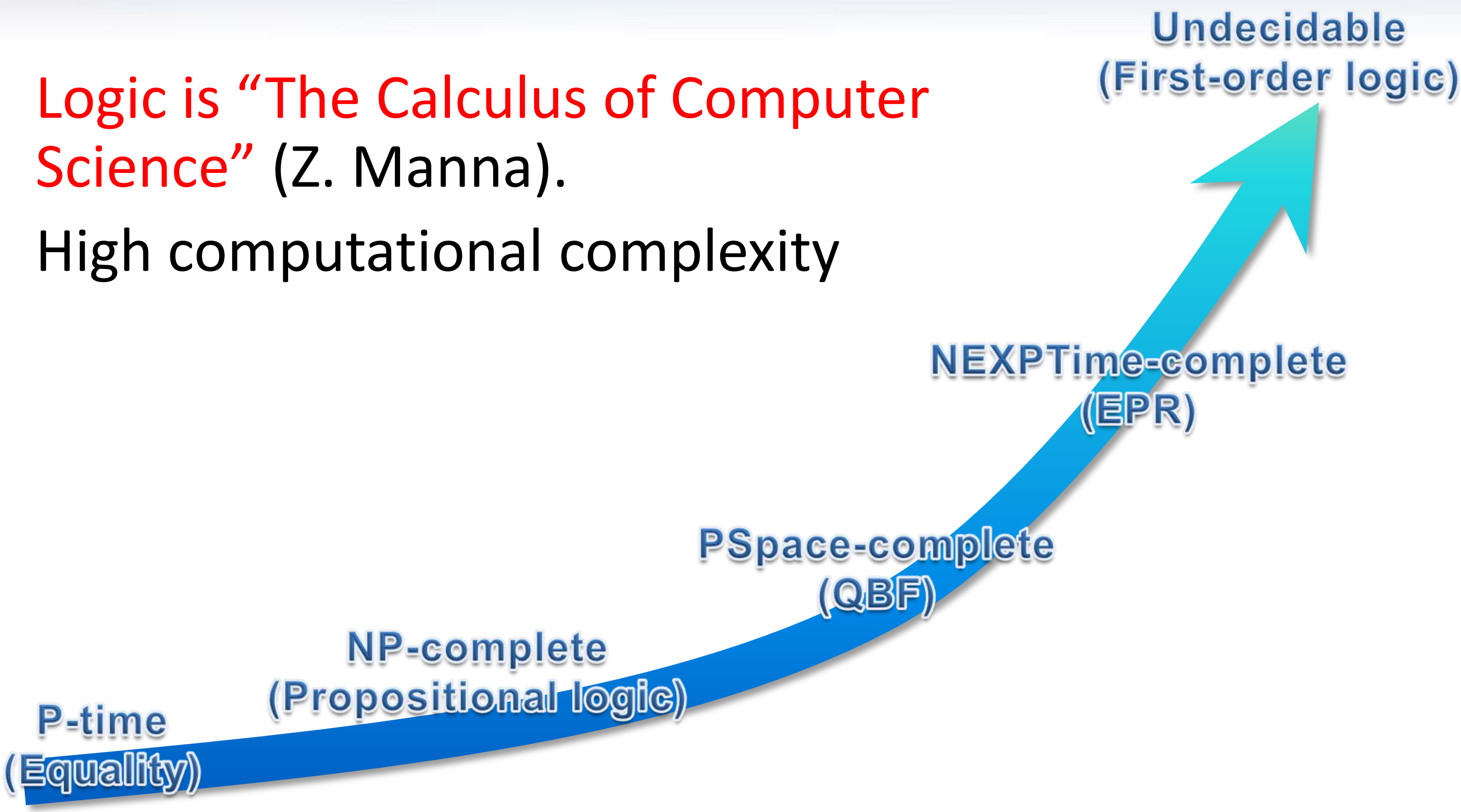
Quantifiers in Satisfiability Modulo Theories

Manchester 2009

Leonardo de Moura
Microsoft Research

Symbolic Reasoning

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



Satisfiability Modulo Theories (SMT)

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

SMT solvers have specialized algorithms for *T*

Satisfiability Modulo Theories (SMT)

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

Satisfiability Modulo Theories (SMT)

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

Arithmetic

Satisfiability Modulo Theories (SMT)

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

Array Theory

Satisfiability Modulo Theories (SMT)

