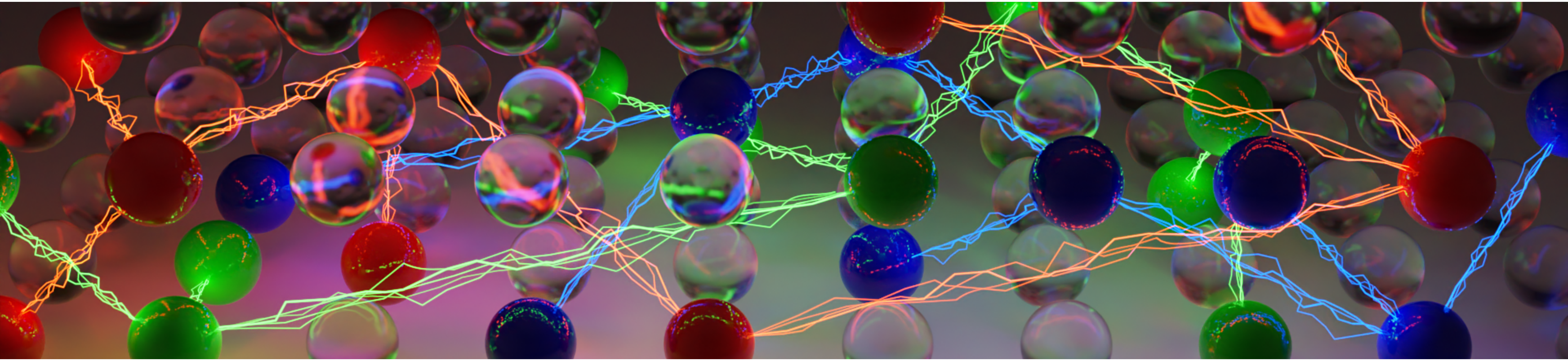


Practical SAT Solving

Lecture 12 – Satisfiability Modulo Theories (SMT)

Markus Iser, Dominik Schreiber | July 21, 2025

Roadmap

- SMT: Motivation and definition
- Some example theories
- Formal framework and decidability
- SMT solving
  - Lazy approach: DPLL(T)
  - Eager approach: The case of Bit Vectors
- (Brief) pragmatics of SMT

Note: This lecture is mostly based on the following slide sets:

https://github.com/biotomas/sat-lecture-kit/blob/main/slides/l10.tex

(motivation, example theories, decidability, DPLL(T) example, bit vectors)

https://resources.mpi-inf.mpg.de/departments/rg1/conferences/vtsa08/slides/barret2_smt.pdf

(formal definitions)

https://alexeyignatiev.github.io/ssa-school-2019/slides/ao-satsmtar19-slides.pdf

(lazy vs. eager, DPLL(T) techniques & properties)

SMT: Motivation

Propositional logic: very low-level for many practical problems

- Linear (integer or real) arithmetic:

$$x + y < 5 \wedge (2x - y > 4 \vee x + y > 7)$$

- Non-linear arithmetic:

$$x^2 + y^2 = 4 \wedge x - y = 3$$

- Arithmetic as actually done by a computer:

$$4294967295 + 1 = 0$$

Natural point of extension: **First Order Logic** with suitable interpretation / semantics

What is SMT?

Satisfiability Modulo Theories (SMT)

Decide the satisfiability of a First Order Logic (FOL) formula with respect to a certain background theory.

- **Syntax:** in most cases, quantifier-free, ground fragment of FOL
  - Set of atomic constants — e.g., 0, 1, `null`
  - Set of k-ary functions $f(x_1, \ldots, x_k)$ $(k \geq 1)$ — e.g., $+$, $\times$, `read`, `write`
    – each x_i is a term, i.e., either a constant or some k'-ary function
  - Set of k-ary propositions $P(x_1, \ldots, x_k)$ — e.g., $=$, $<$
    – $k = 0$: Atom as in propositional logic
    – each x_i is a term
  - Formula: Boolean expression featuring the above propositions as its “variables”

What is SMT?

Satisfiability Modulo Theories (SMT)

Decide the satisfiability of a First Order Logic (FOL) formula with respect to a certain background theory.

