Microsoft Research

# Engineering DPLL(T) + Saturation

## IJCAR 2008

Leonardo de Moura and Nikolaj Bjørner
Microsoft Research

# Satisfiability Modulo Theories (SMT)


- Arithmetic
- Bit-vectors
- Arrays
- …

# Satisfiability Modulo Theories (SMT)

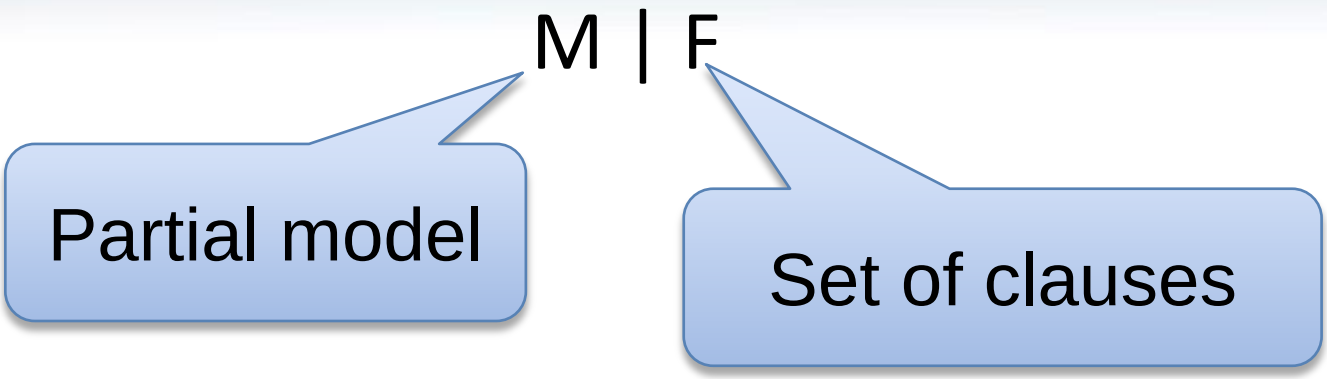
$$x + 2 = y \Rightarrow f(read(write(a, x, 3), y - 2)) = f(y - x + 1)$$

