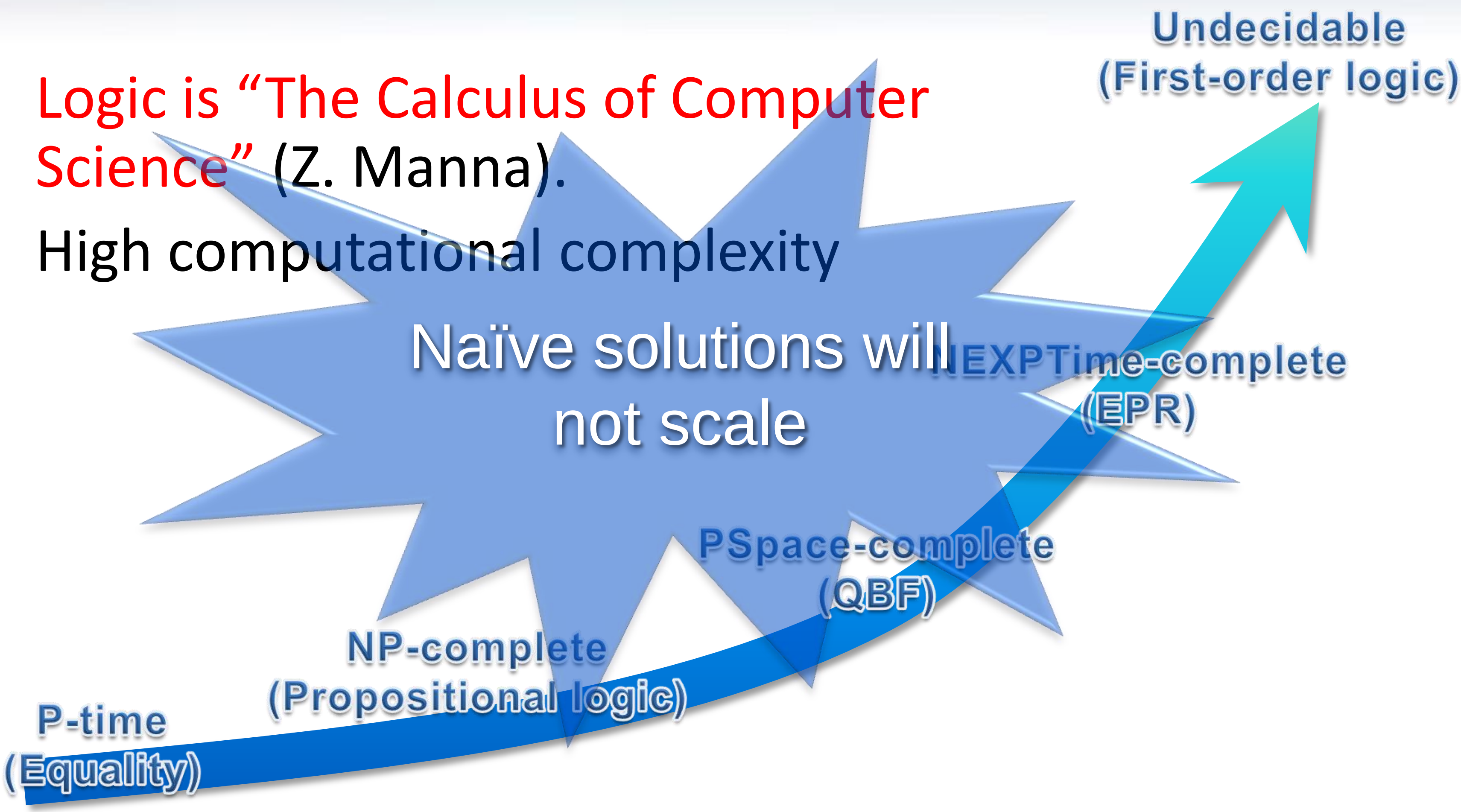
Quantifiers in Satisfiability Modulo Theories

Frontiers of Computational Reasoning 2009 – MSR Cambridge

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)

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

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)

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

Array Theory

Satisfiability Modulo Theories (SMT)

