

# C07 – SMT Solvers

---

Program Verification

FMI · Denisa Diaconescu · Spring 2022

What are SMT solvers?

First-order logic (in 5 min)

Some first-order theories

How SMT solvers works?

**What are SMT solvers?**

---

# The SMT problem

The SMT problem:

*Given a first-order logic formula, with symbols from (possibly several) theories, does it have a model?*

Is the formula satisfiable? If so, how?

The SAT problem is a special case, in which

- the formula is quantifier-free, without function symbols or equality
- no theories are used

SAT solving algorithms are an important ingredient in SMT solvers

# First-order theories

- Whereas the language of SAT solvers is Boolean logic, the language of SMT solvers is **first-order logic**.
- **First-order theories** allow us to capture structures which are used by programs (e.g., arrays, integers) and enable reasoning about them.
- **Validity in first order logic (FOL) is undecidable!**
  - **Lambda calculus** – Alonzo Church (1936)
  - **Turing machines** – Alan Turing (1937)
  - **Recursive functions** – Kurt Gödel (1934) and Stephen Kleene (1936)
- Validity in particular first order theories is (sometimes) decidable.

Combine **propositional satisfiability** search techniques with solvers for **specific first-order theories**:

- Linear arithmetic
- Bit vectors
- Arrays
- ...



SMT solvers are used as core engines in many tools in

- program analysis and verification
- software engineering
- hardware verification
- ...
- symbolic execution
- concolic execution

Many SMT solvers:

- Z3 (Microsoft)
- Yices
- MathSAT
- CVC4
- ...



## First-order logic (in 5 min)

---

- The language includes the **Boolean operations** of propositional logic, but 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**: a 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

## Example (Propositional logic)

- Equality: no
- Predicate symbols:  $x_1, x_2, \dots$
- Constant symbols: none
- Function symbols: none

## Example (Propositional logic)

- Equality: no
- Predicate symbols:  $x_1, x_2, \dots$
- Constant symbols: none
- Function symbols: none

## Example (Elementary Number Theory)

- Equality: yes
- Predicate symbols:  $<$
- Constant symbols: 0
- Function symbols:  $S$  (successor),  $+$ ,  $*$

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

- **Terms**
  - **Variables** and **constants** are terms
  - For each function symbol  $f$  of arity  $n$ , and terms  $t_1, \dots, t_n$ ,  $f(t_1, \dots, t_n)$  is a term
- **Atomic formulas** are expressions of the form
  - $P(t_1, \dots, t_n)$  where  $P$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n$  are terms.
    - Predicate symbols of arity 0 are called **propositional atoms**.
  - $t = t'$  if the logic is with equality where  $t, t'$  are terms.

- **Terms**
  - **Variables** and **constants** are terms
  - For each function symbol  $f$  of arity  $n$ , and terms  $t_1, \dots, t_n$ ,  $f(t_1, \dots, t_n)$  is a term
- **Atomic formulas** are expressions of the form
  - $P(t_1, \dots, t_n)$  where  $P$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n$  are terms.
    - Predicate symbols of arity 0 are called **propositional atoms**.
  - $t = t'$  if the logic is with equality where  $t, t'$  are terms.
- An atomic formula or its negation is called a **literal**.

- **Terms**
  - **Variables** and **constants** are terms
  - For each function symbol  $f$  of arity  $n$ , and terms  $t_1, \dots, t_n$ ,  $f(t_1, \dots, t_n)$  is a term
- **Atomic formulas** are expressions of the form
  - $P(t_1, \dots, t_n)$  where  $P$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n$  are terms.
    - Predicate symbols of arity 0 are called **propositional atoms**.
  - $t = t'$  if the logic is with equality where  $t, t'$  are terms.
- An atomic formula or its negation is called a **literal**.
- **Formulas** are built from literals using the Boolean operators and quantification. If  $\alpha$  is a formula, then for every variables  $x$ 
  - $\forall x. \alpha$  is a formula
  - $\exists x. \alpha$  is a formula



Given a signature  $\Sigma$  and a set  $V$  of variables, a **structure**  $\mathcal{M}$  of  $\Sigma$  consists of:

1. A nonempty set  $M$  called the **domain** of  $\mathcal{M}$
2. For each constant  $c$  in  $\Sigma$ , an element  $c^{\mathcal{M}} \in M$ .
3. For each  $n$ -ary function symbol  $f$  in  $\Sigma$ , an  $n$ -ary function  $f^{\mathcal{M}} : M^n \rightarrow M$ .
4. For each  $n$ -ary predicate symbol  $P$  in  $\Sigma$ , an  $n$ -ary relation  $P^{\mathcal{M}} \subseteq M^n$ .

An **interpretation**  $I$  of the variables in  $V$  into a structure  $\mathcal{M}$  maps each variable  $v \in V$  to an element  $I(v) \in M$ .

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible structure  $\mathcal{M}$  for this signature has the domain  $\mathbb{N}$ , the set of natural numbers,  $\in^{\mathcal{M}} = <$ , and  $\emptyset^{\mathcal{M}} = 0$ .

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible structure  $\mathcal{M}$  for this signature has the domain  $\mathbb{N}$ , the set of natural numbers,  $\in^{\mathcal{M}} = <$ , and  $\emptyset^{\mathcal{M}} = 0$ .

Consider the sentence  $\exists x \forall y \neg (y \in x)$ .

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible structure  $\mathcal{M}$  for this signature has the domain  $\mathbb{N}$ , the set of natural numbers,  $\in^{\mathcal{M}} = <$ , and  $\emptyset^{\mathcal{M}} = 0$ .

