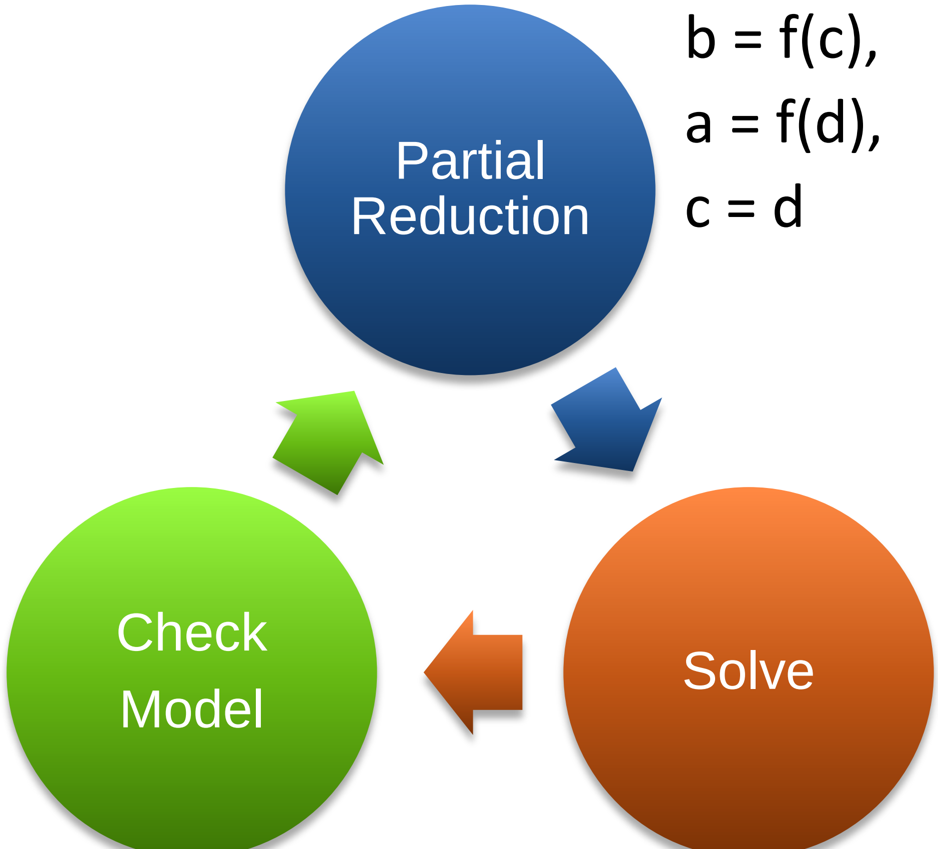
$a = k_{f(d)}$,

$c = d$,

Lazy Reduction: Model guided + Reduction

$f(a) \neq f(b)$,
$b = f(c)$,
$a = f(d)$,
$c = d$

$k_{f(a)} = 0$
$k_{f(b)} = 1$
$b \quad = 2$
$k_{f(c)} = 2$
$a \quad = 3$
$k_{f(d)} = 3$
$c \quad = 4$
$d \quad = 4$



