SMT solving for fun and profit

Haniel Barbosa

2019–09–20, Belo Horizonte, Brazil
Automated reasoning

- Formal Verification
- Program Analysis
- Automatic Testing
- Program Synthesis
My goal is to **increase the applicability** of formal methods via:

**Automated reasoning**

- The Computer Science and Mathematical Logic field concerned with:
  - **deduction** (premises entail Truth of conclusion)
  - **models** (witness of Truth)
  - **proofs** (convincing argument of Truth)

- We devise algorithms to solve problems stated in formal languages
My goal is to **increase the applicability** of formal methods via:

### Automated reasoning

- The Computer Science and Mathematical Logic field concerned with
  - *deduction* (premises entail Truth of conclusion)
  - *models* (witness of Truth)
  - *proofs* (convincing argument of Truth)

- We devise algorithms to solve problems stated in formal languages

### Satisfiability modulo theories (SMT)

- Sweetspot between expressive logics and efficient decision procedures
Automated reasoning

- Formal Verification
- Program Analysis
- Automatic Testing
- Program Synthesis
Formal Verification

Program Analysis

Automatic Testing

Program Synthesis

SMT Solvers

SMT solving for fun and profit
Formal Verification

Program Analysis

Automatic Testing

Program Synthesis

SMT Solvers
Formal Verification

Automatic Testing

SMT Solvers

Program Analysis

Program Synthesis
Outline

▷ Satisfiability
  ▶ Boolean satisfiability
  ▶ Satisfiability modulo theories

▷ Applications
  ▶ Program verification
  ▶ Program synthesis
  ▶ Others
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[ C ::= p \mid \neg p \mid C \lor C \]
\[ \varphi ::= C \mid \varphi \land \varphi \]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \varphi \) to \( \{\top, \bot\} \) such that \( M(\varphi) = \top \) (or \( M \models \varphi \)).
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[
C \ ::= \ p \mid \neg p \mid C \lor C \\
\varphi \ ::= \ C \mid \varphi \land \varphi
\]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \varphi \) to \( \{\top, \bot\} \) such that \( M(\varphi) = \top \) (or \( M \models \varphi \)).

Example

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \) satisfiable?
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[ C ::= p \mid \neg p \mid C \lor C \]

\[ \varphi ::= C \mid \varphi \land \varphi \]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( \mathcal{M} \) mapping every proposition \( \varphi \) to \( \{ \top, \bot \} \) such that \( \mathcal{M}(\varphi) = \top \) (or \( \mathcal{M} \models \varphi \)).

Example

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \) satisfiable? Yes

\[ \mathcal{M}(p) = \top, \mathcal{M}(q) = \top, \mathcal{M}(r) = \bot \Rightarrow \mathcal{M}(\varphi) = \top \]
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[
C ::= p \mid \neg p \mid C \lor C \\
\varphi ::= C \mid \varphi \land \varphi
\]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \varphi \) to \( \{\top, \bot\} \) such that \( M(\varphi) = \top \) (or \( M \models \varphi \)).

Example

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \land (r \lor \neg q) \) satisfiable?
Boolean Satisfiability (SAT)

Propositional formulas in CNF:

\[
C ::= p | \neg p | C \lor C \\
\varphi ::= C | \varphi \land \varphi
\]

Given a formula \( \varphi \) in propositional logic, finding an assignment \( M \) mapping every proposition \( \varphi \) to \{\top, \bot\} such that \( M(\varphi) = \top \) (or \( M \models \varphi \)).