Consider the sentence  $\exists x \forall y \neg (y \in x)$ .

*What does this sentence mean in this structure?*

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible structure  $\mathcal{M}$  for this signature has the domain  $\mathbb{N}$ , the set of natural numbers,  $\in^{\mathcal{M}} = <$ , and  $\emptyset^{\mathcal{M}} = 0$ .

Consider the sentence  $\exists x \forall y \neg (y \in x)$ .

*What does this sentence mean in this structure?*

There is a natural number  $x$  such that no other natural number is smaller than  $x$ .

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible structure  $\mathcal{M}$  for this signature has the domain  $\mathbb{N}$ , the set of natural numbers,  $\in^{\mathcal{M}} = <$ , and  $\emptyset^{\mathcal{M}} = 0$ .

Consider the sentence  $\exists x \forall y \neg (y \in x)$ .

*What does this sentence mean in this structure?*

There is a natural number  $x$  such that no other natural number is smaller than  $x$ .

*Is this sentence true in the structure?*

## Example

Consider the signature with a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible structure  $\mathcal{M}$  for this signature has the domain  $\mathbb{N}$ , the set of natural numbers,  $\in^{\mathcal{M}} = <$ , and  $\emptyset^{\mathcal{M}} = 0$ .

Consider the sentence  $\exists x \forall y \neg (y \in x)$ .

*What does this sentence mean in this structure?*

There is a natural number  $x$  such that no other natural number is smaller than  $x$ .

*Is this sentence true in the structure?*

Since 0 has this property, the sentence is true in this structure.



# Term interpretation

Given

- a structure  $\mathcal{M}$  of  $\Sigma$ , and
- an interpretation  $I$  of  $V$  into  $\mathcal{M}$ ,

We can inductively extend the interpretation  $I$  to all terms over  $\Sigma$  with variables from  $V$ , as follows:

- $\bar{I}(v) = I(v)$  for any  $v \in V$
- $\bar{I}(c) = c^{\mathcal{M}}$  for any constant symbol  $c$  in  $\Sigma$
- $\bar{I}(f(t_1, \dots, t_n)) = f^{\mathcal{M}}(\bar{I}(t_1), \dots, \bar{I}(t_n))$ , for any  $n$ -ary function symbol  $f \in \Sigma$  and any terms  $(t_i)_{1 \leq i \leq n}$

Given

- a structure  $\mathcal{M}$  of  $\Sigma$ , and
- an interpretation  $I$  of  $V$  into  $\mathcal{M}$ ,

We can inductively define a satisfaction relation for  $\Sigma$ -formulas with variables from  $V$ , as follows:

- $\mathcal{M}, I \models t_1 = t_2$  iff  $\bar{I}(t_1) = \bar{I}(t_2)$
- $\mathcal{M}, I \models P(t_1, \dots, t_n)$  iff  $(\bar{I}(t_1), \dots, \bar{I}(t_n)) \in P^{\mathcal{M}}$
- $\mathcal{M}, I \models \phi_1 \wedge \phi_2$  iff  $\mathcal{M}, I \models \phi_1$  and  $\mathcal{M}, I \models \phi_2$
- $\mathcal{M}, I \models \neg\phi$  iff  $\mathcal{M}, I \not\models \phi$
- $\mathcal{M}, I \models \exists x.\phi$  iff there exists  $a \in M$  such that  $\mathcal{M}, I[x \leftarrow a] \models \phi$

# Satisfiability and validity in a structure

Given

- a structure  $\mathcal{M}$  of  $\Sigma$ , and
- a  $\Sigma$  formula  $\phi$  with variables from  $V$

We say that

- $\phi$  is satisfiable in  $\mathcal{M}$  iff  $\mathcal{M}, I \models \phi$  for **some** interpretation  $I$
- $\phi$  is valid in  $\mathcal{M}$  iff  $\mathcal{M}, I \models \phi$  for **any** interpretation  $I$

# Satisfiability and validity in a structure

Given

- a structure  $\mathcal{M}$  of  $\Sigma$ , and
- a  $\Sigma$  formula  $\phi$  with variables from  $V$

We say that

- $\phi$  is satisfiable in  $\mathcal{M}$  iff  $\mathcal{M}, I \models \phi$  for **some** interpretation  $I$
- $\phi$  is valid in  $\mathcal{M}$  iff  $\mathcal{M}, I \models \phi$  for **any** interpretation  $I$

Note that  $\phi$  is valid iff  $\neg\phi$  is not satisfiable. Indeed,

- $\phi$  is valid iff  $\mathcal{M}, I \models \phi$  for any  $I$
- iff there exists no  $I$  such that  $\mathcal{M}, I \not\models \phi$
- iff there exists no  $I$  such that  $\mathcal{M}, I \models \neg\phi$
- iff  $\neg\phi$  is not satisfiable

# Satisfiability, validity, and free variables

Given a formula  $\phi$  and the set  $X$  of its free variables

- If  $I_1, I_2$  interpretations such that  $I_1|_X = I_2|_X$
- Then  $\mathcal{M}, I_1 \models \phi$  iff  $\mathcal{M}, I_2 \models \phi$

Hence, if a formula  $\phi$  does not have free variables

- The satisfaction relation does not depend on interpretations  
We can therefore write  $\mathcal{M} \models \phi$
- satisfiability and validity coincide for  $\phi$

## Satisfiability, validity, and closed formulas

Given a formula  $\phi$  and the set  $\{x_1, \dots, x_n\}$  of its free variables,

- $\phi$  satisfiable in  $\mathcal{M}$  iff  $\mathcal{M} \models \exists x_1 \dots \exists x_n. \phi$
- $\phi$  valid in  $\mathcal{M}$  iff  $\mathcal{M} \models \forall x_1 \dots \forall x_n. \phi$

# Satisfiability and validity in a theory

A **theory** is a set of sentences (no free variables).