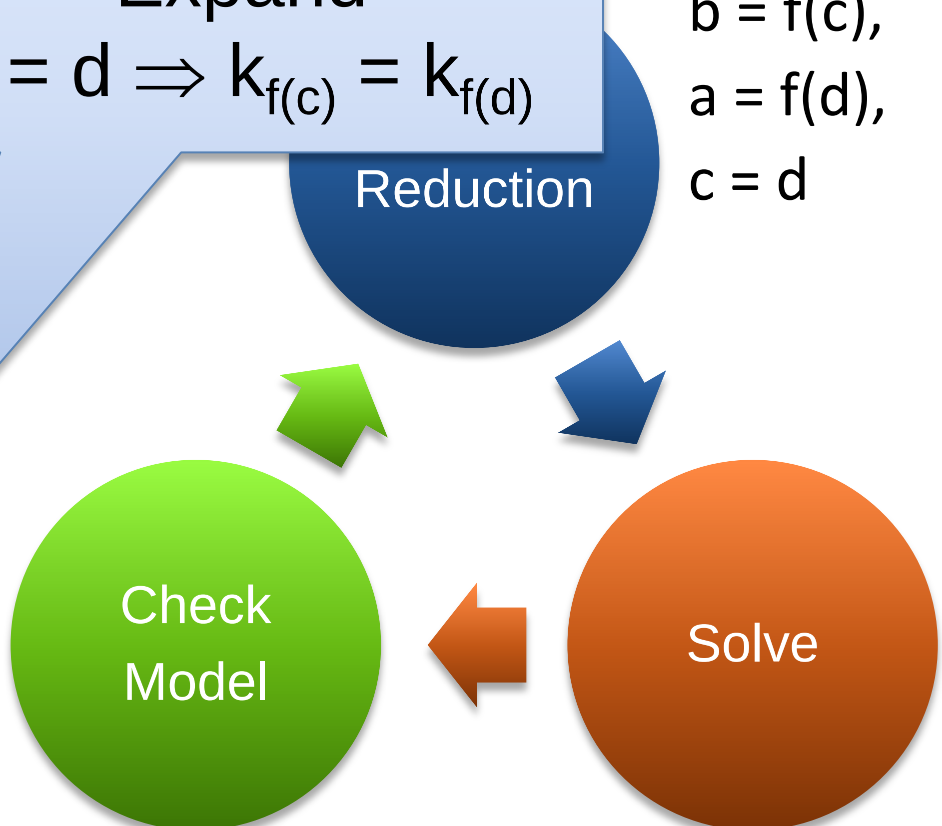
$k_{f(a)} \neq k_{f(b)}$,
$b = k_{f(c)}$,
$a = k_{f(d)}$,
$c = d$,

Lazy Reduction: Model guided + Reduction

Expand

$c = d \Rightarrow k_{f(c)} = k_{f(d)}$

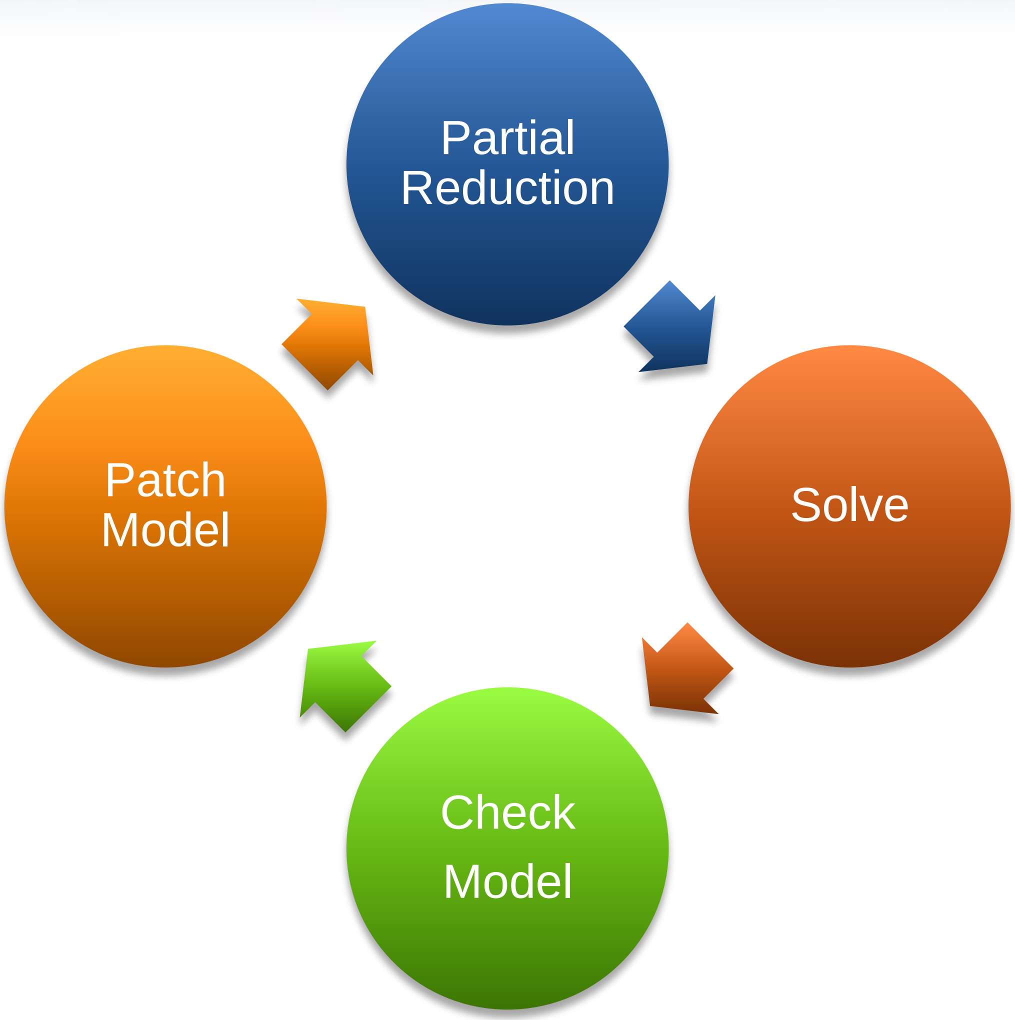
$f(a) \neq f(b),$
$b = f(c),$
$a = f(d),$
$c = d$