Arithmetic

Arrays

Free Functions

# DPLL



# 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

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

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

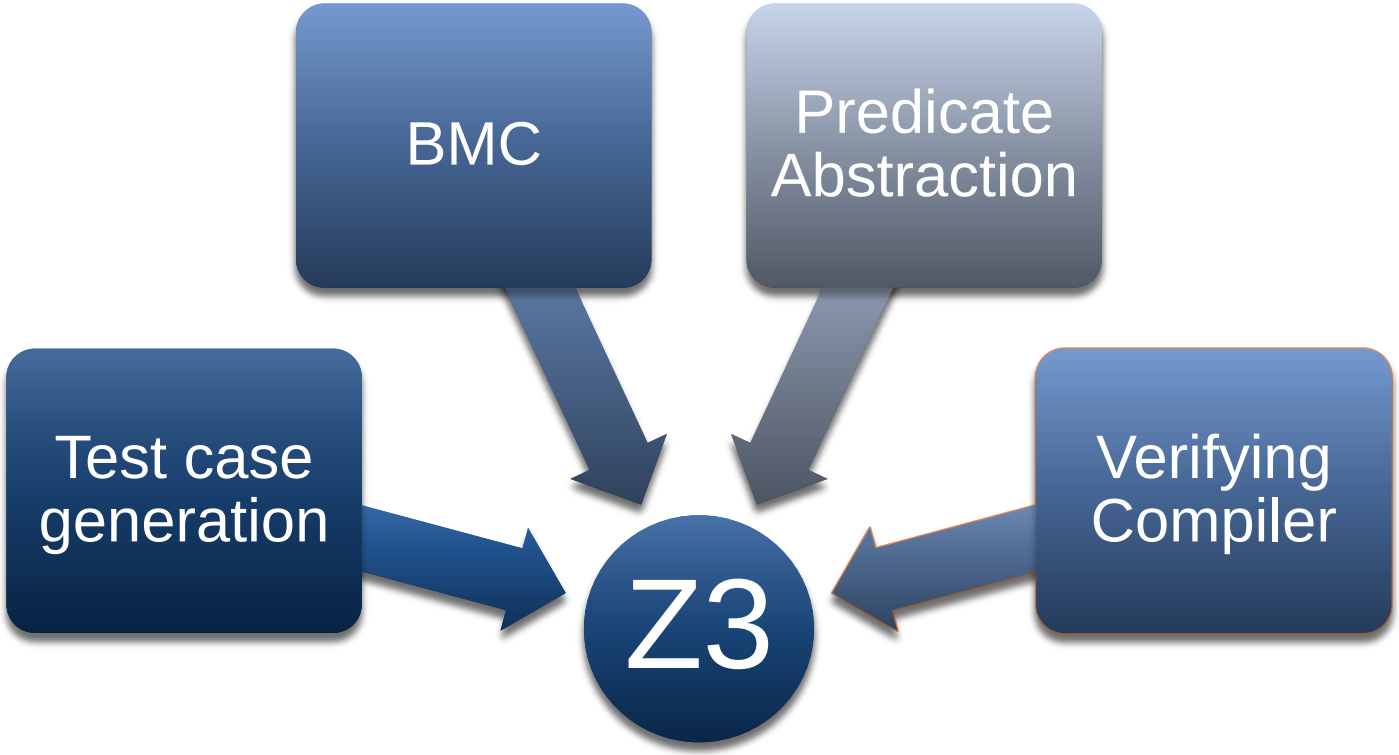
# SMT = DPLL + Theories

- Efficient decision procedures for conjunctions of ground atoms.

