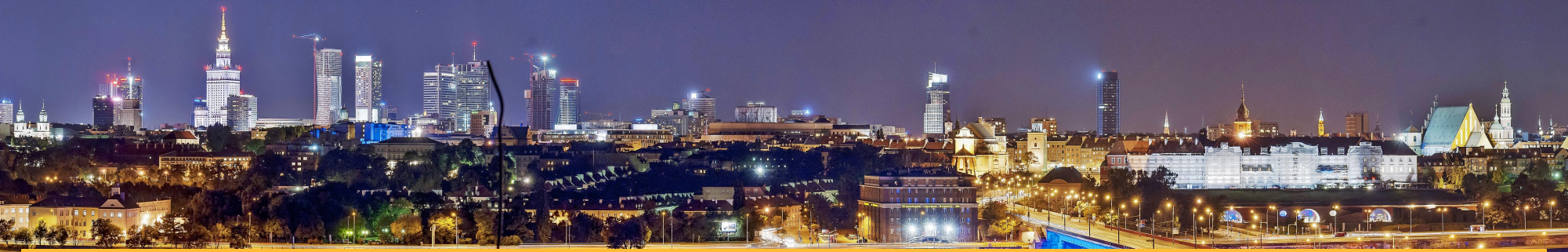


Introduction to Software Verification

Vincent Penelle

<vpenelle@u-bordeaux.fr>

LaBRI, Université de Bordeaux

September 16, 2019

— SAT and SMT —

Propositional Logic: Booleans

Booleans

$\{\text{True}, \text{False}\}$ OR $\{\top, \bot\}$ OR $\{0, 1\}$

And

$\wedge$	$\bot$	$\bot$
$\bot$	$\bot$	$\bot$
$\top$	$\bot$	$\top$

Or

$\vee$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$
$\top$	$\top$	$\top$

Not

	$\neg$
$\bot$	$\top$
$\top$	$\bot$

Propositional Logic: Formulæ

Formula Grammar

$$\phi ::= x \mid \neg\phi \mid \phi \vee \phi' \mid \phi \wedge \phi'$$

Examples:

- $p \vee \neg p$
- $\neg(p \wedge q)$
- $(p \wedge q) \vee (\neg p \wedge \neg q)$
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$

Propositional Logic: Formulæ with added (syntactic) sugar

Formula Grammar

$$\phi ::= x \mid \neg\phi \mid \phi \vee \phi' \mid \phi \wedge \phi' \mid \phi \Rightarrow \phi' \mid \phi \Leftrightarrow \phi'$$

Examples:

- $p \vee \neg p$
- $\neg(p \wedge q)$
- $(p \wedge q) \vee (\neg p \wedge \neg q)$ or $p \Leftrightarrow q$
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$ or $(p \Rightarrow q) \wedge (q \Rightarrow r) \wedge (r \Rightarrow p)$
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$ or $p \Rightarrow (q \wedge \neg r) \wedge \neg r \Rightarrow \neg p$

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

ϕ is satisfiable if and only if $\neg\phi$ is not valid.

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Examples:

- $p \vee \neg p$
- $\neg(p \wedge q)$
- $(p \wedge q) \vee (\neg p \wedge \neg q)$
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Examples:

- $p \vee \neg p$ Valid
- $\neg(p \wedge q)$
- $(p \wedge q) \vee (\neg p \wedge \neg q)$
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Examples:

- $p \vee \neg p$ Valid
- $\neg(p \wedge q)$ Satisfiable: $p = \bot, q = \bot$.
- $(p \wedge q) \vee (\neg p \wedge \neg q)$
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Examples:

- $p \vee \neg p$ Valid
- $\neg(p \wedge q)$ Satisfiable: $p = \bot, q = \bot$.
- $(p \wedge q) \vee (\neg p \wedge \neg q)$ Satisfiable: $p = \top, q = \top$.
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Examples:

- $p \vee \neg p$ Valid
- $\neg(p \wedge q)$ Satisfiable: $p = \bot, q = \bot$.
- $(p \wedge q) \vee (\neg p \wedge \neg q)$ Satisfiable: $p = \top, q = \top$.
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$ Satisfiable: $p = q = r = \top$.
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$

Propositional Logic: Satisfiability and validity

Satisfiability: $\nu \models \phi$

A formula ϕ is **satisfiable** if a assignment of variables ν makes it true.