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

Uninterpreted Functions

Theories

- *A Theory is a set of sentences*

- Alternative definition:
  A Theory is a class of structures

- *Th(M)* is the set of sentences that are true in the structure *M*

SMT: Some Applications @ Microsoft

The Spec# Programming System

HAVOC

ForμLa

Hyper-V
Microsoft | Virtualization

Terminator T-2

VCC

SLAM

Yogi

NModel

Vigilante

SpecExplorer

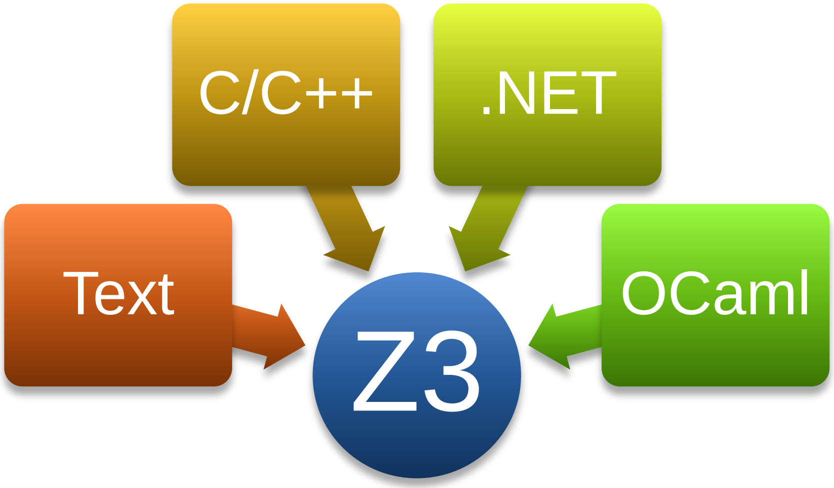
Pex

F7

SAGE

SMT@Microsoft: Solver

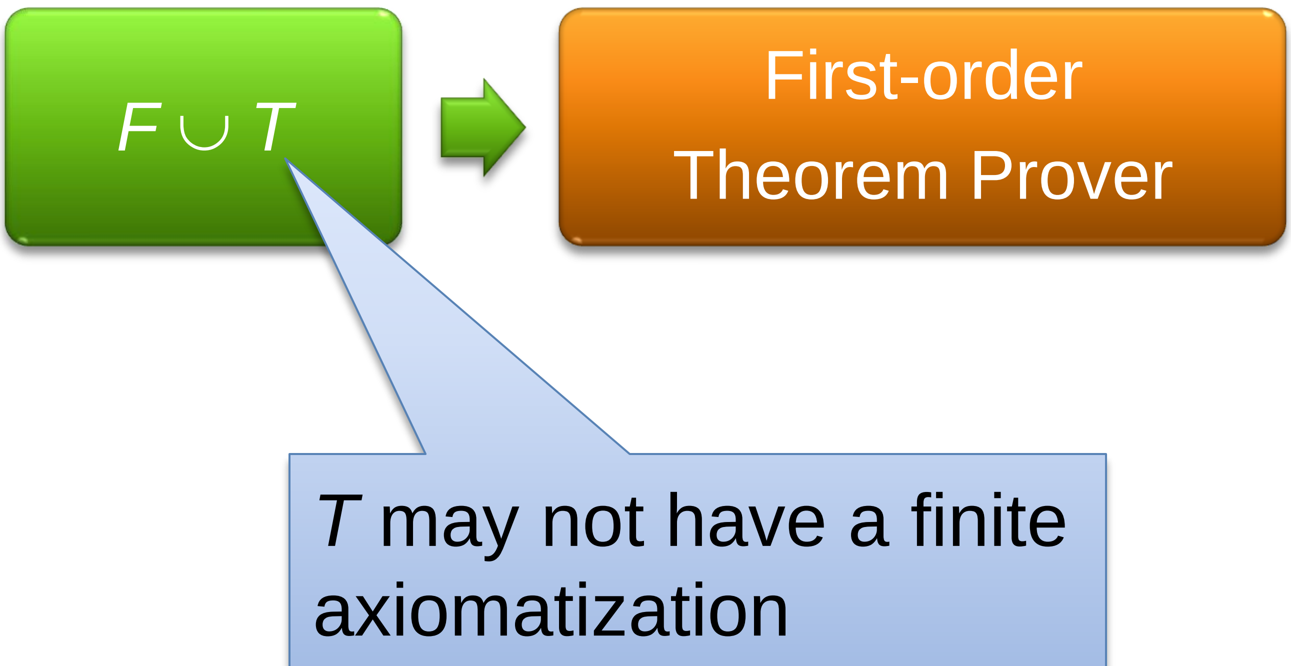
- Z3 is a new solver developed at Microsoft Research.
- Development/Research driven by internal customers.
- Free for academic research.
- Interfaces:



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

Microsoft Research

SMT x First-order provers



SMT x SAT

For some theories, SMT can be reduced to SAT

Higher level of abstraction

$$\mathbf{bvmul_{32}(a,b) = bvmul_{32}(b,a)}$$

Ground formulas

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

Many Applications

Bounded Model Checking

Test-Case Generation

DPLL

M | F

Partial model

Set of clauses

DPLL

- Guessing

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

↓

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

DPLL

- Deducing

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

↓

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

DPLL

- Backtracking

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

↓

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

Solvers = DPLL + Decision Procedures

- Efficient decision procedures for conjunctions of ground atoms.

