1. Satisfiability Modulo Theories

Prof. Roberto Amadini

Department of Computer Science and Engineering, University of Bologna, Italy

Combinatorial Decision Making and Optimization

2nd cycle degree programme in Artificial Intelligence

University of Bologna, Academic Year 2024/25

From SAT to SMT

- Boolean satisfiability problem (SAT) well-studied
  - 1st problem proven NP-complete
- Several applications in AI and other fields
  - *The Silent (R)evolution of SAT*
    https://dl.acm.org/doi/10.1145/3560469
- Based on Propositional Logic
  - Atomic proposition or atoms (facts) can be either true or false
  - Atoms combined via connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
  - Decidable, "efficient", but limited expressiveness
- Real-world applications often require a more expressive logic, e.g., First-Order Logic (FOL)

From SAT to SMT

- FOL formulas are *more general*, e.g., $(\forall x)(x < u \land (\nexists y)\ f(x, y) = u)$
- Many applications do not require "general-purpose" FOL satisfiability: only satisfiability w.r.t. a *fixed* background theory
- E.g., given formula $y = x + 1 \land x < z \land z < y$ one typically checks satisfiability w.r.t. an arithmetical theory, unless otherwise specified...
  - But one might interpret 1 as '1', $+$ as concatenation, $<$ as lex...
  - ...What arithmetic theory? Integers or reals?
- Focusing on a particular theory enables a more efficient solving via specialized decision procedures
  - Especially for quantifier-free formulas

From SAT to SMT

- A formula ϕ can be undecidable! We cannot prove neither ϕ nor $\neg\phi$
  - Undecidable $\neq$ unsatisfiable
- But we can restrict to decidable fragments of an undecidable theory
  - "subsets of the theory"
- **Satisfiability Modulo Theory** (SMT) concerns the study of the satisfiability of formulas w.r.t. some backgrounds theories
  - theory of integer arithmetic, real numbers, arrays, strings, (multi-)sets, trees, lists, bit-vectors, ...
- SMT solvers are used to determine the satisfiability of formulas through different procedures over different background theories
  - E.g., Z3 or CVC5 (formerly CVC4, CVC3, ...)

SMT history

- The roots of SMT date back to late 70s - early 80s
  - Nelson and Oppen, Shostak, Boyer and Moore, ...
- Modern SMT research started in the 90s
- From the 2000s up to now big development in SMT's foundational and practical aspects
  - SMT approaches integrated in various tools for theorem proving, program analysis and testing
  - Following big development in SAT solvers

SMT vs SAT/CP

- SMT extends SAT, and tackles combinatorial problems from an orthogonal perspective w.r.t. CP

- Like SAT solving, SMT historically specialised in theorem proving, software analysis and verification, (dynamic) symbolic execution
- CP more oriented to scheduling, resource allocation, optimization

- Like CP and unlike SAT, SMT employs domain-specific reasoning
- Unlike CP, SMT doesn't generally require finite domains

- SMT solving uses SAT abstractions instead of propagators and natively handles nogoods, which is not always true for CP solvers

Example: Dynamic Symbolic Execution of JavaScript

```
var x = symStr();    ▶ x is a symbolic variable having String type
var y = "length";
if (x[y] >= 2)
  console.log("PC1")
else if (y[x] === "g")
  console.log("PC2")
else
  console.log("PC3")
```

- Dynamic Symbolic Execution (DSE) runs a program with concrete inputs while tracking symbolic expressions (a.k.a. concolic execution)
  - DSE enables automated test generation and bug detection

- Consider the DSE of above JavaScript snippet starting with $\{x \leftarrow \texttt{""}\}$

Example: DSE of JavaScript

```
var x = "";
var y = "length";
if (""[y] >= 2)
  console.log("PC1")
else if (y[""] === "g")
  console.log("PC2")
else
  console.log("PC3")
```

- Property "length" of "" is 0

Example: DSE of JavaScript

```
var x = "";
var y = "length";
if (""[y] >= 2)
  console.log("PC1")
else if (y[""] === "g")
  console.log("PC2")
else
  console.log("PC3")
```

- Property `""` of `"length"` object is `undefined`

Example: DSE of JavaScript

```
1 var x = "";
2 var y = "length";
3 if (""[y] >= 2)
4   console.log("PC1")
5 else if (y[""] === "g")
6   console.log("PC2")
7 else
8   console.log("PC3")
```

▶ $PC = \{\neg(|x| \geq 2), \neg(\texttt{"length"}[x] = \texttt{"g"})\}$

- With input $x \leftarrow$ `""` we reached line 8 with associated path condition $PC = \{\neg(|x| \geq 2), \neg(\texttt{"length"}[x] = \texttt{"g"})\}$
- We negate one of the constraint of PC (e.g., $\neg(|x| \geq 2)$) and we solve $PC' = \{|x| \geq 2, \neg(\texttt{"length"}[x] = \texttt{"g"})\}$
  - A feasible solution is $x \leftarrow$ `"aa"`
  - We need a suitable string solver: PC' has (*reified*) constraints over string length, (in-)equality, indexing, ...