$k_{f(a)} = 0$
$k_{f(b)} = 1$
$b = 2$
$k_{f(c)} = 2$
$a = 3$
$k_{f(d)} = 3$
$c = 4$
$d = 4$

$k_{f(a)} \neq k_{f(b)},$
$b = k_{f(c)},$
$a = k_{f(d)},$
$c = d,$

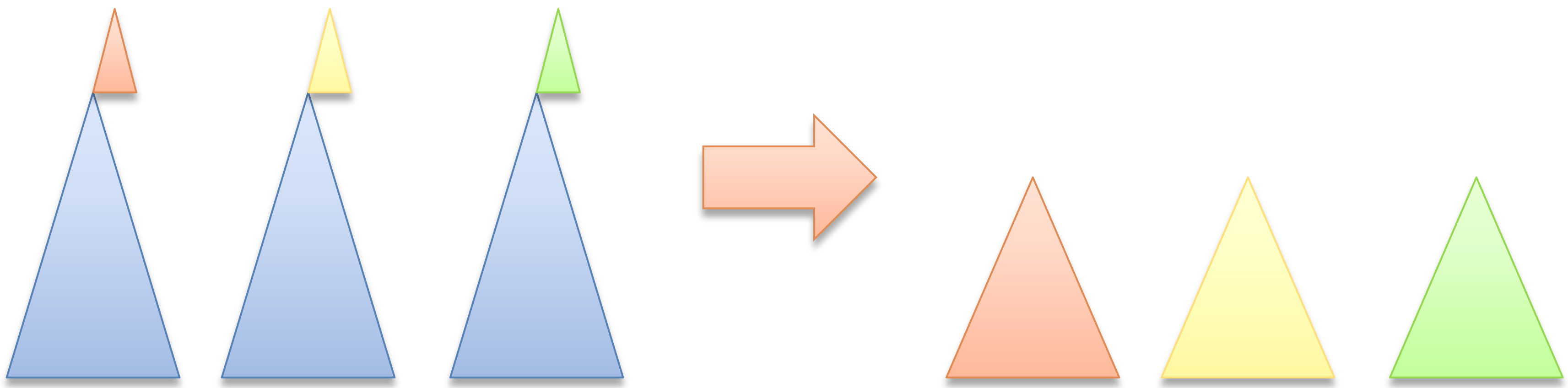
Lazy Reduction: Variation



Simplification Blowup

Reduce Locally, but Expand Globally

due to sharing



Simplification Blowup

Reduce Locally, but Expand Globally

due to sharing

if(c, t, 0) = 2 ➔ c and (t = 2)

Simplification Blowup

Reduce Locally, but Expand Globally

due to sharing

if(c, if(d, 2, t), 0) = 2 ➔ c and (d or t = 2)

if(c, if(d, 2, t), 0) = 0 ➔ (not c) or ((not d) and t = 0)

