

# Formal Methods in Software Engineering

**Satisfiability Modulo Theories** — Spring 2025

Konstantin Chukharev

# §1 First-Order Theories

## Motivation

Consider the signature  $\Sigma = \langle \Sigma^S, \Sigma^F \rangle$  for a fragment of number theory:

- $\Sigma^S = \{\text{Nat}\}$ ,  $\Sigma^F = \{0, 1, +, <\}$
- $\text{rank}(0) = \text{rank}(1) = \langle \text{Nat} \rangle$
- $\text{rank}(+) = \langle \text{Nat}, \text{Nat}, \text{Nat} \rangle$
- $\text{rank}(<) = \langle \text{Nat}, \text{Nat}, \text{Bool} \rangle$

1. Consider the  $\Sigma$ -sentence:  $\forall x : \text{Nat}. \neg(x < x)$

- Is it *valid*, that is, true under *all* interpretations?
- No, e.g., if we interpret  $<$  as *equals* or *divides*.

2. Consider the  $\Sigma$ -sentence:  $\neg \exists x : \text{Nat}. (x < 0)$

- Is it *valid*?
- No, e.g., if we interpret  $\text{Nat}$  as the set of *all* integers.

3. Consider the  $\Sigma$ -sentence:  $\forall x : \text{Nat}. \forall y : \text{Nat}. \forall z : \text{Nat}. (x < y) \wedge (y < z) \rightarrow (x < z)$

- Is it *valid*?
- No, e.g., if we interpret  $<$  as the *successor* relation.

## Motivation [2]

In practice, we often *do not care* about satisfiability or validity in *general*, but rather with respect to a *limited class* of interpretations.

### A practical reason:

- When reasoning in a particular application domain, we typically have *specific* data types/structures in mind (e.g., integers, strings, lists, arrays, finite sets, ...).
- More generally, we are typically *not* interested in *arbitrary* interpretations, but rather in *specific* ones.

*Theories* formalize this domain-specific reasoning: we talk about satisfiability and validity *with respect to a theory* or “*modulo a theory*”.

### A computational reason:

- The validity problem for FOL is *undecidable* in general.
- However, the validity problem for many *restricted* theories, is *decidable*.

# First-Order Theories

Hereinafter, we assume that we have an infinite set of variables  $X$ .

**Definition 1** (Theory): A first-order *theory*  $\mathcal{T}$  is a pair<sup>1</sup>  $\langle \Sigma, M \rangle$ , where

- $\Sigma = \langle \Sigma^S, \Sigma^F \rangle$  is a first-order signature,
- $M$  is a class<sup>2</sup> of  $\Sigma$ -interpretations over  $X$  that is *closed under variable re-assignment*.

**Definition 2:**  $M$  is *closed under variable re-assignment* if every  $\Sigma$ -interpretation that differs from one in  $M$  in the way it interprets the variables in  $X$  is also in  $M$ .

A theory limits the interpretations of  $\Sigma$ -formulas to those from  $M$ .

---

<sup>1</sup>Here, we use **bold** style for  $M$  to denote that it is *not a single* model, but a *collection* of them.

<sup>2</sup>*Class* is a generalization of a set.

## Theory Examples

**Example:** Theory of Real Arithmetic  $\mathcal{T}_{\text{RA}} = \langle \Sigma_{\text{RA}}, \mathcal{M}_{\text{RA}} \rangle$ :

- $\Sigma_{\text{RA}}^S = \{\text{Real}\}$
- $\Sigma_{\text{RA}}^F = \{+, -, \times, \leq\} \cup \{q \mid q \text{ is a decimal numeral}\}$
- All  $\mathcal{I} \in \mathcal{M}_{\text{RA}}$  interpret `Real` as the set of *real numbers*  $\mathbb{R}$ , each  $q$  as the *decimal number* that it denotes, and the function symbols in the usual way.

**Example:** Theory of Ternary Strings  $\mathcal{T}_{\text{TS}} = \langle \Sigma_{\text{TS}}, \mathcal{M}_{\text{TS}} \rangle$ :

- $\Sigma_{\text{TS}}^S = \{\text{String}\}$
- $\Sigma_{\text{TS}}^F = \{\cdot, <\} \cup \{a, b, c\}$
- All  $\mathcal{I} \in \mathcal{M}_{\text{TS}}$  interpret `String` as the set  $\{a, b, c\}^*$  of all finite strings over the characters  $\{a, b, c\}$ , symbol  $\cdot$  as string concatenation (e.g.,  $a \cdot b = ab$ ), and  $<$  as lexicographic order.

## $\mathcal{T}$ -interpretations

**Definition 3** (Reduct): Let  $\Sigma$  and  $\Omega$  be two signatures over variables  $X$ , where  $\Omega \supseteq \Sigma$ , that is,  $\Omega^S \supseteq \Sigma^S$  and  $\Omega^F \supseteq \Sigma^F$ .

Let  $\mathcal{J}$  be an  $\Omega$ -interpretation over  $X$ .

The *reduct*  $\mathcal{J}^\Sigma$  of  $\mathcal{J}$  to  $\Sigma$  is a  $\Sigma$ -interpretation obtained from  $\mathcal{J}$  by restricting it to the symbols in  $\Sigma$ .

**Definition 4** ( $\mathcal{T}$ -interpretation): Given a theory  $\mathcal{T} = \langle \Sigma, \mathcal{M} \rangle$ , a  *$\mathcal{T}$ -interpretation* is any  $\Omega$ -interpretation  $\mathcal{J}$  for some signature  $\Omega \supseteq \Sigma$  such that  $\mathcal{J}^\Sigma \in \mathcal{M}$ .

**Note:** This definition allows us to consider the satisfiability in a theory  $\mathcal{T} = \langle \Sigma, \mathcal{M} \rangle$  of formulas that contain sorts or function symbols not in  $\Sigma$ . These symbols are usually called *uninterpreted* (in  $\mathcal{T}$ ).

## $\mathcal{T}$ -interpretations [2]

**Example:** Consider again the theory of real arithmetic  $\mathcal{T}_{\text{RA}} = \langle \Sigma_{\text{RA}}, \mathcal{M}_{\text{RA}} \rangle$ .

All  $\mathcal{J} \in \mathcal{M}_{\text{RA}}$  interpret `Real` as  $\mathbb{R}$  and function symbols as usual.

Which of the following interpretations are  $\mathcal{T}_{\text{RA}}$ -interpretations?

1.  $\text{Real}^{\mathcal{J}_1} = \mathbb{Q}$ , symbols in  $\Sigma_{\text{RA}}^F$  interpreted as usual. ✗
2.  $\text{Real}^{\mathcal{J}_2} = \mathbb{R}$ , symbols in  $\Sigma_{\text{RA}}^F$  interpreted as usual, and  $\text{String}^{\mathcal{J}_2} = \{0.5, 1.3\}$ . ✓
3.  $\text{Real}^{\mathcal{J}_3} = \mathbb{R}$ , symbols in  $\Sigma_{\text{RA}}^F$  interpreted as usual, and  $\log^{\mathcal{J}_3}$  is the successor function. ✓



## $\mathcal{T}$ -satisfiability, $\mathcal{T}$ -entailment, $\mathcal{T}$ -validity

**Definition 5** ( $\mathcal{T}$ -satisfiability): A  $\Sigma$ -formula  $\alpha$  is *satisfiable in  $\mathcal{T}$* , or  *$\mathcal{T}$ -satisfiable*, if it is satisfied by *some*  $\mathcal{T}$ -interpretation  $\mathcal{I}$ .

**Definition 6** ( $\mathcal{T}$ -entailment): A set  $\Gamma$  of formulas  *$\mathcal{T}$ -entails* a formula  $\alpha$ , if every  $\mathcal{T}$ -interpretation that satisfies all formulas in  $\Gamma$  also satisfies  $\alpha$ .

**Definition 7** ( $\mathcal{T}$ -validity): A formula  $\alpha$  is  *$\mathcal{T}$ -valid*, if it is satisfied by *all*  $\mathcal{T}$ -interpretations.

**Note:** A formula  $\alpha$  is  *$\mathcal{T}$ -valid* iff  $\emptyset \models \alpha$ .

**Example:** Which of the following  $\Sigma_{\text{RA}}$ -formulas is satisfiable or valid in  $\mathcal{T}_{\text{RA}}$ ?

1.  $(x_0 + x_1 \leq 0.5) \wedge (x_0 - x_1 \leq 2)$
2.  $\forall x_0. (x_0 + x_1 \leq 1.7) \rightarrow (x_1 \leq 1.7 - x_0)$
3.  $\forall x_0. \forall x_1. (x_0 + x_1 \leq 1)$

*satisfiable, falsifiable*

*satisfiable, valid*

*unsatisfiable, falsifiable*

## FOL vs Theory

For every signature  $\Sigma$ , entailment and validity in “pure” FOL can be seen as entailment and validity in the theory  $\mathcal{T}_{\text{FOL}} = \langle \Sigma, M_{\text{FOL}} \rangle$  where  $M_{\text{FOL}}$  is the class of *all possible*  $\Sigma$ -interpretations.