Example

Is \( \varphi = (p \lor \neg q) \land (\neg r \lor \neg p) \land q \land (r \lor \neg q) \) satisfiable? **No**

No combination of valuations for these propositions such that \( \varphi \) is \( \top \).
SAT solving

- SAT is NP-complete
- Nevertheless tractable in practice by modern SAT solvers, based on *conflict driven clause learning* (CDCL)
  - mid ’90s: formulas solvable with thousands of variables and clauses
  - now: formulas solvable with millions of variables and clauses
SAT solving

- SAT is NP-complete
- Nevertheless tractable in practice by modern SAT solvers, based on *conflict driven clause learning* (CDCL)
  - mid '90s: formulas solvable with thousands of variables and clauses
  - now: formulas solvable with millions of variables and clauses

DPLL vs CDCL

- CDCL combines *model search* and *proof search*
  - Smart ideas + clever engineering “tricks”
SAT solving
Satisfiability Modulo Theories (SMT)

First-order formulas

\[ t ::= x \mid f(t, \ldots, t) \]

in CNF:

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( \mathcal{T}_1, \ldots, \mathcal{T}_n \), finding a model \( \mathcal{M} \) giving an \textit{interpretation} to all terms and predicates such that \( \mathcal{M} \models \mathcal{T}_1, \ldots, \mathcal{T}_n \ \varphi \)

Example

Is \( \varphi \) satisfiable modulo \textit{equality} and \textit{arithmetic}?
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

\[ t ::= x \mid f(t, \ldots, t) \]
\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models T_1, \ldots, T_n \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\equiv f(0)) \lor x_3 + x_1 > x_3 + 1) \]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:

$t ::= x | f(t, . . . , t)$

$\varphi ::= p(t, . . . , t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi$

Given a formula $\varphi$ in FOL and background theories $T_1, \ldots, T_n$, finding a model $M$ giving an interpretation to all terms and predicates such that $M \models T_1, \ldots, T_n \varphi$

Example

Is $\varphi$ satisfiable modulo equality and arithmetic?

$\varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\equiv f(0)) \lor (x_3 + x_1 > x_3 + 1)$

LIA EUF LIA

$\varphi | = LIA x_1 \equiv 0 | = EUF f(x_1) \equiv f(0) | = LIA x_3 + x_1 \not\geq x_3 + 1$

Therefore $| = EUF \cup LIA \neg \varphi$

SMT solving for fun and profit
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:
\[ t ::= x | f(t, \ldots, t) \]
\[ \varphi ::= p(t, \ldots, t) | \neg \varphi | \varphi \lor \varphi | \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( \mathcal{M} \) giving an \textit{interpretation} to all terms and predicates such that \( \mathcal{M} \models T_1, \ldots, T_n. \varphi \)

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \not\equiv f(0)) \lor x_3 + x_1 > x_3 + 1 \]

\( \varphi \models_{\text{LIA}} x_1 \equiv 0 \)
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:
\[ t ::= x \mid f(t, \ldots, t) \]
\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an *interpretation* to all terms and predicates such that 
\[ M \models T_1, \ldots, T_n \varphi \]

**Example**

Is \( \varphi \) satisfiable modulo *equality* and *arithmetic*?

\[ \varphi = (x_1 \geq 0) \land (x_1 < 1) \land (f(x_1) \neq f(0)) \lor x_3 + x_1 > x_3 + 1) \]

\[ \varphi \models \text{LIA} \quad x_1 \approx 0 \]
\[ x_1 \approx 0 \models \text{EUF} \quad f(x_1) \approx f(0) \]
Satisfiability Modulo Theories (SMT)

First-order formulas

\[ t ::= x \mid f(t, \ldots, t) \]

in CNF:

\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( T_1, \ldots, T_n \), finding a model \( M \) giving an interpretation to all terms and predicates such that

\[ M \models T_1, \ldots, T_n \varphi \]

Example

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = \underbrace{(x_1 \geq 0) \land (x_1 < 1)}_{\text{LIA}} \land \underbrace{(f(x_1) \not\equiv f(0))}_{\text{EUF}} \lor \underbrace{x_3 + x_1 > x_3 + 1}_{\text{LIA}} \]

\[
\begin{align*}
\varphi & \models \text{LIA} \quad x_1 \equiv 0 \\
x_1 \equiv 0 & \models \text{EUF} \quad f(x_1) \equiv f(0) \\
x_1 \equiv 0 & \models \text{LIA} \quad x_3 + x_1 \not\equiv x_3 + 1
\end{align*}
\]
Satisfiability Modulo Theories (SMT)

