# Introduction to Formal Methods

Lecture 4

Satisfiability Modulo Theories

Hossein Hojjat & Fatemeh Ghassemi

October 2, 2018

# Example

- Do `f` and `g` produce the same results for the same values of inputs?

```
int f(int y) {
  int x, z;
  z = y;
  y = x;
  x = z;
  return x * x;
}
```

```
int g(int x) {
  return x * x;
}
```

# Example

- Do `f` and `g` produce the same results for the same values of inputs?

Satisfiable iff programs non-equivalent

$$(z = y \wedge y_1 = x \wedge x_1 = z \wedge ret_1 = x_1 \times x_1) \wedge$$
$$(ret_2 = y \times y) \wedge$$
$$(ret_1 \neq ret_2)$$

```
int f(int y) {
  int x, z;
  z = y;
  y = x;
  x = z;
  return x * x;
}
```

```
int g(int x) {
  return x * x;
}
```

# Example

- Do `f` and `g` produce the same results for the same values of inputs?

Satisfiable iff programs non-equivalent

$$(z = y \wedge y_1 = x \wedge x_1 = z \wedge ret_1 = x_1 \times x_1) \wedge$$
$$(ret_2 = y \times y) \wedge$$
$$(ret_1 \neq ret_2)$$

- We may bit-blast the formula into propositional logic (e.g. 64-bit machine)
- **Problem:** Bit-blasting does not scale
- For encoding multiplication we need quadratical number of clauses
- Using an SMT solver, the formula can be solved as it is

```
int f(int y) {
  int x, z;
  z = y;
  y = x;
  x = z;
  return x * x;
}
```

```
int g(int x) {
  return x * x;
}
```

# Motivation

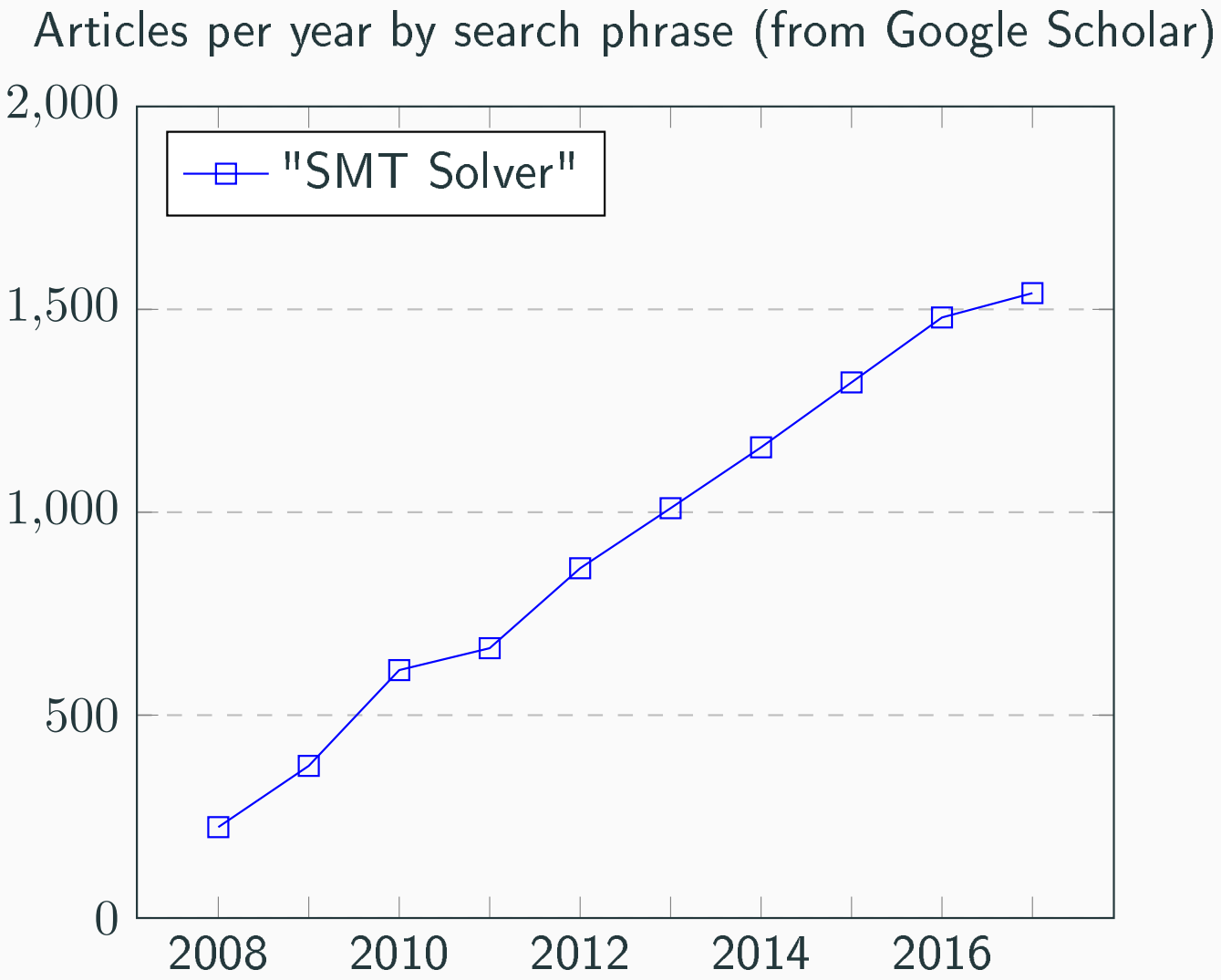
- Boolean engines such as SAT solvers are typical engines of choice for today's industrial verification applications
- However, systems are usually designed and modeled at a higher level than Boolean
- Translation to Boolean logic is simple but laborious and expensive
- A primary goal of research in Satisfiability Modulo Theories (SMT) is to create verification engines that:
  - Can reason natively at a higher level of abstraction
  - Retain the speed and automation of today's Boolean engines