Given a theory  $T$ , a formula  $\varphi$  is

- **$T$ -valid** if  $\varphi$  is satisfied by all models of  $T$  for all interpretations of  $V$ .
- **$T$ -satisfiable** if it is satisfied by some model of  $T$  for some interpretations of  $V$ .

# Drinker's paradox

## Example

$$\exists x(D(x) \rightarrow \forall yD(y))$$

*There is someone in the pub such that  
if he/she is drinking, then everyone in the pub is drinking.*

How is this formula?

1. valid
2. unsatisfiable
3. satisfiable but not valid



# Drinker's paradox:

## Example

$$\exists x(D(x) \rightarrow \forall yD(y))$$

*There is someone in the pub such that  
if he/she is drinking, then everyone in the pub is drinking.*

How is this formula?

1. valid
2. unsatisfiable
3. satisfiable but not valid





The **validity problem** for  $T$  is the problem of deciding, for each formula  $\varphi$ , whether  $\varphi$  is  $T$ -valid.

The **validity problem** for  $T$  is the problem of deciding, for each formula  $\varphi$ , whether  $\varphi$  is  $T$ -valid.

The **satisfiability problem** for  $T$ , or **the problem of satisfiability modulo theories**, is the problem of deciding, for each formula  $\varphi$ , whether  $\varphi$  is  $T$ -satisfiable.

$$\varphi \text{ is } T\text{-valid} \quad \text{iff} \quad \neg\varphi \text{ is } T\text{-unsatisfiable.}$$

# SMT problem

The **validity problem** for  $T$  is the problem of deciding, for each formula  $\varphi$ , whether  $\varphi$  is  $T$ -valid.

The **satisfiability problem** for  $T$ , or the **problem of satisfiability modulo theories**, is the problem of deciding, for each formula  $\varphi$ , whether  $\varphi$  is  $T$ -satisfiable.

$$\varphi \text{ is } T\text{-valid} \quad \text{iff} \quad \neg\varphi \text{ is } T\text{-unsatisfiable.}$$

SMT = satisfiability modulo theories

It is important to make a distinction between SMT and standard first order satisfiability.

For example, is the following sentence satisfiable?

$$\textit{read}(\textit{write}(a, i, v), i) \neq v$$

# Satisfiability modulo theories

It is important to make a distinction between SMT and standard first order satisfiability.

For example, is the following sentence satisfiable?

$$\textit{read}(\textit{write}(a, i, v), i) \neq v$$

If the set of allowable models is unrestricted, then the answer is **yes**.

However, if we only consider models that obey the axioms for *read* and *write* then the answer is **no**.

Quiz time!

<https://www.questionpro.com/t/AT4NiZr2Xb>

## Some first-order theories

---

# First-order theories

A **first-order theory**  $T$  is defined by

- **Signature**  $\Sigma_T$  = set of constant, function, and predicate symbols
  - Have no meaning
- **Axioms**  $A_T$  = set of  $\Sigma_T$ -sentences (no free variables)
  - Provide meaning for the symbols of  $\Sigma_T$

A **fragment** of a theory  $T$  is a syntactically restricted subset of formulas of the theory.



# First-order theories

A **first-order theory**  $T$  is defined by

- **Signature**  $\Sigma_T$  = set of constant, function, and predicate symbols
  - Have no meaning
- **Axioms**  $A_T$  = set of  $\Sigma_T$ -sentences (no free variables)
  - Provide meaning for the symbols of  $\Sigma_T$

A **fragment** of a theory  $T$  is a syntactically restricted subset of formulas of the theory.

## Example

The **quantifier-free fragment** of a theory  $T$  (denoted **QFF**  $T$ ) is the set of formulas without quantifiers that are valid in  $T$ .

# First-order theories

A **first-order theory**  $T$  is defined by

- **Signature**  $\Sigma_T$  = set of constant, function, and predicate symbols
  - Have no meaning
- **Axioms**  $A_T$  = set of  $\Sigma_T$ -sentences (no free variables)
  - Provide meaning for the symbols of  $\Sigma_T$

A **fragment** of a theory  $T$  is a syntactically restricted subset of formulas of the theory.

## Example

The **quantifier-free fragment** of a theory  $T$  (denoted **QFF**  $T$ ) is the set of formulas without quantifiers that are valid in  $T$ .

A theory is **decidable** if for every formula in the theory we can automatically check whether the formula is valid or not.

Similarly for fragments of a theory.

- In principle, SMT can be applied to any theory  $T$ .
- In practice, when people talk about SMT, they are usually referring to a small set of specific theories.
- We will consider a few examples of theories which are of particular interest in program analysis and verification applications.