- Pure first-order logic = reasoning over *all* possible interpretations.
- Reasoning modulo a theory = *restricting* interpretations with some domain constraints.
- Theories make automated reasoning *feasible* in many domains.

## Axiomatization

**Definition 8** (Axiomatic theory): A first-order *axiomatic theory*  $\mathcal{T}$  is defined by a signature  $\Sigma$  and a set  $\mathcal{A}$  of  $\Sigma$ -sentences, or *axioms*.

**Definition 9** ( $\mathcal{T}$ -validity in axiomatic theory): An  $\Omega$ -formula  $\alpha$  is *valid* in an axiomatic theory  $\mathcal{T}$  if it is entailed by the axioms of  $\mathcal{T}$ , that is, every  $\Omega$ -interpretation  $\mathcal{I}$  that satisfies  $\mathcal{A}$  also satisfies  $\alpha$ .

**Note:** Axiomatic theories are a *special case* of the general definition (via  $\mathbf{M}$ ) of theories.

- Given an axiomatic theory  $\mathcal{T}'$  defined by  $\Sigma$  and  $\mathcal{A}$ , we can define a theory  $\mathcal{T} = \langle \Sigma, \mathbf{M} \rangle$  where  $\mathbf{M}$  is the class of all  $\Sigma$ -interpretations that satisfy all axioms in  $\mathcal{A}$ .
- It is not hard to show that a formula  $\alpha$  is valid in  $\mathcal{T}$  *iff* it is valid in  $\mathcal{T}'$ .

**Note:** Not all theories are first-order axiomatizable.

## Non-Axiomatizable Theories

**Note:** Not all theories are first-order axiomatizable.

**Example:** Consider the theory  $\mathcal{T}_{\text{Nat}}$  of the natural numbers, with signature  $\Sigma$  with  $\Sigma^S = \{\text{Nat}\}$ ,  $\Sigma^F = \{0, S, +, <\}$ , and  $M = \{\mathcal{I}\}$  where  $\text{Nat}^{\mathcal{I}} = \mathbb{N}$  and  $\Sigma^F$  is interpreted as usual.

*Any set of axioms* (for example, *Peano axioms*) for this theory is satisfied by *non-standard models*, e.g., interpretations  $\mathcal{I}'$  where  $\text{Nat}^{\mathcal{I}'}$  includes other chains of elements besides the natural numbers.

However, these models *falsify* formulas that are *valid* in  $\mathcal{T}_{\text{Nat}}$ .

For example, “every number is either zero or a successor”:  $\forall x. (x \doteq 0) \vee \exists y. (x \doteq S(y))$ .

- **true** in the *standard* model, i.e.  $\text{Nat}^{\mathcal{I}} = \mathbb{N} = \{0, 1 := S(0), 2 := S(1), \dots\}$ .
- **false** in *non-standard* models, e.g.,  $\text{Nat}^{\mathcal{I}'} = \{0, 1, 2, \dots\} \cup \{\omega, \omega + 1, \dots\}$ 
  - ▶ Intuitively,  $\omega$  is “an infinite element”.
  - ▶ The successor function still applies:  $S(\omega) = \omega + 1$ ,  $S(\omega + 1) = \omega + 2$ , etc.
  - ▶ Even the addition and multiplication still works:  $\omega + 3 = S(S(S(\omega)))$ ,  $\omega \times 2 = \omega + \omega$ .
  - ▶ But  $\omega$  is larger than all standard numbers:  $\omega > 0, \omega > 1, \dots$

## Peano Arithmetic

**Definition 10:** *Peano arithmetic*  $\mathcal{T}_{\text{PA}}$ , or *first-order arithmetic*, is the axiomatic theory of natural numbers with signature  $\Sigma_{\text{PA}}^F = \{0, S, +, \times, =\}$  and *Peano axioms*:

1.  $\forall x. (S(x) \neq 0)$  (zero)
2.  $\forall x. \forall y. (S(x) = S(y)) \rightarrow (x = y)$  (successor)
3.  $F[0] \wedge (\forall x. F[x] \rightarrow F[x + 1]) \rightarrow \forall x. F[x]$  (induction)
4.  $\forall x. (x + 0 = x)$  (plus zero)
5.  $\forall x. \forall y. (x + S(y) = S(x + y))$  (plus successor)
6.  $\forall x. (x \times 0 = 0)$  (times zero)
7.  $\forall x. \forall y. (x \times S(y) = (x \times y) + x)$  (times successor)

Axiom (induction) is the *induction axiom schema*. It stands for an *infinite* set of axioms, one for each  $\Sigma_{\text{PA}}$ -formula  $F$  with one free variable. The notation  $F[\alpha]$  means that  $F$  contains  $\alpha$  as a sub-formula.

The *intended interpretation* (*standard models*) of  $\mathcal{T}_{\text{PA}}$  have the domain  $\mathbb{N}$  and the usual interpretations of the function symbols as  $0_{\mathbb{N}}$ ,  $S_{\mathbb{N}}$ ,  $+_{\mathbb{N}}$ , and  $\times_{\mathbb{N}}$ .

## Presburger Arithmetic

**Note:** Satisfiability and validity in  $\mathcal{T}_{\mathcal{P}_A}$  is undecidable. Therefore, we need a more restricted theory of arithmetic that does not include multiplication.

**Definition 11:** *Presburger arithmetic*  $\mathcal{T}_{\mathbb{N}}$  is the axiomatic theory of natural numbers with signature  $\Sigma_{\mathbb{N}}^F = \{0, S, +, =\}$  and the *subset* of *Peano axioms*:

1.  $\forall x. (S(x) \neq 0)$  (zero)
2.  $\forall x. \forall y. (S(x) = S(y)) \rightarrow (x = y)$  (successor)
3.  $F[0] \wedge (\forall x. F[x] \rightarrow F[x + 1]) \rightarrow \forall x. F[x]$  (induction)
4.  $\forall x. (x + 0 = x)$  (plus zero)
5.  $\forall x. \forall y. (x + S(y) = S(x + y))$  (plus successor)

**Note:** Presburger arithmetic is decidable.

## Completeness of Theories

**Definition 12:** A  $\Sigma$ -theory  $\mathcal{T}$  is *complete* if for every  $\Sigma$ -sentence  $\alpha$ , either  $\alpha$  or  $\neg\alpha$  is valid in  $\mathcal{T}$ .

**Note:** In a complete  $\Sigma$ -theory, every  $\Sigma$ -sentence is either valid or unsatisfiable.

**Example:** Any theory  $\mathcal{T} = \langle \Sigma, M \rangle$  where all interpretations in  $M$  only differ in how they interpret the variables (e.g.,  $\mathcal{T}_{\text{RA}}$ ) is *complete*.

**Example:** The axiomatic (mono-sorted) theory of *monoids* with  $\Sigma^F = \{ \cdot, \varepsilon \}$  and axioms

$$\forall x. \forall y. \forall z. (x \cdot y) \cdot z \doteq x \cdot (y \cdot z) \quad \forall x. (x \cdot \varepsilon \doteq x) \quad \forall x. (\varepsilon \cdot x \doteq x)$$

is *incomplete*. For example, the sentence  $\forall x. \forall y. (x \cdot y \doteq y \cdot x)$  is *true* in some monoids (e.g. the addition of integers *is* commutative) but *false* in others (e.g. the concatenation of strings *is not* commutative).

## Completeness of Theories [2]

**Example:** The axiomatic (mono-sorted) theory of *dense linear orders without endpoints* with  $\Sigma^F = \{<\}$  and the following axioms is *complete*.

$$\forall x. \forall y. (x < y) \rightarrow \exists z. ((x < z) \wedge (z < y)) \quad (\text{dense})$$

$$\forall x. \forall y. ((x < y) \vee (y < x) \vee (x = y)) \quad (\text{linear})$$

$$\forall x. \neg(x < x) \quad \forall x. \forall y. \forall z. ((x < y) \wedge (y < z) \rightarrow (x < z)) \quad (\text{orders})$$

$$\forall x. \exists y. (y < x) \quad \forall x. \exists y. (x < y) \quad (\text{without endpoints})$$



## Decidability

Recall that a set  $A$  is *decidable* if there exists a *terminating* procedure that, given an input element  $a$ , returns (after *finite* time) either “yes” if  $a \in A$  or “no” if  $a \notin A$ .

**Definition 13:** A theory  $\mathcal{T} = \langle \Sigma, M \rangle$  is *decidable* if the set of all  $\mathcal{T}$ -valid  $\Sigma$ -formulas is decidable.