Example: DSE of JavaScript

```
1  var x = symVar();
2  var y = "length";
3  if (x[y] >= 2)          ▶ PC′ = {|x| ≥ 2, ¬("length"[x] = "g")}
4    console.log("PC1")
5  else if (y[x] === "g")
6    console.log("PC2")    ▶ PC″ = {¬(|x| ≥ 2), "length"[x] = "g"}
7  else
8    console.log("PC3")    ▶ PC = {¬(|x| ≥ 2), ¬("length"[x] = "g")}
```

- With $x \leftarrow$ `"aa"` we reach line 4 with associated PC'; we then iterate the procedure to solve $PC'' = \{\neg(|x| \geq 2), \texttt{"length"}[x] = \texttt{"g"}\}$
  - A feasible solution is $x \leftarrow$ `"3"`, because `"length"["3"] === "g"`
  - PC'' hard to solve if we don't know that $x \in \{\texttt{"0"}, \texttt{"1"}, \dots\}$
- With $x \leftarrow$ `"3"` we reach line 6: the set of inputs $\{\{x \leftarrow \texttt{""}\}, \{x \leftarrow \texttt{"aa"}\}, \{x \leftarrow \texttt{"3"}\}\}$ covers all the lines

Example: DSE of JavaScript

```
var x = symVar();
var y = "length";
if (x[y] >= 2)    ▶ PC′ = {|x| ≥ 2, ¬("length"[x] = "g")}
  console.log("PC1")
else if (y[x] === "g")
  console.log("PC2")    ▶ PC″ = {¬(|x| ≥ 2), "length"[x] = "g"}
else
  console.log("PC3")    ▶ PC = {¬(|x| ≥ 2), ¬("length"[x] = "g")}
```

- Example: ArathaJS
  - `https://github.com/ArathaJS/aratha`
  - `./run-analysis demo.js`
  - No longer developed / maintained :-(

SMT and Program Analysis

- Faithfully modelling JS, and in general programming languages semantics, is hard but not strictly necessary
- Difficult constructs and PCs can be ignored or approximated
  - This affects correctness (we might not achieve maximum coverage)
  - Acceptable if *"good enough"* coverage is reached in short time
- SMT solvers mostly used for software analysis: stronger reasoning capabilities and expressiveness w.r.t SAT, CP or MIP solvers
  - Achieved by leveraging FOL over multiple background theories

SMT Preliminaries

Formal preliminaries

- Let's recall (introduce?) the relevant FOL concepts and notation
  - We shall always assume FOL with equality
- The signature of a FOL is the set $\Sigma = \Sigma^F \cup \Sigma^P$ of its non-logical symbols, i.e., functions in Σ^F and predicates in Σ^P
  - We denote Σ_k^F (resp. Σ_k^P) the functions (predicates) with arity $k \geq 0$
  - So, $\Sigma^P = \bigcup_k \Sigma_k^P$ and $\Sigma^F = \bigcup_k \Sigma_k^F$
- The 0-arity functions of Σ_0^F are constant symbols
- The 0-arity predicates of Σ_0^P are propositional symbols
  - E.g., propositional logic has $\Sigma^F = \emptyset$ and $\Sigma^P = \Sigma_0^P \supseteq \{\bot, \top\}$
- We will consider only quantifier-free fragments (no $\exists, \forall$)
  - All variables are free variables

Terms and Formulas

- The set $\mathbb{T}^{\Sigma}$ of terms Σ is defined as:
  - $c \in \Sigma_0^F \implies c \in \mathbb{T}^{\Sigma}$
  - $f \in \Sigma_k^F$ and $t_1, \ldots, t_k \in \mathbb{T}^{\Sigma} \implies f(t_1, \ldots, t_k) \in \mathbb{T}^{\Sigma}$
  - $\varphi \in \mathbb{F}^{\Sigma}$ and $t_1, t_2 \in \mathbb{T}^{\Sigma} \implies ite(\varphi, t_1, t_2) \in \mathbb{T}^{\Sigma}$

- The set $\mathbb{F}^{\Sigma}$ of formulas of Σ is defined as:

$$\left.\begin{array}{l}
\bullet\ \bot, \top \in \mathbb{F}^{\Sigma} \\
\bullet\ t_1, t_2 \in \mathbb{T}^{\Sigma} \implies t_1 = t_2 \in \mathbb{F}^{\Sigma} \\
\bullet\ A \in \Sigma_0^P \implies A \in \mathbb{F}^{\Sigma} \\
\bullet\ p \in \Sigma_k^P \text{ and } t_1, \ldots, t_k \in \mathbb{T}^{\Sigma} \implies p(t_1, \ldots, t_k) \in \mathbb{F}^{\Sigma}
\end{array}\right\} \text{Atomic formulas}$$

  - $\varphi \in \mathbb{F}^{\Sigma} \implies \neg\varphi \in \mathbb{F}^{\Sigma}$
  - $\varphi_1, \varphi_2 \in \mathbb{F}^{\Sigma} \implies \varphi_1 \to \varphi_2,\ \varphi_1 \leftrightarrow \varphi_2,\ \varphi_1 \wedge \varphi_2,\ \varphi_1 \vee \varphi_2 \in \mathbb{F}^{\Sigma}$