# Motivation

Many applications naturally need features beyond propositional logic

- First-order terms: variables, constants, function symbols
  $f(a) = b \rightarrow g(b) = a$
- Theories: e.g. integer arithmetic, sets, floating points
  $n < 10 \rightarrow n \times m \leq 20$
- Quantifiers: First-order "forall" $\forall$ and "exists" ($\exists$)

# Growth of SMT



# Satisfiability Modulo Theories (SMT)

Is formula $\phi$ satisfiable modulo theory $T$?

$$x + 2 = y \rightarrow f\,(\,\mathit{select}\,(\,\mathit{store}\,(a, x, 3), y - 2)) = f\,(\,y - x + 1)$$

Array Theory

Arithmetic

Uninterpreted Functions

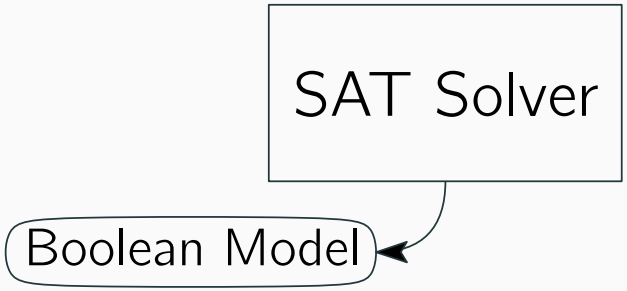
# From SAT to SMT


- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$

# From SAT to SMT

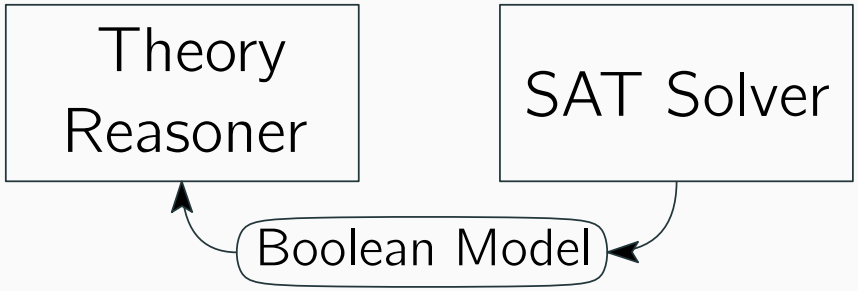

- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$
- To SAT solver:
  $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \big(\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\big)$

# From SAT to SMT



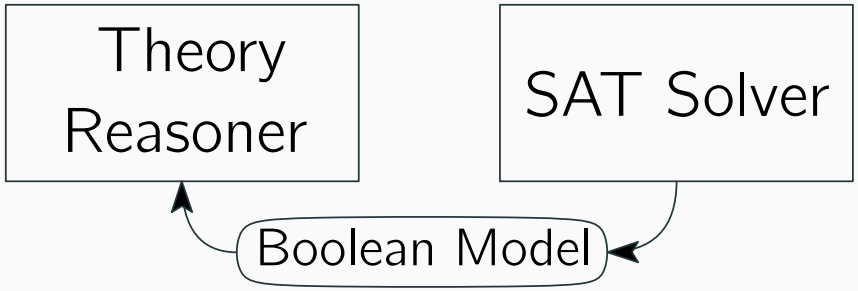

- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$
- To SAT solver:
  $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \big(\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\big)$
- Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$