**Definition 14:** A *fragment* of  $\mathcal{T}$  is a *syntactically-restricted subset* of  $\mathcal{T}$ -valid  $\Sigma$ -formulas.

**Example:** The *quantifier-free* fragment of  $\mathcal{T}$  is the set of all  $\mathcal{T}$ -valid  $\Sigma$ -formulas *without quantifiers*.

**Example:** The *linear* fragment of  $\mathcal{T}_{\text{RA}}$  is the set of all  $\mathcal{T}$ -valid  $\Sigma_{\text{RA}}$ -formulas *without multiplication* ( $\times$ ).

## Axiomatizability

**Definition 15:** A theory  $\mathcal{T} = \langle \Sigma, M \rangle$  is *recursively axiomatizable* if  $M$  is the class of all interpretations satisfying a *decidable set* of first-order axioms  $\mathcal{A}$ .

**Theorem 1** (Lemma): Every recursively axiomatizable theory  $\mathcal{T}$  admits a procedure  $E_{\mathcal{T}}$  that *enumerates* all  $\mathcal{T}$ -valid formulas.

**Theorem 2:** For every *complete* and *recursively axiomatizable* theory  $\mathcal{T}$ , validity in  $\mathcal{T}$  is decidable.

**Proof:** Given a formula  $\alpha$ , use  $E_{\mathcal{T}}$  to enumerate all valid formulas. Since  $\mathcal{T}$  is complete, either  $\alpha$  or  $\neg\alpha$  will eventually (after *finite* time) be produced by  $E_{\mathcal{T}}$ . □

## **§2 Introduction to SMT**

## Common Theories in SMT

Satisfiability Modulo Theories (SMT) traditionally focuses on theories with *decidable quantifier-free fragments*.

SMT is concerned with (un)satisfiability, but recall that a formula  $\alpha$  is  *$\mathcal{T}$ -valid* iff  $\neg\alpha$  is  *$\mathcal{T}$ -unsatisfiable*.

Checking the (un)satisfiability of quantifier-free formulas in main background theories *efficiently* has a large number of applications in:

- hardware and software verification
- model checking
- symbolic execution
- compiler validation
- type checking
- planning and scheduling
- software synthesis
- cyber-security
- verifiable machine learning
- analysis of biological systems

Further, we are going to study:

- A few of those *theories* and their *decision procedures*.
- *Proof systems* to reason *modulo theories* automatically.

## From Quantifier-Free Formulas to Conjunctions of Literals

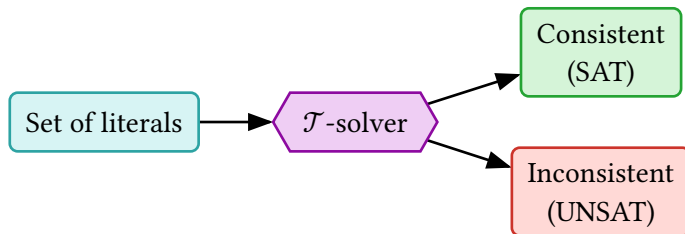
**Theorem 3:** The satisfiability of *quantifier-free* formulas in a theory  $\mathcal{T}$  is *decidable* iff the satisfiability in  $\mathcal{T}$  of *conjunctions of literals* is decidable.

Here, *literal* is an atom or its negation. For example:  $(a \doteq b)$ ,  $\neg(a + 1 < b)$ ,  $(f(b) \doteq g(f(a)))$ .

**Proof:** A quantifier-free formula can be transformed into disjunctive normal form (DNF), and its satisfiability reduces to checking satisfiability of conjunctions of literals. Conversely, a conjunction of literals is a special case of a quantifier-free formula. Thus, the two satisfiability problems are equivalent.  $\square$

## Theory Solvers

**Definition 16** ( $\mathcal{T}$ -solver): A *theory solver*, or  $\mathcal{T}$ -*solver*, is a specialized decision procedure for the satisfiability of conjunctions of literals in a theory  $\mathcal{T}$ .



# Theory of Uninterpreted Functions

**Definition 17:** Given a signature  $\Sigma$ , the most general theory consists of the class of *all*  $\Sigma$ -interpretations. In fact, this is a *family* of theories parameterized by the signature  $\Sigma$ .

It is known as the theory of *equality with uninterpreted functions*  $\mathcal{T}_{\text{EUF}}$ , or the *empty theory*, since it contains no *sentences*.

**Example:**  $(a \doteq b) \wedge (f(a) \doteq b) \wedge \neg(g(a) \doteq g(f(a)))$  Is this formula satisfiable in  $\mathcal{T}_{\text{EUF}}$ ?

Both validity and satisfiability are undecidable in  $\mathcal{T}_{\text{EUF}}$ .

- Validity in  $\mathcal{T}_{\text{EUF}}$  is *semi-decidable* — this is just a validity in FOL.
- Since a formula  $\alpha$  is  $\mathcal{T}$ -satisfiable iff  $\neg\alpha$  is not  $\mathcal{T}$ -valid,  $\mathcal{T}_{\text{EUF}}$ -satisfiability is *co-recognizable*.

However, the satisfiability of *conjunctions of  $\mathcal{T}_{\text{EUF}}$ -literals* is *decidable*, in polynomial time, using the *congruence closure* algorithm.

# Theory of Real Arithmetic

**Definition 18:** The theory of *real arithmetic*  $\mathcal{T}_{\text{RA}}$  is a theory of inequalities over the real numbers.

- $\Sigma^S = \{\text{Real}\}$
- $\Sigma^F = \{+, -, \times, <\} \cup \{q \mid q \text{ is a decimal numeral}\}$
- $\mathcal{M}$  is the class of interpretations that interpret  $\text{Real}$  as the set of *real numbers*  $\mathbb{R}$ , and the function symbols in the usual way.

Satisfiability in the full  $\mathcal{T}_{\text{RA}}$  is *decidable* (in worst-case doubly-exponential time).

Restricted fragments of  $\mathcal{T}_{\text{RA}}$  can be decided more efficiently.

**Example:** Quantifier-free linear real arithmetic (QF\_LRA) is the theory of *linear* inequalities over the reals, where  $\times$  can only be used in the form of *multiplication by constants* (decimal numerals).

The satisfiability of conjunctions of literals in QF\_LRA is *decidable* in *polynomial time*.



# Theory of Integer Arithmetic

**Definition 19:** The theory of *integer arithmetic*  $\mathcal{T}_{\text{IA}}$  is a theory of inequalities over the integers.

- $\Sigma^S = \{\text{Int}\}$
- $\Sigma^F = \{+, -, \times, <\} \cup \{n \mid n \text{ is an integer numeral}\}$
- $\mathcal{M}$  is the class of interpretations that interpret Int as the set of *integers*  $\mathbb{Z}$ , and the function symbols in the usual way.

Satisfiability in  $\mathcal{T}_{\text{IA}}$  is *not even semi-decidable*!

Satisfiability of quantifier-free  $\Sigma$ -formulas in  $\mathcal{T}_{\text{IA}}$  is *undecidable* as well.

*Linear integer arithmetic* (LIA, also known as *Presburger arithmetic*) is decidable, but not efficiently (in worst-case triply-exponential time). Its quantifier-free fragment (QF\_LIA) is NP-complete.

# Theory of Arrays with Extensionality

**Definition 20:** The theory of *arrays*  $\mathcal{T}_{AX}$  is useful for modelling RAM or array data structures.

- $\Sigma^S = \{A, I, E\}$  (arrays, indices, elements)
- $\Sigma^F = \{\text{read}, \text{write}\}$ , where  $\text{rank}(\text{read}) = \langle A, I, E \rangle$  and  $\text{rank}(\text{write}) = \langle A, I, E, A \rangle$

Let  $a$  be a variable of sort A, variable  $i$  of sort I, and variable  $v$  of sort E.

- $\text{read}(a, i)$  denotes the value stored in array  $a$  at index  $i$ .
- $\text{write}(a, i, v)$  denotes the array that stores value  $v$  at index  $i$  and is otherwise identical to  $a$ .

**Example:**  $\text{read}(\text{write}(a, i, v), i) \doteq_E v$

- Is this formula *intuitively* valid/satisfiable/unsatisfiable in  $\mathcal{T}_A$ ?

**Example:**  $\forall i. (\text{read}(a, i) \doteq_E \text{read}(a', i)) \rightarrow (a \doteq_A a')$

- Is this formula *intuitively* valid/satisfiable/unsatisfiable in  $\mathcal{T}_A$ ?

## Theory of Arrays with Extensionality [2]

**Definition 21:** The theory of arrays  $\mathcal{T}_{\text{AX}} = \langle \Sigma, M \rangle$  is finitely axiomatizable.