# First-order theories

Theory	Description	Full Fragment	No Quantifiers
$T_E$	Equality	NO	YES
$T_{PA}$	Peano arithmetic	NO	NO
$T_N$	Presburger arithmetic	YES	YES
$T_Z$	Linear Integers	YES	YES
$T_R$	Reals (with *)	YES	YES
$T_Q$	Rationals (without *)	YES	YES
$T_{RDS}$	Recursive Data Structures	NO	YES
$T_{RDS}^+$	Acyclic Recursive Data Structures	YES	YES
$T_A$	Arrays	NO	YES
$T_A^=$	Arrays with extensionality	NO	YES

source: *The Calculus of Computation*, Manna and Bradley

# Theory of Equality

## Signature $\Sigma_E$ :

- Any function, predicate, and constant
- The predicate  $=$  which is **interpreted** (i.e., defined via axioms)

## Axioms $A_E$ :

- $\forall x. x = x$
- $\forall x, y. x = y \rightarrow y = x$
- $\forall x, y, z. x = y \wedge y = z \rightarrow x = z$
- $\forall x_1, \dots, \forall x_n, y_1, \dots, y_n. x_1 = y_1 \wedge \dots \wedge x_n = y_n \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$

# Theory of Equality

Signature  $\Sigma_E$ :

- Any function, predicate, and constant
- The predicate  $=$  which is **interpreted** (i.e., defined via axioms)

Axioms  $A_E$ :

- $\forall x. x = x$
- $\forall x, y. x = y \rightarrow y = x$
- $\forall x, y, z. x = y \wedge y = z \rightarrow x = z$
- $\forall x_1, \dots, \forall x_n, y_1, \dots, y_n. x_1 = y_1 \wedge \dots \wedge x_n = y_n \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$

$T_E$  is undecidable

QFF  $T_E$  is decidable

## Example

$$(a = b) \wedge (b = c) \rightarrow (g(f(a), b) = g(f(c), a))$$

Exercise: Is this valid? If yes, prove it.

# Arithmetic: Natural numbers and Integers

Natural numbers  $\mathbb{N} = \{0, 1, 2, \dots\}$

Integers  $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$

Three theories:

- **Peano arithmetic**  $T_{PA}$ 
  - Natural numbers with addition (+), multiplication (\*), equality (=)
- **Presburger arithmetic**  $T_N$ 
  - Natural numbers with addition (+), equality (=)
- **Theory of integers**  $T_Z$ 
  - Integers with addition (+), subtraction (-), comparison (>), equality (=), multiplication by constants



# Theory of Peano Arithmetic

## Signature $\Sigma_E$ :

- Constants:  $0, 1$
- Binary functions:  $+, *$
- Predicate:  $=$

## Axioms $A_{PA}$ :

- $\forall x. \neg(x + 1 = 0)$
- $\forall x. x + 0 = x$
- $\forall x. x * 0 = 0$
- $\forall x, y. x + 1 = y + 1 \rightarrow x = y$
- $\forall x, y. x + (y + 1) = (x + y) + 1$
- $\forall x, y. x * (y + 1) = (x * y) + x$
- For each formula  $\varphi(x, \bar{y})$  in the language  $\Sigma_E$ ,  
 $\forall \bar{y}. \varphi(0, \bar{y}) \wedge (\forall x. \varphi(x, \bar{y}) \rightarrow \varphi(x + 1, \bar{y})) \rightarrow \forall x. \varphi(x, \bar{y})$  (induction)

# Theory of Peano Arithmetic

## Signature $\Sigma_E$ :

- Constants:  $0, 1$
- Binary functions:  $+, *$
- Predicate:  $=$

## Axioms $A_{PA}$ :

- $\forall x. \neg(x + 1 = 0)$
- $\forall x. x + 0 = x$
- $\forall x. x * 0 = 0$
- $\forall x, y. x + 1 = y + 1 \rightarrow x = y$
- $\forall x, y. x + (y + 1) = (x + y) + 1$
- $\forall x, y. x * (y + 1) = (x * y) + x$
- For each formula  $\varphi(x, \bar{y})$  in the language  $\Sigma_E$ ,  
 $\forall \bar{y}. \varphi(0, \bar{y}) \wedge (\forall x. \varphi(x, \bar{y}) \rightarrow \varphi(x + 1, \bar{y})) \rightarrow \forall x. \varphi(x, \bar{y})$

(induction)

$T_{PA}$  is undecidable  
*QFF*  $T_{PA}$  is undecidable

## Example

Is the formula

$$3 * x + 2 = 2 * y \text{ in } T_{PA}?$$

## Example

Is the formula

$$3 * x + 2 = 2 * y \text{ in } T_{PA}?$$

Yes! It can be written as

$$(1 + 1 + 1) * x + 1 + 1 = (1 + 1) * y$$

# Theory of Presburger Arithmetic

## Signature $\Sigma_N$ :

- Constants: 0, 1
- Binary functions: +
- Predicate: =

NO multiplication!

## Axioms $A_N$ :

- $\forall x. \neg(x + 1 = 0)$
- $\forall x. x + 0 = x$
- $\forall x, y. x + 1 = y + 1 \rightarrow x = y$
- $\forall x, y. x + (y + 1) = (x + y) + 1$
- For each formula  $\varphi(x, \bar{y})$  in the language  $\Sigma_E$ ,  
 $\forall \bar{y}. \varphi(0, \bar{y}) \wedge (\forall x. \varphi(x, \bar{y}) \rightarrow \varphi(x + 1, \bar{y})) \rightarrow \forall x. \varphi(x, \bar{y})$

(induction)

# Theory of Presburger Arithmetic

## Signature $\Sigma_N$ :

- Constants: 0, 1
- Binary functions: +
- Predicate: =

NO multiplication!