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

Efficient algorithms

Difference Logic	Belmann-Ford
Uninterpreted functions	Congruence closure
Linear arithmetic	Simplex

Model Generation

- How to represent the model of satisfiable formulae?
- Functor:
  - Given a model *M* for *T*
  - Generate a model *M'* for *F* (modulo *T*)
- Example:

F: f(a) = 0 and a > b and f(b) > f(a) + 1

M':

Symbol	Interpretation
a	1
b	0
f	ite(x=1, 0, 2)

Model Generation

- How to represent the model of satisfiable formulae?
- Functor:
  - Given a model *M* for *T*
  - Generate a model *M’* for *F*
- Example:

F: f(a) = 0 and a > b and f(b) > f(a) + 1

Interpretation is given using *T*-symbols

M’:

Symbol	Interpretation
a	1
b	0
f	ite(x=1, 0, 2)

Model Generation

- How to represent the model of satisfiable formulae?
- Functor:
  - Given a model *M* for *T*
  - Generate a model *M'* for *F* (mo
- Example:

F: f(a) = 0 and a > b and f(b) > f(a) + 1

Non ground term (lambda expression)

M':

Symbol	Interpretation
a	1
b	0
f	ite(x=1, 0, 2)

Model Checking

M':

Symbol	Interpretation
a	1
b	0
f	ite(x=1, 0, 2)

Is ∀x: f(x) > 0 satisfied by *M'*?

Yes,
not (ite(k=1,0,2) > 0) is unsatisfiable

Model Checking

M':

Symbol	Interpretation
a	1
b	0
f	ite(x=1, 0, 2)

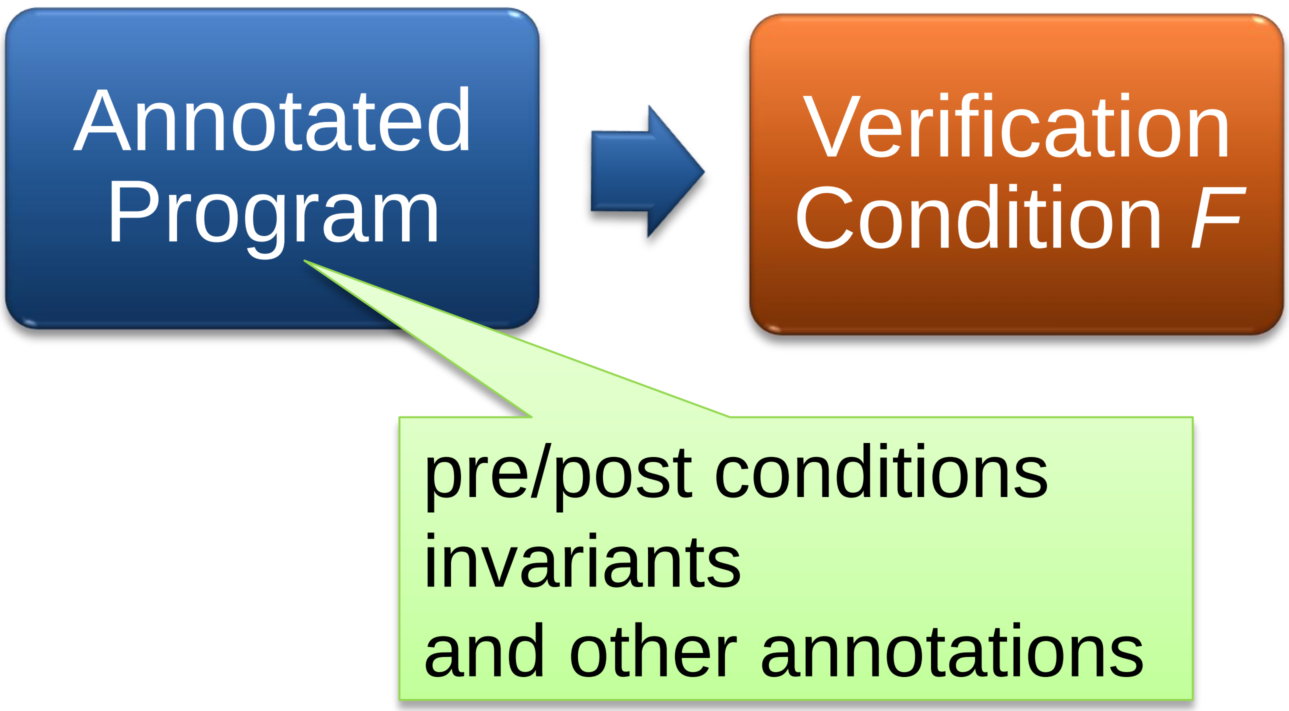
Is ∀x: f(x) > 0 satisfied by *M'*?