First-order formulas in CNF:
\[ t ::= x \mid f(t, \ldots, t) \]
\[ \varphi ::= p(t, \ldots, t) \mid \neg \varphi \mid \varphi \lor \varphi \mid \forall x_1 \ldots x_n. \varphi \]

Given a formula \( \varphi \) in FOL and background theories \( \mathcal{T}_1, \ldots, \mathcal{T}_n \), finding a model \( \mathcal{M} \) giving an interpretation to all terms and predicates such that \( \mathcal{M} \models \mathcal{T}_1, \ldots, \mathcal{T}_n \varphi \)

**Example**

Is \( \varphi \) satisfiable modulo equality and arithmetic?

\[ \varphi = \underbrace{(x_1 \geq 0) \land (x_1 < 1)}_{\text{LIA}} \land \underbrace{(f(x_1) \not= f(0))}_{\text{EUF}} \lor \underbrace{x_3 + x_1 > x_3 + 1}_{\text{LIA}} \]

\[
\begin{align*}
\varphi & \models_{\text{LIA}} x_1 \approx 0 \\
x_1 \approx 0 & \models_{\text{EUF}} f(x_1) \approx f(0) \\
x_1 \approx 0 & \models_{\text{LIA}} x_3 + x_1 \not\approx x_3 + 1 \\
\end{align*}
\]

Therefore \( \models_{\text{EUF} \cup \text{LIA}} \neg \varphi \)
SMT solving

- Decidability depends on the theories being used

- Efficient decision procedures
  - Equality and uninterpreted functions (Congruence Closure)
  - Linear arithmetic (Simplex)
  - Bit-vectors (Bit-blasting)
  - Combination of theories (Nelson-Oppen)
  - ...

- Boolean search leverages SAT solvers

- Users may define their own theories
  - New operators as uninterpreted functions + Axioms
CDCL($\mathcal{T}$) architecture

- **Rewriter** simplifies terms
  - $x + 0 \rightarrow x$
  - $a \not\sim a \rightarrow \bot$
  - $(\text{str.replace} \ x \ (\text{str.++} \ x \ x) \ y) \rightarrow x$

- **SAT solver** enumerates models for Boolean skeleton of formula

- **Theory solvers** check consistency in the theory

- **Instantiation module** selects relevant instances
Many SMT solvers around

- **Z3**
  - Primarily developed at Microsoft Research
  - Open source, C++
  - Available at https://github.com/Z3Prover/z3

- **CVC4**
  - Primarily developed at Stanford University and The University of Iowa
  - Open source, C++
  - Available at https://github.com/CVC4/CVC4

- **Other noteworthy SMT solvers:**
  - veriT
  - Yices
  - MathSAT
  - Boolector
  - ...

SMT solving for fun and profit
Proving program equivalence

The programs are equivalent if the following formula is valid:

\[
\text{in}_0 \approx \text{in}_0 \land \phi_a \land \phi_b \to \text{out}_2 \approx \text{out}_0
\]

Here are the two programs:

```c
1 int power3(int in)
2 {
3     int i, out_a;
4     out_a = in;
5     for (i = 0; i < 2; i++)
6         out_a = out_a * in;
7     return out_a;
8 }

1 int power3_new(int in)
2 {
3     int out_b;
4     out_b = (in * in) * in;
5     return out_b;
6 }
```
Proving program equivalence

Static single assignment form and loop unrolling:

\[ \varphi_a = (out0_a \simeq in0_a) \land (out1_a \simeq out0_a \ast in0_a) \land (out2_a \simeq out1_a \ast in0_a) \]

\[ \varphi_b = out0_b \simeq ((in0_b \ast in0_b) \ast in0_b) \]
Proving program equivalence

```c
int power3(int in)
{
    int i, out_a;
    out_a = in;
    for (i = 0; i < 2; i++)
        out_a = out_a * in;
    return out_a;
}
```

```c
int power3_new(int in)
{
    int out_b;
    out_b = (in * in) * in;
    return out_b;
}
```

Static single assignment form and loop unrolling:

\[ \varphi_a = (out0_a \simeq in0_a) \land (out1_a \simeq out0_a \ast in0_a) \land (out2_a \simeq out1_a \ast in0_a) \]
\[ \varphi_b = out0_b \simeq ((in0_b \ast in0_b) \ast in0_b) \]

The programs are equivalent if the following formula is valid:

\[ in0_a \simeq in0_b \land \varphi_a \land \varphi_b \rightarrow out2_a \simeq out0_b \]
What if we complicate things a bit?