$M$  is the class of interpretations that satisfy the following axioms:

1.  $\forall a. \forall i. \forall v. (\text{read}(\text{write}(a, i, v), i) \doteq_{\text{E}} v)$
2.  $\forall a. \forall i. \forall j. \forall v. \neg(i \doteq_{\text{I}} j) \rightarrow (\text{read}(\text{write}(a, i, v), j) \doteq_{\text{E}} \text{read}(a, j))$
3.  $\forall a. \forall b. (\forall i. (\text{read}(a, i) \doteq_{\text{E}} \text{read}(b, i))) \rightarrow (a \doteq_{\text{A}} b)$

**Note:** The last axiom is called *extensionality* axiom. It states that two arrays are equal if they have the same values at all indices. It can be omitted to obtain a theory of arrays *without extensionality*  $\mathcal{T}_{\text{A}}$ .

Validity and satisfiability in  $\mathcal{T}_{\text{AX}}$  is *undecidable*.

There are several *decidable fragments* of  $\mathcal{T}_{\text{A}}$ .

# Survey of Decidability and Complexity

Theory	Description	Full	QF	Full complexity	QFC complexity
PL	Propositional Logic	—	yes	NP-complete	$\Theta(n)$
$\mathcal{T}_{\text{EUF}}$	Equality	no	yes	undecidable	$\mathcal{O}(n \log n)$
$\mathcal{T}_{\text{PA}}$	Peano Arithmetic	no	no	undecidable	undecidable
$\mathcal{T}_{\mathbb{N}}$	Presburger Arithmetic	yes	yes	$\Omega(2^{2^n}), \mathcal{O}(2^{2^{kn}})$	NP-complete
$\mathcal{T}_{\mathbb{Z}}$	Linear Integers (LIA)	yes	yes	$\Omega(2^{2^n}), \mathcal{O}(2^{2^{kn}})$	NP-complete
$\mathcal{T}_{\mathbb{R}}$	Reals	yes	yes	$\mathcal{O}(2^{2^{kn}})$	$\mathcal{O}(2^{2^{kn}})$
$\mathcal{T}_{\mathbb{Q}}$	Linear Rationals	yes	yes	$\Omega(2^n), \mathcal{O}(2^{2^{kn}})$	PTIME
$\mathcal{T}_{\text{RDS}}$	Recursive Data Structures	no	yes	undecidable	$\mathcal{O}(n \log n)$
$\mathcal{T}_{\text{ARDS}}$	Acyclic RDS	yes	yes	not elementary recursive	$\Theta(n)$
$\mathcal{T}_{\text{A}}$	Arrays	no	yes	undecidable	NP-complete
$\mathcal{T}_{\text{AX}}$	Arrays with Extensionality	no	yes	undecidable	NP-complete

## Survey of Decidability and Complexity [2]

Legend for the table:

- “**Full**” denotes the decidability of a complete theory *with* quantifiers.
- “**QF**” denotes the decidability of a *quantifier-free* theory.
- “**Full complexity**” denotes the complexity of the satisfiability in a complete theory *with quantifiers*.
- “**QFC complexity**” denotes the complexity of the satisfiability in a *quantifier-free conjunctive* fragment.
- For complexities,  $n$  is the size of the input formula,  $k$  is some positive integer.
- “*Not elementary recursive*” means the runtime cannot be bounded by a fixed-height stack of exponentials.

## §3 Difference Logic

## Difference Logic

**Definition 22:** *Difference logic* (DL) is a fragment of linear integer arithmetic consisting of conjunctions of literals of the very restricted form:

$$x - y \bowtie c$$

where  $x$  and  $y$  are integer variables,  $c$  is a numeral, and  $\bowtie \in \{=, <, \leq, >, \geq\}$ .

A solver for difference logic consists of three steps:

1. Literals normalization.
2. Conversion to a graph.
3. Cycle detection.

## Decision Procedure for DL

**Step 1:** Rewrite each literal using  $\leq$  by applying the following rules:

1.  $(x - y = c) \longrightarrow (x - y \leq c) \wedge (x - y \geq c)$
2.  $(x - y \geq c) \longrightarrow (y - x \leq -c)$
3.  $(x - y > c) \longrightarrow (y - x < -c)$
4.  $(x - y < c) \longrightarrow (x - y \leq c - 1)$

**Step 2:** Construct a weighted directed graph  $G$  with a vertex for each variable and an edge  $x \xrightarrow{c} y$  for each literal  $(x - y \leq c)$ .

**Step 3:** Check for *negative cycles* in  $G$ .

- Use, for example, the Bellman-Ford algorithm.
- If  $G$  contains a negative cycle, the set of literals is *inconsistent* (UNSAT).
- Otherwise, the set of literals is *consistent* (SAT).



## Difference Logic Example

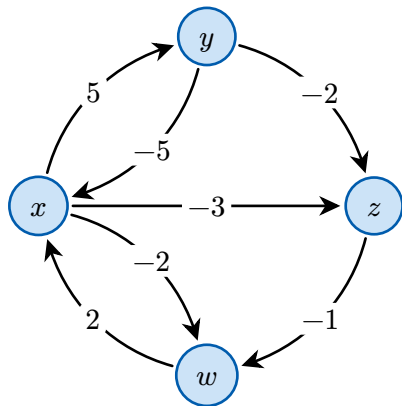
Consider the following set of difference logic literals:

$$(x - y = 5) \wedge (z - y \geq 2) \wedge (z - x > 2) \wedge (w - x = 2) \wedge (z - w < 0)$$

Normalize the literals:

- $(x - y = 5) \implies (x - y \leq 5) \wedge (y - x \leq -5)$
- $(z - y \geq 2) \implies (y - z \leq -2)$
- $(z - x > 2) \implies (x - z \leq -3)$
- $(w - x = 2) \implies (w - x \leq 2) \wedge (x - w \leq -2)$
- $(z - w < 0) \implies (z - w \leq -1)$

**UNSAT** because of the negative cycle:  $x \xrightarrow{-3} z \xrightarrow{-1} w \xrightarrow{2} x$ .



## §4 Equality

# Theory of Equality with Uninterpreted Functions

**Definition 23:** The theory of equality with uninterpreted functions  $\mathcal{T}_{\text{EUF}}$  is defined by the signature  $\Sigma^F = \{\dot{=}, f, g, h, \dots\}$  (*interpreted* equality and *uninterpreted* functions) and the following axioms:

1.  $\forall x. x \dot{=} x$  (reflexivity)
2.  $\forall x. \forall y. (x \dot{=} y) \rightarrow (y \dot{=} x)$  (symmetry)
3.  $\forall x. \forall y. \forall z. (x \dot{=} y) \wedge (y \dot{=} z) \rightarrow (x \dot{=} z)$  (transitivity)
4.  $\forall x. \forall y. (x = y) \rightarrow (f(x) \dot{=} f(y))$  (function congruence)

## Flattening

**Definition 24:** A literal is *flat* if it is of the form:

- $x \doteq y$
- $\neg(x \doteq y)$
- $x \doteq f(z)$

where  $x$  and  $y$  are variables,  $f$  is a function symbol, and  $z$  is a tuple of 0 or more variables.

**Note:** Any set of literals can be converted to an equisatisfiable set of *flat* literals by introducing *new* variables and equating non-equational atoms to *true*.

**Example:** Consider the set of literals:  $\{x + y > 0, y \doteq f(g(z))\}$ .

We can convert it to an equisatisfiable set of flat literals by introducing fresh variables  $v_i$ :

$$\{ v_1 \doteq v_2 > v_3, \quad v_1 \doteq \text{true}, \quad v_2 \doteq x + y, \quad v_3 \doteq 0, \quad y \doteq f(v_4), \quad v_4 \doteq g(z) \}$$

Hereinafter, we will assume that all literals are *flat*.

## Notation and Assumptions

---

- We abbreviate  $\neg(s \doteq t)$  with  $s \not\doteq t$ .
- For tuples  $\mathbf{u} = \langle u_1, \dots, u_n \rangle$  and  $\mathbf{v} = \langle v_1, \dots, v_n \rangle$ , we abbreviate  $(u_1 \doteq v_1) \wedge \dots \wedge (u_n \doteq v_n)$  with  $\mathbf{u} = \mathbf{v}$ .
- $\Gamma$  is used to refer to the “current” proof state in rule premises.
- $\Gamma, s \doteq t$  is an abbreviation for  $\Gamma \cup \{s \doteq t\}$ .
- If applying a rule  $R$  does not change  $\Gamma$ , then  $R$  *is not applicable* to  $\Gamma$ , that is,  $\Gamma$  is *irreducible* w.r.t.  $R$ .

## Satisfiability Proof System for QF\_UF

Let QF\_UF be the quantifier-free fragment of FOL over some signature  $\Sigma$ .