## Axioms $A_N$ :

- $\forall x. \neg(x + 1 = 0)$
- $\forall x. x + 0 = x$
- $\forall x, y. x + 1 = y + 1 \rightarrow x = y$
- $\forall x, y. x + (y + 1) = (x + y) + 1$
- For each formula  $\varphi(x, \bar{y})$  in the language  $\Sigma_E$ ,  
 $\forall \bar{y}. \varphi(0, \bar{y}) \wedge (\forall x. \varphi(x, \bar{y}) \rightarrow \varphi(x + 1, \bar{y})) \rightarrow \forall x. \varphi(x, \bar{y})$

(induction)

$T_N$  is decidable  
QFF  $T_N$  is decidable

## Signature $\Sigma_Z$ :

- Constants:  $\dots, -3, -2, -1, 0, 1, 2, 3, \dots$
- Unary function:  $\dots, -3*, -2*, 2*, 3*, \dots$   
(intended meaning  $2 * x$  is  $x + x$ ,  $-3 * x$  is  $-x - x - x$ )
- Binary functions:  $+$  and  $-$
- Predicate:  $=$  and  $>$

$T_Z$  is decidable

*QFF*  $T_Z$  is decidable

# Theory of Arrays

## Signature $\Sigma_A$ :

- Functions:
  - $read(-, -)$ 
    - written for simplicity as  $-[-]$
    - e.g.  $a[i]$
  - $write(-, -, -)$ 
    - e.g.  $write(a, v, i)$  denotes the array  $a'$  where  $a'[v] = i$  and all other entries are the same as  $a$
- Predicate:  $=$

## Axioms $A_A$ :

- Same as  $A_E$
- $\forall a, i, j. i = j \rightarrow a[i] = a[j]$
- $\forall a, v, i, j. i = j \rightarrow write(a, i, v)[j] = v$
- $\forall a, v, i, j. i \neq j \rightarrow write(a, i, v)[j] = a[j]$



# Theory of Arrays

## Signature $\Sigma_A$ :

- Functions:
  - $read(-, -)$ 
    - written for simplicity as  $-[-]$
    - e.g.  $a[i]$
  - $write(-, -, -)$ 
    - e.g.  $write(a, v, i)$  denotes the array  $a'$  where  $a'[v] = i$  and all other entries are the same as  $a$
- Predicate:  $=$

## Axioms $A_A$ :

- Same as  $A_E$
- $\forall a, i, j. i = j \rightarrow a[i] = a[j]$
- $\forall a, v, i, j. i = j \rightarrow write(a, i, v)[j] = v$
- $\forall a, v, i, j. i \neq j \rightarrow write(a, i, v)[j] = a[j]$

$T_A$  is undecidable  
QFF  $T_A$  is decidable

## How SMT solvers works?

---

There are two main approaches for SMT solvers:

- The eager approach
  - Tries to find ways of encoding an entire SMT problem into SAT.
  - There are a variety of techniques
  - For some theories, this works quite well.

There are two main approaches for SMT solvers:

- The eager approach
  - Tries to find ways of encoding an entire SMT problem into SAT.
  - There are a variety of techniques
  - For some theories, this works quite well.
- The lazy approach
  - Tries to combine SAT and theory reasoning.
  - The basis for most modern SMT solvers.

1. How to solve conjunctions of literals in a theory?
  - Use a Theory solver
2. How to combine a theory solver and a SAT solver to reason about arbitrary formulas?
  - The DPLL(T) framework
3. How to combine theory solvers for several theories?
  - The Nelson-Oppen method and its variants

- Given a theory  $T$ , a Theory solver for  $T$  takes as input a set (interpreted as an implicit conjunction)  $\varphi$  of literals and determines whether  $\varphi$  is  $T$ -satisfiable.
  - $\varphi$  is  $T$ -satisfiable if there is some model  $\mathcal{M}$  of  $T$  such that  $\varphi$  holds in  $\mathcal{M}$ .
- In order to integrate a Theory solver into a modern SMT solver, it is helpful if the Theory solver can do more than just check satisfiability.

# Characteristics of Theory solvers

Some desirable characteristics of Theory solvers include:

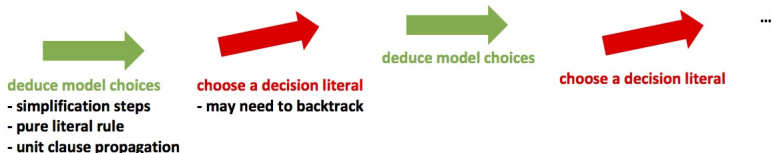
- **Incrementality** – easy to add new literals or backtrack to a previous state
- **Layered/Lazy** – able to detect simple inconsistencies quickly, able to detect difficult inconsistencies eventually
- **Equality Propagating** – if Theory solvers can detect when two terms are equivalent, this greatly simplifies theory combination
- **Model Generating** – when reporting  $T$ -satisfiable, the Theory solver also provides a concrete value for each variable or function symbol
- **Proof Generating** – when reporting  $T$ -unsatisfiable, the Theory solver also provides a checkable proof

# Propositional Abstraction

- An **atom** is a formula without propositional connectives or quantifiers
  - depending on the signature  $f(a) = b, m * n \leq 42$  could be atoms; 42 is not
  - a propositional atom is an uninterpreted constant symbol of sort Bool
- A (first-order) **literal** is an atom or its negation
- For a given signature  $\Sigma$ , we define a signature  $\Sigma^P$  containing only:
  - the **propositional  $\Sigma$ -atoms**
  - a **fresh propositional atom** for each non-propositional  $\Sigma$ -atom
- We then fix an injective mapping from the non-propositional  $\Sigma$ -atoms to the  $\Sigma^P$ -atoms.
- For a  $\Sigma$ -formula  $\varphi$ , the formula  $\varphi^P$  is the **propositional abstraction** of  $\varphi$ , given by replacing all non-propositional  $\Sigma$ -atoms in  $\varphi$  with their image under this mapping.
- An  $\Sigma$ -formula  $\varphi$  is **propositional unsatisfiable** if  $\varphi^P \models \perp$ .
- An  $\Sigma$ -formula  $\varphi$  **propositionally entails** an  $\Sigma$ -formula  $\psi$  if  $\varphi^P \models \psi^P$ .
  - Note that  $\varphi^P \models \psi^P$  implies  $\varphi \models \psi$ , but **not necessarily vice-versa**.