- **Syntax:** in most cases, quantifier-free, ground fragment of FOL
  - Set of atomic constants — e.g., 0, 1, `null`
  - Set of k-ary functions $f(x_1, \ldots, x_k)$ $(k \geq 1)$ — e.g., $+$, $\times$, `read`, `write`
    – each x_i is a term, i.e., either a constant or some k'-ary function
  - Set of k-ary propositions $P(x_1, \ldots, x_k)$ — e.g., $=$, $<$
    – $k = 0$: Atom as in propositional logic
    – each x_i is a term
  - Formula: Boolean expression featuring the above propositions as its “variables”
- **Semantics:** depends on chosen background theory
  - Many theories feature equality, i.e., a special proposition $P_=(x, y) \Leftrightarrow x = y$
  - Each theory adds some set of axioms that must hold

Theory: Equality with Uninterpreted Functions (EUF)

- Equality proposition “$=$” comes with some implicit axioms:
  1. Reflexivity: $\forall x : \; x = x$
  2. Symmetry: $\forall x \, \forall y : \; x = y \rightarrow y = x$
  3. Transitivity: $\forall x \, \forall y \, \forall z : \; x = y \land y = z \rightarrow x = z$
  4. Congruence: $\forall k \, \forall f(x_1, \ldots, x_k) \, \forall x_1, \ldots, x_k \, \forall y_1, \ldots, y_k :$
     $\bigwedge_{i=1}^{k} x_i = y_i \rightarrow f(x_1, \ldots, x_k) = f(y_1, \ldots, y_k)$
- Functions are left uninterpreted and thus carry no inherent meaning apart from syntactical footprint

Theory: Equality with Uninterpreted Functions (EUF)

- Equality proposition "$=$" comes with some implicit axioms:
  1. Reflexivity: $\forall x : \; x = x$
  2. Symmetry: $\forall x \, \forall y : \; x = y \rightarrow y = x$
  3. Transitivity: $\forall x \, \forall y \, \forall z : \; x = y \wedge y = z \rightarrow x = z$
  4. Congruence: $\forall k \, \forall f(x_1, \ldots, x_k) \, \forall x_1, \ldots, x_k \, \forall y_1, \ldots, y_k :$
     $\bigwedge_{i=1}^{k} x_i = y_i \rightarrow f(x_1, \ldots, x_k) = f(y_1, \ldots, y_k)$
- Functions are left uninterpreted and thus carry no inherent meaning apart from syntactical footprint
- Examples:
  - $(z \neq x) \wedge (z \neq y)$ — Satisfiable for ≥ 3 objects
  - $h(a, g(f(b), f(c))) = d \wedge h(b, g(f(a), f(c))) \neq d \wedge a = b$ — Unsatisfiable

Theory: Equality with Uninterpreted Functions (EUF)

- Equality proposition “$=$” comes with some implicit axioms:
  1. Reflexivity: $\forall x : \; x = x$
  2. Symmetry: $\forall x \, \forall y : \; x = y \rightarrow y = x$
  3. Transitivity: $\forall x \, \forall y \, \forall z : \; x = y \land y = z \rightarrow x = z$
  4. Congruence: $\forall k \, \forall f(x_1, \ldots, x_k) \, \forall x_1, \ldots, x_k \, \forall y_1, \ldots, y_k :$
     $\bigwedge_{i=1}^{k} x_i = y_i \rightarrow f(x_1, \ldots, x_k) = f(y_1, \ldots, y_k)$
- Functions are left uninterpreted and thus carry no inherent meaning apart from syntactical footprint
- Examples:
  - $(z \neq x) \land (z \neq y)$ — Satisfiable for ≥ 3 objects
  - $h(a, g(f(b), f(c))) = d \land h(b, g(f(a), f(c))) \neq d \land a = b$ — Unsatisfiable
- Useful to abstract away non-supported constructions / operations
- Also called Theory of Equality

