Satisfiability Modulo Theories

Tjark Weber

UPPSALA UNIVERSITY

Mobility Seminar
January 20, 2012
Introduction

Satisfiability Modulo Theories =

Propositional satisfiability + background theories
Example: Job-Shop Scheduling

Given: \( n \) jobs, each composed of \( m \) tasks of varying duration, that must be performed consecutively on \( m \) machines; a total maximum time \( \text{max} \).

\[
\begin{array}{c|cc}
   d_{i,j} & \text{Machine 1} & \text{Machine 2} \\
   \hline
   \text{Job 1} & 2 & 1 \\
   \text{Job 2} & 3 & 1 \\
   \text{Job 3} & 2 & 3 \\
\end{array}
\]

\( \text{max} = 8 \)

Is there a schedule such that the end-time of every task is \( \leq \text{max} \)?
The job-shop scheduling problem has a straightforward encoding in propositional logic + linear integer arithmetic.

A schedule is specified by the start time $t_{i,j}$ for the $j$-th task of every job $i$.

Precedence constraints:

$$t_{i,1} \geq 0 \land t_{i,2} \geq t_{i,1} + d_{i,1} \land t_{i,2} + d_{i,2} \leq \text{max} \quad (\text{for } i = 1, 2, 3)$$

Resource constraints:

$$(t_{1,j} \geq t_{2,j} + d_{2,j} \lor t_{2,j} \geq t_{1,j} + d_{1,j}) \land$$

$$(t_{1,j} \geq t_{3,j} + d_{3,j} \lor t_{3,j} \geq t_{1,j} + d_{1,j}) \land$$

$$(t_{2,j} \geq t_{3,j} + d_{3,j} \lor t_{3,j} \geq t_{2,j} + d_{2,j}) \quad (\text{for } j = 1, 2)$$
Job-Shop Scheduling: Solution

SMT formula encoding

<table>
<thead>
<tr>
<th></th>
<th>Machine 1</th>
<th>Machine 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Job 3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

max = 8

Solution:

\[ t_{1,1} = 5, \ t_{1,2} = 7 \]
\[ t_{2,1} = 2, \ t_{2,2} = 6 \]
\[ t_{3,1} = 0, \ t_{3,2} = 3 \]
Background Theories

- EUF
- Arithmetic
- Arrays
- Bit-vectors
- Quantifiers
- Algebraic data types
- ...

\[ x = y \implies f(x) = f(y) \]

\[ y < 0 \implies x + y < x \]

\[ \text{select}(\text{store}(a, i, x), i) = x \]

\[ 2 \cdot x = x \ll 1 \]
SMT solvers are the core engine of many tools for program analysis, testing and verification.
Dynamic Symbolic Execution

Task: To find input that can steer program execution into specific branches.
Program Model Checking

Task: To prove/refute conjectures about the values of program variables in order to characterize a finite-state abstraction.
Static Program Analysis

Task: To check feasibility of certain program paths.
Task: To prove verification conditions that arise from claims of functional correctness.
We’ve seen what SMT solvers are good for. How do you actually interact with them?
The SMT-LIB Language

SMT solvers provide a **textual interface**. Most solvers support a standard language, SMT-LIB.

SMT-LIB defines

- concrete syntax for input formulas, and
- a command-based scripting language.

Solver-specific syntax is often available to extend SMT-LIB, e.g., for data types.
SMT-LIB: Example

; This example illustrates basic arithmetic and
; uninterpreted functions

(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
(push)
(assert (= x y))
(check-sat)
(pop)
(exit)
Inter-Process Communication

- File-based (the basic solution)
- Stream-based (when you need online functionality)
- Web interface (mostly for quick experiments)
- In-memory API (the tightly integrated approach)
- ...
A Virtuous Circle

enable

SMT solvers

evaluates

Applications

generate

SMT-COMP

are used in

Benchmarks

SMT solvers enable Applications, which generate SMT-COMP. SMT-COMP evaluates benchmarks that are used in Applications, thus forming a virtuous circle.
Choosing an SMT Solver

There are many good SMT solvers: Barcelogic, CVC, MathSAT, OpenSMT, Yices, Z3, ... 

Differences:

- Supported background theories
- Platform, license, API, incrementality, quantifiers, ...
- Performance (cf. SMT-COMP)
Choosing an SMT Solver

Speaker’s pick: Z3

- Good all-round solver, expressive input language
- Excellent performance
- Many other features: MaxSMT, fixed-point constraints, proofs
- Users can define custom theory solvers
- However, closed source (but free for academic use)

Your mileage may vary.
So far, we have considered SMT solvers as a black box.

SMT formula \rightarrow \text{sat (model)}\quad \text{unsat (proof)}

This view is sufficient for many applications!