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

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

FormuLa

Hyper-V
Microsoft | Virtualization

Terminator T-2

VCC

SLAM

Yogi

NModel

Vigilante

SpecExplorer

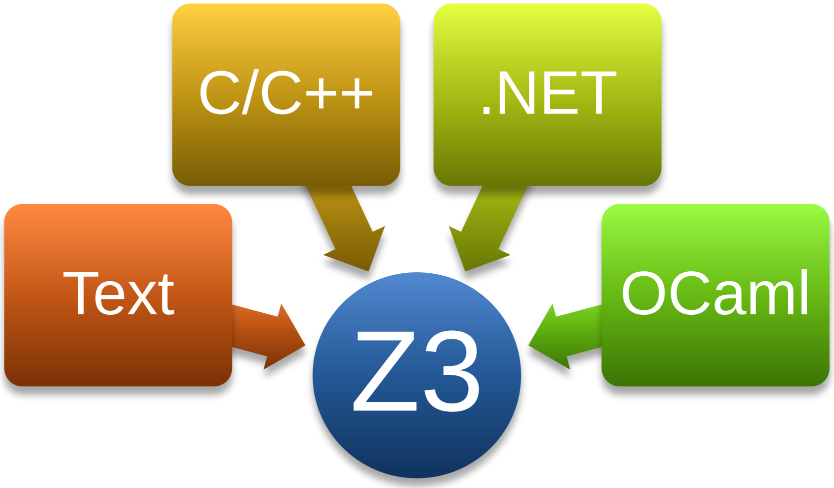
Pex

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

$F \cup T$

First-order Theorem Prover

T may not have a finite axiomatization

SMT x SAT

For some theories, SMT can be reduced to SAT

Higher level of abstraction

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

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

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

$$\Downarrow$$

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

DPLL

- Deducing

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

$$\Downarrow$$

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

DPLL

- Backtracking

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

$$\Downarrow$$

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

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

Verifying Compilers

Annotated Program → Verification Condition *F*

pre/post conditions
invariants
and other annotations

Verification conditions: Structure

∀ Axioms
(non-ground)

BIG
and-or
tree
(ground)

Control & Data
Flow

Main Challenge

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

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

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

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

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

Pattern:

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 patterns share several instructions.

Combine code sequences in a code tree

E-matching: Limitations

- E-matching needs ground seeds.

  $\forall x: p(x)$,

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

E-matching: Limitations

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

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

$g(a) = c,$

$g(b) = c,$

$a \neq b$

Pattern is too restrictive

E-matching: Limitations

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

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

$g(a) = c,$

$g(b) = c,$

$a \neq b$

More "liberal" pattern

E-matching: Limitations

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

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

$g(a) = c,$

$g(b) = c,$

$a \neq b,$

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

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

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided patterns.
- Matching loops:

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

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

  f(a) = c

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided patterns.
- Matching loops:

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

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

$f(a) = c$

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

E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided patterns.
- Matching loops:

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

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

$f(a) = c$

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

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

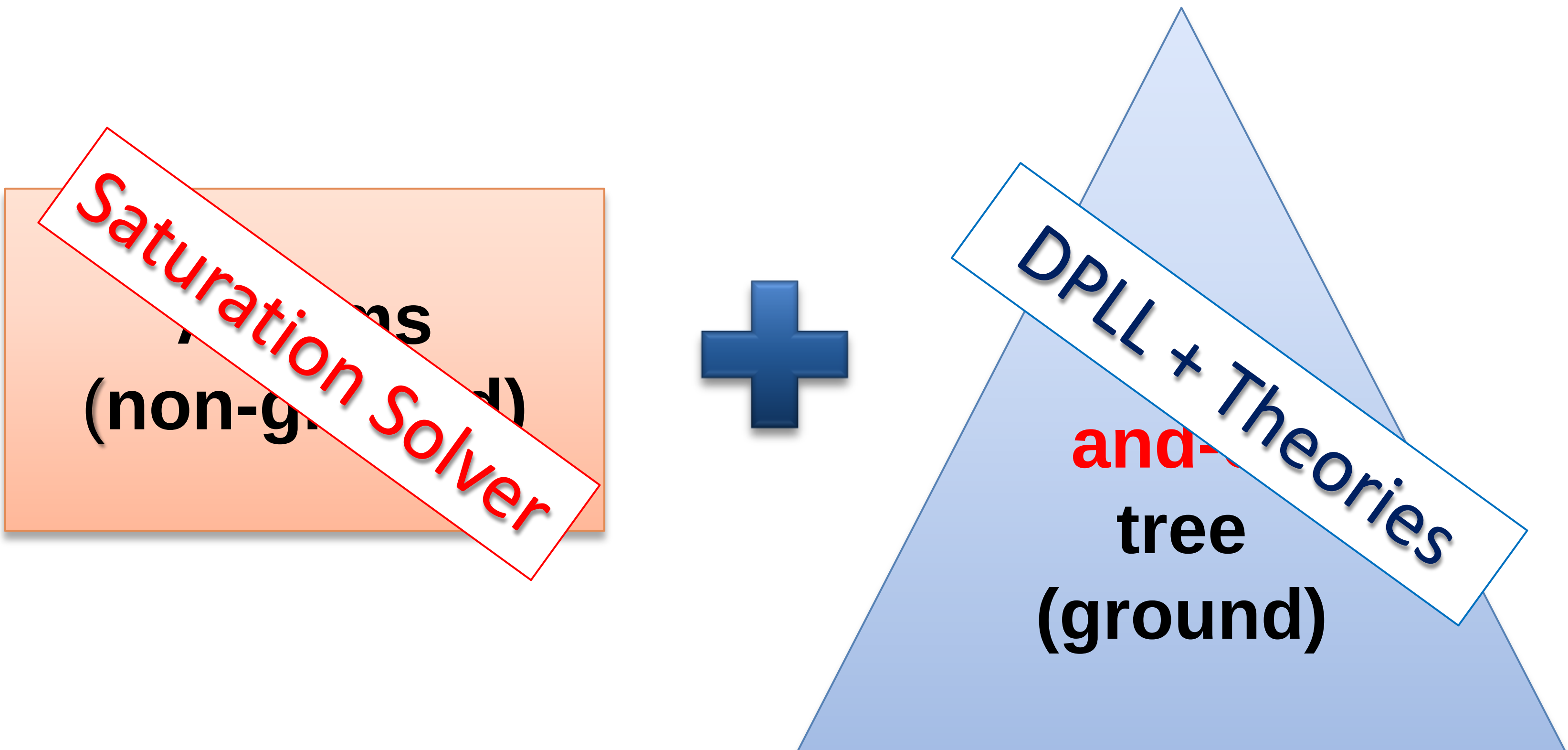
E-matching: Limitations

- E-matching needs ground seeds.
- Bad user provided patterns.
- Matching loops.
- 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

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

DPLL(Γ): Deduce I

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

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

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

DPLL(Γ): Backtracking

p(a), r(a) | p(a)∨q(a), ¬p(a)∨¬r(a), $\cancel{p(a) \triangleright r(a)}$, …

p(a) is removed from M

¬p(a) | p(a)∨q(a), ¬p(a)∨¬r(a), …

DPLL(Γ): Hypothesis Elimination

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

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

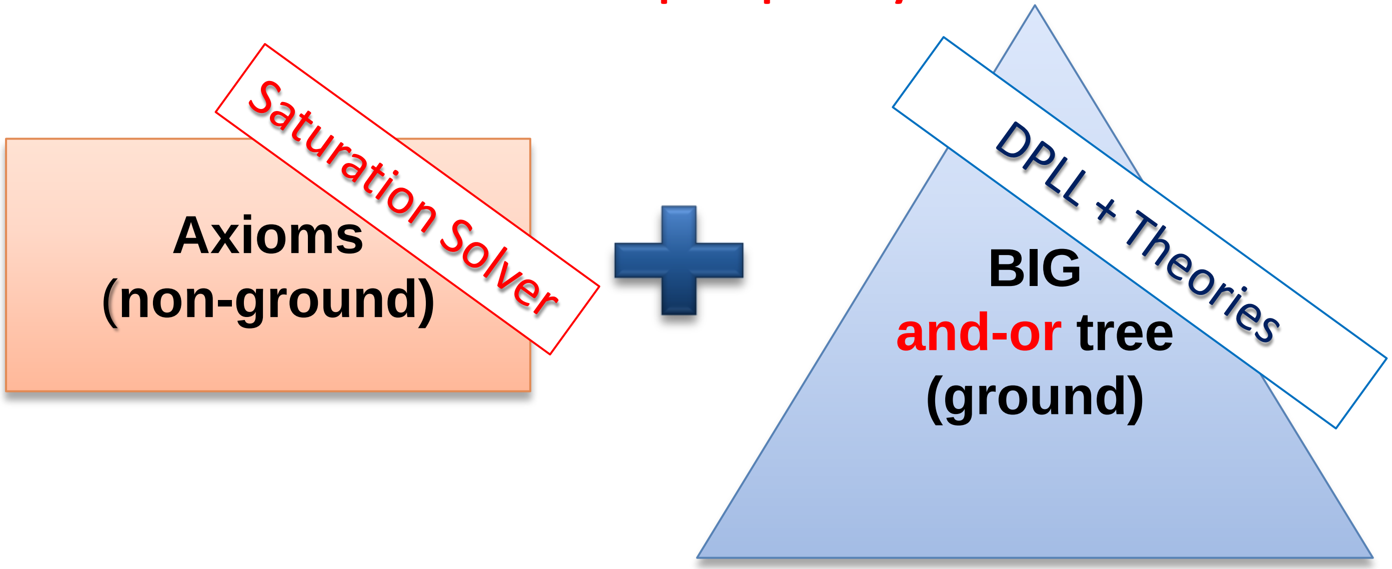
DPLL(Γ): Improvement

- Saturation solver ignores non-unit ground clauses.