Theory: Presburger Arithmetic

Arithmetic over natural numbers with addition only

- Constants: $0, 1$ · Functions: $+$ · Predicates: $=$
- Axioms:
  1. EUF axioms
  2. Null: $\forall x: \; x + 1 \neq 0$
  3. Successor: $\forall x, y: \; x + 1 = y + 1 \rightarrow x = y$
  4. Induction: $P(0) \wedge (\forall x: \; P(x) \rightarrow P(x + 1)) \rightarrow (\forall x: \; P(x))$
  5. Plus Zero: $\forall x: \; x + 0 = x$
  6. Plus successor: $\forall x, y: \; x + (y + 1) = (x + y) + 1$

Theory: Peano Arithmetic

Arithmetic over natural numbers with addition and multiplication

- Constants: $0, 1$ · Functions: $+, \times$ · Predicates: $=$
- Axioms:
  1. EUF axioms
  2. Null: $\forall x : \; x + 1 \neq 0$
  3. Successor: $\forall x, y : \; x + 1 = y + 1 \rightarrow x = y$
  4. Induction: $P(0) \land (\forall x : \; P(x) \rightarrow P(x + 1)) \rightarrow (\forall x : \; P(x))$
  5. Plus Zero: $\forall x : \; x + 0 = x$
  6. Plus successor: $\forall x, y : \; x + (y + 1) = (x + y) + 1$
  7. Times Zero: $\forall x : \; x \times 0 = 0$
  8. Times successor: $\forall x, y : \; x \times (y + 1) = (x \times y) + x$

Theory: Arrays

Basic reasoning over arrays (and memory in general)

- Functions: $\texttt{read}(a, i), \texttt{write}(a, i, v)$ · Predicates: $=$
- Axioms:
  1. EUF axioms
  2. Read over write #1: $\forall a, v, i, j: \; i = j \rightarrow \texttt{read}(\texttt{write}(a, i, v), j) = v$
  2. Read over write #2: $\forall a, v, i, j: \; i \neq j \rightarrow \texttt{read}(\texttt{write}(a, i, v), j) = \texttt{read}(a, j)$
  3. Extensionality: $\forall a, b: \; a = b \leftrightarrow (\forall i: \; \texttt{read}(a, i) = \texttt{read}(b, i))$

SMT Definitions (semi-formal)

Signatures and Models

A signature Σ is a set of constants, functions, and predicates.
A model M of Σ is a pair of a set D, called the domain of M, and a mapping
– from each constant $c \in \Sigma$ to some $d \in D$;
– from each k-ary function $f \in \Sigma$ to some function $\phi : D^k \to D$; and
– from each k-ary predicate $P \in \Sigma$ to some *relation* $\mathcal{P} \subseteq D^k$.

Σ-formula, Σ-theories

A Σ-formula is a FOL formula over the according symbols of Σ.
A Σ-theory $\mathcal{T}$ is a set of sentences, each of which is a Σ-formula.

$\mathcal{T}$-Satisfiability and $\mathcal{T}$-Validity

A Σ-formula F is $\mathcal{T}$-satisfiable iff there is a model M of $\mathcal{T}$ such that $\mathcal{T} \cup \{F\}$ is `true` under M.
A Σ-formula F is $\mathcal{T}$-valid iff $\mathcal{T} \cup \{F\}$ is `true` under *all* models M of $\mathcal{T}$.

Decidability of SMT

Definition: Theory Decidability

A theory $\mathcal{T}$ is decidable if and only if the $\mathcal{T}$-satisfiability of every Σ-formula is decidable.

Theory	Decidable?	Quantor-free Fragment decidable?	Conjunction of literals decidable?
Uninterpreted Functions	–	✓	✓
Peano Arithmetic	–	–	✓
Presburger Arithmetic	✓	✓	✓
Arrays	–	✓	✓

SMT Solving

For SMT solving, we differentiate **two general approaches**:

SMT Solving

For SMT solving, we differentiate **two general approaches**:

- **Eager approach**: Find a direct translation of $\mathcal{T} \cup F$ to propositional logic; perform SAT solving [1]
  - Promising for "Boolean theories" like arrays, bit vectors
  - Need to encode full theory in advance
  - Theory-specific encodings required

SMT Solving

For SMT solving, we differentiate **two general approaches**:

- **Eager approach**: Find a direct translation of $\mathcal{T} \cup F$ to propositional logic; perform SAT solving [1]
  - Promising for "Boolean theories" like arrays, bit vectors
  - Need to encode full theory in advance
  - Theory-specific encodings required
- **Lazy approach**: Perform propositional reasoning over the Boolean skeleton of F; lazily check whether a found propositional model is consistent with $\mathcal{T}$.
  - Known as DPLL($\mathcal{T}$) in literature [3]
  - Numerous optimizations lead to close interaction between SAT solver and theory solver
  - Modular and flexible architecture

Lazy Approach: Example

Σ-Formula F (linear integer arithmetic):

$$y \geq 1 \land (x < 0 \lor y < 1) \land (x \geq 0 \lor y < 0)$$

Lazy Approach: Example

Σ-Formula F (linear integer arithmetic):

$$y \geq 1 \land (x < 0 \lor y < 1) \land (x \geq 0 \lor y < 0)$$

Boolean skeleton:

$$A \land (B \lor C) \land (D \lor E)$$

Lazy Approach: Example

Σ-Formula F (linear integer arithmetic):

$$y \geq 1 \land (x < 0 \lor y < 1) \land (x \geq 0 \lor y < 0)$$

Boolean skeleton:

$$A \land (B \lor C) \land (D \lor E)$$

Satisfying assignment found by SAT solver:

$$A, \quad \neg B, \quad C, \quad \neg D, \quad E$$

Lazy Approach: Example

Σ-Formula F (linear integer arithmetic):

$$y \geq 1 \land (x < 0 \lor y < 1) \land (x \geq 0 \lor y < 0)$$

Boolean skeleton:

$$A \land (B \lor C) \land (D \lor E)$$

Satisfying assignment found by SAT solver:

$$A, \quad \neg B, \quad C, \quad \neg D, \quad E$$

Inconsistent subset of according $\mathcal{T}$-literals:

$$y \geq 1, \quad y < 1, \quad y < 0$$

Lazy Approach: Example

Σ-Formula F (linear integer arithmetic):

$$y \geq 1 \wedge (x < 0 \vee y < 1) \wedge (x \geq 0 \vee y < 0)$$

Boolean skeleton:

$$A \wedge (B \vee C) \wedge (D \vee E)$$

Satisfying assignment found by SAT solver:

$$A, \quad \neg B, \quad C, \quad \neg D, \quad E$$

Inconsistent subset of according $\mathcal{T}$-literals:

$$y \geq 1, \quad y < 1, \quad y < 0$$

Exclude this inconsistency:

$$\neg(y \geq 1) \vee \neg(y < 1)$$

Lazy Approach: Example

Σ-Formula F (linear integer arithmetic):

$$y \geq 1 \wedge (x < 0 \vee y < 1) \wedge (x \geq 0 \vee y < 0)$$

Boolean skeleton:

$$A \wedge (B \vee C) \wedge (D \vee E)$$

Satisfying assignment found by SAT solver:

$$A, \quad \neg B, \quad C, \quad \neg D, \quad E$$

Inconsistent subset of according $\mathcal{T}$-literals:

$$y \geq 1, \quad y < 1, \quad y < 0$$

Exclude this inconsistency:

$$\neg(y \geq 1) \vee \neg(y < 1)$$

Next Boolean skeleton:

$$A \wedge (B \vee C) \wedge (D \vee E) \wedge (\neg A \vee \neg C)$$

...

Lazy Approach

Optimizations of DPLL(T):

- Already check theory consistency of a partial assignment as it is being constructed
- Let theory solver guide search by returning consequences implied by a partial assignment
- Upon inconsistency, instead of a full restart, backtrack to a point where the assignment was still consistent

Lazy Approach

Optimizations of DPLL(T):

- Already check theory consistency of a partial assignment as it is being constructed
- Let theory solver guide search by returning consequences implied by a partial assignment
- Upon inconsistency, instead of a full restart, backtrack to a point where the assignment was still consistent

DPLL(T) follows modular approach:

- SAT solver and theory solver communicate via relatively simple API
  – most recently, IPASIR-UP ("User Propagators") [2]
- Theory solver only receives conjunctions of literals
  – Satisfiability of such conjunctions is decidable in most theories
- New theory? $\rightarrow$ just plug in a new theory solver
- SAT solver can be embedded with little effort

Bit Vectors via Eager Approach: Motivation

```
int x, y;
...
if (x - y > 0) {
  assert(x > y);
  ...
}
```

Can this assertion fail?