Below is a simple *satisfiability proof system*  $R_{\text{UF}}$  for QF\_UF:

<b>REFL</b> $\frac{x \text{ occurs in } \Gamma}{\Gamma := \Gamma, x \doteq x}$	<b>CONG</b> $\frac{x \doteq f(u) \in \Gamma \quad y \doteq f(v) \in \Gamma \quad u = v \in \Gamma}{\Gamma := \Gamma, x \doteq y}$
<b>SYMM</b> $\frac{x \not\doteq y \in \Gamma}{\Gamma := \Gamma, y \doteq x}$	<b>CONTR</b> $\frac{x \doteq y \in \Gamma \quad x \not\doteq y \in \Gamma}{\text{UNSAT}}$
<b>TRANS</b> $\frac{x \not\doteq y \in \Gamma \quad y \doteq z \in \Gamma}{\Gamma := \Gamma, x \doteq z}$	<b>SAT</b> $\frac{\text{No other rules apply}}{\text{SAT}}$

Is  $R_{\text{UF}}$  *sound*?

Is  $R_{\text{UF}}$  *terminating*?

## Example Derivation in $R_{UF}$

<b>REFL</b> $\frac{x \text{ occurs in } \Gamma}{\Gamma := \Gamma, x \doteq x}$	<b>TRANS</b> $\frac{x \not\doteq y \in \Gamma \quad y \doteq z \in \Gamma}{\Gamma := \Gamma, x \doteq z}$	<b>CONTR</b> $\frac{x \doteq y \in \Gamma \quad x \not\doteq y \in \Gamma}{\text{UNSAT}}$
<b>SYMM</b> $\frac{x \not\doteq y \in \Gamma}{\Gamma := \Gamma, y \doteq x}$	<b>CONG</b> $\frac{x \doteq f(u) \in \Gamma \quad y \doteq f(v) \in \Gamma \quad u = v \in \Gamma}{\Gamma := \Gamma, x \doteq y}$	<b>SAT</b> $\frac{\text{No other rules apply}}{\text{SAT}}$

**Example:** Determine the satisfiability of the following set of literals:  $a \doteq f(f(a))$ ,  $a \doteq f(f(f(a)))$ ,  $g(a, f(a)) \not\doteq g(f(a), a)$ . Flatten the literals and construct the following proof:

$a \doteq f(a_1), a_1 \doteq f(a), a \doteq f(a_2), a_2 \doteq f(a_1), a_3 \not\doteq a_4, a_3 \doteq g(a, a_1), a_4 \doteq g(a_1, a)$	<b>REFL</b>
$a_1 \doteq a_1$	<b>CONG</b> applied to $a \doteq f(a_1), a_2 \doteq f(a_1), a_1 \doteq a_1$
$a \doteq a_2$	<b>CONG</b> applied to $a_1 \doteq f(a), a \doteq f(a_2), a \doteq a_2$
$a_1 \doteq a$	<b>SYMM</b>
$a \doteq a_1$	<b>CONG</b> applied to $a_3 \doteq g(a, a_1), a_4 \doteq g(a_1, a), a \doteq a_1, a_1 \doteq a$
$a_3 \doteq a_4$	<b>CONTR</b> applied to $a_3 \doteq a_4, a_3 \not\doteq a_4$
<b>UNSAT</b>	

## Soundness of $R_{UF}$

**Theorem 4** (Refutation soundness): A literal set  $\Gamma_0$  is unsatisfiable if  $R_{UF}$  derives UNSAT from it.

**Proof:** All rules except SAT are satisfiability-preserving.

If a derivation from  $\Gamma_0$  ends with UNSAT, then  $\Gamma_0$  must be unsatisfiable. □

**Theorem 5** (Solution soundness): A literal set  $\Gamma_0$  is satisfiable if  $R_{UF}$  derives SAT from it.

**Proof:** Let  $\Gamma$  be a proof state to which SAT applies. From  $\Gamma$ , we can construct an interpretation  $\mathcal{J}$  that satisfies  $\Gamma_0$ . Let  $s \sim t$  iff  $(s \doteq t) \in \Gamma$ . One can show that  $\sim$  is an equivalence relation.

Let the domain of  $\mathcal{J}$  be the equivalence classes  $E_1, \dots, E_k$  of  $\sim$ .

- For every variable or a constant  $t$ , let  $t^{\mathcal{J}} = E_i$  if  $t \in E_i$  for some  $i$ . Otherwise, let  $t^{\mathcal{J}} = E_1$ .
- For every unary function symbol  $f$ , and equivalence class  $E_i$ , let  $f^{\mathcal{J}}$  be such that  $f^{\mathcal{J}}(E_i) = E_j$  if  $f(t) \in E_j$  for some  $t \in E_i$ . Otherwise, let  $f^{\mathcal{J}}(E_i) = E_1$ . Define  $f^{\mathcal{J}}$  for non-unary  $f$  similarly.

We can show that  $\mathcal{J} \models \Gamma$ . This means that  $\mathcal{J}$  models  $\Gamma_0$  as well since  $\Gamma_0 \subseteq \Gamma$ . □



## Termination in $R_{UF}$

**Theorem 6:** Every derivation strategy for  $R_{UF}$  terminates.

**Proof:**  $R_{UF}$  adds to the current state  $\Gamma$  only equalities between variables of  $\Gamma_0$ .

So, at some point it will run out of new equalities to add.

□

## Completeness of $R_{UF}$

**Theorem 7** (Refutation completeness): Every derivation strategy applied to an unsatisfiable state  $\Gamma_0$  ends with UNSAT.

**Proof:** Let  $\Gamma_0$  be an unsatisfiable state. Suppose there was a derivation from  $\Gamma_0$  that did not end with UNSAT. Then, by the termination theorem, it would have to end with SAT. But then  $R_{UF}$  would be not be solution sound.  $\square$

**Theorem 8** (Solution completeness): Every derivation strategy applied to a satisfiable state  $\Gamma_0$  ends with SAT.

**Proof:** Let  $\Gamma_0$  be a satisfiable state. Suppose there was a derivation from  $\Gamma_0$  that did not end with SAT. Then, by the termination theorem, it would have to end with UNSAT. But then  $R_{UF}$  would be not be refutation sound.  $\square$

## §5 Arrays

## Theory of Arrays

**Definition 25:** The theory of *arrays*  $\mathcal{T}_{AX}$  is defined by the signature  $\Sigma^S = \{A, I, E\}$  (arrays, indices, elements),  $\Sigma^F = \{\text{read}, \text{write}\}$  and the following axioms:

1.  $\forall a. \forall i. \forall v. (\text{read}(\text{write}(a, i, v), i) \doteq_E v)$
2.  $\forall a. \forall i. \forall j. \forall v. \neg(i \doteq_I j) \rightarrow (\text{read}(\text{write}(a, i, v), j) \doteq_E \text{read}(a, j))$
3.  $\forall a. \forall b. (\forall i. (\text{read}(a, i) \doteq_E \text{read}(b, i))) \rightarrow (a \doteq_A b)$

## Example

```
void ReadBlock(int data[], int x, int len) {  
    int i = 0;  
    int next = data[0];  
    for (; i < next && i < len; i = i + 1) {  
        if (data[i] == x)  
            break;  
        else  
            Process(data[i]);  
    }  
    assert(i < len);  
}
```

One pass through this code can be translated into the following  $\mathcal{T}_A$  formula:

$$(i \doteq 0) \wedge (\text{next} \doteq \text{read}(\text{data}, 0)) \wedge (i < \text{next}) \wedge \\ \wedge (i < \text{len}) \wedge (\text{read}(\text{data}, i) \doteq x) \wedge \neg(i < \text{len})$$

## Satisfiability Proof System for QF\_AX

The satisfiability proof system  $R_{AX}$  for  $\mathcal{T}_{AX}$  *extends* the proof system  $R_{UF}$  for  $\mathcal{T}_{UF}$  with the following rules:

$$\begin{array}{c} \mathbf{RINTRO1} \frac{b \doteq \text{write}(a, i, v) \in \Gamma}{\Gamma := \Gamma, v \doteq \text{read}(b, i)} \\[2ex] \mathbf{RINTRO2} \frac{b \doteq \text{write}(a, i, v) \in \Gamma \quad u \doteq \text{read}(x, j) \in \Gamma \quad x \in \{a, b\}}{\Gamma := \Gamma, i \doteq j \quad \Gamma := \Gamma, i \neq j, u \doteq \text{read}(a, j), u \doteq \text{read}(b, j)} \\[2ex] \mathbf{EXT} \frac{a \neq b \in \Gamma \quad a \text{ and } b \text{ are arrays}}{\Gamma := \Gamma, u \neq v, u \doteq \text{read}(a, k), v \doteq \text{read}(b, k)} \end{array}$$

- **RINTRO1**: After writing  $v$  at index  $i$ , the reading at the same index  $i$  gives us back the value  $v$ .
- **RINTRO2**: After writing  $v$  in  $a$  at index  $i$ , the reading from  $a$  or  $b$  at index  $j$  *splits* in two cases: (1)  $i$  equals  $j$ , (2)  $a$  and  $b$  have the same value  $u$  at position  $j$ .
- **EXT**: If two arrays  $a$  and  $b$  are distinct, they must differ at some index  $k$ .

