Satisfiability Modulo Theory (SMT)

Testing, Quality Assurance, and Maintenance
Winter 2019

Prof. Arie Gurfinkel
Satisfiability Modulo Theory (SMT)

Satisfiability is the problem of determining whether a formula $F$ has a model

- if $F$ is *propositional*, a model is a truth assignment to Boolean variables
- if $F$ is *first-order formula*, a model assigns values to variables and interpretation to all the function and predicate symbols

**SAT Solvers**

- check satisfiability of propositional formulas

**SMT Solvers**

- check satisfiability of formulas in a *decidable* first-order theory (e.g., linear arithmetic, uninterpreted functions, array theory, bit-vectors)
SAT solvers have been the focus of increased recent attention thanks to technological advances and industrial applications. Yet, they draw on a combination of some of the most fundamental areas in computer science as well as discoveries from the past century of symbolic logic. They combine the problem of Boolean Satisfiability with domains, such as those studied in convex optimization and term-manipulating symbolic systems. They involve the decision problem, completeness and incompleteness of logical theories, and finally complexity theory. In this article, we present an overview of the field of Satisfiability Modulo Theories, and some of its applications.

key driving factor [4]. An important ingredient is a common interchange format for benchmarks, called SMT-LIB [33], and the classification of benchmarks into various categories depending on which theories are required. Conversely, a growing number of applications are able to generate benchmarks in the SMT-LIB format to further inspire improving SMT solvers.

There is a relatively long tradition of using SMT solvers in select and specialized contexts. One prolific case is theorem proving systems such as ACL2 [26] and PVS [32]. These use decision procedures to discharge lemmas encountered during interactive proofs. SMT solvers have also been used for a long time in the context of program verification and extended static checking [21], where verification is focused on assertion checking. Recent progress in SMT solvers, however, has enabled their use in a set of diverse applications, including interactive theorem provers and extended static checkers, but also in the context of scheduling, planning, test-case generation, model-based testing and program development, static program analysis, program synthesis, and run-time analysis, among several others.

We begin by introducing a motivating application and a simple instance of it that we will use as a running example.

1.1 An SMT Application - Scheduling

Consider the classical job shop scheduling decision problem. In this problem, there are $n$ jobs, each composed of $m$ tasks of varying duration that have to be performed consecutively on $m$ machines. The start of a new task can be delayed as long as needed in order to wait for a machine to become available, but tasks cannot be interrupted once
Example

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

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

Arithmetic
Example

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

Array theory
Example

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

Uninterpreted function
Example

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

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

By arithmetic, this is equivalent to

\[ b + 2 = c \land f(\text{read}(\text{write}(a, b, 3), b)) \neq f(3) \]
Example

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

By arithmetic, this is equivalent to

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

then, by the array theory axiom: \( \text{read}(\text{write}(v, i, x), i) = x \)

\[ b + 2 = c \land f(3) \neq f(3) \]
Example

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

By arithmetic, this is equivalent to

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

then, by the array theory axiom: \( \text{read}(\text{write}(v, i, x), i) = x \)

\[ b + 2 = c \land f(3) \neq f(3) \]

then, the formula is unsatisfiable
Example 2

\[ x \geq 0 \land f(x) \geq 0 \land y \geq 0 \land f(y) \geq 0 \land x \neq y \]
Example 2

\[ x \geq 0 \land f(x) \geq 0 \land y \geq 0 \land f(y) \geq 0 \land x \neq y \]

This formula is **satisfiable**
Example 2

\[ x \geq 0 \land f(x) \geq 0 \land y \geq 0 \land f(y) \geq 0 \land x \neq y \]

This formula is **satisfiable**:

Example model:

\[ x \rightarrow 1 \]
\[ y \rightarrow 2 \]
\[ f(1) \rightarrow 0 \]
\[ f(2) \rightarrow 1 \]
\[ f(\ldots) \rightarrow 0 \]
SMT - Milestones

<table>
<thead>
<tr>
<th>Year</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1977</td>
<td>Efficient Equality Reasoning</td>
</tr>
<tr>
<td>1979</td>
<td>Theory Combination Foundations</td>
</tr>
<tr>
<td>1979</td>
<td>Arithmetic + Functions</td>
</tr>
<tr>
<td>1982</td>
<td>Combining Canonizing Solvers</td>
</tr>
<tr>
<td>1992-8</td>
<td>Systems: PVS, Simplify, STeP, SVC</td>
</tr>
<tr>
<td>2002</td>
<td>Theory Clause Learning</td>
</tr>
<tr>
<td>2005</td>
<td>SMT competition</td>
</tr>
<tr>
<td>2006</td>
<td>Efficient SAT + Simplex</td>
</tr>
<tr>
<td>2007</td>
<td>Efficient Equality Matching</td>
</tr>
<tr>
<td>2009</td>
<td>Combinatory Array Logic, …</td>
</tr>
</tbody>
</table>

Includes progress from SAT:

SAT + Theory Solvers = SMT

15KLOC + 285KLOC = Z3

**Graph:**
- Z3 (of '07) Time On Boogie Regression
- Z3 Time On VCC Regression
- Simplify (of '01) time

**Time:**
- Nov 08
- March 09
SAT/SMT Revolution

Solve any computational problem by effective reduction to SAT/SMT

• iterate as necessary
SMT : Basic Architecture

SAT + Theory Solvers = SMT

- Equality + UF
- Arithmetic
- Bit-vectors
- ...
SAT + Theory solvers

**Basic Idea**

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

**Abstract (aka “naming” atoms)**

\[ p_1, \ p_2, (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]
SAT + Theory solvers

Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]
SAT + Theory solvers

Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \]
\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]
SAT + Theory solvers

Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, (p_3 \lor p_4) \]

\[ p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \neg p_3, \ p_4 \]

\[ x \geq 0, \ y = x + 1, \]
\[ \neg(y > 2), \ y < 1 \]
SAT + Theory solvers

Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \quad p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]

\[ x \geq 0, \ y = x + 1, \ \neg(y > 2), \ y < 1 \]

Unsatisfiable

\[ x \geq 0, \ y = x + 1, \ y < 1 \]

Theory Solver
SAT + Theory solvers

Basic Idea

\[ x \geq 0, \ y = x + 1, \ (y > 2 \lor y < 1) \]

Abstract (aka “naming” atoms)

\[ p_1, \ p_2, \ (p_3 \lor p_4) \quad p_1 \equiv (x \geq 0), \ p_2 \equiv (y = x + 1), \]
\[ p_3 \equiv (y > 2), \ p_4 \equiv (y < 1) \]

SAT Solver

Assignment

\[ p_1, \ p_2, \ \neg p_3, \ p_4 \]

x \geq 0, \ y = x + 1, \neg(y > 2), \ y < 1

New Lemma

\[ \neg p_1 \lor \neg p_2 \lor \neg p_4 \]

Unsatisfiable

x \geq 0, \ y = x + 1, \ y < 1

Theory Solver
SAT + Theory solvers

New Lemma
\neg p_1 \lor \neg p_2 \lor \neg p_4

Unsatisfiable
x \geq 0, y = x + 1, y < 1

AKA Theory conflict

Theory Solver