Bit Vectors via Eager Approach: Motivation

```
int x, y;
...
if (x - y > 0) {
  assert(x > y);
  ...
}
```

Can this assertion fail?

– Linear Integer Arithmetic: $x - y > 0 \wedge \neg(x > y)$ is unsatisfiable.

Bit Vectors via Eager Approach: Motivation

```
int x, y;
...
if (x - y > 0) {
  assert(x > y);
  ...
}
```

Can this assertion fail?

– Linear Integer Arithmetic: $x - y > 0 \land \neg(x > y)$ is unsatisfiable.

– Computer: assertion fails if `x = 2147483648` and `y = 1`!

Bit Vector via Eager Approach: Theory (informal)

Bit Vector (BV) theory: Express numeric variables as bit vectors. Reason over them.

- Bit vector v has bits $v_0, \ldots, v_{n-1}$, (bit) length $n = |v|$, (unsigned) value $\langle v \rangle = \Sigma_{i=0}^{|v|-1} 2^i v_i$
- Positional manipulation functions, like $\texttt{concat}(a, b) := (a_0, \ldots, a_{n_a-1}, b_0, \ldots, b_{n_b-1})$, $\texttt{zero\_extend}(a, k) := (a_0, \ldots, a_{n-1}, 0, \ldots, 0)$ (k zeroes), $\texttt{leftshift}(a, k)$, $\texttt{rightshift}(a, k)$, etc.
- Bitwise operation functions, like $\texttt{not}(a)$, $\texttt{and}(a, b)$, $\texttt{or}(a, b)$, $\texttt{xor}(a, b)$
- Arithmetic operation functions, like $\texttt{add}(a, b)$, $\texttt{sub}(a, b)$, $\texttt{mul}(a, b)$
- Comparison predicates, like $=$, $<_{\text{signed}}$, $<_{\text{unsigned}}$, etc.

Above assertion example: $(0_{(32)} <_{\text{signed}} \texttt{sub}(x, y)) \land (x \leq_{\text{signed}} y)$

SMT solver for BV theory?
— eager approach is natural due to intrinsically Boolean structure

Bit Vector via Eager Theory: Encoding

Propositional encoding F of a bit vector formula Φ:

- Initialize F as the Boolean skeleton of Φ,
  substituting each predicate P with a Boolean abstraction variable $AV(P)$
- For each added abstraction variable $AV(P)$, extend F by two kinds of constraints:
  – constraints that express the predicate P
  – constraints for each term in P
  (using n Boolean variables $v_0, \ldots, v_{n-1}$ for each term corresponding to a bit vector v of length n)

Some (simple) examples for constraints:

$$AV(x = y) \leftrightarrow \left(\bigwedge_{i=0}^{|x|-1} x_i \leftrightarrow y_i \right)$$

$$AV(\text{and}(a, b)) \leftrightarrow \left(\bigwedge_{i=0}^{|x|-1} \text{and}(a, b)_i \leftrightarrow (a_i \wedge b_i) \right)$$

Bit Vector via Eager Theory: Remarks

- Some constraints may require case distinction over bit vector values
- Some constraints are expensive to encode
- Incremental schemes possible to save encoding effort
  - Under- or over-approximate encoding, react based on SAT/UNSAT
  - Add constraints lazily – counter-example guided abstraction refinement (CEGAR)
  - Approximate expensive operations (like $\texttt{mul}(a, b)$) by replacing them with uninterpreted functions
- Further reading: [4]

SMT in Practice

Example: Swap two integers without third variable

```
int x, y, oldx, oldy;
...
oldx = x;
oldy = y;
x = x + y;
y = x - y;
x = x - y;
assert(y == oldx && x == oldy);
```