Yes,
not (ite(k=1,0,2) > 0) is unsatisfiable

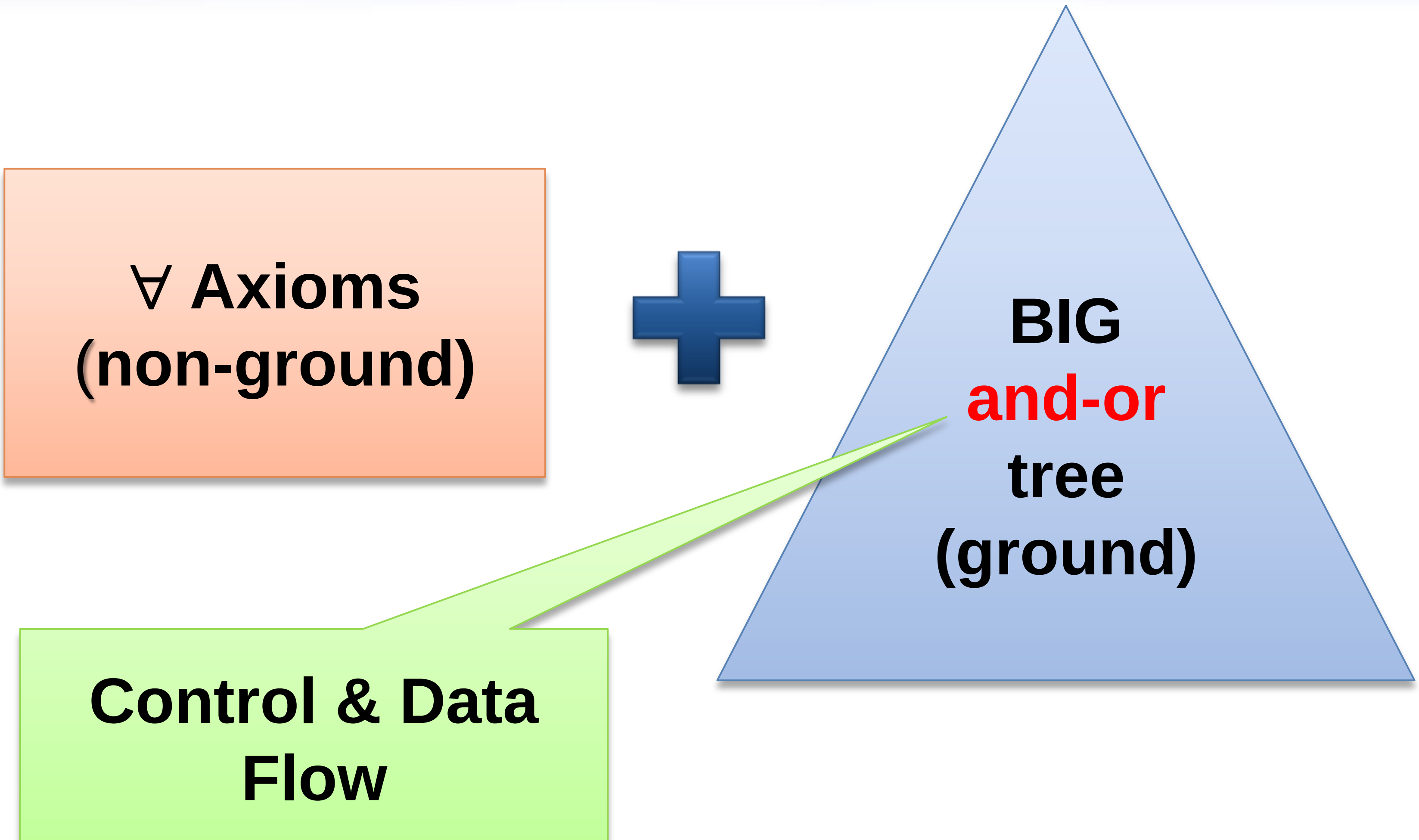
- Negated quantifier
- Replaced *f* by its interpretation
- Replaced *x* by fresh constant *k*

Microsoft Research

Verifying Compilers



Verification conditions: Structure



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

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

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)

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!