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

- Examples:
  - Congruence closure
  - Dual Simplex
  - Bellman-Ford
  - …

# SMT: many applications at MS…






Microsoft Research

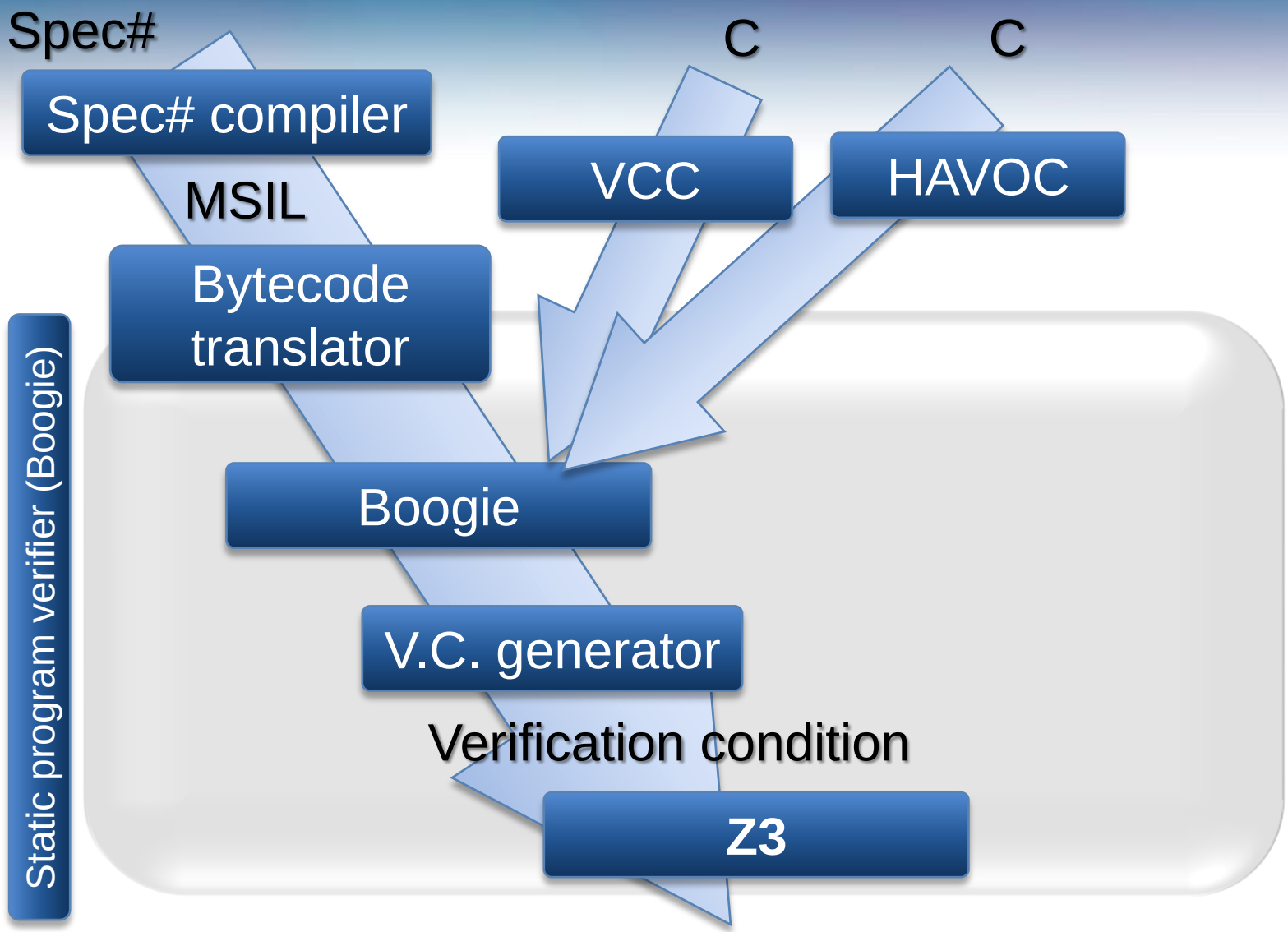