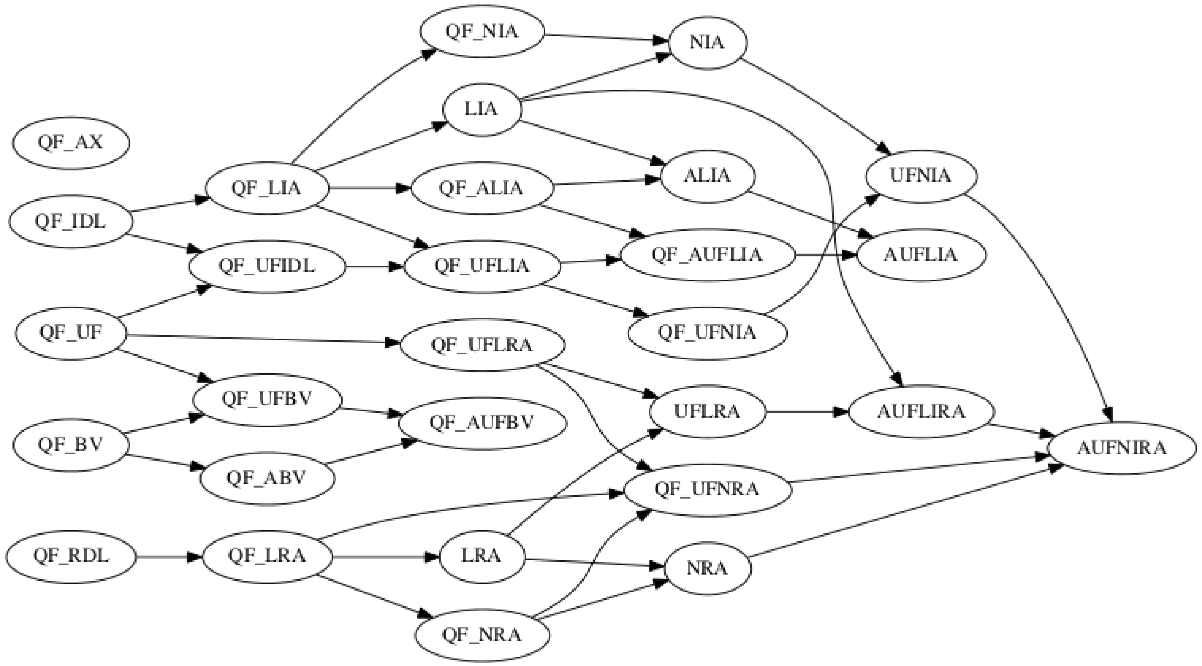
Example from `https://smt-lib.org/examples.shtml`

```
(set-logic QF_BV)
(set-option :produce-models true)

(declare-const x_0 (_ BitVec 32))
(declare-const x_1 (_ BitVec 32))
(declare-const x_2 (_ BitVec 32))
(declare-const y_0 (_ BitVec 32))
(declare-const y_1 (_ BitVec 32))
(assert (= x_1 (bvadd x_0 y_0)))
(assert (= y_1 (bvsub x_1 y_0)))
(assert (= x_2 (bvsub x_1 y_1)))

(assert (not
  (and (= x_2 y_0)
       (= y_1 x_0))))
(check-sat)
; unsat
(exit)
```

SMT: Concluding Remarks

SMT is a vast area – we barely scratched the surface.

- Standardization of different theories & logics and their interactions
- SMT solvers support subsets of theories
  - Completely different reasoning needed for different theories, applications
- Increasingly relevant research topic: Proofs for SMT solvers
- Definitive resource surrounding SMT: `http://smt-lib.org/`



References I

[1] Robert Brummayer und Armin Biere. „Boolector: An efficient SMT solver for bit-vectors and arrays“. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2009, S. 174–177.

[2] Katalin Fazekas u. a. „IPASIR-UP: User Propagators for CDCL“. In: *Theory and Applications of Satisfiability Testing (SAT)*. 2023. DOI: `10.4230/LIPIcs.SAT.2023.8`.

[3] Robert Nieuwenhuis, Albert Oliveras und Cesare Tinelli. „Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)“. In: *Journal of the ACM (JACM)* 53.6 (2006), S. 937–977.

[4] Samuel Teuber, Marko Kleine Büning und Carsten Sinz. „An Incremental Abstraction Scheme for Solving Hard SMT-Instances over Bit-Vectors“. In: *arXiv preprint arXiv:2008.10061* (2020).