Formal preliminaries

- An atomic formula is also called an atom
- A literal is either:
  - an atomic formula (positive literal), or
  - the negation of one (negative literal)
- A clause is a disjunction $\ell_1 \vee \cdots \vee \ell_k$ of literals
  - A unit clause is a clause consisting of a single literal $\neq \bot, \top$
- A formula is in Conjunctive Normal Form (CNF) if it is the conjunction $c_1 \wedge \cdots \wedge c_k$ of $k \geq 0$ clauses
  - Also denoted $\{c_1, \ldots, c_k\}$ or simply $c_1, \ldots, c_k$

Semantics

- The semantics of a formula denotes its *"meaning"*, i.e., a truth value in $\{true, false\}$, by means of a certain interpretation

- A model for Σ is a pair $\mathcal{M} = \langle M, (\cdot)^{\mathcal{M}} \rangle$ where set M is the universe of $\mathcal{M}$ and a mapping $(\cdot)^{\mathcal{M}}$ such that:
  - $f^{\mathcal{M}} \in \{\varphi \mid \varphi : M^k \to M\}$ for each function $f \in \Sigma_k^F$
    - In particular $c^{\mathcal{M}} \in M$ for each constant $c \in \Sigma_0^F$
  - $p^{\mathcal{M}} \in \{\varphi \mid \varphi : M^k \to \{true, false\}\}$ for each predicate $f \in \Sigma_k^P$
    - In particular $B^{\mathcal{M}} \in \{true, false\}$ for each proposition $B \in \Sigma_0^P$

- The $(\cdot)^{\mathcal{M}}$ extension to terms and formulas is called interpretation:
  - $\perp^{\mathcal{M}} = false$, $\top^{\mathcal{M}} = true$, $(t_1 = t_2)^{\mathcal{M}} = true \iff t_1^{\mathcal{M}} = t_2^{\mathcal{M}}$
  - $f(t_1, \dots, t_k)^{\mathcal{M}} = f^{\mathcal{M}}(t_1^{\mathcal{M}}, \dots, t_k^{\mathcal{M}})$
  - $p(t_1, \dots, t_k)^{\mathcal{M}} = p^{M}(t_1^{\mathcal{M}}, \dots, t_k^{\mathcal{M}})$
  - $ite(\varphi, t_1, t_2)^{\mathcal{M}} = \begin{cases} t_1^{\mathcal{M}} & \text{if } \varphi^{\mathcal{M}} = true \\ t_2^{\mathcal{M}} & \text{if } \varphi^{\mathcal{M}} = false \end{cases}$

Satisfiability

- $\mathcal{M}$ satisfies (resp. falsifies) $\varphi \in \mathbb{F}^\Sigma$ if $\varphi^\mathcal{M} = \mathit{true}$ (resp. $\varphi^\mathcal{M} = \mathit{false}$)
- A Σ-theory is a (*possibly infinite*) set $\mathcal{T}$ of Σ-models
- $\varphi \in \mathbb{F}^\Sigma$ is $\mathcal{T}$-satisfiable if there exists a model $\mathcal{M} \in \mathcal{T}$ satisfying φ
  - $\{\varphi_1, \ldots, \varphi_k\} \subseteq \mathbb{F}^\Sigma$ is $\mathcal{T}$-consistent iff $\varphi_1 \wedge \cdots \wedge \varphi_k$ is $\mathcal{T}$-satisfiable
- $\Gamma \subseteq \mathbb{F}^\Sigma$ $\mathcal{T}$-entails φ iff every $\mathcal{M} \in \mathcal{T}$ that satisfies Γ also satisfies φ
  - If Γ $\mathcal{T}$-entails φ, we write $\Gamma \models_\mathcal{T} \varphi$
  - Γ is $\mathcal{T}$-consistent iff $\Gamma \not\models_\mathcal{T} \bot$
- $\varphi \in \mathbb{F}^\Sigma$ is $\mathcal{T}$-valid iff $\emptyset \models_\mathcal{T} \varphi$, i.e., every $\mathcal{M} \in \mathcal{T}$ satisfies φ
  - A $\mathcal{T}$-valid clause $c = \ell_1 \vee \cdots \vee \ell_k$ is called theory lemma
  - φ is $\mathcal{T}$-consistent $\iff$ $\neg\varphi$ is not $\mathcal{T}$-valid

Example

- Suppose Σ defined by $\Sigma_0^F = \{a, b, c, d\}, \Sigma_2^F = \{f, g\}, \Sigma_1^P = \{p\}$
- Let $\mathcal{M}_1, \mathcal{M}_2$ be 2 models having universe $\mathcal{P}(\mathbb{Z})$ and such that:
  - $a^{\mathcal{M}_1} = \emptyset, b^{\mathcal{M}_1} = \{2x \mid x \in \mathbb{Z}\}, c^{\mathcal{M}_1} = \{2x+1 \mid x \in \mathbb{Z}\}, d^{\mathcal{M}_1} = \mathbb{Z}$
  - $a^{\mathcal{M}_2} = \{0\}, b^{\mathcal{M}_2} = \{x \in \mathbb{Z} \mid x > 0\}, c^{\mathcal{M}_2} = \{x \in \mathbb{Z} \mid x < 0\}, d^{\mathcal{M}_2} = \mathbb{Z}$
  - $f^{\mathcal{M}_1} = f^{\mathcal{M}_2} = \cup, \quad g^{\mathcal{M}_1} = g^{\mathcal{M}_2} = \cap, \quad p^{\mathcal{M}_1}(X) = p^{\mathcal{M}_2}(X) \Leftrightarrow X = \emptyset$
- Consider theory $\mathcal{T} = \{\mathcal{M}_1, \mathcal{M}_2\}$ and provide example(s) of:
  - A formula $\mathcal{T}$-satisfiable and not atomic
  - A set of ≥ 2 formulas not $\mathcal{T}$-consistent
  - A set of ≥ 2 formulas that $\mathcal{T}$-entails a not $\mathcal{T}$-valid formula
  - A $\mathcal{T}$-lemma of ≥ 3 clauses