# Verifying Compilers

A verifying compiler uses *automated reasoning* to check the correctness of a program that is compiles.

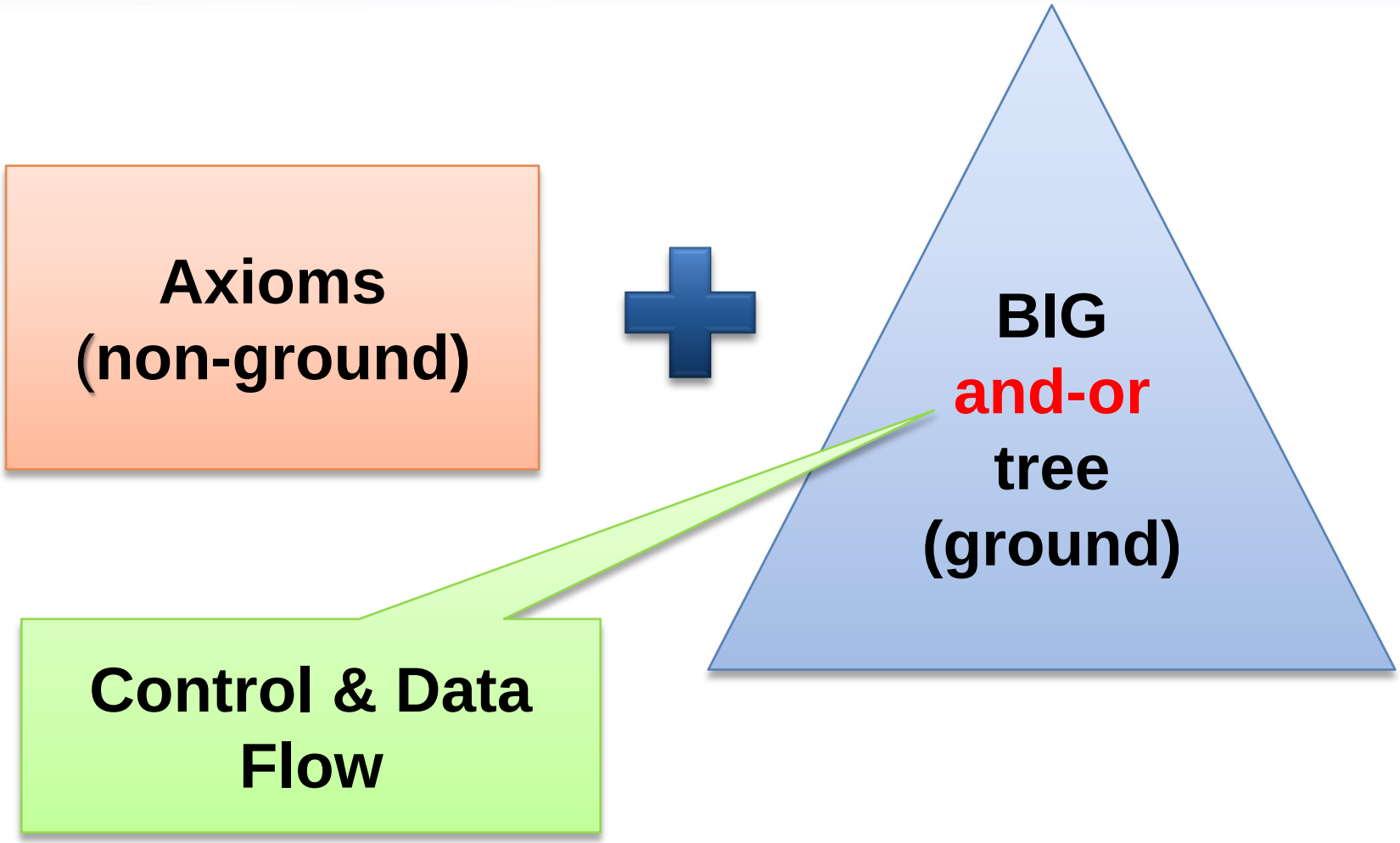
Correctness is specified by *types, assertions, . . . and other redundant annotations* that accompany the program.