Simplification Blowup

Reduce Locally, but Expand Globally

Solutions:

1. Apply only to unshared terms
2. Use a budget
3. k-level window

Application: Verifying Compilers

Annotated Program → Verification Condition *F*

pre/post conditions
invariants
and other annotations

Main Challenge

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

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

Microsoft Research

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

Microsoft Research

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

Non ground clauses + interpreted symbols

There is no sound and refutationally complete procedure for

linear arithmetic + unintepreted function symbols

Microsoft Research

Quantifiers: Approaches used in Z3

- Heuristic quantifier instantiation
- Quantifier Elimination
- Complete quantifier instantiation
- Model based quantifier instantiation
- Superposition Calculus

E-matching & Quantifier instantiation

- SMT solvers use heuristic quantifier instantiation.
- E-matching (matching modulo equalities).
- Example:

$\forall$ x: f(g(x)) = x { f(g(x)) }

a = g(b),

b = c,

f(a) $\neq$ c

Trigger

Microsoft Research

E-matching & Quantifier instantiation

- SMT solvers use heuristic quantifier instantiation.
- E-matching (matching modulo equalities).
- Example:

$\forall$ x: f(g(x)) = x { f(g(x)) }

a = g(b),

b = c,

f(a) $\neq$ c

x=b

f(g(b)) = b

Equalities and ground terms come from the partial model M

Microsoft Research

E-matching: why do we use it?

- Integrates smoothly with DPLL.
- Software verification problems are big & shallow.
- Decides useful theories:
  - Arrays
  - Partial orders
  - …

Microsoft Research

Efficient E-matching

- E-matching is NP-Hard.
- In practice

Problem	Indexing Technique
Fast retrieval	E-matching code trees
Incremental E-Matching	Inverted path index

Microsoft Research

E-matching code trees

Trigger:

f(x1, g(x1, a), h(x2), b)

Compiler

Instructions:

1. init(f, 2)
2. check(r4, b, 3)
3. bind(r2, g, r5, 4)
4. compare(r1, r5, 5)
5. check(r6, a, 6)
6. bind(r3, h, r7, 7)
7. yield(r1, r7)

Similar triggers share several instructions.

Combine code sequences in a code tree

E-matching: Limitations

- E-matching needs ground seeds.

  $\forall x: p(x)$,

  $\forall x: \text{not } p(x)$

Microsoft Research

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided triggers:

  $\forall x$: f(g(x))=x { f(g(x)) }

  g(a) = c,

  g(b) = c,

  $a \neq b$

Trigger is too restrictive

Microsoft Research

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided triggers:

  $\forall x: f(g(x))=x \{ g(x) \}$

  $g(a) = c,$

  $g(b) = c,$

  $a \neq b$

More "liberal" trigger

Microsoft Research

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided triggers:

$\forall x: f(g(x))=x \; \{ g(x) \}$

$g(a) = c,$

$g(b) = c,$

$a \neq b,$

$f(g(a)) = a,$

$f(g(b)) = b \Longrightarrow a=b$

Microsoft Research

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided triggers.
- It is not refutationally complete.

False positives

Complete Quantifier Instantiation

Essentially Uninterpreted Fragment.

Universal variables only occur as arguments of uninterpreted symbols.

$$\forall x\text{: } f(x) + 1 > g(f(x))$$

$$\forall x,y\text{: } f(x+y) = f(x) + f(y)$$

Microsoft Research

Complete Quantifier Instantiation

Almost Uninterpreted Fragment.

