Quantifiers in Satisfiability Modulo Theories
Frontiers of Computational Reasoning 2009 – MSR Cambridge

Leonardo de Moura
Microsoft Research
Logic is “The Calculus of Computer Science” (Z. Manna).

High computational complexity

Naïve solutions will not scale

P-time (Equality)

NP-complete (Propositional logic)

PSPACE-complete (QBF)

EXPTIME-complete (EPR)

Undecidable (First-order logic)

Quantifiers in Satisfiability Modulo Theories
Satisfiability Modulo Theories (SMT)

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

SMT solvers have specialized algorithms for $T$.
$b + 2 = c \text{ and } f(\text{read}(\text{write}(a, b, 3), c-2)) \neq f(c-b+1)$
Quantifiers in Satisfiability Modulo Theories

Arithmetic

\[ b + 2 = c \quad \text{and} \quad f(\text{read}(\text{write}(a,b,3), c-2)) \neq f(c-b+1) \]
Satisfiability Modulo Theories (SMT)

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

Array Theory
\[ b + 2 = c \quad \text{and} \quad f(read(write(a,b,3), c-2)) \neq f(c-b+1) \]
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 $M$
Z3 is a new solver developed at Microsoft Research.
Development/Research driven by internal customers.
Free for academic research.
Interfaces:

- C/C++
- .NET
- Text
- OCaml

http://research.microsoft.com/projects/z3
SMT x First-order provers

\[ F \cup T \]

First-order Theorem Prover

\[ T \text{ may not have a finite axiomatization} \]
For some theories, SMT can be reduced to SAT

Higher level of abstraction

\[ \text{bvmul}_{32}(a,b) = \text{bvmul}_{32}(b,a) \]
For most SMT solvers, $F$ is a set of ground formulas.

Many Applications

Bounded Model Checking

Test-Case Generation
Quantifiers in Satisfiability Modulo Theories
Guessing

\[ p \mid p \lor q, \neg q \lor r \]

\[ p, \neg q \mid p \lor q, \neg q \lor r \]
Deducing

\[ p \mid p \lor q, \neg p \lor s \]

\[ p, s \mid p \lor q, \neg p \lor s \]
Backtracking

\[ p, \neg s, q \mid p \lor q, s \lor q, \neg p \lor \neg q \]

\[ p, s \mid p \lor q, s \lor q, \neg p \lor \neg q \]
Efficient decision procedures for conjunctions of ground atoms.

\[ a = b, \ a < 5 \mid \neg a = b \lor f(a) = f(b), \ a < 5 \lor a > 10 \]

### Efficient algorithms

<table>
<thead>
<tr>
<th>Difference Logic</th>
<th>Belmann-Ford</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uninterpreted functions</td>
<td>Congruence closure</td>
</tr>
<tr>
<td>Linear arithmetic</td>
<td>Simplex</td>
</tr>
</tbody>
</table>
Annotated Program \quad \rightarrow \quad \text{Verification Condition } F

pre/post conditions
invariants
and other annotations
Verification conditions: Structure

∀ Axioms (non-ground)

Control & Data Flow

BIG and-or tree (ground)
Main Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime

∀ h, o, f:
IsHeap(h) ∧ o ≠ null ∧ read(h, o, alloc) = t
⇒
read(h, o, f) = null ∨ read(h, read(h, o, f), alloc) = t
Main Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
  \[ \forall o, f:\]
  \[ o \neq \text{null} \land \text{read}(h_0, o, \text{alloc}) = t \implies \]
  \[ \text{read}(h_1, o, f) = \text{read}(h_0, o, f) \lor (o, f) \in M \]
Main Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions

\[ \forall i,j: i \leq j \Rightarrow \text{read}(a,i) \leq \text{read}(b,j) \]
Main Challenge

- Quantifiers, quantifiers, quantifiers, ...
- Modeling the runtime
- Frame axioms
- User provided assertions
- Theories
  \[ \forall x: p(x,x) \]
  \[ \forall x,y,z: p(x,y), p(y,z) \implies p(x,z) \]
  \[ \forall x,y: p(x,y), p(y,x) \implies x = y \]
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

Quantifiers in Satisfiability Modulo Theories
SMT solvers use **heuristic quantifier instantiation**.

**E-matching** (matching modulo equalities).

**Example:**

\[ \forall x: f(g(x)) = x \{ f(g(x)) \} \]

- \( a = g(b) \),
- \( b = c \),
- \( f(a) \neq c \)
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

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
  - …
E-efficient E-matching

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Indexing Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast retrieval</td>
<td>E-matching code trees</td>
</tr>
<tr>
<td>Incremental E-Matching</td>
<td>Inverted path index</td>
</tr>
</tbody>
</table>
E-matching code trees

Pattern:
f(x₁, g(x₁, a), h(x₂), b)

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.

Quantifiers in Satisfiability Modulo Theories
E-matching: Limitations

- E-matching needs ground seeds.
  - $\forall x: p(x)$,
  - $\forall x: \neg p(x)$
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:
  \[ \forall x: f(g(x)) = x \quad \{ g(x) \} \]
  \[ g(a) = c, \]
  \[ g(b) = c, \]
  \[ a \neq b \]

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:
  \[
  \forall x: f(x) = g(f(x)) \quad \{f(x)\}
  \]
  \[
  \forall x: g(x) = f(g(x)) \quad \{g(x)\}
  \]
  \[
  f(a) = c
  \]
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**
Tight integration: DPLL + Saturation solver.
Inference rule:

\[
\frac{C_1 \ldots C_n}{C}
\]

DPLL(\(\Gamma\)) is parametric.