## Example Derivation in $R_{AX}$

$$\begin{array}{l}
 \mathbf{RINTRO1} \frac{b \doteq \text{write}(a, i, v) \in \Gamma}{\Gamma := \Gamma, v \doteq \text{read}(b, i)} \quad \mathbf{EXT} \frac{a \not\equiv b \in \Gamma \quad a \text{ and } b \text{ are arrays}}{\Gamma := \Gamma, u \not\equiv v, u \doteq \text{read}(a, k), v \doteq \text{read}(b, k)} \\
 \mathbf{RINTRO2} \frac{b \doteq \text{write}(a, i, v) \in \Gamma \quad u \doteq \text{read}(x, j) \in \Gamma \quad x \in \{a, b\}}{\Gamma := \Gamma, i \doteq j \quad \Gamma := \Gamma, i \not\equiv j, u \doteq \text{read}(a, j), u \doteq \text{read}(b, j)}
 \end{array}$$

**Example:** Determine the satisfiability of  $\{\text{write}(a_1, i, \text{read}(a_1, i)) \doteq \text{write}(a_2, i, \text{read}(a_2, i)), a_1 \not\equiv a_2\}$ .

First, flatten the literals:

$$\begin{aligned}
 & \{\text{write}(a_1, i, \text{read}(a_1, i)) \doteq \text{write}(a_2, i, \text{read}(a_2, i))\} \longrightarrow \\
 & \longrightarrow \{a'_1 \doteq a'_2, a'_1 \doteq \text{write}(a_1, i, \text{read}(a_2, i)), a'_2 \doteq \text{write}(a_2, i, \text{read}(a_1, i)), a_1 \not\equiv a_2\} \longrightarrow \\
 & \longrightarrow \{a'_1 \doteq a'_2, a'_1 \doteq \text{write}(a_1, i, v_2), v_2 \doteq \text{read}(a_2, i), a'_2 \doteq \text{write}(a_2, i, v_1), v_1 \doteq \text{read}(a_1, i), a_1 \not\equiv a_2\}
 \end{aligned}$$

## Example Derivation in $R_{AX}$ [2]

1.  $a'_1 \doteq a'_2, a'_1 \doteq \text{write}(a_1, i, v_2), v_2 \doteq \text{read}(a_2, i), a'_2 \doteq \text{write}(a_2, i, v_1), v_1 \doteq \text{read}(a_1, i), a_1 \not\doteq a_2$
2. (by REFL)  $a_1 \doteq a_1$
3. (by REFL)  $a_2 \doteq a_2$
4. (by EXT)  $u_1 \not\doteq u_2, u_1 \doteq \text{read}(a_1, n), u_2 \doteq \text{read}(a_2, n)$
5. (by RINTRO2) split

- 
6.  $i \doteq n$
  7. (by CONG)  $v_1 \doteq u_1$
  8. (by SYMM)  $u_1 \doteq v_1$
  9. (by CONG)  $v_2 \doteq u_2$
  10. (by RINTRO1)  $v_2 \doteq \text{read}(a'_1, i)$
  11. (by RINTRO1)  $v_1 \doteq \text{read}(a'_2, i)$
  12. (by REFL)  $i \doteq i$
  13. (by CONG)  $v_1 \doteq v_2$
  14. (by TRANS)  $u_1 \doteq u_2$
  15. (by CONTR) UNSAT

- 
6.  $i \not\doteq n, u_1 \doteq \text{read}(a'_1, n)$
  7. (by RINTRO2) split

- 
8.  $i \doteq n$
  9. (by CONTR) UNSAT

- 
8.  $i \not\doteq n, u_2 \doteq \text{read}(a'_2, n)$
  9. (by RELF)  $n \doteq n$
  10. (by CONG)  $u_1 \doteq u_2$
  11. (by CONTR) UNSAT



## §6 *Arithmetic*

## Theory of Real Arithmetic

**Definition 26:** The theory of *real arithmetic*  $\mathcal{T}_{\text{RA}}$  is defined by the signature  $\Sigma_{\text{RA}}^S = \{\text{Real}\}$ ,  $\Sigma_{\text{RA}}^F = \{+, -, \times, \leq\} \cup \{q \mid q \text{ is a decimal numeral}\}$  and the class of interpretations  $\mathcal{M}_{\text{RA}}$  that interpret  $\text{Real}$  as the set of *real numbers*  $\mathbb{R}$ , and the function symbols in the usual way.

*Quantifier-free linear real arithmetic* (QF\_LRA) is the theory of *linear inequalities* over the reals, where  $\times$  can only be used in the form of *multiplication by constants (decimal numerals)*.

# Linear Programming

**Definition 27:** A *linear program* (LP) consists of:

1. An  $m \times n$  matrix  $\mathbf{A}$ , the *constraint matrix*.
2. An  $m$ -dimensional vector  $\mathbf{b}$ .
3. An  $n$ -dimensional vector  $\mathbf{c}$ , the *objective function*.

Let  $\mathbf{x}$  be a vector of  $n$  variables.

**Goal:** Find a solution  $\mathbf{x}$  that *maximizes*  $\mathbf{c}^T \mathbf{x}$  subject to the linear constraints  $\mathbf{Ax} \leq \mathbf{b}$  (and<sup>3</sup>  $\mathbf{x} \geq \mathbf{0}$ ).

**Note:** All **bold**-styled symbols denote *vectors* or *matrices*, e.g.,  $\mathbf{x}$ ,  $\mathbf{A}$ ,  $\mathbf{0}$ .

---

<sup>3</sup>The constraint  $\mathbf{x} \geq \mathbf{0}$  is introduced when LP is expressed in *standard form*, explained later in these slides.

## Example and Terminology

**Example:** Maximize  $2x_2 - x_1$  subject to:

$$x_1 + x_2 \leq 3$$

$$2x_1 - x_2 \leq -5$$

Here,  $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ ,  $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & -1 \end{bmatrix}$ ,  $\mathbf{b} = \begin{bmatrix} 3 \\ -5 \end{bmatrix}$ ,  $\mathbf{c} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$ .

Find  $\mathbf{x}$  that maximizes  $\mathbf{c}^T \mathbf{x}$  subject to  $\mathbf{Ax} \leq \mathbf{b}$ .

**Definition 28:** An assignment of  $\mathbf{x}$  is a *feasible solution* if it satisfies  $\mathbf{Ax} \leq \mathbf{b}$ .

- Is  $\mathbf{x} = \langle 0, 0 \rangle$  a feasible solution? ✗
- Is  $\mathbf{x} = \langle -2, 1 \rangle$  a feasible solution? ✓

**Definition 29:** For a given assignment  $\mathbf{x}$ , the value  $\mathbf{c}^T \mathbf{x}$  is the *objective value*, or *cost*, of  $\mathbf{x}$ .

- What is the objective value of  $\mathbf{x} = \langle -2, 1 \rangle$ ?

## Example and Terminology [2]

**Definition 30:** An *optimal solution* is a feasible solution with a *maximal* objective value among all feasible solutions.

**Definition 31:** If a linear program has no feasible solutions, it is *infeasible*.

**Definition 32:** The linear program is *unbounded* if the objective value of the optimal solution is  $\infty$ .

## Geometric Interpretation

**Definition 33:** A *polytope* is a generalization of 3-dimensional polyhedra to higher dimensions.

**Definition 34:** A polytope  $P$  is *convex* if every point on the line segment connecting any two points in  $P$  is also within  $P$ .

Formally, for all  $a, b \in \mathbb{R}^n \cap P$ , and for all  $\lambda \in [0; 1]$ , it holds that  $\lambda a + (1 - \lambda)b \in P$ .

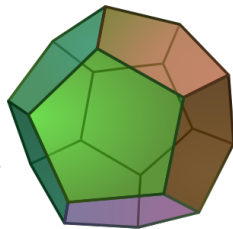
**Note:** For an  $m \times n$  constraint matrix  $A$ , the set of points  $P = \{x \mid Ax \leq b\}$  forms a *convex polytope* in  $n$ -dimensional space.

**LP goal:** find a point  $x$  *inside the polytope* that maximizes  $c^T x$  for a given  $c$ .

**Note:** LP is *infeasible* iff the polytope is *empty*.

**Note:** LP is *unbounded* iff the polytope is *open* in the direction of the objective function.

**Note:** The *optimal solution* for a bounded LP lies on a *vertex* of the polytope.



# Satisfiability as Linear Programming

Our goal is to use LP to check the satisfiability of *sets of linear  $\mathcal{T}_{\text{RA}}$ -literals*.