Relax restriction on the occurrence of universal variables.

$$\text{not } (x \leq y)$$

$$\text{not } (x \leq t)$$

$$f(x + c)$$

$$x =_c t$$

…

Microsoft Research

Complete Quantifier instantiation

- If F is in the almost uninterpreted fragment
- Convert F into an equisatisfiable (modulo T) set of ground clauses F^*
- F^* may be infinite
- It is a decision procedure if F^* is finite
- Subsumes EPR, Array Property Fragment, Stratified Vocabularies for Many Sorted Logic

Microsoft Research

Generating F* (for the essentially unintepreted fragment)

- F induces a system Δ_F of set constraints
- $S_{k,i}$ set of ground instances for variable x_i in clause C_k
- $A_{f,j}$ set of ground j-th arguments of f

j-th argument of f in clause C_k	Set Constraint
a ground term t	$t \in A_{f,j}$
$t\ [x_1,\ldots,x_n]$	$t\ [S_{k,1},\ldots,S_{k,n}] \in A_{f,j}$
x_i	$S_{k,i} \in A_{f,j}$

- F^* is generated using the least solution of Δ_F
- $F^* = \{\ C_k\ [S_{k,1},\ldots,S_{k,n}]\ \mid\ C_k \in F\ \}$

Generating F* (for the essentially unintepreted fragment)

- F induces a system Δ_F of set const
- $S_{k,i}$ set of ground instances for va
- $A_{f,j}$ set of ground j-th arguments

j-th argument of f in clause C_k	Set Con
a ground term t	$t \in A_{f,j}$
$t\ [x_1,\ldots,x_n]$	$t\ [S_{k,1},\ldots$ $,_n] \in A_{f,j}$
x_i	$S_{k,i} \in$ $_{f,j}$

We assume the least solution is not empty

- F^* is generated using the least solution of Δ_F
- $F^* = \{\ C_k\ [S_{k,1},\ldots,S_{k,n}]\ \mid\ C_k \in F\ \}$

Microsoft Research

Generating F*: Example

F

$g(x_1, x_2) = 0 \vee h(x_2) = 0,$
$g(f(x_1),b) + 1 < f(x_1),$
$h(b) = 1, \quad f(a) = 0$

Δ_F

$S_{1,1}= A_{g,1},\ S_{1,2}= A_{g,2},\ S_{1,2}= A_{h,1}$
$S_{2,1}= A_{f,1},\ \ f(S_{2,1}) \subseteq A_{g,1},\ \ b \in A_{g,2}$
$b \in A_{h,1},\ \ a \in A_{f,1}$

Least solution

$S_{1,1}= A_{g,1} = \{ f(a) \}$
$S_{1,2}= A_{g,2} = A_{h,1} = \{b\}$
$S_{2,1}= A_{f,1}= \{a\}$

F*

$g(f(a), b) = 0 \vee h(b) = 0,$
$g(f(a),b) + 1 < f(a),$
$h(b) = 1, \quad f(a) = 0$

Model Checking Quantifiers

M

$a \rightarrow 2,\ b \rightarrow 2,\ c \rightarrow 3$

$f(x) \rightarrow 2$

$h(x) \rightarrow if(x=2, 0, 1)$

$g(x,y) \rightarrow if(x=0 \wedge y=2,-1, 0)$

Does M satisfies?

$\forall x_1, x_2 : g(x_1, x_2) = 0 \vee h(x_2) = 0$

$\forall x_1, x_2: if(x_1=0 \wedge x_2=2,-1,0) = 0 \vee if(x_2=2,0,1) = 0$ is valid

$\exists x_1, x_2: if(x_1=0 \wedge x_2=2,-1,0) \neq 0 \wedge if(x_2=2,0,1) \neq 0$ is unsat

$if(s_1=0 \wedge s_2=2,-1,0) \neq 0 \wedge if(s_2=2,0,1) \neq 0$ is unsat

Microsoft Research

Model-based Quantifier Instantiation

Suppose M does not satisfy a clause C[x] in F.

Add an instance C[t] which "blocks" this spurious model.
Issue: how to find t?

Use a clause C[t] that is in F*.

Quantifier Elimination

- Linear Real/Integer Arithmetic
- Recursive Datatypes
- (some support for) Non-Linear Arithmetic

Model Base Quantifier Instantiation
Demo

Microsoft Research

Conclusion

- SMT is hot at Microsoft
- Z3 is available for non-commercial use
- Main challenges:
  - Combining Engines
  - Quantifiers
- 95% transpiration + 5% inspiration
- Future: “Opening the Black Box”