# Recall DPLL/CDCL Algorithms

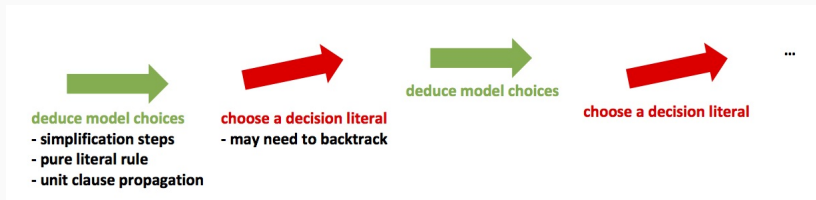


...until...

- conflict reached
  - backtrack - try flipping a decision literal
  - (if CDCL) learn new clause, back-jump
- model found
  - return the model

# Adapting DPLL to DPLL(T)

Run DPLL on the **propositional abstraction**  $\varphi^P$  of the  $T$ -input formula  $\varphi$



...until...

- **conflict reached:** backtrack/jump, learn clauses as usual
- **model found** (represented by a set  $\Gamma$  of literals)
  - It is not necessarily a  $T$ -model!
  - Ask theory solver: is  $\Gamma$   $T$ -satisfiable?
    - If yes, we are done.
    - If no, backtrack in the original search.
  - (CDCL) get a  $T$ -unsatisfiable subset for clause learning/back-jumping

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$
- Theory solver detects  $[1, \neg 2]$  **T-unsat**

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$
- Theory solver detects  $[1, \neg 2]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2]$

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$
- Theory solver detects  $[1, \neg 2]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2]$
- SAT solver returns model  $[1, 2, 3, \neg 4]$



# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$
- Theory solver detects  $[1, \neg 2]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2]$
- SAT solver returns model  $[1, 2, 3, \neg 4]$
- Theory solver detects  $[1, 3, \neg 4]$  **T-unsat**

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$
- Theory solver detects  $[1, \neg 2]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2]$
- SAT solver returns model  $[1, 2, 3, \neg 4]$
- Theory solver detects  $[1, 3, \neg 4]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2, \neg 1 \vee \neg 3 \vee 4]$

# Adapting DPLL to DPLL(T)

## Example

$$\underbrace{g(a) = c}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{\neg 2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{\neg 4}$$

- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4]$  (i.e.  $[[1], [-2, 3], [-4]]$ )
- SAT solver returns model  $[1, \neg 2, \neg 4]$
- Theory solver detects  $[1, \neg 2]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2]$
- SAT solver returns model  $[1, 2, 3, \neg 4]$
- Theory solver detects  $[1, 3, \neg 4]$  **T-unsat**
- Call SAT solver with input  $[1, \neg 2 \vee 3, \neg 4, \neg 1 \vee 2, \neg 1 \vee \neg 3 \vee 4]$
- SAT solver detects **unsat**

# Theory combination

- Given a theory  $T$ , a **Theory solver** for  $T$  takes as input a set (interpreted as an implicit conjunction)  $\varphi$  of literals and determines whether  $\varphi$  is  $T$ -satisfiable.
- We are often interested in using **two or more theories at the same time**.
- **Can we combine two theory solvers to get a theory solver for the combined theory?**

# Theory combination

- Given a theory  $T$ , a **Theory solver** for  $T$  takes as input a set (interpreted as an implicit conjunction)  $\varphi$  of literals and determines whether  $\varphi$  is  $T$ -satisfiable.
- We are often interested in using **two or more theories at the same time**.
- **Can we combine two theory solvers to get a theory solver for the combined theory?**

## Example

The following formula uses both  $T_E$  and  $T_Z$

$$\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

# The Nelson-Oppen Method

A very general method for combining theory solvers is the Nelson-Oppen method.

This method is applicable when:

1. The signatures  $\Sigma_i$  are disjoint.
2. The theories  $T_i$  are stably-infinite.
  - A  $\Sigma$ -theory  $T$  is stably-infinite if every  $T$ -satisfiable quantifier-free  $\Sigma$ -formula is satisfiable in an infinite model.
3. The formulas to be tested for satisfiability are conjunctions of quantifier-free literals.

Extensions exist that can relax each of these restrictions in some cases.

# The Nelson-Oppen Method

Some definitions:

- A member of  $\Sigma_i$  is an *i*-symbol.
- A term  $t$  is an *i*-term if it starts with an *i*-symbol.
- An atomic *i*-formula is
  - an application of an *i*-predicate,
  - an equation whose lhs is an *i*-term, or
  - an equation whose lhs is a variable and whose rhs is an *i*-term
- An *i*-literal is an atomic *i*-formula or the negation of one.
- An occurrence of a term  $t$  in either an *i*-term or an *i*-literal is *i*-alien if it is a  $j$ -term with  $i \neq j$  and all of its super-terms (if any) are *i*-terms.
- An expression is pure if it contains only variables and *i*-symbols for some  $i$ .