Some statistics

- Grand challenge: Microsoft Hypervisor
- 70k lines of dense C code
- VCs have several Mb
- Thousands of non ground clauses
- Developers are willing to wait at most 5 min per VC

Many Approaches

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

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

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

E-matching: why do we use it?

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

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

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),

  $\forall x$: not p(x)

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

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

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

$\Rightarrow$ a=b

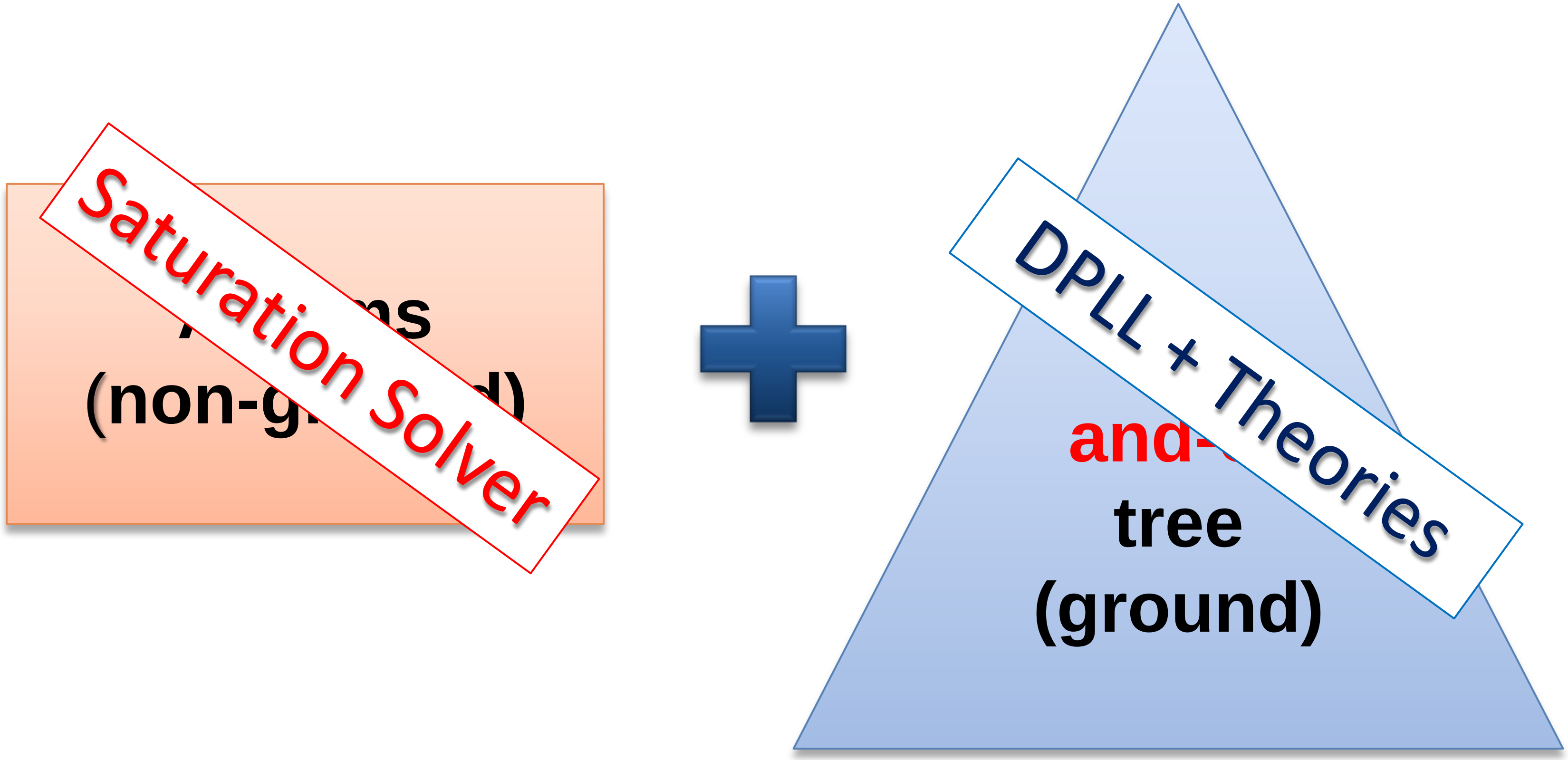
E-matching: Limitations

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

False positives

DPLL(Γ)

- Tight integration: DPLL + Saturation solver.