Example

- Suppose Σ defined by $\Sigma_0^F = \{a, b, c, d\}, \Sigma_2^F = \{f, g\}, \Sigma_1^P = \{p\}$
- Let $\mathcal{M}_1, \mathcal{M}_2$ be 2 models having universe $\mathcal{P}(\mathbb{Z})$ and such that:
  - $a^{\mathcal{M}_1} = \emptyset, b^{\mathcal{M}_1} = \{2x \mid x \in \mathbb{Z}\}, c^{\mathcal{M}_1} = \{2x + 1 \mid x \in \mathbb{Z}\}, d^{\mathcal{M}_1} = \mathbb{Z}$
  - $a^{\mathcal{M}_2} = \{0\}, b^{\mathcal{M}_2} = \{x \in \mathbb{Z} \mid x > 0\}, c^{\mathcal{M}_2} = \{x \in \mathbb{Z} \mid x < 0\}, d^{\mathcal{M}_2} = \mathbb{Z}$
  - $f^{\mathcal{M}_1} = f^{\mathcal{M}_2} = \cup, \quad g^{\mathcal{M}_1} = g^{\mathcal{M}_2} = \cap, \quad p^{\mathcal{M}_1}(X) = p^{\mathcal{M}_2}(X) \Leftrightarrow X = \emptyset$
- Consider theory $\mathcal{T} = \{\mathcal{M}_1, \mathcal{M}_2\}$ and provide example(s) of:
  - A formula $\mathcal{T}$-satisfiable and not atomic: $p(a) \vee p(g(b, c))$
  - A set of ≥ 2 formulas not $\mathcal{T}$-consistent: $\{p(a), p(d)\}$
  - A set of ≥ 2 formulas that $\mathcal{T}$-entails a not $\mathcal{T}$-valid formula $\{p(a), \neg p(c)\} \models_{\mathcal{T}} (a = g(b, c))$
  - A $\mathcal{T}$-lemma of ≥ 3 clauses: $p(b) \vee p(c) \vee d = f(f(a, b), c)$

Expansion

- We check the $\mathcal{T}$-satisfiability of formulas with quantifier-free variables
  - In other terms, we find a consistent assignment values $\Rightarrow$ variables
- Because no quantification is involved, variables can be seen as "additional constants" not in Σ_0^F
- More generally, given signature Σ we can consider formulas with uninterpreted symbols, i.e., symbols not in Σ
  - Variable $\equiv$ uninterpreted constant $\equiv$ uninterpreted function
- Given Σ-model $\mathcal{M} = \langle M, (\cdot)^{\mathcal{M}} \rangle$ and $\Sigma' \supseteq \Sigma$, an expansion $\mathcal{M}'$ to Σ' of that model is any Σ'-model $\mathcal{M}' = \langle M', (\cdot)^{\mathcal{M}'} \rangle$ such that:
  - $M' = M$
  - $s^{\mathcal{M}'} = s^{\mathcal{M}}$ for each $s \in \Sigma$

Expansion

- Instead of a Σ-theory $\mathcal{T}$, we (sometimes implicitly) consider the theory $\mathcal{T}' = \{\mathcal{M}' \mid \mathcal{M}' \text{ is an expansion of a } \Sigma\text{-model } \mathcal{M}\ \}$
- The ground $\mathcal{T}$-satisfiability problem is determining, given Σ-theory $\mathcal{T}$, the $\mathcal{T}$-satisfiability of ground formulas over a Σ-expansion $\mathcal{T}'$
    - ground formula $\equiv$ formula with no variables: because uninterpreted constants play the role of variables, our formulas are always ground
- Because φ is $\mathcal{T}$-satisfiable $\iff$ $\neg\varphi$ is not $\mathcal{T}$-valid, the ground $\mathcal{T}$-satisfiability problem has a dual validity problem
    - E.g., $x > 5$ satisfiable iff $x \leq 5$ not valid