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

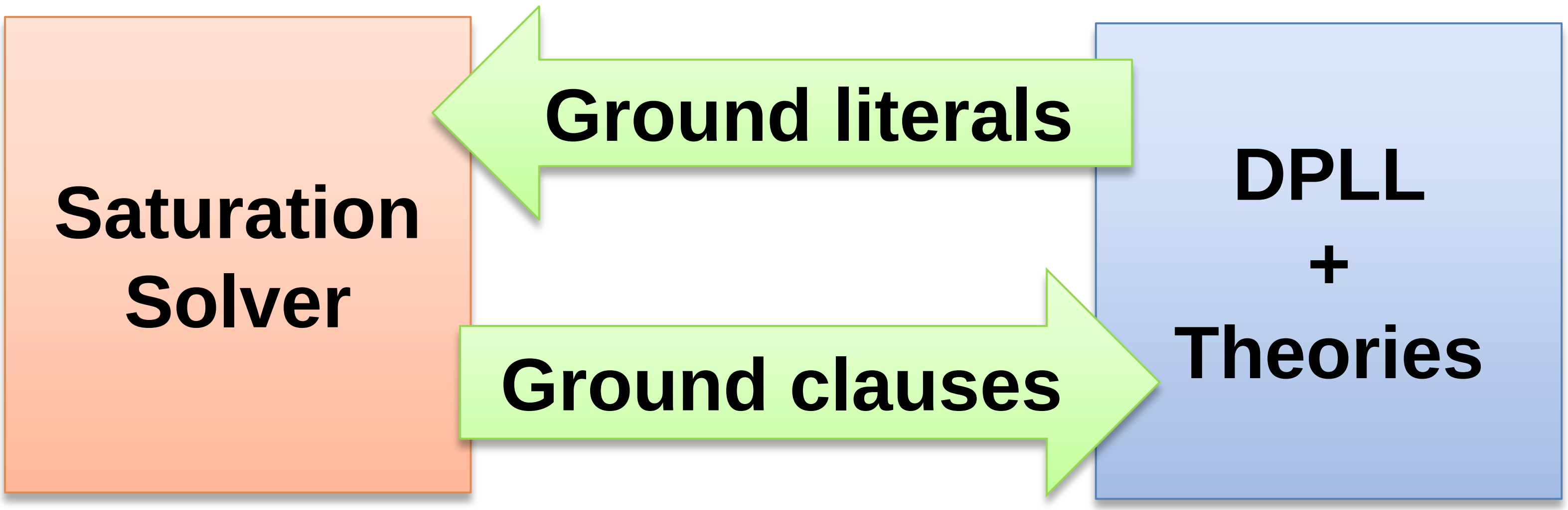
DPLL(Γ): Improvement

- 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): Contraction rules

- Contraction rules are very important.
- Examples:
  - Subsumption
  - Demodulation
  - …
- Contraction rules with a single premise are easy.

DPLL(Γ): Contraction rules

- Contraction rules with several premises.
- Example:

  $p(a) \triangleright r(x)$, $r(x) \vee s(x)$

  $r(x)$ subsumes $r(x) \vee s(x)$

- Problem: $p(a) \triangleright r(x)$ can be deleted during backtracking.

DPLL(Γ): Contraction rules

- Contraction rules with several premises.
- Example:

  $p(a) \triangleright r(x), \quad r(x) \vee s(x)$

- Naïve solution: use hypothesis elimination.

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

Microsoft Research

DPLL(Γ): Contraction rules

- Contraction rules with several premises.
- Example:

  $p(a) \triangleright r(x), \quad r(x) \vee s(x)$

- Solution: disable $r(x) \vee s(x)$ until $p(a)$ is removed from the partial model M.

DPLL(Γ): Problems

- Interpreted symtbols

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

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

DPLL(Γ): Problems

- Ground equations (duplication of work)
  - Superposition
  - Congruence closure

VCs have a huge number of ground equalities

- Partial solution: E-graph (congruence closure) → canonical set of rewriting rules.

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.