# Conversion to separate form

Given a conjunction of literals  $\varphi$ , we want to convert it into a **separate form**: a  $T$ -equisatisfiable conjunction of literals  $\varphi_1 \wedge \varphi_2 \wedge \dots \wedge \varphi_n$  where each  $\varphi_i$  is a  $\Sigma_i$ -formula.

We have the following **algorithm**:

1. Let  $\psi$  be **some literal** in  $\varphi$ .
2. If  $\psi$  is a **pure  $i$ -literal**, for some  $i$ , remove  $\psi$  from  $\varphi$  and add  $\psi$  to  $\varphi_i$ .  
If  $\varphi$  is **empty then stop**; otherwise **goto step 1**.
3. Otherwise,  $\psi$  is an  **$i$ -literal** for some  $i$ .  
Let  $t$  be a **term occurring  $i$ -alien** in  $\psi$ .  
Replace  $t$  in  $\varphi$  with a **new variable**  $z$  and add  $z = t$  to  $\varphi$ .  
**Goto step 1**.



## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := ?$
- $\varphi_Z := ?$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := ?$
- $\varphi_Z := 1 \leq x$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := ?$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := ?$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(2) \wedge y = 1$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := ?$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = \cancel{1 \leq x \wedge x \leq 2} \wedge \cancel{f(x) \neq f(y)} \wedge \cancel{f(x) \neq f(2)} \wedge y = 1$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(2) \wedge y = 1$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(z) \wedge y = 1 \wedge z = 2$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$



## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(z) \wedge y = 1 \wedge z = 2$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(z) \wedge y = 1 \wedge z = 2$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(z) \wedge y = 1 \wedge z = 2$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi = 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(y) \wedge f(x) \neq f(z) \wedge y = 1 \wedge z = 2$$

We convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

# The Nelson-Oppen Method

- As each  $\varphi_i$  is a  $\Sigma_i$ -formula, we can run a Theory solver  $Sat_i$  for each  $\varphi_i$ .
- If any  $Sat_i$  reports that  $\varphi_i$  is unsatisfiable, then  $\varphi$  is unsatisfiable.
- The converse is not true in general!
- We need a way for the decision procedures to communicate with each other about shared variables.
- If  $S$  is a set of terms and  $\sim$  is an equivalence relation on  $S$ , then the arrangement of  $S$  induced by  $\sim$  is

$$Ar_{\sim} = \{x = y \mid x \sim y\} \cup \{x \neq y \mid x \not\sim y\}$$

# The Nelson-Oppen Method

Suppose that  $T_1$  and  $T_2$  are theories with disjoint signatures  $\Sigma_1$  and  $\Sigma_2$ .

Let  $T = \bigcup T_i$  and  $\Sigma = \bigcup \Sigma_i$ .

Given a  $\Sigma$ -formula  $\varphi$  and decision procedures  $Sat_1$  and  $Sat_2$  for  $T_1$  and  $T_2$ , we wish to determine if  $\varphi$  is  $T$ -satisfiable.

The non-deterministic Nelson-Oppen algorithm:

1. Convert  $\varphi$  to its **separate form**  $\varphi_1 \wedge \varphi_2$ .
2. Let  $S$  be the **set of variables shared** between  $\varphi_1$  and  $\varphi_2$ .  
Guess an **equivalence relation**  $\sim$  on  $S$ .
3. **Run**  $Sat_1$  on  $\varphi_1 \cup Ar_\sim$ .
4. **Run**  $Sat_2$  on  $\varphi_2 \cup Ar_\sim$ .

# The Nelson-Oppen Method

Suppose that  $T_1$  and  $T_2$  are theories with disjoint signatures  $\Sigma_1$  and  $\Sigma_2$ .

Let  $T = \bigcup T_i$  and  $\Sigma = \bigcup \Sigma_i$ .

Given a  $\Sigma$ -formula  $\varphi$  and decision procedures  $Sat_1$  and  $Sat_2$  for  $T_1$  and  $T_2$ , we wish to determine if  $\varphi$  is  $T$ -satisfiable.

The non-deterministic Nelson-Oppen algorithm:

1. Convert  $\varphi$  to its **separate form**  $\varphi_1 \wedge \varphi_2$ .
2. Let  $S$  be the **set of variables shared** between  $\varphi_1$  and  $\varphi_2$ .  
Guess an **equivalence relation**  $\sim$  on  $S$ .
3. **Run**  $Sat_1$  on  $\varphi_1 \cup Ar_\sim$ .
4. **Run**  $Sat_2$  on  $\varphi_2 \cup Ar_\sim$ .

If there exists an equivalence relation  $\sim$  such that both  $Sat_1$  and  $Sat_2$  succeed, then  $\varphi$  is  **$T$ -satisfiable**.

If no such equivalence relation exists, then  $\varphi$  is  **$T$ -unsatisfiable**.

# The Nelson-Oppen Method

Suppose that  $T_1$  and  $T_2$  are theories with disjoint signatures  $\Sigma_1$  and  $\Sigma_2$ .

Let  $T = \bigcup T_i$  and  $\Sigma = \bigcup \Sigma_i$ .

Given a  $\Sigma$ -formula  $\varphi$  and decision procedures  $Sat_1$  and  $Sat_2$  for  $T_1$  and  $T_2$ , we wish to determine if  $\varphi$  is  $T$ -satisfiable.