Example

- Let's take Σ as $\Sigma_0^F = \{a, b, c, d\}, \Sigma_1^F = \{f, g\}, \Sigma_2^P = \{p\}$ and a Σ-model $\mathcal{M} = \langle [0, 2\pi), (\cdot)^{\mathcal{M}} \rangle$ s.t.
  - $a^{\mathcal{M}} = 0, b^{\mathcal{M}} = \frac{\pi}{2}, c^{\mathcal{M}} = \pi, d^{\mathcal{M}} = \frac{3}{2}\pi,$
  - $f^{\mathcal{M}} = \sin$, $g^{\mathcal{M}} = \cos$, $p^{\mathcal{M}}(x, y) \Leftrightarrow x > y$
- By expanding Σ with uninterpreted constants $x, y, z, \ldots$ we can check the satisfiability of arbitrarily complex ground formulas
  - $p(f(y), g(g(d))) \vee p(a, f(b))$, $g(x) \iff g(c) \wedge f(g(z))$, $\ldots$
- E.g., is $p(g(x), f(d))$ is $\mathcal{M}$-satisfiable?
  - Let $\Sigma' = \Sigma \cup \{x\}$, and expansion $\mathcal{M}'$ of $\mathcal{M}$ s.t. $x^{\mathcal{M}'} = \frac{1}{2}\pi$
  - $p^{\mathcal{M}'}(g(x), f(d)) \equiv g^{\mathcal{M}}(x^{\mathcal{M}'}) > f^{\mathcal{M}}(d^{\mathcal{M}}) \equiv \cos(\frac{1}{2}\pi) > \sin(\frac{3}{2}\pi) \equiv 0 > -1 \equiv$ *true*

Axiomatic definition

- A theory can be defined axiomatically
- A (minimal) set of formulas $\Lambda \subseteq \mathbb{F}^{\Sigma}$ called axioms is given, and the corresponding theory is the set of all models of Λ
  - That is, $\mathcal{T}_{\Lambda} = \{\mathcal{M} \mid \forall \varphi \in \Lambda : \varphi^{M} = \mathit{true}\}$
- E.g., Peano axioms. Given Σ with constant 0 e unary function S:
  - $(\forall x)\ \neg(S(x) = 0)$
  - $(\forall x)(\forall y)\ S(x) = S(y) \rightarrow x = y$
  - $(\varphi(0) \wedge (\forall x)(\varphi(x) \rightarrow \varphi(S(x))) \rightarrow (\forall x)\ \varphi(x) \quad$ for any $\varphi \in \mathbb{F}^{\Sigma}$
- A theory satisfying Peano axioms is called arithmetic theory
- By adding $+$ and $*$, we can prove many arithmetic theorems
  - But not all of them! See Gödel incompleteness theorems

FOL Theories vs SMT Theories

- Naming alert!
- In FOL, a theory $\mathcal{T}$ is defined as a set of formulas that is closed under logical deduction: $\Gamma \models \varphi$ implies that $\varphi \in \mathcal{T}$ for each $\Gamma \subseteq \mathcal{T}$
  - E.g., the theory of arithmetic consists of all formulas derivable from the Peano axioms
- In SMT, a theory is defined as a set of models interpreting a given FOL signature in a domain-specific way.
  - SMT solvers check satisfiability of formulas w.r.t. a given theory $\mathcal{T}$: a formula ϕ is $\mathcal{T}$-satisfiable if there is a model $\mathcal{M} \in \mathcal{T}$ such that $\mathcal{M} \models \varphi$

Many-sorted logic

- Most of SMT applications involve different data types or sorts
- It may be convenient to formalize SMT problems with a many-sorted FOL having:
  - a set of sort symbols $\mathcal{S}$, representing different domains
  - a set of sorted variables uniquely associated with a sort $\sigma \in \mathcal{S}$
  - a sorted signature Σ including a set $\Sigma^{\mathcal{S}} \subseteq \mathcal{S}$ of sort symbols
  - corresponding semantics for variables and sorted signatures...
- ...Let's just assume that sort $\approx$ type without adding new formalism

Some theories of interest

Theories of interest

Let's have an overview of some SMT theories of interest:

- Uninterpreted functions
- Arithmetic
- Arrays
- Bit-vectors
- Strings

EUF Theory

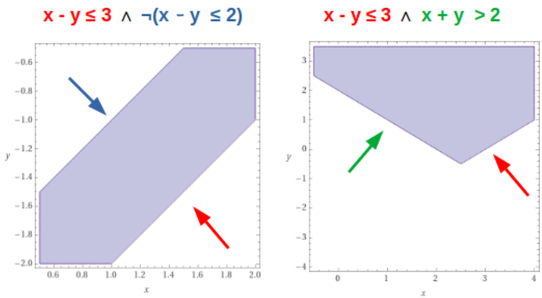
- EUF = Equality with Uninterpreted Functions theory ($\mathcal{T}_{EUF}$)
- No restrictions on how Σ-symbols interpretation, only the congruence closure property enforced: $\mathbf{x} = \mathbf{y} \Rightarrow f(\mathbf{x}) = f(\mathbf{y})$
  - for each $\mathbf{x} = (x_1, \ldots, x_k), \mathbf{y} = (y_1, \ldots, y_k)$ and $f \in \Sigma_k^F$
  - a.k.a. empty theory: its set of axioms is $\emptyset$
    - Why congruence closure cannot be expressed as a FOL axiom?
- Why $\mathcal{T}_{EUF}$? To abstract complex or "black-box" functions
- E.g., consider $a * (f(b) + f(c)) = d \wedge b * (f(a) + f(c)) \neq d \wedge a = b$
  - We don't need any arithmetic theory to prove it unsatisfiable!
- Just abstract $+$ and $*$ with fresh uninterpreted functions g and h: $h(a, g(f(b), f(c))) = d \wedge h(b, g(f(a), f(c))) \neq d \wedge a = b$