DPLL(Γ)

- Inference rule:

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

- DPLL(Γ) is parametric.
- Examples:
  - Resolution
  - Superposition calculus
  - …

DPLL(Γ)

M | F

Partial model

Set of clauses

DPLL(Γ): Deduce I

$p(a) \mid p(a) \lor q(a),\ \forall x{:}\ \neg p(x) \lor r(x),\ \forall x{:}\ p(x) \lor s(x)$

DPLL(Γ): Deduce I

$p(a) \mid p(a) \vee q(a), \neg p(x) \vee r(x), p(x) \vee s(x)$

DPLL(Γ): Deduce I

$p(a) \mid p(a) \vee q(a), \neg p(x) \vee r(x), p(x) \vee s(x)$

Resolution

$p(a) \mid p(a) \vee q(a), \neg p(x) \vee r(x), p(x) \vee s(x), r(x) \vee s(x)$

DPLL(Γ): Deduce II

- Using ground atoms from M:

$$M \mid F$$

- Main issue: backtracking.
- Hypothetical clauses:

$$H \triangleright C$$

Track literals from M used to derive C

(hypothesis) Ground literals

(regular) Clause

DPLL(Γ): Deduce II

$$p(a) \mid p(a) \vee q(a),\ \neg p(x) \vee r(x)$$

$$\frac{p(a), \quad \neg p(x) \vee r(x)}{r(a)}$$

$$p(a) \mid p(a) \vee q(a),\ \neg p(x) \vee r(x),\ p(a) \triangleright r(a)$$

DPLL(Γ): Backtracking

$p(a), r(a) \mid p(a) \lor q(a), \neg p(a) \lor \neg r(a), \mathbf{p(a) \triangleright r(a)}, \ldots$

DPLL(Γ): Backtracking

$$p(a), r(a) \mid p(a) \lor q(a), \neg p(a) \lor \neg r(a), \; p(a) \rhd r(a), \ldots$$

p(a) is removed from M

$$\neg p(a) \mid p(a) \lor q(a), \neg p(a) \lor \neg r(a), \ldots$$

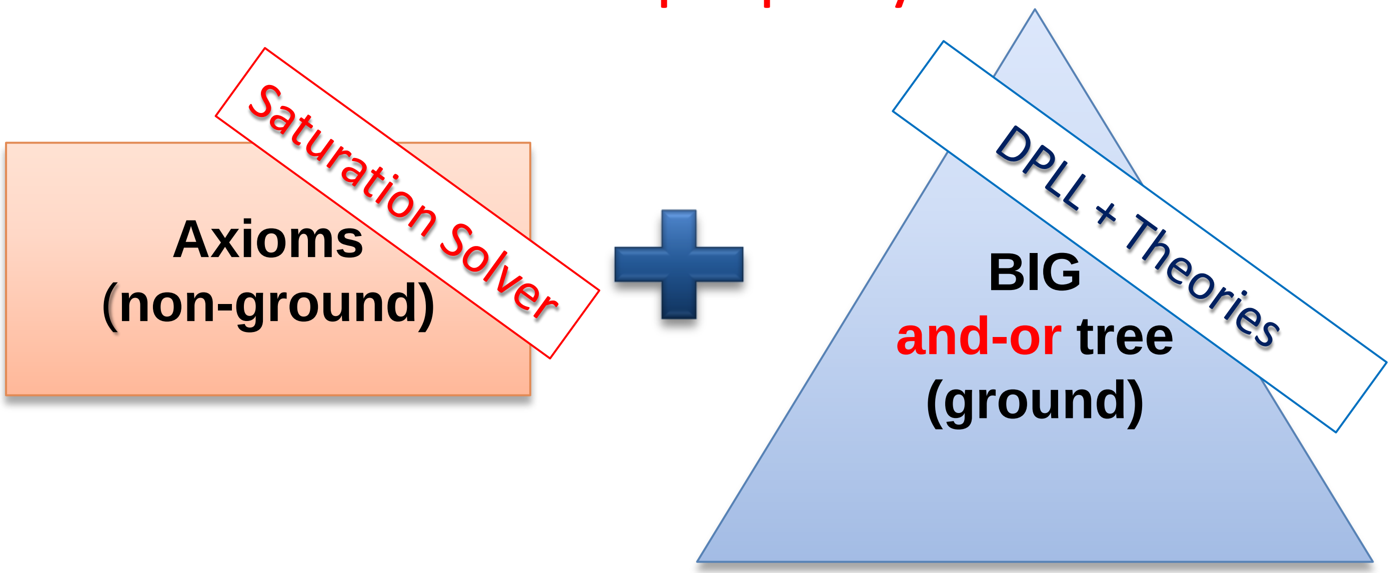
DPLL(Γ): Improvement

- Saturation solver ignores non-unit ground clauses.