# From SAT to SMT



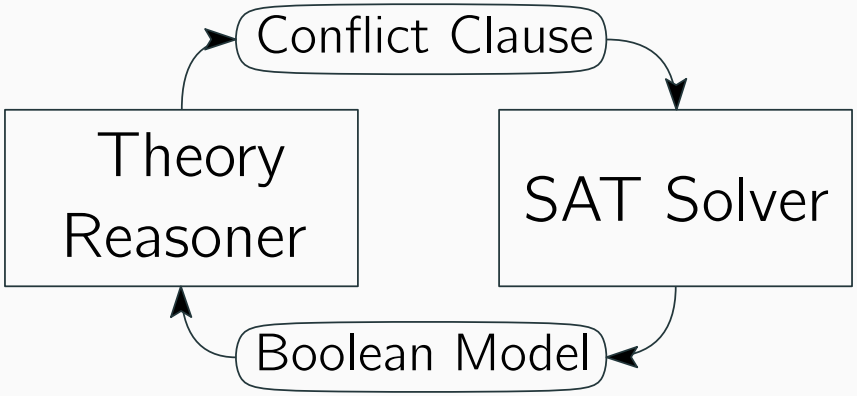

- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$
- To SAT solver:
  $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \big(\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\big)$
- Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
- Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$

# From SAT to SMT



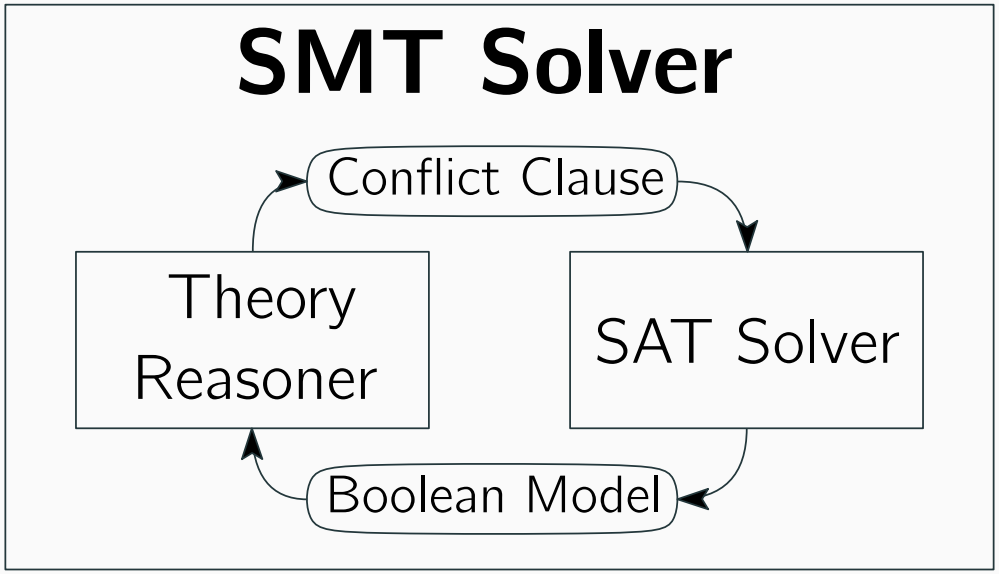

- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$
- To SAT solver:
  $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \big(\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\big)$
- Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
- Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$
  **unsatisfiable**

# From SAT to SMT




- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$
- To SAT solver:
  $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \big(\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\big)$
- Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
- Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$
  **unsatisfiable**
- New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

# From SAT to SMT




- Input:
  $a \leq b \wedge b \leq a + x \wedge x = 0 \wedge \big(f(a) \neq f(b) \vee (q(a) \wedge \neg q(b + x))\big)$
- To SAT solver:
  $p_{a \leq b} \wedge p_{b \leq a+x} \wedge p_{x=0} \wedge \big(\neg p_{f(a)=f(b)} \vee (p_{q(a)} \wedge \neg p_{q(b+x)})\big)$
- Boolean model: $p_{a \leq b}, p_{b \leq a+x}, p_{x=0}, \neg p_{f(a)=f(b)}$
- Theory reasoner: $a \leq b, b \leq a + x, x = 0, f(a) \neq f(b)$
  **unsatisfiable**
- New clause: $\neg p_{a \leq b} \vee \neg p_{b \leq a+x} \vee \neg p_{x=0} \vee p_{f(a)=f(b)}$