Validity

A formula ϕ is **valid** if all assignments of variables makes it true.

Examples:

- $p \vee \neg p$ Valid
- $\neg(p \wedge q)$ Satisfiable: $p = \bot, q = \bot$.
- $(p \wedge q) \vee (\neg p \wedge \neg q)$ Satisfiable: $p = \top, q = \top$.
- $(\neg p \vee q) \wedge (\neg q \vee r) \wedge (\neg r \vee p)$ Satisfiable: $p = q = r = \top$.
- $(\neg p \vee (q \wedge \neg r)) \wedge (r \vee \neg p)$ Unsatisfiable.

SAT

SAT is NP-complete

Problem **SAT**:

Input: A propositional formula ϕ.

Output: Is ϕ satisfiable?

SAT is **NP-complete**.

It means that **verifying** if a given valuation ν satisfies ϕ is polynomial, but **finding** such a ν given ϕ is not (a priori).

SAT

SAT is NP-complete

Problem **SAT**:

Input: A propositional formula ϕ.

Output: Is ϕ satisfiable?

SAT is **NP-complete**.

It means that **verifying** if a given valuation ν satisfies ϕ is polynomial, but **finding** such a ν given ϕ is not (a priori).

Hopefully, there are good heuristics which allow to solve efficiently satisfiability problems up to a not too ridiculous size.
Examples: Glucose (developed in LaBRI), MiniSAT, Lingeling, Sat4j, etc...

SAT

SAT is NP-complete

Problem **SAT**:

Input: A propositional formula ϕ.

Output: Is ϕ satisfiable?

SAT is **NP-complete**.

It means that **verifying** if a given valuation ν satisfies ϕ is polynomial, but **finding** such a ν given ϕ is not (a priori).

Hopefully, there are good heuristics which allow to solve efficiently satisfiability problems up to a not too ridiculous size.
Examples: Glucose (developed in LaBRI), MiniSAT, Lingeling, Sat4j, etc...

Therefore, modelling problems and programs in propositional logic, and check the satisfiability of the obtained formula with a SAT solver to verify a program!

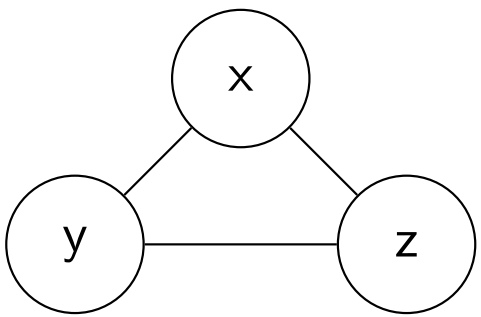
How does it work?

Formulæ must be in CNF (not obvious) $\rightarrow$ Exercise.
Explain Unit propagation and split rules. Say it is Davis, Putnam, Logemann, Loveland algorithm.

Modelling with SAT

Can I color a graph with **two** colors such that no two adjacent nodes are the same color?

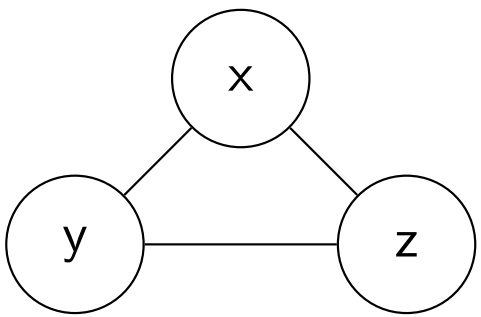
One variable per node: x stands for "x in red", $\neg x$ for "x in green".



Modelling with SAT

Can I color a graph with **two** colors such that no two adjacent nodes are the same color?

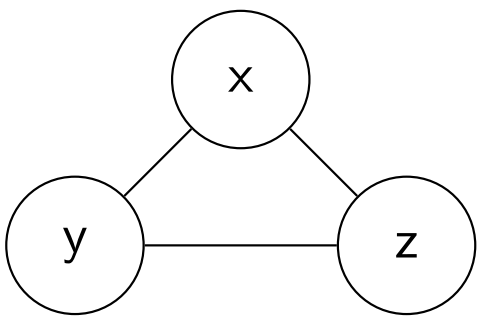
One variable per node: x stands for "x in red", $\neg x$ for "x in green".