Tony Hoare 2004

# Verification architecture



# 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!**

# 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

Pattern



Microsoft Research

# E-matching & Quantifier instantiation

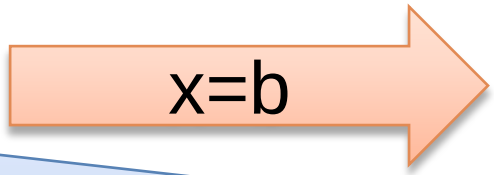
- SMT solvers use heuristic quantifier instantiation.
- E-matching (matching modulo equalities).
- Example:

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

  a = g(b),

  b = c,

  f(a) $\neq$ c




f(g(b)) = b

Equalities and ground terms come from the partial model M:

M | F

# E-matching: why do we use it?

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

# 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 patterns:

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

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:

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

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

  $f(a) = c$

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

# E-matching: Limitations

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

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

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

  $f(a) = c$

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

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

# E-matching: Limitations

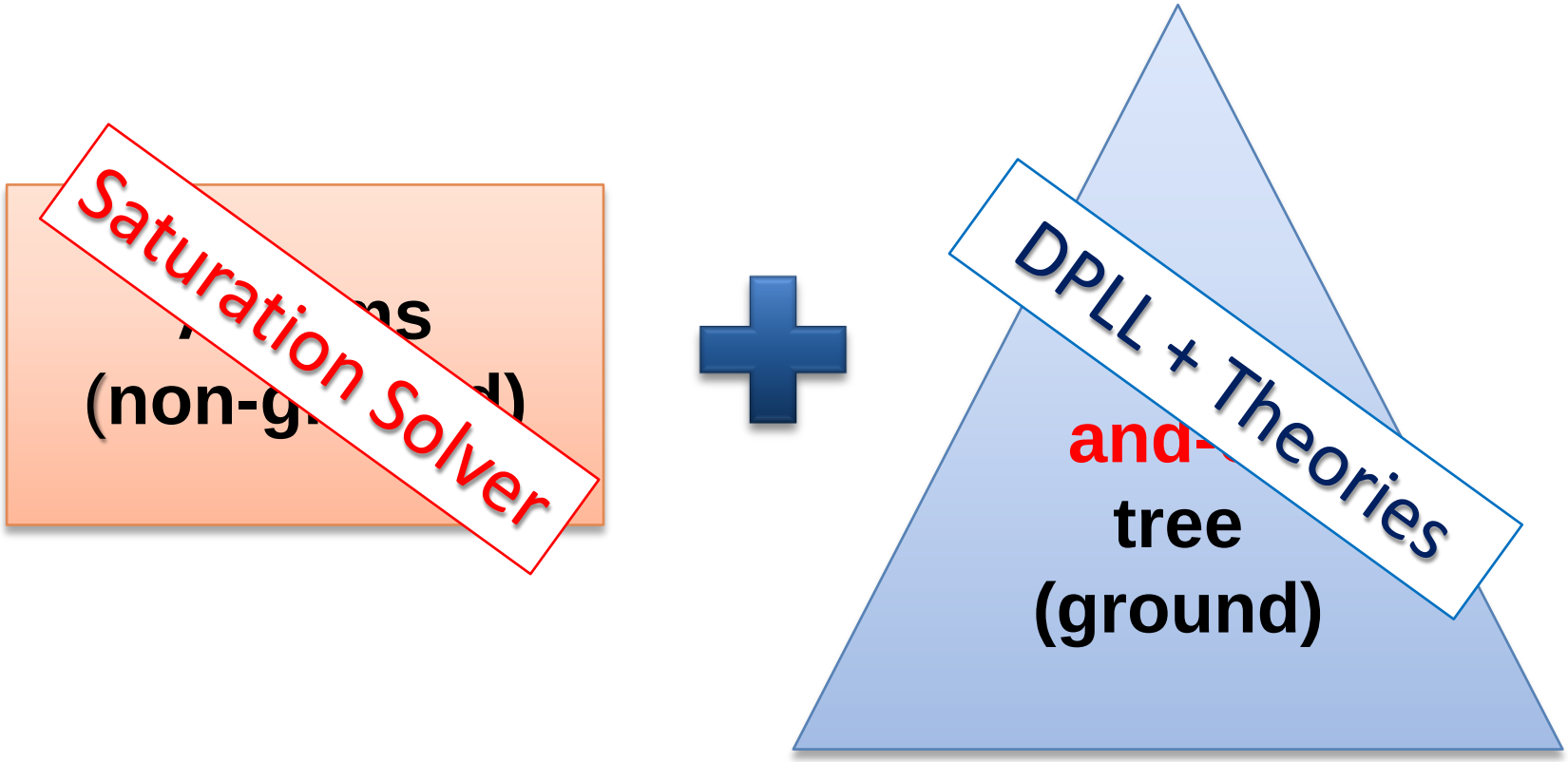
- E-matching needs ground seeds.
- Bad user provided patterns.
- Matching loops.
- It is not refutationally complete.

# Z3: Beyond E-matching

- Decidable fragments:
  - EPR (this morning)
  - Array property fragment
  - More coming soon
- DPLL($\Gamma$): DPLL + Saturation (this talk)

# DPLL(Γ)

- Tight integration: DPLL + Saturation solver.






Microsoft Research

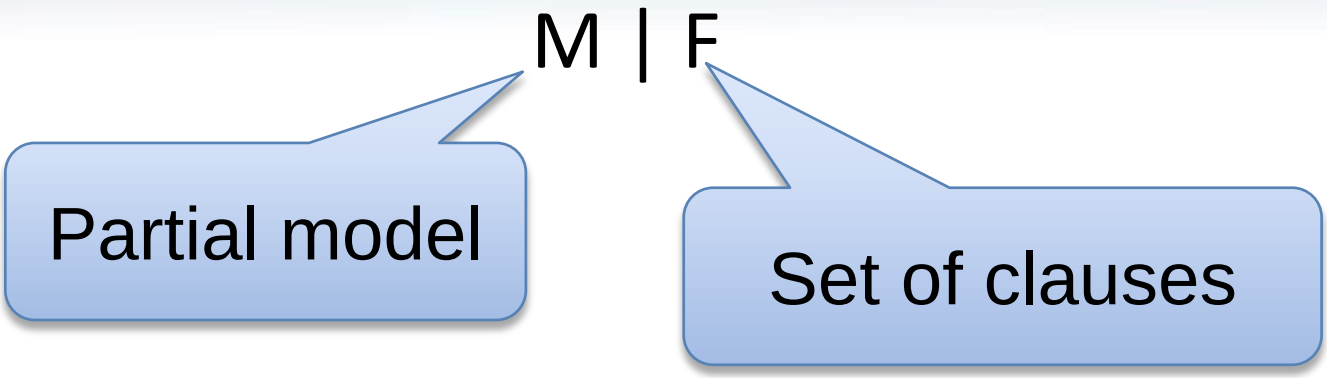
# DPLL($\Gamma$)