Add(Int x, y) {
    z := x; i := 0;
    assume(y > 0);
    while (i < y) {
        z := z + 1;
        i := i + 1;
    }
    return z;
}

Post-condition:
\forall x, y : z = x + y

Result is the sum of the inputs
What if we complicate things a bit?

Add(Int x, y) {
    z := x; i := 0;
    assume(y > 0);
    while (i < y) {
        z := z + 1;
        i := i + 1;
    }
    return z;
}

Post-condition:
\[ \forall x, y : z = x + y \]

Verification:

\[
\begin{align*}
    z &= x \land i = 0 \land y > 0 \\
    Inv(x, y, z, i) \land i < y \land z' &= z + 1 \land i' = i + 1 \\
    Inv(x, y, z, i) \land i \geq y &\rightarrow z = x + y
\end{align*}
\]

SMT solving for fun and profit
What if we complicate things a bit?

Add(Int x, y) {
    z := x; i := 0;
    assume(y > 0);
    while (i < y) {
        z := z + 1;
        i := i + 1;
    }
    return z;
}

Post-condition:
\forall x, y : z = x + y

Verification:

\begin{align*}
z &= x \land i = 0 \land y > 0 & \rightarrow & \text{Inv}(x, y, z, i) \\
\text{Inv}(x, y, z, i) \land i < y \land z' = z + 1 \land i' = i + 1 & \rightarrow & \text{Inv}(x, y, z', i') \\
\text{Inv}(x, y, z, i) \land i \geq y & \rightarrow & z = x + y
\end{align*}
How to generate loop invariants?

State of the art:
- IC3 [Bradley 2011], based on SMT
  - Only applicable to theories that admit quantifier elimination
  - Highly heuristic

Alternative approaches using program synthesis
- Can be used for any theory supported by an SMT solver
- Allow tighter control of the search
Syntax-Guided Synthesis (SyGuS) [Alur et al. 2013]

- Specification is given by $\mathcal{T}$-formula: $\exists f. \forall \bar{x}. \varphi[f, \bar{x}]$
- Syntactic restrictions given by context-free grammar $R$

Diagram:
- Specification
- Syntax restrictions
- Synthesizer
- Program
Syntax-Guided Synthesis (SyGuS) [Alur et al. 2013]

- Specification is given by $\mathcal{T}$-formula: $\exists f. \forall \bar{x}. \varphi[f, \bar{x}]$
- Syntactic restrictions given by context-free grammar $R$
- Commonly solved via enumerative CEGIS [Solar-Lezama et al. 2006]

$I ::= 0 | 1 | x | y | I + I | I - I$

Solution enumerator

Candidate $f(x,y)=x$

Solution verifier

$\exists f. \forall \bar{x}. f(x, y) > x + 1$

Counter-Example $f(x=0, y=1)$
Scalability issues of enumerative techniques

For this bit-vector grammar, enumerating

- Terms of size = 1 : .05 seconds
- Terms of size = 2 : .6 seconds
- Terms of size = 3 : 48 seconds
- Terms of size = 4 : 5.8 hours
- Terms of size = 5 : ??? (100+ days)
Divide-and-conquer (D&C)  

- Generate partial solutions correct on subset of input
- Combine using conditionals

**Step 1:** Propose terms until all points covered

<table>
<thead>
<tr>
<th>Partial Solutions</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1, 1)</td>
</tr>
<tr>
<td>1</td>
<td>(1, 2)</td>
</tr>
<tr>
<td>( x )</td>
<td>(2, 1)</td>
</tr>
<tr>
<td>( y )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

**Step 2:** Generate predicates

<table>
<thead>
<tr>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 \geq 1 )</td>
</tr>
<tr>
<td>( 1 \geq 1 )</td>
</tr>
<tr>
<td>( x \geq 1 )</td>
</tr>
<tr>
<td>( x \geq 2 )</td>
</tr>
<tr>
<td>( x \geq y )</td>
</tr>
</tbody>
</table>