# SMT-LIB

- SMT-LIB is a language for specifying input to SMT solvers
- Basic instructions:

| | |
|---|---|
| `(declare-fun x () Int)` | declare an integer constant `x` |
| `(assert (> x 0))` | add `x > 0` to known facts |
| `(check-sat)` | check if there exist an assignment that makes all known facts true |
| `(get-model)` | print this assignment |

# http://rise4fun.com/z3

z3 Microsoft Research

Is this formula satisfiable?

```
 1 (declare-fun x () Int)
 2 (declare-fun x1 () Int)
 3 (declare-fun y () Int)
 4 (declare-fun y1 () Int)
 5 (declare-fun z () Int)
 6 (declare-fun ret1 () Int)
 7 (declare-fun ret2 () Int)
 8 (assert (and (= z y)(= y1 x)(= x1 z)(= ret1 (* x1 x1))(= ret2 (* y y))(not (= ret1 ret2))))
 9 (check-sat)
10 (exit)
11
```

tutorial

home video permalink

'▶' shortcut: Alt+B

unsat

samples
smtc_arith
doc_examples

about Z3 - Efficient Theorem Prover

Z3 is a high-performance theorem prover. Z3 supports arithmetic, fixed-size bit-vectors, extensional arrays, datatypes, uninterpreted functions, and quantifiers.

# SMT Solvers: Language

- Language of SAT solvers is Boolean logic
- Language of SMT solvers is **First-Order Logic** (FOL)
- FOL includes the Boolean operations of Boolean logic, instead of propositional variables, more complicated expressions are allowed
- A first-order language must specify its signature: the set of constant, function, and predicate symbols that are allowed
- Each predicate and function symbol has an associated arity: natural number indicating how many arguments it takes
  - Equality is a special predicate symbol of arity 2
  - Constant symbols can also be thought of as functions whose arity is 0

# First-Order Languages: Examples

**Propositional Logic**

- Predicate symbols: $x_1, x_2, \ldots$
- Constant symbols: none
- Function symbols: none

**Elementary Number Theory**

- Predicate symbols: $<$
- Constant symbols: $0$
- Function symbols: $S$ (successor), $+$, $\times$

# First-Order Logic: Syntax

**Terms**

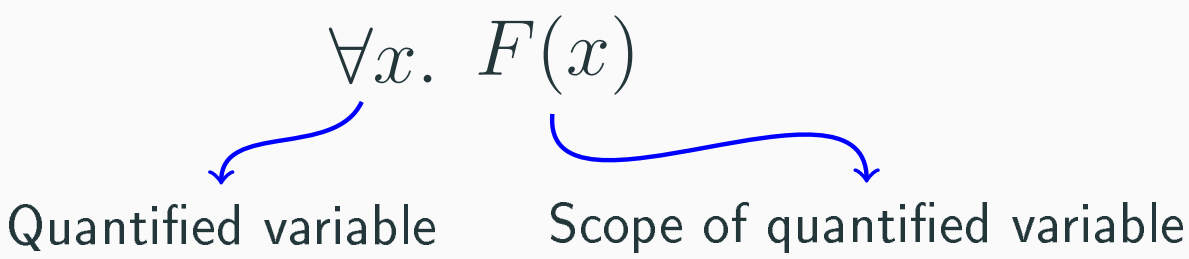
- Variables and constants are terms
- For each function symbol $f$ of arity $n$, and terms $t_1$,...,$t_n$, $f(t_1, \cdots, t_n)$ is a term

- Atom: an expression of the form: $P(t_1, \cdots, t_n)$ where $P$ is a predicate symbol of arity $n$ and $t_1, \cdots, t_n$ are terms
- An atom or its negation is called a literal
- Formulas are built from literals using the Boolean operators and quantification

# Quantifiers

existential quantifier: $\exists x.F(x)$ "there exists an $x$ such that $F(x)$"

universal quantifier: $\forall x.F(x)$ "for all $x$, $F(x)$"




- Every occurrence of a variable $x$ in a formula $\forall x.F(x)$ (or $\exists x.F(x)$) is called a bound occurrence
- Occurrences which are not bound are called free
- Closed formula: no free variables
- Open formula: some free variables
- Ground formula: no variables

# FOL example 1

- Fermat's Last Theorem: No three positive integers $a$, $b$, $c$ satisfy the equation $a^n + b^n = c^n$ for any integer $n$ greater than $2$
- Assuming universe is integers, how do we express this theorem in FOL using function $\hat{}$ and predicates $>$, $=$?