Arithmetic Theories

- Theory over numbers are clearly very used and useful
- Let $\Sigma \equiv (0, 1, +, -, \leq)$ and $\mathcal{T}_{\mathcal{Z}}$ interpreting Σ symbols in the usual way. $\mathcal{T}_{\mathcal{Z}}$ is a.k.a. Presburger arithmetic
  - We can define $\mathcal{T}_{\mathcal{R}}$ interpreting Σ symbols over reals
- The satisfiability of ground formulas for $\mathcal{T}_{\mathcal{Z}}$ and $\mathcal{T}_{\mathcal{R}}$ is decidable
  - There exist procedures to decide if a formula is true/false
- Ground satisfiability in $\mathcal{T}_{\mathcal{R}}$ is decidable in polynomial time
  - Simplex method is exponential but works fine in practice
- $\mathcal{T}_{\mathcal{Z}}$-satisfiability is harder: NP-complete in general

Arithmetic Theories

- $\mathcal{T}_{\mathbb{Z}}$ fragments have more efficient decision procedures, e.g.:
- Difference logic: every atom must be $x - y \bowtie k$ with $\bowtie \in \{=, \leq\}$, x, y variables and k integer
- UTPVI ("unit two variable per inequality") every atom $x \pm y \bowtie k$



Arithmetic Theories

- Things are much harder with multiplication:
  - The integer case becomes undecidable (*Peano arithmetic*)
  - The real case becomes doubly-exponential
- Another non-trivial case is the floating-point arithmetic, e.g.
  - For $\mathcal{T}_{\mathcal{Z}}$ and $\mathcal{T}_{\mathcal{R}}$, $(x+y)+z = x+(y+z)$ is valid (*associativity*)
  - For IEEE754 floating points, this is no longer true! E.g. for a sound floating-point model $\mathcal{M}$ s.t. $x^{\mathcal{M}} = 1, y^{\mathcal{M}} = 10^{100}, z^{\mathcal{M}} = -10^{100}$ we have $((x+y)+z)^{\mathcal{M}'} = \cdots = 0 \neq 1 = \cdots = x+(y+z)^{\mathcal{M}'}$
    - *"Catastrophic cancellation"*
  - *Why?*
- For floating points, $+$ and $\cdot$ are still commutative but not necessarily associative nor distributive

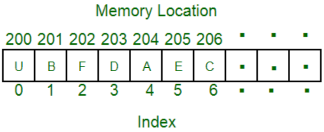
Arithmetic Theories

```python
x = 1
y = 1e100 # 10^100
z = -1e100 # 10^-100
print ("x =", x)
print ("y =", y)
print ("z =", z)
print ("x + y =", x + y)
print ("(x + y) + z =", (x + y) + z)
print ("x + (y + z) =",  x + (y + z))
```

```
x = 1
y = 1e+100
z = -1e+100
x + y = 1e+100
(x + y) + z = 0.0
x + (y + z) = 1.0
```

Theory of Arrays

- Arrays are homogeneous and indexed collections of elements



- Let $\Sigma_{\mathcal{A}}$ be a signature with 2 interpreted functions read and write:
    - $read(a, i)$ returns the value of $a[i]$
    - $write(a, i, v)$ returns the array obtained by replacing $a[i]$ with v

- The theory of array $\mathcal{T}_{\mathcal{A}}$ (with extensionality) is defined by:
    - (i) $(\forall a)(\forall i)(\forall v)\ read(write(a, i, v), i) = v$
    - (ii) $(\forall a)(\forall i)(\forall j)(\forall v)\ i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$
    - (iii) $(\forall a)(\forall a')\ ((\forall i)\ read(a, i) = read(a', i)) \rightarrow a = a' \qquad (extensionality)$

Theory of Arrays

- Exercise: is the following formula $\mathcal{T}_{\mathcal{A}}$-satisfiable?

$$
\begin{gathered}
write(a, i, x) \neq b \ \wedge \ a = b \ \wedge \ i = j \ \wedge \\
read(b, i) = y \ \wedge \ read(write(b, i, x), j) = y
\end{gathered}
$$

- Hint: remember the axioms:
  - (i) $(\forall a)(\forall i)(\forall v)\ read(write(a, i, v), i) = v$
  - (ii) $(\forall a)(\forall i)(\forall j)(\forall v)\ i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$
  - (iii) $(\forall a)(\forall a')\ ((\forall i)\ read(a, i) = read(a', i)) \rightarrow a = a'$

Theory of Arrays

- Exercise: is the following formula $\mathcal{T}_{\mathcal{A}}$-satisfiable?

$$
\begin{array}{c}
write(a, i, x) \neq b \ \wedge \ a = b \ \wedge \ i = j \ \wedge \\
read(b, i) = y \ \wedge \ read(write(b, i, x), j) = y
\end{array}
$$

- Hint: remember the axioms:

  (i) $(\forall a)(\forall i)(\forall v)\ read(write(a, i, v), i) = v$

  (ii) $(\forall a)(\forall i)(\forall j)(\forall v)\ i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$

  (iii) $(\forall a)(\forall a')\ ((\forall i)\ read(a, i) = read(a', i)) \rightarrow a = a'$