**Step 3:** Combine! \( \text{if } (x \geq y) \text{ then } x \text{ else } y \)

Only applicable for **plainly separable** specifications
SMT solver labels points while accounting for dependencies

Decision tree learning combines predicates and labels into candidates

Significant gains over state-of-the-art SyGuS solvers for invariant synthesis and competitive against model checkers
Other applications of SMT-based program synthesis

- Superoptimization [Schkufza et al. 2013, Nötzli et al. 2019, ...]
- Programming by examples [Gulwani 2011, Feng et al. 2017 ...]
- Circuit synthesis [Eldib et al. 2016, ...]
- ...

SMT solving for fun and profit
Other applications – Graph coloring

Given a graph \( G = (V, E) \), can the vertices be colored with \( k \) colors s.t. for each \( (v, w) \in E \) the vertices \( v \) and \( w \) have different colors?
Other applications – Graph coloring

Given a graph $G = (V, E)$, can the vertices be colored with $k$ colors s.t. for each $(v, w) \in E$ the vertices $v$ and $w$ have different colors?

▶ Variables:
  - Integer variables $x_i$ for each vertex

▶ Constraints:
  - $1 \leq x_i \leq k$
  - $x_i \neq x_j$ for each $(x_i, x_j) \in E$
Other applications – Graph coloring with optimization

Given a graph $G = (V, E)$, can the vertices be colored with $k$ colors s.t. for each $(v, w) \in E$ the vertices $v$ and $w$ have different colors?

- **Variables:**
  - Integer variables $x_i$ for each vertex

- **Constraints:**
  - $1 \leq x_i \leq k$
  - $x_i \neq x_j$ for each $(x_i, x_j) \in E$

Minimize the number of colors? **MaxSMT!**
Other applications – networks and security

- Microsoft Azure and Amazon Web Services investing heavily in formal verification for access policies

- Encoding of semantics of access policies into SMT

- Verifying properties detects misconfigurations of policies

- SMT solvers Z3 and CVC4 invoked millions time a day at AWS for policy verification
How to profit from SMT solvers?

Determine for your favorite application:

- How to encode (parts of) my problem in SMT? Which theories to use?
- Leverage existing solvers
  - Standard input language
  - APIs available in several languages (Python, OCaml, C++, ...)
  - Continuously maintained (Z3, CVC4, ...)
- If need be, how to extend the SMT solver?
  - New formal calculus
  - New algorithm
  - Implementation techniques

Fun!!!
How to profit from SMT solvers?

Determine for your favorite application:

- How to encode (parts of) my problem in SMT? Which theories to use?
How to profit from SMT solvers?

Determine for your favorite application:

- **How to encode (parts of) my problem in SMT? Which theories to use?**

- **Leverage existing solvers**
  - Standard input language
  - APIs available in several languages (Python, OCaml, C++, ...)
  - Continuously maintained (Z3, CVC4, ...)

Fun!!!
How to profit from SMT solvers?

Determine for your favorite application:

▷ How to encode (parts of) my problem in SMT? Which theories to use?

▷ Leverage existing solvers
  ▶ Standard input language
  ▶ APIs available in several languages (Python, OCaml, C++, ...)
  ▶ Continuously maintained (Z3, CVC4, ...)

▷ If need be, how to extend the SMT solver?
  ▶ New formal calculus
  ▶ New algorithm
  ▶ Implementation techniques
How to profit from SMT solvers?

Determine for your favorite application:

- How to encode (parts of) my problem in SMT? Which theories to use?

- Leverage existing solvers
  - Standard input language
  - APIs available in several languages (Python, OCaml, C++, ...)
  - Continuously maintained (Z3, CVC4, ...)

- If need be, how to extend the SMT solver?
  - New formal calculus
  - New algorithm
  - Implementation techniques
SMT solving for fun and profit

Haniel Barbosa

2019–09–20, Belo Horizonte, Brazil
References


References


References