# FOL example 2

- Consider the axiom schema of unrestricted comprehension in naive set theory:

  "There exists a set whose members are precisely those objects that satisfy predicate $P$"

- Using predicates *IsSet*, $\in$, $P$ express this in FOL

# Theory

- FOL is very expressive, powerful and undecidable in general
- Many application domains do not need the full power of FOL
- First-order theories are useful for reasoning about specific applications
  - e.g., programs with arithmetic operations over integers
- Specialized, efficient decision procedures

# Theory

A First-order theory $T$ consists of:

- Signature $\Sigma_T$ : set of constant, function, and predicate symbols
  - Have no meaning
- Axioms $A_T$ : set of closed formulas over $\Sigma_T$
  - Provide meaning for symbols of $\Sigma_T$

# Theory of Equality $T_=$

- Also known as theory of equality with uninterpreted functions
- Uninterpreted functions are useful as an abstraction or over-approximation mechanism

**Signature**

- $=$ binary predicate, interpreted by axioms
- all constant, function, and predicate symbols

$$\Sigma_= = \{=, a, b, c, \cdots, f, g, h, \cdots, p, q, r\}$$

# Theory of Equality $T_=$

**Axioms**

- $\forall x.\ x = x$ (reflexivity)
- $\forall x, y.\ x = y \rightarrow y = x$ (symmetry)
- $\forall x, y, z.\ x = y \wedge y = z \rightarrow x = z$ (transitivity)
- $\forall x_1, \cdots, x_n, y_1, \cdots, y_n. \bigwedge x_i = y_i \rightarrow f(x_1, \cdots, x_n) = f(y_1, \cdots, y_n)$ (function congruence)
- $\forall x_1, \cdots, x_n, y_1, \cdots, y_n. \bigwedge x_i = y_i \rightarrow (p(x_1, \cdots, x_n) \leftrightarrow p(y_1, \cdots, y_n))$ (predicate congruence)

# Theory of Presburger Arithmetic

- Presburger arithmetic: allows only addition over natural numbers
- $\Sigma_{\mathbb{N}} = \{0, 1, =, +\}$

**Axioms ($A_{\mathbb{N}}$)**

- $\forall x.\ \neg(x + 1 = 0)$ (zero)
- $\forall x.\ x + 0 = 0$ (plus zero)
- $\forall x, y.\ x + 1 = y + 1 \rightarrow x = y$ (successor)
- $\forall x, y.\ x + (y + 1) = (x + y) + 1$ (plus successor)
- $F[0] \wedge (\forall x. F[x] \rightarrow F[x + 1]) \rightarrow \forall x. F[x]$ (induction)

# Theory of Arrays

$\Sigma_A = \{\mathit{select}, \mathit{store}\}$

- $\mathit{select}(a, i)$ binary function that returns the value of array $a$ at index $i$
- $\mathit{store}(a, i, v)$ ternary function that returns an array identical to a except that at index $i$ it has value $v$

**Axioms**

$$\forall a.\forall i.\forall v.(\mathit{select}(\mathit{store}(a, i, v), i) = v)$$

$$\forall a.\forall i.\forall j.\forall v.(i \neq j \rightarrow \mathit{select}(\mathit{store}(a, i, v), j) = \mathit{select}(a, j))$$

# Combination of Theories

Given

- theory $T_1$ with signature $\Sigma_{T_1}$ and axioms $A_{T_1}$
- theory $T_2$ with signature $\Sigma_{T_2}$ and axioms $A_{T_2}$
- an SMT solver for $T_1$
- an SMT solver for $T_2$

Can we produce a solver for $T_1 \cup T_2$?

- $T_1 \cup T_2$ with signature $\Sigma_{T_1} \cup \Sigma_{T_2}$ and axioms $A_{T_1} \cup A_{T_2}$

# Nelson-Oppen Framework

Framework for deciding combined theories under certain assumptions, e.g, only for quantifier-free theories

Examples

- theory of arrays and bitvectors
- theory of arrays and integers

# Reference


- Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli: "Satisfiability modulo theories", In Armin Biere, Hans van Maaren, and Toby Walsh, editors, Handbook of Satisfiability, volume 4, chapter 8. IOS Press, 2009.