The non-deterministic Nelson-Oppen algorithm:

1. Convert  $\varphi$  to its **separate form**  $\varphi_1 \wedge \varphi_2$ .
2. Let  $S$  be the **set of variables shared** between  $\varphi_1$  and  $\varphi_2$ .  
Guess an **equivalence relation**  $\sim$  on  $S$ .
3. **Run**  $Sat_1$  on  $\varphi_1 \cup Ar_\sim$ .
4. **Run**  $Sat_2$  on  $\varphi_2 \cup Ar_\sim$ .

If there exists an equivalence relation  $\sim$  such that both  $Sat_1$  and  $Sat_2$  succeed, then  $\varphi$  is  **$T$ -satisfiable**.

If no such equivalence relation exists, then  $\varphi$  is  **$T$ -unsatisfiable**.

The generalization to more than two theories is straightforward.



# The Nelson-Oppen Method

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

We first convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

# The Nelson-Oppen Method

## Example

Consider the following  $\Sigma_E \cup \Sigma_Z$ :

$$\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$$

We first convert  $\varphi$  to a separate form:

- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

The shared variables are  $\{x, y, z\}$ .

There are 5 possible arrangements based on equivalence classes of  $x, y$ , and  $z$  (see *Bell number*).

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

1.  $\{x = y, x = z, y = z\}$
2.  $\{x = y, x \neq z, y \neq z\}$
3.  $\{x \neq y, x = z, y \neq z\}$
4.  $\{x \neq y, x \neq z, y = z\}$
5.  $\{x \neq y, x \neq z, y \neq z\}$

# The Nelson-Oppen Method

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

1.  $\{x = y, x = z, y = z\}$

inconsistent with  $T_E$

2.  $\{x = y, x \neq z, y \neq z\}$

3.  $\{x \neq y, x = z, y \neq z\}$

4.  $\{x \neq y, x \neq z, y = z\}$

5.  $\{x \neq y, x \neq z, y \neq z\}$

# The Nelson-Oppen Method

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

1.  $\{x = y, x = z, y = z\}$  inconsistent with  $T_E$
2.  $\{x = y, x \neq z, y \neq z\}$  inconsistent with  $T_E$
3.  $\{x \neq y, x = z, y \neq z\}$
4.  $\{x \neq y, x \neq z, y = z\}$
5.  $\{x \neq y, x \neq z, y \neq z\}$

# The Nelson-Oppen Method

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

1.  $\{x = y, x = z, y = z\}$  inconsistent with  $T_E$
2.  $\{x = y, x \neq z, y \neq z\}$  inconsistent with  $T_E$
3.  $\{x \neq y, x = z, y \neq z\}$  inconsistent with  $T_E$
4.  $\{x \neq y, x \neq z, y = z\}$
5.  $\{x \neq y, x \neq z, y \neq z\}$

# The Nelson-Oppen Method

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

- |                                       |                         |
|---------------------------------------|-------------------------|
| 1. $\{x = y, x = z, y = z\}$          | inconsistent with $T_E$ |
| 2. $\{x = y, x \neq z, y \neq z\}$    | inconsistent with $T_E$ |
| 3. $\{x \neq y, x = z, y \neq z\}$    | inconsistent with $T_E$ |
| 4. $\{x \neq y, x \neq z, y = z\}$    | inconsistent with $T_Z$ |
| 5. $\{x \neq y, x \neq z, y \neq z\}$ |                         |

# The Nelson-Oppen Method

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

- |                                       |                         |
|---------------------------------------|-------------------------|
| 1. $\{x = y, x = z, y = z\}$          | inconsistent with $T_E$ |
| 2. $\{x = y, x \neq z, y \neq z\}$    | inconsistent with $T_E$ |
| 3. $\{x \neq y, x = z, y \neq z\}$    | inconsistent with $T_E$ |
| 4. $\{x \neq y, x \neq z, y = z\}$    | inconsistent with $T_Z$ |
| 5. $\{x \neq y, x \neq z, y \neq z\}$ | inconsistent with $T_Z$ |



# The Nelson-Oppen Method

## Example

- $\varphi := 1 \leq x \wedge x \leq 2 \wedge f(x) \neq f(1) \wedge f(x) \neq f(2)$
- $\varphi_E := f(x) \neq f(y) \wedge f(x) \neq f(z)$
- $\varphi_Z := 1 \leq x \wedge x \leq 2 \wedge y = 1 \wedge z = 2$

- |                                       |                         |
|---------------------------------------|-------------------------|
| 1. $\{x = y, x = z, y = z\}$          | inconsistent with $T_E$ |
| 2. $\{x = y, x \neq z, y \neq z\}$    | inconsistent with $T_E$ |
| 3. $\{x \neq y, x = z, y \neq z\}$    | inconsistent with $T_E$ |
| 4. $\{x \neq y, x \neq z, y = z\}$    | inconsistent with $T_Z$ |
| 5. $\{x \neq y, x \neq z, y \neq z\}$ | inconsistent with $T_Z$ |

Conclusion:  $\varphi$  is  $T_E \cup T_Z$ -unsatisfiable!

Recall the ingredients:

- Theory solvers for different theories
- Combine a Theory solver and a SAT solver
- Combine Theory solvers for different theories

- Lecture Notes on "Program Analysis", ETH Zurich, Martin Vechev.
- Lecture Notes on "Techniques for Program Analysis and Verification", Stanford, Clark Barrett.
- Lecture Notes on "Program verification", ETH Zurich, Alexander Summers.
- Lecture Notes on "Computer-Aided Reasoning for Software Engineering", University of Washington, Emina Torlak.