Each node has one color: $\neg(x \wedge \neg x)$.

Modelling with SAT

Can I color a graph with **two** colors such that no two adjacent nodes are the same color?

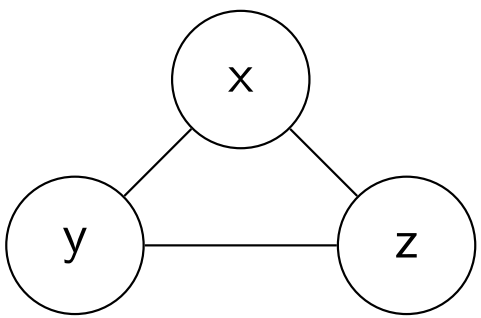


One variable per node: x stands for "x in red", $\neg x$ for "x in green".

Each node has one color: $\neg(x \wedge \neg x)$. **True**

Modelling with SAT

Can I color a graph with **two** colors such that no two adjacent nodes are the same color?



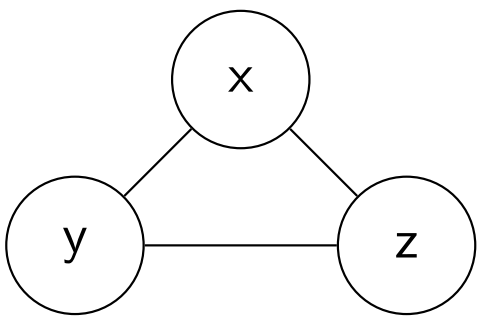
One variable per node: x stands for "x in red", $\neg x$ for "x in green".

Each node has one color: $\neg(x \wedge \neg x)$. **True**

Each edge has different colors for its ends: $\neg(x \wedge y) \wedge \neg(\neg x \wedge \neg y)$.

Modelling with SAT

Can I color a graph with **two** colors such that no two adjacent nodes are the same color?



One variable per node: x stands for "x in red", $\neg x$ for "x in green".

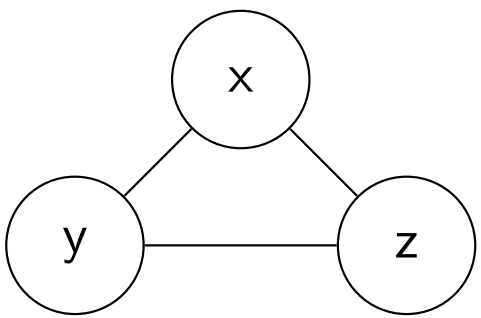
Each node has one color: $\neg(x \wedge \neg x)$. **True**

Each edge has different colors for its ends: $\neg(x \wedge y) \wedge \neg(\neg x \wedge \neg y)$.

Total formula: $\neg(x \wedge y) \wedge \neg(\neg x \wedge \neg y) \wedge \neg(x \wedge z) \wedge \neg(\neg x \wedge \neg z) \wedge \neg(z \wedge y) \wedge \neg(\neg z \wedge \neg y)$.

Modelling with SAT

Can I color a graph with **two** colors such that no two adjacent nodes are the same color?



One variable per node: x stands for "x in red", $\neg x$ for "x in green".

Each node has one color: $\neg(x \wedge \neg x)$. **True**

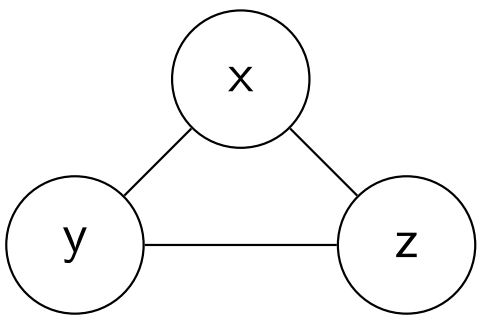
Each edge has different colors for its ends: $\neg(x \wedge y) \wedge \neg(\neg x \wedge \neg y)$.

Total formula: $\neg(x \wedge y) \wedge \neg(\neg x \wedge \neg y) \wedge \neg(x \wedge z) \wedge \neg(\neg x \wedge \neg z) \wedge \neg(z \wedge y) \wedge \neg(\neg z \wedge \neg y)$.
Unsatisfiable!