Examples:
- Resolution
- Superposition calculus
- ...

Quantifiers in Satisfiability Modulo Theories
Quantifiers in Satisfiability Modulo Theories

DPLL(Γ)

Partial model

M | F

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

\[ p(a) \mid p(a) \lor q(a), \lnot p(x) \lor r(x), p(x) \lor s(x) \]
DPLL(Γ): Deduce I

\[ p(a) \mid p(a) \lor q(a), \neg p(x) \lor r(x), p(x) \lor s(x) \]

Resolution

\[ p(a) \mid p(a) \lor q(a), \neg p(x) \lor r(x), p(x) \lor s(x), r(x) \lor s(x) \]
Using ground atoms from $M$:

$M \mid F$

Main issue: backtracking.

Hypothetical clauses:

$H \triangleright C$

*Hypothesis* Ground literals

*Regular* Clause

Track literals from $M$ used to derive $C$
p(a) \mid p(a) \lor q(a), \neg p(x) \lor r(x)

\hline
\underline{p(a), \neg p(x) \lor r(x)}
\hline
\hline
\begin{align*}
p(a) \mid p(a) \lor q(a), \neg p(x) \lor r(x), & p(a) \triangleright r(a) \\
r(a)
\end{align*}
DPLL(Γ): Backtracking

\[ p(a), r(a) \mid p(a) \lor q(a), \neg p(a) \lor \neg r(a), \ p(a) \triangleright r(a), \ldots \]
Quantifiers in Satisfiability Modulo Theories

DPLL(Γ): Backtracking

p(a), r(a) | p(a) ∨ q(a), ¬p(a) ∨ ¬r(a), p(a) ▷ r(a), ...

p(a) is removed from M

¬p(a) | p(a) ∨ q(a), ¬p(a) ∨ ¬r(a), ...

Microsoft Research
DPLL(\Gamma): Hypothesis Elimination

\[ p(a), r(a) \mid p(a) \lor q(a), \neg p(a) \lor \neg r(a), p(a) \Rightarrow r(a), \ldots \]

\[ p(a), r(a) \mid p(a) \lor q(a), \neg p(a) \lor \neg r(a), \neg p(a) \lor r(a), \ldots \]
Saturation solver ignores non-unit ground clauses.

\[ p(a) \mid p(b) \lor q(a), \neg p(x) \lor r(x) \]
Saturation solver ignores non-unit ground clauses.

It is still refutationally complete if:

- $\Gamma$ has the reduction property.
Saturation solver ignores non-unit ground clauses.

It is still refutationally complete if:

- $\Gamma$ has the reduction property.
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:
  \[ p(a) \Rightarrow r(x), \quad r(x) \lor s(x) \]

  \( r(x) \) subsumes \( r(x) \lor s(x) \)

- Problem: \( p(a) \Rightarrow r(x) \) can be deleted during backtracking.
Contraction rules with several premises.

Example:
\[ p(a) \triangleright r(x), \quad r(x) \lor s(x) \]

Naïve solution: use hypothesis elimination.
\[ \neg p(a) \lor r(x), \quad r(x) \lor s(x) \]
DPLL(Γ): Contraction rules

- Contraction rules with several premises.
- Example:
  \[ p(a) \triangleright r(x), \quad r(x) \lor s(x) \]
- Solution: disable \( r(x) \lor s(x) \) until \( p(a) \) is removed from the partial model \( M \).
Interpreted symbols

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

It is refutationally complete if

- Interpreted symbols only occur in ground clauses
- Non ground clauses are variable inactive
- “Good” ordering is used
DPLL(\(\Gamma\)): Problems

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

VCs have a huge number of ground equalities

Partial solution: E-graph (congruence closure) \(\rightarrow\) canonical set of rewriting rules.
There is no sound and refutationally complete procedure for linear arithmetic + uninterpreted function symbols
Universal variables only occur as arguments of uninterpreted symbols.

\[ \forall x: f(x) + 1 > g(f(x)) \]  

\[ \forall x, y: f(x+y) = f(x) + f(y) \]
Almost unintepreted fragment

- Relax restriction on the occurrence of universal variables.

\[
\begin{align*}
\text{not } (x \leq y) \\
\text{not } (x \leq t) \\
f(x + c) \\
x =_c t \\
\ldots
\end{align*}
\]
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
Compactness

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

If we view $T$ as a set of sentences
Apply compactness to $T \cup F^*$
\begin{itemize}
\item \( \forall x: f(f(x)) > f(x) \)
\item \( \forall x: f(x) < a \)
\item \( f(0) = 0 \)
\item \( f(f(0)) > f(0), f(f(f(0))) > f(f(0)), \ldots \)
\item \( f(0) < a, f(f(0)) < a, \ldots \)
\item \( f(0) = 0 \)
\end{itemize}

Satisfiable if \( T \) is \( \text{Th}(\mathbb{Z}) \), but unsatisfiable \( T \) is the class of structures \( \text{Exp}(\mathbb{Z}) \)
CEGAR-like loop for quantifiers

Generate candidate model

Instantiate quantifiers

Model check

Quantifiers in Satisfiability Modulo Theories
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

Quantifiers in Satisfiability Modulo Theories
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!