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

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

$$f(x + c)$$

$$x =_c t$$

$$\ldots$$

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

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 $T \cup F^*$

Microsoft Research

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)

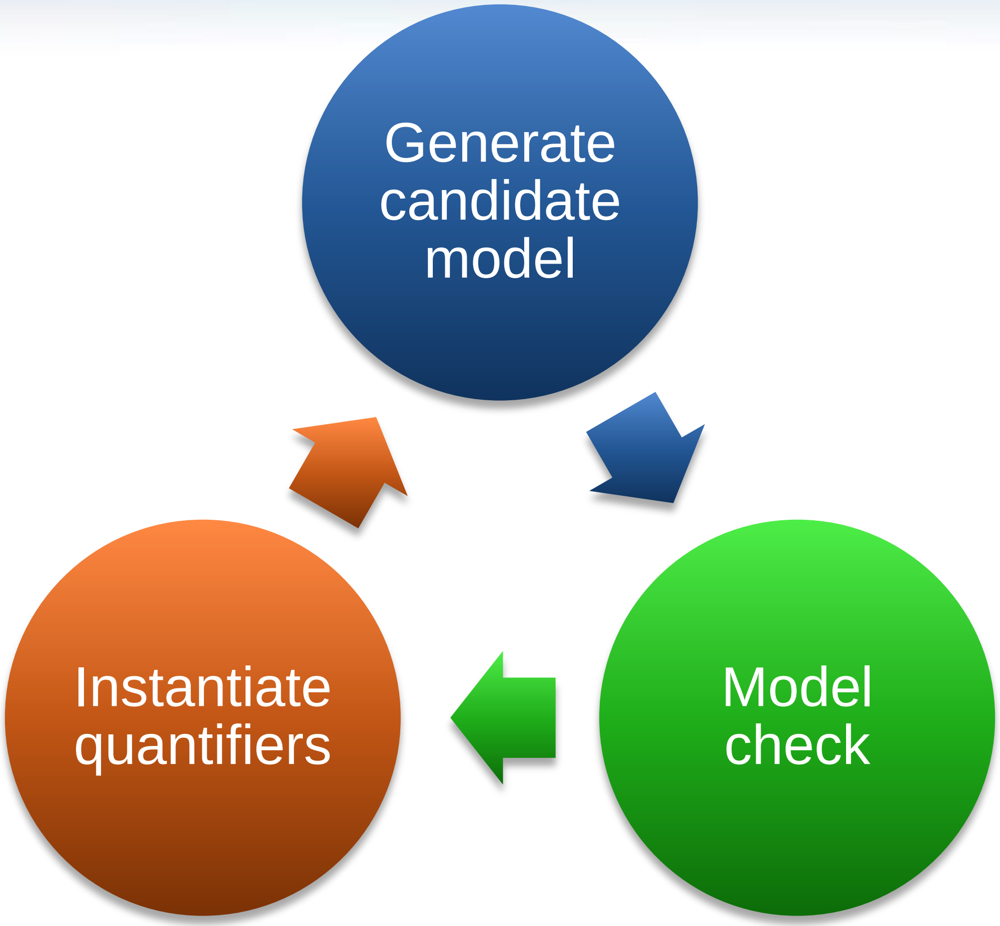
$\Updownarrow$

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

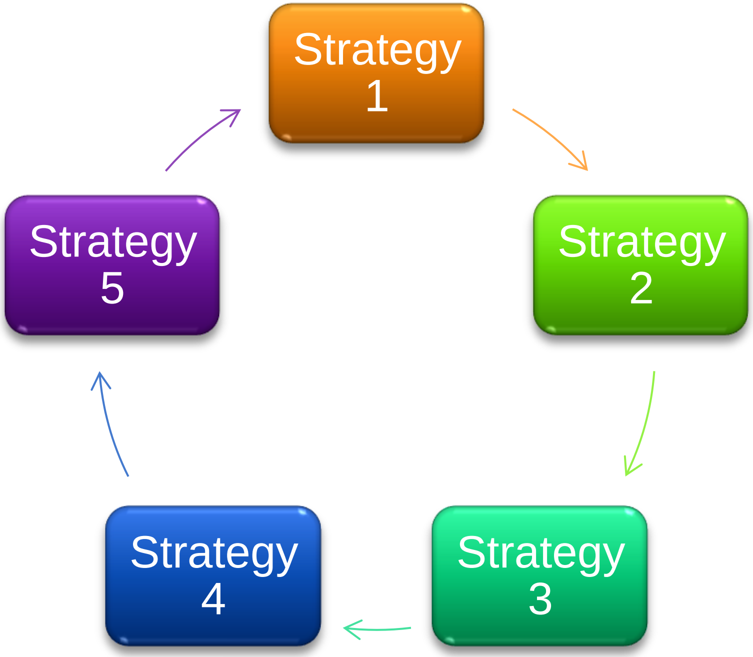


What is the best approach?

- **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!

Microsoft Research