Modelling with SAT (bis)

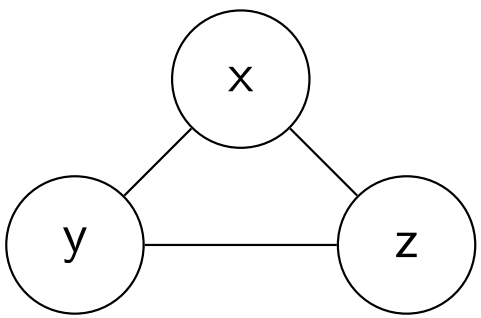
Can I color a graph with **three** colors such that no two adjacent nodes are the same color?

One variable per node: x stands for "x in red", x for "x in green" and x stands for x in blue".



Modelling with SAT (bis)

Can I color a graph with **three** colors such that no two adjacent nodes are the same color?

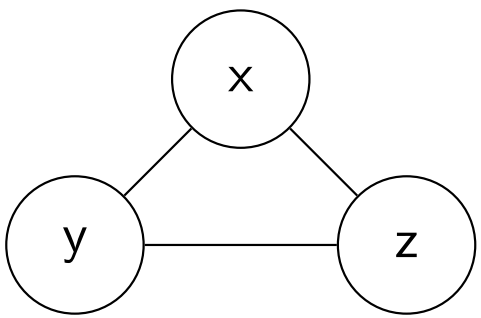


One variable per node: x stands for "x in red", x for "x in green" and x stands for x in blue".

Each node has exactly one color:
$(x \vee x \vee x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x)$.

Modelling with SAT (bis)

Can I color a graph with **three** colors such that no two adjacent nodes are the same color?



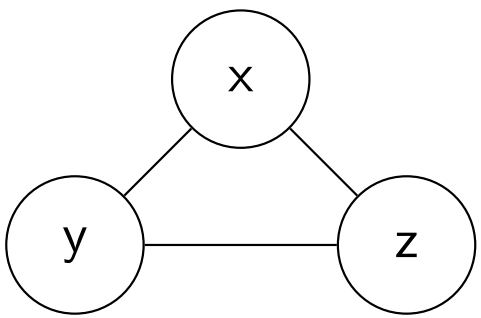
One variable per node: x stands for "x in red", x for "x in green" and x stands for x in blue".

Each node has exactly one color:
$(x \vee x \vee x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x)$.

Each edge has different colors for its ends:
$\neg(x \wedge y) \wedge \neg(x \wedge y) \wedge \neg(x \wedge y)$.

Modelling with SAT (bis)

Can I color a graph with **three** colors such that no two adjacent nodes are the same color?



One variable per node: x stands for "x in red", x for "x in green" and x stands for x in blue".

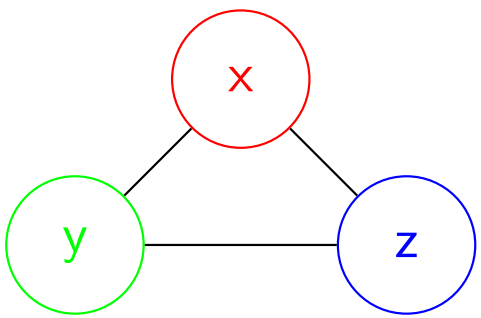
Each node has exactly one color:
$(x \vee x \vee x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x)$.

Each edge has different colors for its ends:
$\neg(x \wedge y) \wedge \neg(x \wedge y) \wedge \neg(x \wedge y)$.

Total formula: a bit big, but straightforward

Modelling with SAT (bis)

Can I color a graph with **three** colors such that no two adjacent nodes are the same color?



One variable per node: x stands for "x in red", x for "x in green" and x stands for x in blue".

Each node has exactly one color:
$(x \vee x \vee x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x) \wedge (\neg x \vee \neg x)$.

Each edge has different colors for its ends:
$\neg(x \wedge y) \wedge \neg(x \wedge y) \wedge \neg(x \wedge y)$.

Total formula: a bit big, but straightforward
Satisfiable!

Exercises

Here put some modelling exercises.

Quantifier-free First Order Logic over a universe

We consider a set $\mathcal{U}$ called **universe**.