$$p(a) \mid p(a) \lor q(a),\ \neg p(x) \lor r(x)$$

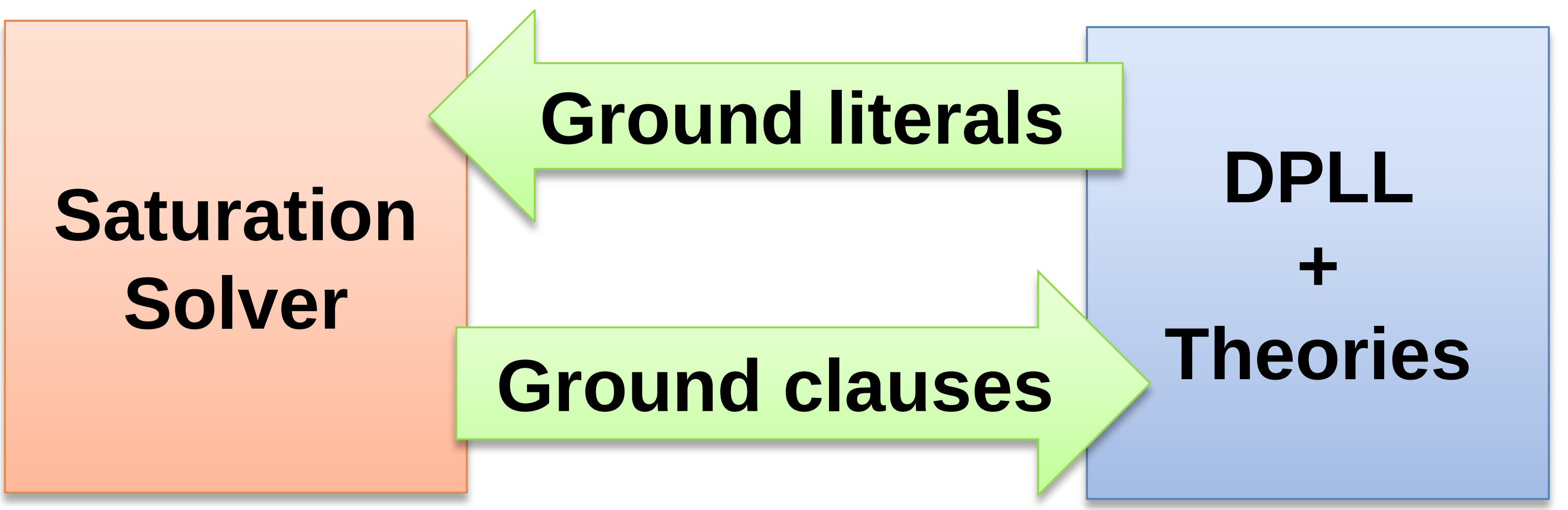
DPLL(Γ): Improvement

- Saturation solver ignores non-unit ground clauses.
- It is still refutanionally complete if:
  - Γ has the reduction property.



DPLL(Γ): Improvement

- Saturation solver ignores non-unit ground clauses.
- It is still refutanionally complete if:
  - Γ has the reduction property.



DPLL(Γ): Problem

- Interpreted symtbols

  $\neg(f(a) > 2)$, $f(x) > 5$

- It is refutationally complete if
  - Interpreted symbols only occur in ground clauses
  - Non ground clauses are variable inactive
  - “Good” ordering is used

Non ground clauses + interpreted symbols

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

Essentially unintepreted fragment

- Universal variables only occur as arguments of uninterpreted symbols.

$$\forall x\colon f(x) + 1 > g(f(x))$$ ✔

$$\forall x,y\colon f(x+y) = f(x) + f(y)$$ 🚫

Almost unintepreted fragment

- Relax restriction on the occurrence of universal variables.

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

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

$$f(x + c)$$

$$x =_c t$$

…

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

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
- $F^* = \{\ C_k\ [S_{k,1},\ldots,S_{k,n}]\ \mid\ C_k \in F\ \}$

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*

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

Refutationally complete procedure

- Compactness

  A set F of first order sentences is unsatisifiable iff it contains an unsatisfiable finite subset

- If we view T as a set of sentences

  Apply compactness to $T \cup F^*$

Example

$\forall x$: f(f(x)) > f(x)

$\forall x$: f(x) < a

f(0) = 0

Satisfiable if *T* is Th(Z), but unsatisfiable *T* is the the class of structures Exp(Z)