- Inference rule:

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

- DPLL($\Gamma$) is parametric.
- Examples:
  - Resolution
  - Superposition calculus
  - …

# DPLL(Γ)



# DPLL(Γ): Deduce I

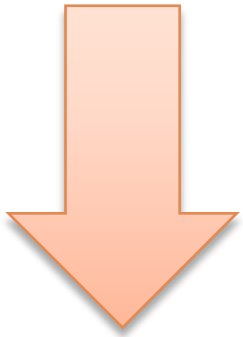
$p(a) \mid p(a) \vee q(a),\ \forall x: \neg p(x) \vee r(x),\ \forall x: p(x) \vee s(x)$

# 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:

$$H \rhd C$$

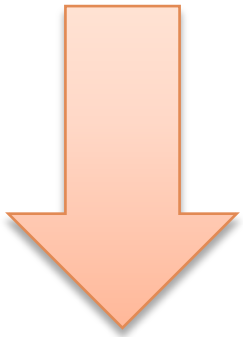
**Track literals from M used to derive C**

**(hypothesis) Ground literals**

**(regular) Clause**

# DPLL(Γ): Deduce II

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



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

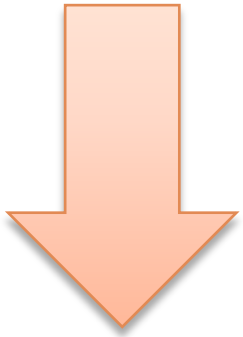
$p(a) \mid p(a) \vee q(a),\ \neg p(x) \vee r(x),\ p(a) \rhd r(a)$

# DPLL(Γ): Backtracking

p(a), r(a) | p(a)∨q(a), ¬p(a)∨¬r(a), **p(a)▷r(a)**, …

# DPLL(Γ): Backtracking

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

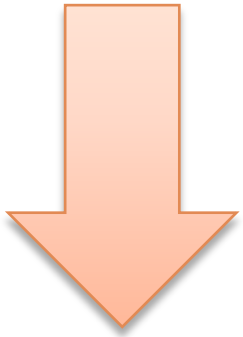


**p(a) is removed from M**

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

# DPLL(Γ): Hypothesis Elimination

$p(a), r(a) \mid p(a) \vee q(a), \neg p(a) \vee \neg r(a),\ \mathbf{p(a) \rhd r(a)}, \ldots$