Predicate

A n-ary predicate over $\mathcal{U}$ is an application from $\mathcal{U}^n$ to $\{\top, \bot\}$.

Quantifier-free FO($\mathcal{U}$)

$$\phi ::= P(e_1, \cdots, e_{|P|}) \mid \neg\phi \mid \phi \vee \phi' \mid \phi \wedge \phi'$$

where $e_1, \cdots, e_{|P|}$ are **expressions** over $\mathcal{U}$ (depends on $\mathcal{U}$).

Examples:

- Graphs: $(Red(x) \wedge Edge(x, y) \wedge Reachable(y, z)) \Rightarrow (Blue(z) \wedge \neg Edge(z, x))$
- Arithmetic: $= (+(3, \times(x, 5)), \times(y, /(z, 20)) \Rightarrow \leq (+(z, y), x)$

Quantifier-free First Order Logic over a universe

We consider a set $\mathcal{U}$ called **universe**.

Predicate

A n-ary predicate over $\mathcal{U}$ is an application from $\mathcal{U}^n$ to $\{\top, \bot\}$.

Quantifier-free FO($\mathcal{U}$)

$$\phi ::= P(e_1, \cdots, e_{|P|}) \mid \neg\phi \mid \phi \vee \phi' \mid \phi \wedge \phi'$$

where $e_1, \cdots, e_{|P|}$ are **expressions** over $\mathcal{U}$ (depends on $\mathcal{U}$).

Examples:

- Graphs: $(Red(x) \wedge Edge(x, y) \wedge Reachable(y, z)) \Rightarrow (Blue(z) \wedge \neg Edge(z, x))$
- Arithmetic: $3 + 5 \times x = y \times (z/20) \vee (z + y) \Rightarrow x$

First Order Logic over a universe

We consider a set $\mathcal{U}$ called **universe**.

Predicate

A n-ary predicate over $\mathcal{U}$ is an application from $\mathcal{U}^n$ to $\{\top, \bot\}$.

FO($\mathcal{U}$)

$$\phi ::= P(e_1, \cdots, e_{|P|}) \mid \neg\phi \mid \phi \vee \phi' \mid \phi \wedge \phi' \mid \exists x, \phi \mid \forall x, \phi$$

where $e_1, \cdots, e_{|P|}$ are **expressions** over $\mathcal{U}$ (depends on $\mathcal{U}$).

Examples:

- Graphs:
  $\exists x, y, \forall z, (Red(x) \wedge Edge(x, y) \wedge Reachable(y, z)) \Rightarrow (Blue(z) \wedge \neg Edge(z, x))$
- Arithmetic: $\forall x, \exists y, z, 3 + 5 \times x = y \times (z/20) \vee (z + y) \Rightarrow x$

SAT Modulo Theory (SMT)

Problem SMT over a universe $\mathcal{U}$ and a Theory T

Input: A formula ϕ in FO($\mathcal{U}, T$).

Output: Is ϕ satisfiable?

Such a formula may even be over **several universes** (provided variable names do not conflict).
Provided there are algorithms to decide the theory over the universe, a SMT problem is **decidable** may be solved by combining a SAT solver and such algorithms.
For full-FO, however, in general, the SMT problem is undecidable.

SAT Modulo Theory (SMT)

Problem SMT over a universe $\mathcal{U}$ and a Theory T

Input: A formula ϕ in FO($\mathcal{U}, T$).

Output: Is ϕ satisfiable?

Such a formula may even be over **several universes** (provided variable names do not conflict).
Provided there are algorithms to decide the theory over the universe, a SMT problem is **decidable** may be solved by combining a SAT solver and such algorithms.
For full-FO, however, in general, the SMT problem is undecidable.

There are today efficient SMT solvers: Z3 (Microsoft), Alt-Ergo (OcamlPro), Barcelogic, etc.

SAT Modulo Theory (SMT)

Problem SMT over a universe $\mathcal{U}$ and a Theory T

Input: A formula ϕ in FO($\mathcal{U}, T$).

Output: Is ϕ satisfiable?

Such a formula may even be over **several universes** (provided variable names do not conflict).
Provided there are algorithms to decide the theory over the universe, a SMT problem is **decidable** may be solved by combining a SAT solver and such algorithms.
For full-FO, however, in general, the SMT problem is undecidable.