**Step 1:** Convert equalities to inequalities.

- A linear  $\mathcal{T}_{\text{RA}}$ -equality can be written to have the form  $\mathbf{a}^T \mathbf{x} = \mathbf{b}$ .
- We rewrite this further as  $\mathbf{a}^T \mathbf{x} \geq \mathbf{b}$  and  $\mathbf{a}^T \mathbf{x} \leq \mathbf{b}$ .
- And finally to  $-\mathbf{a}^T \mathbf{x} \leq -\mathbf{b}$  and  $\mathbf{a}^T \mathbf{x} \leq \mathbf{b}$ .

**Step 2:** Handle inequalities.

- A  $\mathcal{T}_{\text{RA}}$ -literal of the form  $\mathbf{a}^T \mathbf{x} \leq \mathbf{b}$  is already in the desired form.
- A  $\mathcal{T}_{\text{RA}}$ -literal of the form  $\neg(\mathbf{a}^T \mathbf{x} \leq \mathbf{b})$  is transformed as follows:

$$\neg(\mathbf{a}^T \mathbf{x} \leq \mathbf{b}) \longrightarrow (\mathbf{a}^T \mathbf{x} > \mathbf{b}) \longrightarrow (-\mathbf{a}^T \mathbf{x} < -\mathbf{b}) \longrightarrow (-\mathbf{a}^T \mathbf{x} + y \leq -\mathbf{b}), (y > 0)$$

where  $y$  is a fresh variable used for all negated inequalities.

**Example:**  $\neg(2x_1 - x_2 \leq 3)$  rewrites to  $-2x_1 + x_2 + y \leq -3, y > 0$

- If there are no negated inequalities, add the inequality  $y \leq 1$ , where  $y$  is a fresh variable.

## Satisfiability as Linear Programming [2]

- In either case, we end up with a set of the form  $\mathbf{a}^T \mathbf{x} \leq \mathbf{b} \cup \{y > 0\}$

**Step 3:** Check the satisfiability of  $\mathbf{a}^T \mathbf{x} \leq \mathbf{b} \cup \{y > 0\}$ .

Encode it as LP: maximize  $y$  subject to  $\mathbf{a}^T \mathbf{x} \leq \mathbf{b}$ .

The final system is *satisfiable* iff the *optimal value* for  $y$  is *positive*.



## Methods for Solving LP

- *Simplex* (Dantzig, 1947) — exponential time  $\mathcal{O}(2^n)$
- *Ellipsoid* (Khachiyan, 1979) — polynomial time  $\mathcal{O}(n^6)$
- *Projective* (Karmarkar, 1984) — polynomial time  $\mathcal{O}(n^{3.5})$
- And many more tricky algorithms approaching  $\mathcal{O}(n^{2.5})$

**Note:** Although the Simplex method is the *oldest* and the *least efficient in theory*, it can be implemented to be *quite efficient in practice*. It remains the most popular and we will focus on it next.

## Standard Form

Any LP can be transformed to *standard form*:

$$\begin{aligned} & \text{maximize } \sum_{j=1}^n c_j x_j \\ & \text{such that } \sum_{j=1}^m a_{ij} x_j \leq b_i \text{ for } i = 1, \dots, m \\ & \quad x_j \geq 0 \text{ for } j = 1, \dots, n \end{aligned}$$

**Example:** Next, we are going to use the following running example LP:

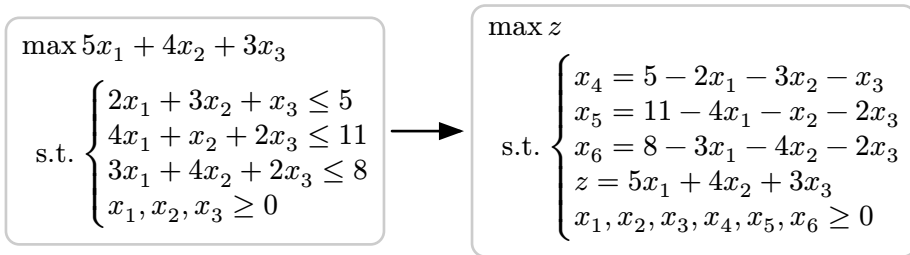
$$\begin{aligned} & \text{maximize } 5x_1 + 4x_2 + 3x_3 \\ & \text{such that } \begin{cases} 2x_1 + 3x_2 + x_3 \leq 5 \\ 4x_1 + x_2 + 2x_3 \leq 11 \\ 3x_1 + 4x_2 + 2x_3 \leq 8 \\ x_1, x_2, x_3 \geq 0 \end{cases} \end{aligned}$$

## Slack Variables

- Observe the first inequality:  $2x_1 + 3x_2 + x_3 \leq 5$
- Define a *new variable* to represent the *slack*:

$$x_4 = 5 - 2x_1 - 3x_2 - x_3, \quad x_4 \geq 0$$

- Do this for each constraint, so that everything becomes *equalities*.
- Define a new variable to represent the *objective value*:  $z = 5x_1 + 4x_2 + 3x_3$



The diagram illustrates the transformation of a linear programming problem. On the left, a box contains the original problem with a maximization objective and three inequality constraints. An arrow points to the right, where a second box shows the transformed problem with the same objective, but the constraints are now equalities, each with a new slack variable added. The variables  $x_4, x_5, x_6$  represent the slack for each constraint, and the objective variable  $z$  is also included in the non-negativity constraints.

$$\begin{array}{l} \max 5x_1 + 4x_2 + 3x_3 \\ \text{s.t.} \begin{cases} 2x_1 + 3x_2 + x_3 \leq 5 \\ 4x_1 + x_2 + 2x_3 \leq 11 \\ 3x_1 + 4x_2 + 2x_3 \leq 8 \\ x_1, x_2, x_3 \geq 0 \end{cases} \end{array} \quad \longrightarrow \quad \begin{array}{l} \max z \\ \text{s.t.} \begin{cases} x_4 = 5 - 2x_1 - 3x_2 - x_3 \\ x_5 = 11 - 4x_1 - x_2 - 2x_3 \\ x_6 = 8 - 3x_1 - 4x_2 - 2x_3 \\ z = 5x_1 + 4x_2 + 3x_3 \\ x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{cases} \end{array}$$

**Note:** Optimal solution remains optimal for the new problem.

# The Simplex Strategy

- Start with a feasible solution.
  - For our example, assign 0 to all variables.  
 $x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0$
  - Assign the introduced variables their computed values.  
 $x_4 \mapsto 5, x_5 \mapsto 11, x_6 \mapsto 8, z \mapsto 0$
- Iteratively improve the objective value.
  - Go from  $\mathbf{x}$  to  $\mathbf{x}'$  only if  $z(\mathbf{x}) \leq z(\mathbf{x}')$ .

What can we improve here?

One option is to make  $x_1$  larger, leaving  $x_2$  and  $x_3$  unchanged:

- $x_1 = 1 \rightarrow x_4 = 3, x_5 = 7, x_6 = 1, z = 5$  ✓
- $x_1 = 2 \rightarrow x_4 = 1, x_5 = 3, x_6 = 2, z = 10$  ✓
- $x_1 = 3 \rightarrow x_4 = -1, \dots$  ✗ *no longer feasible!*

$$\begin{cases} x_4 = 5 - 2x_1 - 3x_2 - x_3 \\ x_5 = 11 - 4x_1 - x_2 - 2x_3 \\ x_6 = 8 - 3x_1 - 4x_2 - 2x_3 \\ z = 5x_1 + 4x_2 + 3x_3 \end{cases}$$

## The Simplex Strategy [2]

We can't increase  $x_1$  *too much*. Let's increase it as much as possible, *without compromising feasibility*.

$$\begin{array}{l} x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0 \\ \left\{ \begin{array}{l} x_4 = 5 - 2x_1 - 3x_2 - x_3 \\ x_5 = 11 - 4x_1 - x_2 - 2x_3 \\ x_6 = 8 - 3x_1 - 4x_2 - 2x_3 \\ z = 5x_1 + 4x_2 + 3x_3 \end{array} \right. \end{array} \longrightarrow \left\{ \begin{array}{l} x_1 \leq \frac{5}{2} \\ x_1 \leq \frac{11}{4} \\ x_1 \leq \frac{8}{3} \end{array} \right.$$

Select the *tightest bound*,  $x_1 \leq \frac{5}{2}$ .

- New assignment:  $x_1 \mapsto \frac{5}{2}, x_2 \mapsto x_3 \mapsto x_4 \mapsto 0, x_5 \mapsto 1, x_6 \mapsto \frac{1}{2}, z \mapsto \frac{25}{2}$
- This indeed improves the objective value  $z$ .