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

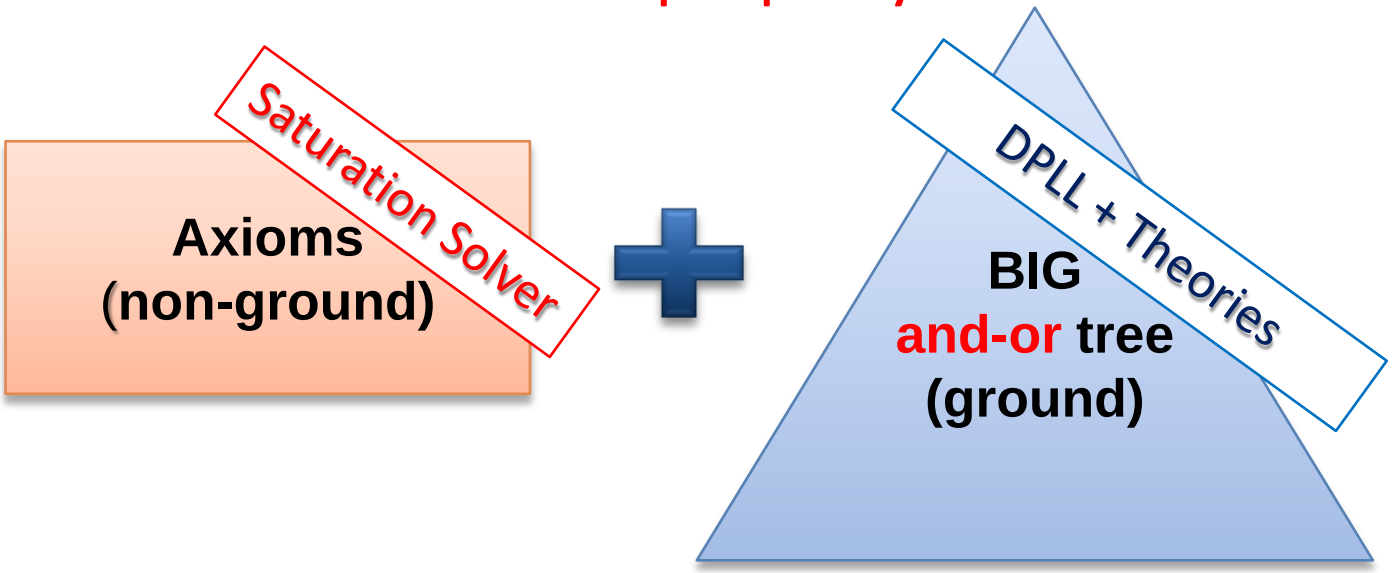
# DPLL(Γ): Improvement

- Saturation solver ignores non-unit ground clauses.

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

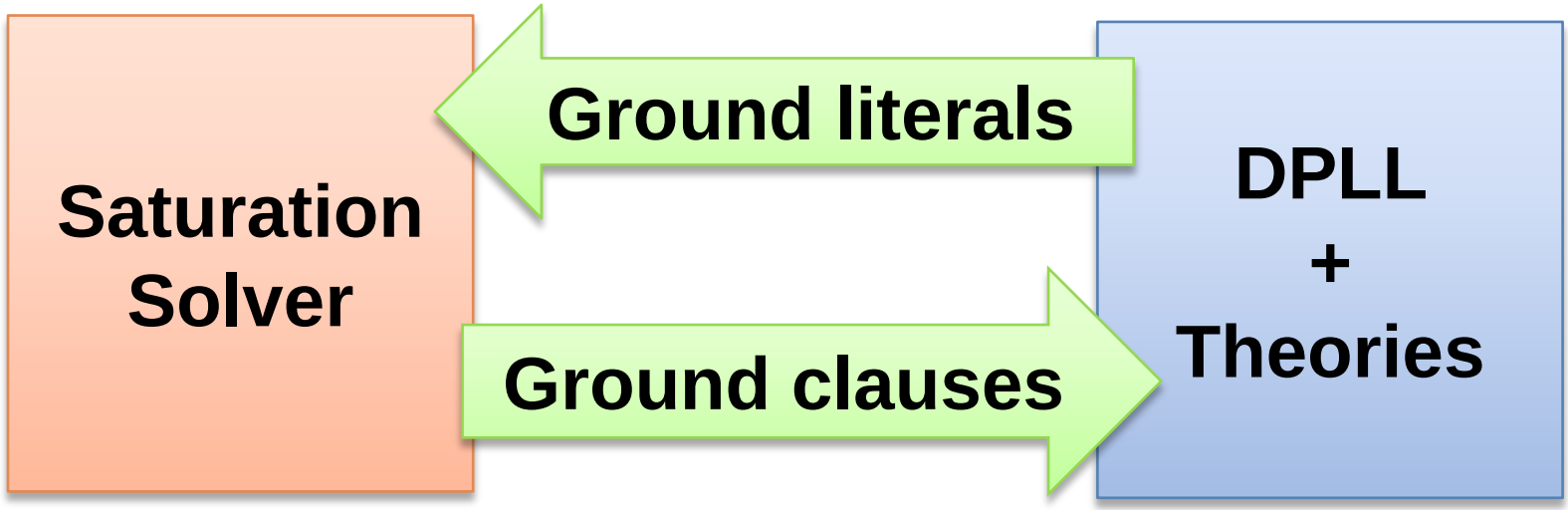
# DPLL(Γ): Improvement

- 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:

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

  r(x) subsumes r(x)∨s(x)

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

# DPLL(Γ): Contraction rules

- Contraction rules with several premises.
- Example:

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

- Naïve solution: use hypothesis elimination.