There are today efficient SMT solvers: Z3 (Microsoft), Alt-Ergo (OcamlPro), Barcelogic, etc.

Modelling programs in SMT is easier than with SAT, and allows to range over infinite domains!

In this course, we will use **Z3**.

SAT Modulo Theory (SMT)

Problem SMT over a universe $\mathcal{U}$ and a Theory T

Input: A formula ϕ in FO($\mathcal{U}, T$).

Output: Is ϕ satisfiable?

Such a formula may even be over **several universes** (provided variable names do not conflict).
Provided there are algorithms to decide the theory over the universe, a SMT problem is **decidable** may be solved by combining a SAT solver and such algorithms.
For full-FO, however, in general, the SMT problem is undecidable.

There are today efficient SMT solvers: Z3 (Microsoft), Alt-Ergo (OcamlPro), Barcelogic, etc.

Modelling programs in SMT is easier than with SAT, and allows to range over infinite domains!

In this course, we will use **Z3**.
Note: Z3 supports quantifiers, but doesn't give a decision procedure for FO

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((x + y \leq 0) \vee \neg a) \wedge ((x = 1) \vee b), \;\; a, b \in \mathbb{B}, \;\; x, y \in \mathbb{R} \quad (1)$$

Solving procedure

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((x + y \leq 0) \vee \neg a) \wedge ((x = 1) \vee b), \quad a, b \in \mathbb{B}, \quad x, y \in \mathbb{R} \qquad (1)$$

Solving procedure

- Identify the boolean parts of the formula;

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((x + y \leq 0) \vee \neg a) \wedge ((x = 1) \vee b), \;\; a, b \in \mathbb{B}, \;\; x, y \in \mathbb{R} \quad (1)$$

Solving procedure

- Identify the boolean parts of the formula;
- Identify the arithmetic parts of the formula;

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((\underbrace{x + y \leq 0}_{c}) \vee \neg a) \wedge (\underbrace{(x = 1)}_{d} \vee b), \quad a, b \in \mathbb{B}, \quad x, y \in \mathbb{R} \qquad (1)$$
$$c, d \in \mathbb{B}$$

Solving procedure

- Identify the boolean parts of the formula;
- Identify the arithmetic parts of the formula;
- Substitute the QF_LRA clauses by boolean clauses;

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((\underbrace{x + y \leq 0}_{c}) \vee \neg a) \wedge (\underbrace{(x = 1)}_{d} \vee b), \quad a, b \in \mathbb{B}, \quad x, y \in \mathbb{R} \quad (1)$$
$$c, d \in \mathbb{B}$$

Solving procedure

- Identify the boolean parts of the formula;
- Identify the arithmetic parts of the formula;
- Substitute the QF_LRA clauses by boolean clauses;
- Check satisfiability of the formula with a SAT-solver (enumerate all possible boolean models);

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((\underbrace{x + y \leq 0}_{c}) \vee \neg a) \wedge (\underbrace{(x = 1)}_{d} \vee b), \quad \begin{matrix} a, b \in \mathbb{B}, \ x, y \in \mathbb{R} \\ c, d \in \mathbb{B} \end{matrix} \qquad (1)$$

Solving procedure

- Identify the boolean parts of the formula;
- Identify the arithmetic parts of the formula;
- Substitute the QF_LRA clauses by boolean clauses;
- Check satisfiability of the formula with a SAT-solver (enumerate all possible boolean models);
- If some boolean models are found satisfiable, check if the model is reachable in the QF_LRA logic (*e.g.* using Simplex);

How does it work?

SMT-solvers combines the efficiency of a SAT-solver and theory-specific decision procedure. For example, the following formula combine propositional logic with unquantified linear real arithmetic (QF_LRA):

$$a \wedge ((\underbrace{x + y \leq 0}_{c}) \vee \neg a) \wedge (\underbrace{(x = 1)}_{d} \vee b), \quad \underbrace{a, b \in \mathbb{B}}_{c,\, d \in \mathbb{B}}, \quad x, y \in \mathbb{R} \qquad (1)$$

Solving procedure

- Identify the boolean parts of the formula;
- Identify the arithmetic parts of the formula;
- Substitute the QF_LRA clauses by boolean clauses;
- Check satisfiability of the formula with a SAT-solver (enumerate all possible boolean models);
- If some boolean models are found satisfiable, check if the model is reachable in the QF_LRA logic (*e.g.* using Simplex);
- Return the result.