We’ll now talk about what happens inside SMT solvers.
\[ \theta := \emptyset; \quad \text{\textit{partial Boolean valuation}} \]

while (true) {
    \[ \theta := \theta \cup \text{propagate}(\phi, \theta); \quad \text{\textit{deduce consequences}} \]
    
    if([\phi]_\theta == \text{true}) {
        return SATISFIABLE;
    } else if([\phi]_\theta == \text{false}) {
        \theta := \text{backtrack}(\phi, \theta); \quad \text{\textit{try a different branch}}
        if(\theta == \emptyset) { return UNSATISFIABLE; }
    } else {
        \theta := \theta \cup \text{decide}(\phi, \theta); \quad \text{\textit{branch on unassigned variable}}
    }
}

Tjark Weber
Satisfiability Modulo Theories
\( \Gamma := \text{abstraction function} \) that maps atomic formulas to Boolean variables;
\( \varphi := \Gamma(\varphi); \)
while(true) {
    \( \vartheta := \text{dpll}(\varphi); \)
    if(\( \vartheta = \text{ UNSATISFIABLE} \)) { return UNSATISFIABLE; }
    \( \Theta := \Gamma^{-1}(\vartheta); \)
    if(\( T(\Theta) = \text{ SATISFIABLE} \)) { return SATISFIABLE; }
    \( \varphi := \varphi \land \neg\vartheta; \quad // \text{theory lemma} \)
}
Combining Theory Solvers

Nelson-Oppen combination method: for disjoint, stably infinite theories it is sufficient to propagate equalities between variables.

Example: \( x \leq y \land y \leq x \land P(f(x) - f(y)) \land \neg P(0) \)
Combining Theory Solvers

Nelson-Oppen combination method: for disjoint, stably infinite theories it is sufficient to propagate equalities between variables.

Example: $x \leq y \land y \leq x \land \neg P\left(f(x) - f(y)\right) \land \neg P\left(0\right)$

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x \leq y$</td>
<td>$v_1 = f(x)$</td>
</tr>
<tr>
<td>$y \leq x$</td>
<td>$v_2 = f(y)$</td>
</tr>
<tr>
<td>$v_3 = v_1 - v_2$</td>
<td>$P(v_3)$</td>
</tr>
<tr>
<td>$v_4 = 0$</td>
<td>$\neg P(v_4)$</td>
</tr>
</tbody>
</table>
Combining Theory Solvers

Nelson-Oppen combination method: for disjoint, stably infinite theories it is sufficient to propagate equalities between variables.

Example:  \( x \leq y \land y \leq x \land P(f(x) - f(y)) \land \neg P(0) \)

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leq y )</td>
<td>( v_1 = f(x) )</td>
</tr>
<tr>
<td>( y \leq x )</td>
<td>( v_2 = f(y) )</td>
</tr>
<tr>
<td>( v_3 = v_1 - v_2 )</td>
<td>( P(v_3) )</td>
</tr>
<tr>
<td>( v_4 = 0 )</td>
<td>( \neg P(v_4) )</td>
</tr>
<tr>
<td>( x = y )</td>
<td></td>
</tr>
</tbody>
</table>

Tjark Weber
Satisfiability Modulo Theories
Combining Theory Solvers

Nelson-Oppen combination method: for disjoint, stably infinite theories it is sufficient to propagate equalities between variables.

Example: \( x \leq y \land y \leq x \land P(f(x) - f(y)) \land \neg P(0) \)

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leq y )</td>
<td>( v_1 = f(x) )</td>
</tr>
<tr>
<td>( y \leq x )</td>
<td>( v_2 = f(y) )</td>
</tr>
<tr>
<td>( v_3 = v_1 - v_2 )</td>
<td>( P(v_3) )</td>
</tr>
<tr>
<td>( v_4 = 0 )</td>
<td>( \neg P(v_4) )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( v_1 = v_2 )</td>
</tr>
</tbody>
</table>
Combining Theory Solvers

Nelson-Oppen combination method: for disjoint, stably infinite theories it is sufficient to propagate equalities between variables.

Example: \( x \leq y \land y \leq x \land P(f(x) - f(y)) \land \neg P(0) \)

\[
\begin{align*}
\text{Arithmetic} & \\
x \leq y & \quad \text{EUF} \\
y \leq x & \\
v_3 = v_1 - v_2 & \quad v_1 = f(x) \\
v_4 = 0 & \quad v_2 = f(y) \\
x = y & \quad P(v_3) \\
v_3 = v_4 & \quad \neg P(v_4) \\
v_1 = v_2 &
\end{align*}
\]
Combining Theory Solvers

Nelson-Oppen combination method: for disjoint, stably infinite theories it is sufficient to propagate equalities between variables.

Example: \( x \leq y \land y \leq x \land P(f(x) - f(y)) \land \neg P(v) \)

<table>
<thead>
<tr>
<th>Arithmetic</th>
<th>EUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \leq y )</td>
<td>( v_1 = f(x) )</td>
</tr>
<tr>
<td>( y \leq x )</td>
<td>( v_2 = f(y) )</td>
</tr>
<tr>
<td>( v_3 = v_1 - v_2 )</td>
<td>( P(v_3) )</td>
</tr>
<tr>
<td>( v_4 = 0 )</td>
<td>( \neg P(v_4) )</td>
</tr>
<tr>
<td>( x = y )</td>
<td>( v_1 = v_2 )</td>
</tr>
<tr>
<td>( v_3 = v_4 )</td>
<td>( \perp )</td>
</tr>
</tbody>
</table>
SMT solvers are expressive and easy to use. They scale orders of magnitude beyond custom ad hoc solvers.

Use them!

Do not write your own constraint solver.