Theory of Arrays

- Exercise: is the following formula $\mathcal{T}_{\mathcal{A}}$-satisfiable?

$$write(a, i, x) \neq a \ \wedge \ read(a, i) = y \ \wedge \ read(write(a, i, x), i) = y$$

- Hint: remember the axioms:

(i) $(\forall a)(\forall i)(\forall v)\ read(write(a, i, v), i) = v$
(ii) $(\forall a)(\forall i)(\forall j)(\forall v)\ i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$
(iii) $(\forall a)(\forall a')\ ((\forall i)\ read(a, i) = read(a', i)) \rightarrow a = a'$

Theory of Arrays

- Exercise: is the following formula $\mathcal{T}_{\mathcal{A}}$-satisfiable?

$$\mathit{write}(a, i, x) \neq a \ \wedge \ \mathit{read}(a, i) = y \ \wedge \ x = y$$

- Hint: remember the axioms:
  (i) $(\forall a)(\forall i)(\forall v) \ \mathit{read}(\mathit{write}(a, i, v), i) = v$
  (ii) $(\forall a)(\forall i)(\forall j)(\forall v) \ i \neq j \rightarrow \mathit{read}(\mathit{write}(a, i, v), j) = \mathit{read}(a, j)$
  (iii) $(\forall a)(\forall a') \ ((\forall i) \ \mathit{read}(a, i) = \mathit{read}(a', i)) \rightarrow a = a'$

Theory of Arrays

- Exercise: is the following formula $\mathcal{T}_{\mathcal{A}}$-satisfiable?

$$write(a, i, x) \neq a \ \wedge \ read(a, i) = x$$

- Hint: remember the axioms:

(i) $(\forall a)(\forall i)(\forall v) \ read(write(a, i, v), i) = v$
(ii) $(\forall a)(\forall i)(\forall j)(\forall v) \ i \neq j \rightarrow read(write(a, i, v), j) = read(a, j)$
(iii) $(\forall a)(\forall a') \ ((\forall i) \ read(a, i) = read(a', i)) \rightarrow a = a'$

- Let $a' = write(a, i, x)$. Then, $a' \neq a \xrightarrow{(iii)} (\exists j) \ read(a', j) \neq read(a, j)$. We can prove the unsatisfiability by applying case splitting:
  - $i \neq j \xrightarrow{(ii)} read(a', j) = read(a, j)$ contradicting $read(a', j) \neq read(a, j)$
  - $i = j \longrightarrow read(a', i) \neq read(a, i) \xrightarrow{(i)} x \neq read(a, i)$ contradicting $read(a, i) = x$

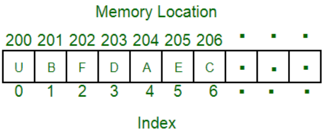
SMT-LIB Encoding

```
; Signature expansion.
(declare-fun a () (Array Int Real))
(declare-fun b () (Array Int Real))
(declare-fun i () Int)
(declare-fun j () Int)
(declare-fun x () Real)
(declare-fun y () Real)
; Formulas. select = read, store = write
(assert (not (= (store a i x) b)))
(assert (= (select b i) y))
(assert (= (select (store b i x) j) y))
(assert (= a b))
(assert (= i j))
; Checking satisfiability.
(check-sat)
```

arrays.smt2

Theory of Arrays

- The full $\mathcal{T}_{\mathcal{A}}$ theory is undecidable, but there are decidable fragments
  - Undecidability comes from universal quantifiers
- Useful theory for SW/HW verification
- In particular, arrays often used to abstract memory locations
  - Main advantage: the abstraction depends on the number of accesses to the memory rather than its actual size

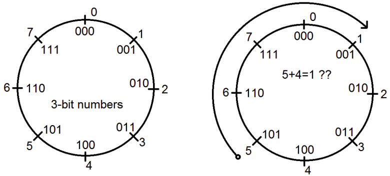


Theory of bit-vectors

- The theory of bit-vectors $\mathcal{T}_{\mathcal{BV}}$ naturally handles verification of programs and circuits
- $\mathcal{T}_{\mathcal{BV}} \neq \mathcal{T}_{\mathcal{A}}$: constants of $\mathcal{T}_{\mathcal{BV}}$ are vectors of bits with fixed length
- Typical operations: string-like (selection, slicing, concatenation, ...), logical (*bit-wise* NOT, OR, AND...), arithmetic $(+, -, \cdot, \ldots)$
- Straightforward reduction to SAT (bit-blasting)
- Exercise: if a, b, c are bit-vectors is $a[0:1] \neq b[0:1] \wedge (a \mid b) = c \wedge c[0] = 0 \wedge a[1] + b[1] = 0$ a $\mathcal{T}_{\mathcal{BV}}$-satisfiable formula?