How does it work?

As any modern boolean SAT solvers most tools are based on the **Davis-Putnam** and **Davis-Logemann-Loveland** (DPLL) procedures:

- Input formula is in Conjunctive Normal Form (CNF);
- Solver combines search with backtracking and deduction.

However, it is not enough to connect an efficient DPLL solver and Simplex to get an efficient DPLL(LRA) solver, we need:

- Theory Propagation,
- Do not wait for a complete Boolean model (parallelize),
- Preprocessing,
- Conversion in CNF,
- Theory layering,
- etc.

Core of SMT-solvers: SMT-Lib

SMT-LIB

Started in **2003**, SMT-LIB is an international initiative aimed at facilitating research and development in **Satisfiability Modulo Theories**. It has been created with the expectation that the availability of common standards and a library of benchmarks would greatly facilitate the evaluation and the comparison of SMT systems, and advance the state of the art in the field in the same way as, for instance, the **TPTP** library has done for theorem proving, or the **SATLIB** library has done initially for SAT.

SMT-LIB Goals

- Provide a standard descriptions of background theories used in SMT systems;
- Develop and promote common input and output languages for SMT solvers;
- Establish and make available to the research community a large library of benchmarks for SMT solvers.

SMT-LIB: Supported Theories

Theory versus Logic

- A **theory** defines a vocabulary of sorts and functions, and it associates a sort with relevant literals.
- A **logic** is defined to gather one or more theories, together with some restrictions on the kind of expressions that may be used within the logic.

- **Core**: Core theory, defining the basic Boolean operators.
- **Ints**: Integer numbers.
- **Reals**: Real numbers.
- **Reals_Ints**: Real and integer numbers.
- **Fixed_Size_BitVectors**: Arbitrary fixed-size bit vectors.
- **ArraysEx**: Functional arrays with extensionality.

SMT-LIB: Supported Logics (1/2)

Boolean Logics

QF_UF: Unquantified uninterpreted formulas.

Logic with Arithmetic

QF_NIA: Quantifier-free integer arithmetic.

QF_LIA: Unquantified linear integer arithmetic. Adds linear arithmetic to the QF_UF logic. In essence, Boolean combinations of inequations between linear polynomials over integer variables.

QF_LRA: Unquantified linear real arithmetic. Adds linear arithmetic over the reals to QF_UF logic.

LRA: Linear integer arithmetic. Simply extends QF_LRA by allowing quantifiers.

Logics for Difference Arithmetic

QF_IDL: Difference Logic over the integers without quantifier. In essence, Boolean combinations of inequations of the form $x - y < b$ where x and y are integer variables and b is an integer constant.

QF_RDL: Difference Logic over the reals without quantifier. In essence, Boolean combinations of inequations of the form $x - y < b$ where x and y are real variables and b is a rational constant.

SMT-LIB: Supported Logics (2/2)

Logics with Bit-Vectors and Arrays

QF_BV: Closed quantifier-free formulas over the theory of fixed-size bitvectors.
QF_AX: Closed quantifier-free formulas over the theory of arrays with extensionality.
QF_ABV: Closed quantifier-free formulas over the theory of bitvectors and bitvector arrays.
QF_UFBV: Unquantified formulas over bitvectors with uninterpreted sort function and symbols.
QF_AUFBV: Closed quantifier-free formulas over the theory of bitvectors and bitvector arrays extended with free sort and function symbols.

Composite Logics with Arithmetic

QF_UFLIA: Unquantified linear integer arithmetic with uninterpreted functions.
QF_AUFLIA: Extends QF_LIA linear arithmetic logic, with addition of arrays and arbitrary uninterpreted functions.
QF_UFNRA: Unquantified non-linear real arithmetic with uninterpreted functions.
AUFLIA: Extends QF_AUFLIA by allowing quantifiers.
AUFLIRA: Allows quantifiers and includes Int and Real sorts, but is limited to linear arithmetic.
AUFNIRA: Allows quantifiers and general Int+Real arithmetic, arrays, and uninterpreted functions.