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

# DPLL(Γ): Contraction rules

- Contraction rules with several premises.
- Example:

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

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

# DPLL(Γ): Problems

- Interpreted symtbols

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

**Disclaimer:**
**Doesn't occur very often**

- Solution: use E-matching for non-ground clauses containing interpreted symbols.

# DPLL(Γ): Problems

- Transitivity + monotonicity

  $\neg p(x,y) \vee \neg p(y,z) \vee p(x,z)$

  $\neg p(x,y) \vee p(f(x), f(y))$

**Saturation engine diverges**

- No satisfactory solution yet.

# DPLL(Γ): Problems

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

Our problems have a huge number of ground equalities

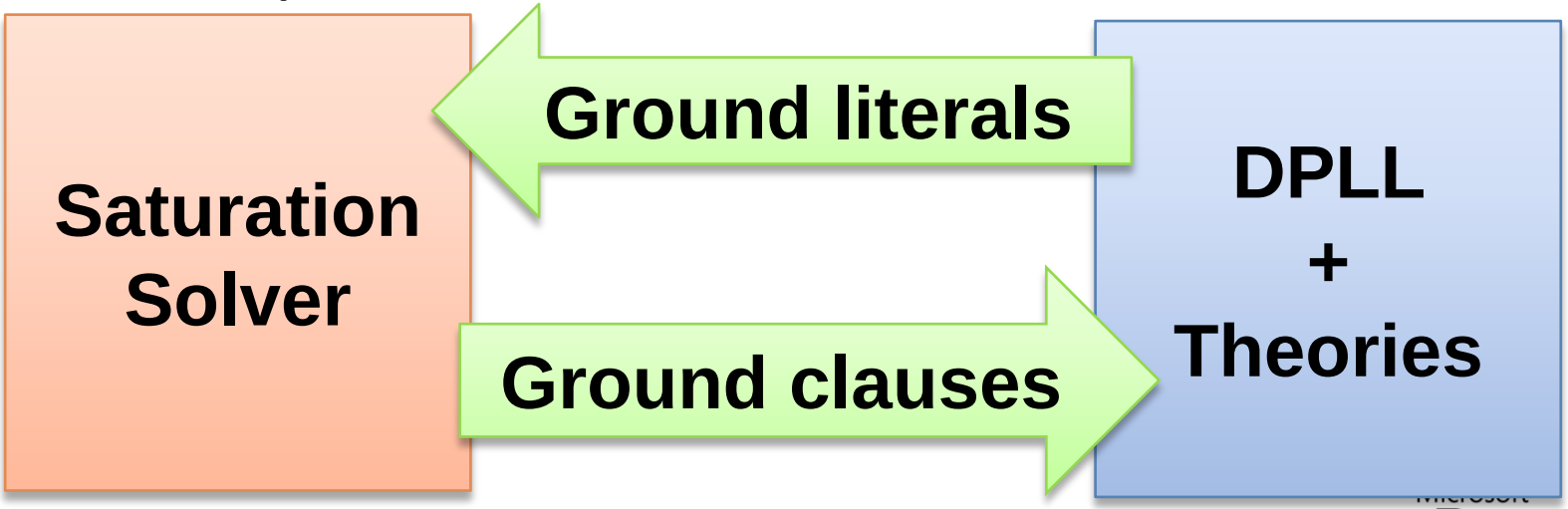
- Partial solution: E-graph (congruence closure) → canonical set of rewriting rules [17].

# Related Work

- Harvey
- SPASS + T
- SMELS
- LASCA

# Future work

- Better superposition calculus engine
- Variable inactivity (Bonacina)
  - Assumption: saturated set of non-ground clauses is variable inactive and doesn't contain interpreted functions.



# Conclusion

- Tight integration: DPLL + Saturation.
- Non-unit ground clauses are delegated to DPLL.
- Good for software verification.
- Detecting unsound set of axioms.
- Implemented in Z3.2.
- Z3.2 won all $\forall$-divisions in SMT-COMP'08.