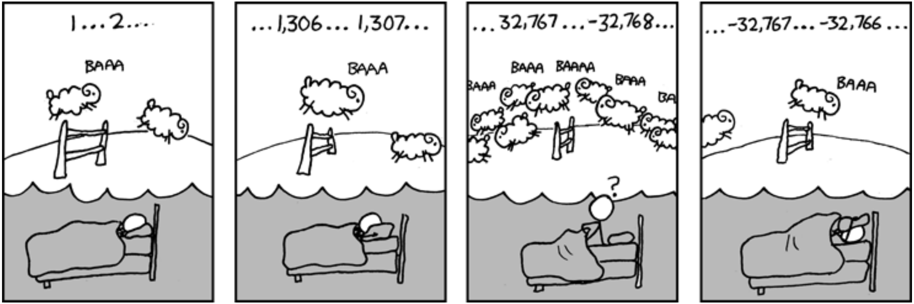
Theory of bit-vectors

- Bit-vectors better than integers or reals to model machine operations
- E.g., $x = 200 \wedge y = x + 100 \wedge y > x$ with x, y unsigned 8-bit integers
  - The formula is valid if we consider *"classical"* arithmetic theories
  - But machines operate differently! 8-bit unsigned integers are enclosed in $[0, 2^8 - 1] = [0, 255]$ so $y = x + 100 = 300$ is out of range
- In these cases typically $y = (x + 100) \mod 2^8 = 44$ so $y < x$
  - "wraparound"



Wraparound

- This also applies for signed integers, e.g., a 16-bit signed int can only be in $[-2^{15}, 2^{15} - 1] = [-32768, 32767]$ so e.g. $32767 + 3 = -32766$



`https://imgs.xkcd.com/comics/cant_sleep.png`

Theory of strings

- Over the last years the need for theory of strings emerged
  - Strings can enable vulnerabilities, especially in web programs
  - `https://mosca2023.github.io`
- Before, string solving typically handled with automata or bit-vectors
  - Automata limit the expressiveness and may be inefficient
  - Bit-vectors impose a fixed limit on string length
- Theory of strings can handle complex operations on unbounded-length strings natively, often in conjunction with other theories
  - E.g. arithmetic theory for string length, or regular expressions
- Some CP proposals too (over bounded-length strings)
  - `https://github.com/ArathaJS/aratha`

Theory of strings

- The theory of word equations is fundamental for string solving
- Fixed an alphabet $\mathcal{S}$, a word equation has form $L = R$ with L, R are concatenations of (uninterpreted) string constants
  - In other terms, L, R concatenate string variables and strings of $\mathcal{S}^*$
  - E.g. $X \cdot world \cdot Z = hello \cdot Y$
- The general theory of word equations is undecidable
  - Equivalent to arithmetic theory
- The quantifier-free theory of word equations is decidable
- Exercise: are the following formulas satisfiable?
  - $XY = YX \;\wedge\; X \neq Y$
  - $aX = Xb \;\wedge\; a \neq b$

Theory of strings

- The theory of word equations is fundamental for string solving
- Fixed an alphabet $\mathcal{S}$, a word equation has form $L = R$ with L, R are concatenations of (uninterpreted) constants
  - In other terms, L, R concatenate string variables and strings of $\mathcal{S}^*$
  - E.g. $X \cdot world \cdot Z = hello \cdot Y$
- The general theory of word equations is undecidable
  - Equivalent to arithmetic theory
- The quantifier-free theory of word equations is decidable
- Exercise: are the following formulas satisfiable?
  - $XY = YX \ \wedge \ X \neq Y$ $\quad X = \epsilon, Y = a$
  - $aX = Xb \ \wedge \ a \neq b$ $\quad X$ starts with a and ends with b, so $X = aX_1b$, thus $aaX_1b = aX_1bb$, $aaaX_2bb = aaX_2bbb, \ldots, a^{k+1}X_kb^k = a^kX_kb^{k+1}$ with $|X_k| < |a| + |b|$: unsat because both a (as prefix) and b (suffix) should fit into X_k

SMT in practice

- In practice, theories not isolated: real-world applications often need a combination of arithmetic, strings, arrays, ... e.g.
    - $a = b + 2 \wedge A = write(B, a, 4) \wedge (read(A, b + 3) = 2 \vee f(a - 1) \neq f(b))$
    - $aX = Yb \ \wedge \ X \in \mathcal{L}(d \mid c^* ab) \ \wedge \ |Y| = 2 \cdot |X|$

- The goal is to efficiently combine decision procedures for each theory
    - Efficient procedures already exist for many theories of interest

- Decidability issues
    - ...But we can always restrict to fragments

Take-home messages

- SMT extends SAT to solve formulas in (quantifier-free) FOL
  - functions, (constants), predicates with arity > 1
- Similar/orthogonal to CP, it tackles combinatorial problems from a "more logical" perspective (formulas)
  - More oriented to problems derived from software analysis
  - Less optimization-oriented
- Eventually, SMT solving eagerly or lazily relies on SAT solving
  - SAT : machine language $\approx$ SMT : higher-level language
- Several theories of interest developed and studied over last decades
  - EUF, arithmetic, arrays, bit-vectors, strings, ...

Resources

- Handbook of Satisfiability – Chapter 12 "*Satisfiability Modulo Theories*" by C. Barrett, R. Sebastiani, S.A. Seshia, C. Tinelli
  - Search "Satisfiability Modulo Theories - EECS at UC Berkeley"
- Barrett, Clark, and Cesare Tinelli. "Satisfiability modulo theories." Handbook of model checking. Springer, Cham, 2018. 305-343.
- SAT/SMT schools
  - `https://sat-smt.in/`
- ...