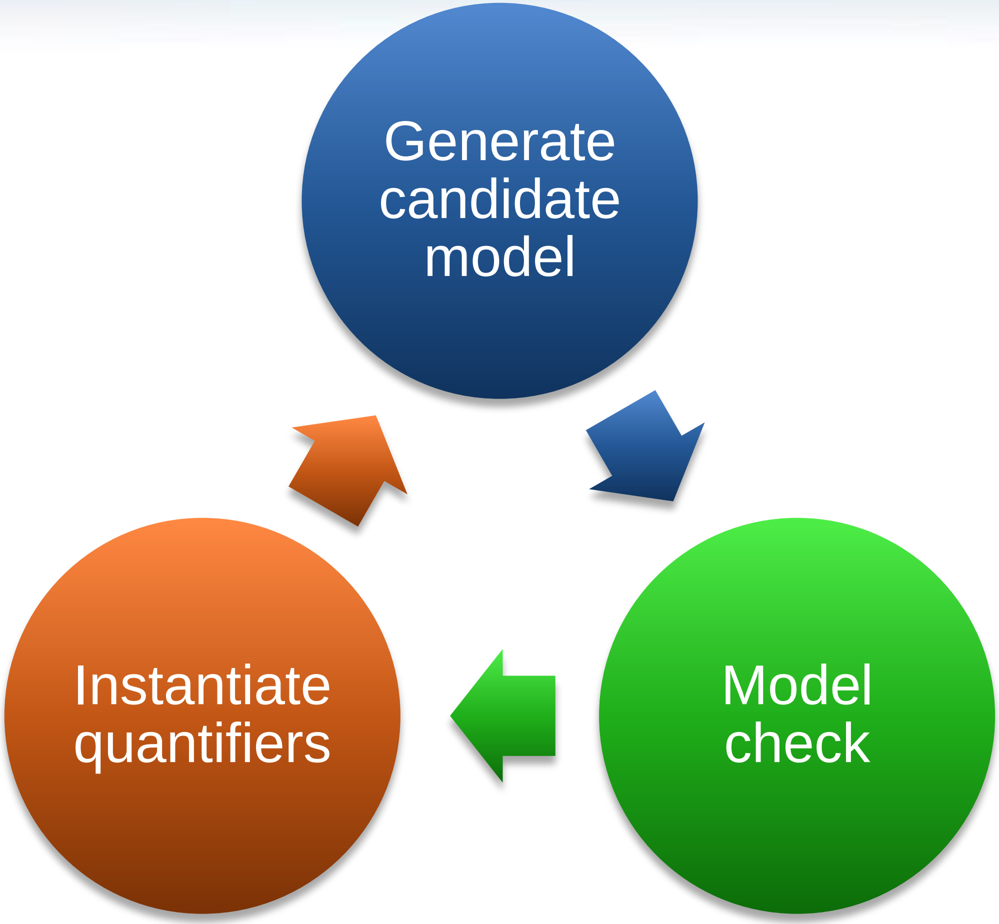
⇕

f(f(0)) > f(0), f(f(f(0))) > f(f(0)), ...

f(0) < a, f(f(0)) < a, ...

f(0) = 0

CEGAR-like loop for quantifiers



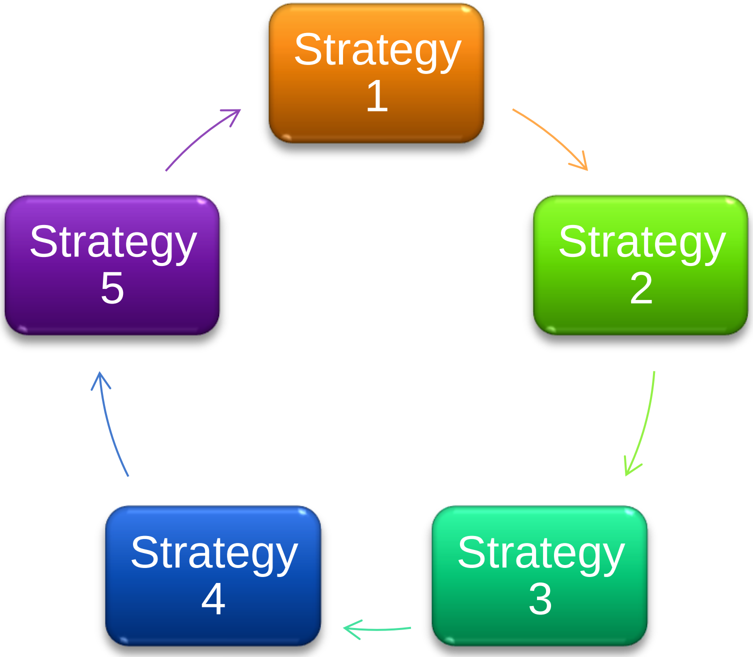
Microsoft Research

What is the best approach?

- **There is no winner**
- Portfolio of algorithms/techniques

Parallel Z3

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



Conclusion

- Some VCs produced by verifying compilers are very challenging
- Most VCs contain many non ground formulas
- Z3 2.0 won all ∀-divisions in SMT-COMP'08
- Many challenges
- Many approaches/algorithms

Thank You!