## The Simplex Strategy [3]

Current assignment:

- $x_1 \mapsto \frac{5}{2}, x_2 \mapsto x_3 \mapsto x_4 \mapsto 0, x_5 \mapsto 1, x_6 \mapsto \frac{1}{2}, z \mapsto \frac{25}{2}$

How do we continue?

For the first iteration we had:

- A *feasible solution*.
- An *equation system* where the variables with positive values are expressed in terms of variables with 0 value.

Does the current *equation system* satisfy this property? **No** ✗

$$\begin{cases} x_4 = 5 - 2x_1 - 3x_2 - x_3 \\ x_5 = 11 - 4x_1 - x_2 - 2x_3 \\ x_6 = 8 - 3x_1 - 4x_2 - 2x_3 \\ z = 5x_1 + 4x_2 + 3x_3 \end{cases}$$

## The Simplex Strategy [4]

What should we change?

- Initially,  $x_1$  was 0 and  $x_4$  was positive.
- Now,  $x_1$  is positive and  $x_4$  is 0.

Isolate  $x_1$  and *eliminate* it from right-hand-side:

$$\bullet \quad x_4 = 5 - 2x_1 - 3x_2 - x_3 \quad \longrightarrow \quad x_1 = \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4$$

$$x_1 \mapsto \frac{5}{2}, x_2 \mapsto x_3 \mapsto x_4 \mapsto 0$$

$$\begin{cases} x_4 = 5 - 2x_1 - 3x_2 - x_3 \\ x_5 = 11 - 4x_1 - x_2 - 2x_3 \\ x_6 = 8 - 3x_1 - 4x_2 - 2x_3 \\ z = 5x_1 + 4x_2 + 3x_3 \end{cases}$$

$$\begin{cases} x_4 = 5 - 2x_1 - 3x_2 - x_3 \\ x_5 = 11 - 4x_1 - x_2 - 2x_3 \\ x_6 = 8 - 3x_1 - 4x_2 - 2x_3 \\ z = 5x_1 + 4x_2 + 3x_3 \end{cases}$$



$$\begin{cases} x_1 = \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4 \\ x_5 = 1 + 5x_2 + \quad \quad \quad + 2x_4 \\ x_6 = \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_4 \\ z = \frac{25}{2} - \frac{7}{2}x_2 + \frac{1}{2}x_3 - \frac{5}{2}x_4 \end{cases}$$

## The Simplex Strategy [5]

How can we improve  $z$  further?

- **Option 1:** decrease  $x_2$  or  $x_4$ , but we can't since  $x_2, x_4 \geq 0$ .
- **Option 2:** increase  $x_3$ . *By how much?*

$x_3$ 's bounds:  $x_3 \leq 5, x_3 \leq \infty, x_3 \leq 1$ .

We increase  $x_3$  to its tightest bound 1.

- New assignment:  $x_1 \mapsto 2, x_2 \mapsto 0, x_3 \mapsto 1, x_4 \mapsto 0, x_5 \mapsto 0, x_6 \mapsto 0$ .
- This gives  $z = 13$ , which is again an improvement.

As before, we switch  $x_6$  and  $x_3$ , and *eliminate*  $x_3$  from the right-hand-side:

$$\left\{ \begin{array}{l} x_1 = \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4 \\ x_5 = 1 + 5x_2 + \quad \quad + 2x_4 \\ x_6 = \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_4 \\ z = \frac{25}{2} - \frac{7}{2}x_2 + \frac{1}{2}x_3 - \frac{5}{2}x_4 \end{array} \right. \longrightarrow \left\{ \begin{array}{l} x_1 = 2 - 2x_2 - 2x_4 + x_6 \\ x_5 = 1 + 5x_2 + 2x_4 \\ x_3 = 1 + x_2 + 3x_4 - 2x_6 \\ z = 13 - 3x_2 - x_4 - x_6 \end{array} \right.$$

$$x_1 \mapsto \frac{5}{2}, x_2 \mapsto 0, x_3 \mapsto 0, x_4 \mapsto 0$$

$$\left\{ \begin{array}{l} x_1 = \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4 \\ x_5 = 1 + 5x_2 + \quad \quad + 2x_4 \\ x_6 = \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_4 \\ z = \frac{25}{2} - \frac{7}{2}x_2 + \frac{1}{2}x_3 - \frac{5}{2}x_4 \end{array} \right.$$



## The Simplex Strategy [6]

Can we improve  $z$  again?

- No, because  $x_2, x_4, x_6 \geq 0$ , and all *appear with negative signs* in the objective function.

So, we are done, and the optimal value of  $z$  is 13.

The optimal solution is then  $x_1 \mapsto 2, x_2 \mapsto 0, x_3 \mapsto 1$ .

$$x_1 \mapsto 2, x_2 \mapsto 0, x_3 \mapsto 1,$$

$$x_4 \mapsto 0, x_6 \mapsto 0$$

$$\begin{cases} x_1 = 2 - 2x_2 - 2x_4 + x_6 \\ x_5 = 1 + 5x_2 + 2x_4 \\ x_3 = 1 + x_2 + 3x_4 - 2x_6 \\ z = 13 - 3x_2 - x_4 - x_6 \end{cases}$$

# The Simplex Algorithm

$$\begin{aligned} &\text{maximize } \sum_{j=1}^n c_j x_j \\ &\text{such that } \sum_{j=1}^m a_{ij} x_j \leq b_i \text{ for } i = 1, \dots, m \\ &\quad x_j \geq 0 \text{ for } j = 1, \dots, n \end{aligned}$$

1. Introduce slack variables  $x_{n+1}, \dots, x_{n+m}$ .
2. Set  $x_{n+i} = b_i - \sum_{j=1}^n a_{ij} x_j$  for  $i = 1, \dots, m$ .
3. Start with initial, *feasible* solution. (commonly,  $x_1 \mapsto 0, \dots, x_n \mapsto 0$ )
4. While some summands in the current objective function have *positive coefficients*, update the feasible solution to improve the objective value. Otherwise, stop.
5. Update the equations to *maintain the invariant* that all right-hand-side values have value 0.
6. Go to 4.

§7 CDCL( $\mathcal{T}$ )

## CDCL( $\mathcal{T}$ ) Architecture

$$\text{CDCL}(\mathcal{T}) = \text{CDCL}(X) + \mathcal{T}\text{-solver}$$

CDCL( $X$ ):

- Very *similar to a SAT solver*, enumerates Boolean models.
- Not allowed: pure literal rule (and other SAT specific heuristics).
- Required: incremental addition of clauses.
- Desirable: partial model detection.

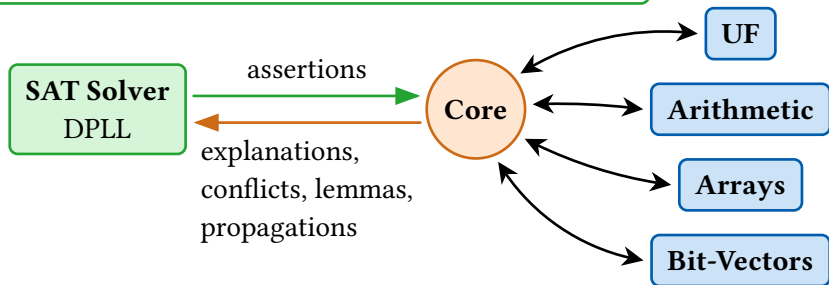
$\mathcal{T}$ -solver:

- Checks the  $\mathcal{T}$ -satisfiability of conjunctions of literals.
- Computes *theory propagations*.
- Produces *explanations* of  $\mathcal{T}$ -unsatisfiability/propagation.
- Must be *incremental* and *backtrackable*.

# Typical SMT Solver Architecture

## SAT Solver:

- Only sees *Boolean skeleton* of a problem.
- Builds *partial model* by assigning truth values to literals
- Sends these literals to the core as *assertions*



## Core:

- Sends each assertion to the appropriate theory
- Sends deduced literals to other theories/SAT solver
- Handles *theory combination*

## Theory Solvers:

- Check  $\mathcal{T}$ -satisfiability of sets of theory literals
- Incremental
- Backtrackable
- Conflict generation
- Theory propagation

## §8 Combining Theories

# Motivation

---

TODO

## TODO

- ☒ theory of arrays  $\mathcal{T}_A$
- ☒ satisfiability proof system for  $\mathcal{T}_A$
- ☒ example of derivation in  $R_{AX}$
- ☐ soundness, termination, completeness of  $R_{AX}$
- ☐ RDS solver
- ☐ Bit-vector solver
- ☐ String solver
- ☒ LRA
- ☒ Linear programming
- ☐ Simplex algorithm
- ☐ Combination of theories