软件分析与架构设计

可满足性模理论（SMT）

何冬杰

重庆大学

First Order Logic (FOL)

❑A first-order language $\mathcal{L}$ consists of:

- ➢Logical symbols:
  - Variables: $x, y, z, \ldots$
  - Logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \ldots$
  - Quantifiers: $\forall, \exists$
  - Parentheses
- ➢Terms:
  - Every variable is a term
  - Every constant symbol is a term
  - Every function $\mathrm{f}(t_1, \cdots, t_n)$ is a term
- ➢Non-logical symbols:
  - Constant symbols: $\mathrm{a}, \mathrm{b}, \mathrm{c}, \ldots$
  - N-ary function symbols: $\mathrm{f}, \mathrm{g}, \mathrm{h}, \ldots$
  - N-ary predicate symbols: $\mathrm{p}, \mathrm{q}, \mathrm{r}, \ldots$
- ➢Well-formed formulas:
  - Atomic formulas: Predicate $P(t_1, \ldots, t_n)$
  - If $\varphi, \text{and } \psi$ are formulas, then
    - ❖ $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi), \ldots$
  - If φ is a formula, x is a variable, then
    - ❖ $\forall \mathrm{x}\, \varphi, \exists \mathrm{x}\, \varphi$

❑Semantics (Interpretation):

- ➢Interprets variables, functions, and predicates over domain $\boldsymbol{D}$
- ➢A formula φ is **satisfiable** if $I \vDash \varphi$ for **some** interpretation I
- ➢A formula φ is **valid if** if $\boldsymbol{I} \vDash \boldsymbol{\varphi}$ for **all** interpretations I

Satisfiability Modulo Theories (SMT)

- **SMTs as generalizations of SAT**
  - SAT: reason propositional logic
  - SMT: aims to reason first order logic (FOL)
    - But satisfiability of FOL is **undecidable**
    - SMT, instead, targets decidable or domain-specific fragments
- **SMT: propositional logic + domain-specific reasoning**
  - deciding the satisfiability of a (ground) First Order formula with respect to a **background theory**
    - Software verification needs reasoning about equality, arithmetic, data structures, …
  - Example: $g(a) = c \wedge ((f(g(a)) \neq f(c)) \vee g(a) = d) \wedge (c \neq d)$
    - Need Equality with Uninterpreted Functions (EUF) theory
  - SMT is a **sweet spot** between expressiveness and efficiency
- **Wide range of applications:**
  - SMT solvers as backend "workhorse" engines of many techniques and tools
  - Model checking, Test generation, Software verification, Synthesis, …

SMT in Practice

- **Beginnings of SMT: Early 2000s**
- **GOOD NEWS: efficient decision procedures for sets of ground literals exist for various theories of interest**
- **PROBLEM: in practice, we need to deal with:**
  - arbitrary Boolean combinations of literals ($\wedge, \vee, \neg$)
  - multiple theories: combine decision procedures for each theory
    - In practice, theories are not isolated
    - E.g., Software verifications needs arithmetic, arrays, bitvectors, …
    - Formulas of the following form usually arise:
      - $a = b + 2 \wedge A = write(B, a + 1, 4) \wedge (read(A, b + 3) = 2 \vee f(a - 1) \neq f(b + 1))$
  - quantifiers $\forall, \exists$
- **KEY practice: exploit SAT solvers to improve performance**
  - Eager approach (a.k.a. "bit-blasting"): encode SMT into SAT
  - Lazy approach: plug SAT solver with a decision procedure

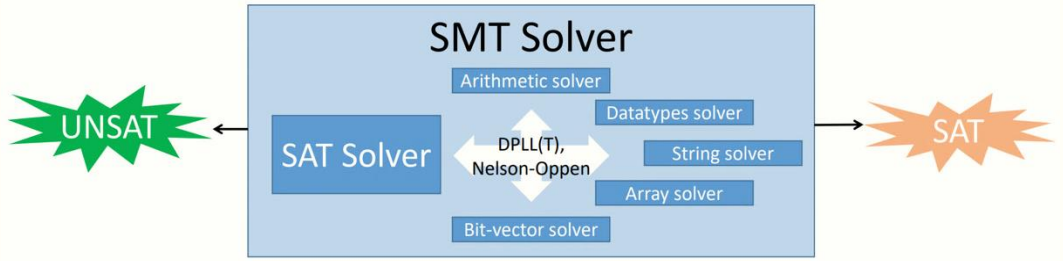
Eager Approach

- **Methodology: translate problem into equisatisfiable propositional formula and use off-the-shelf SAT solver**
  - Still dominant for bit-vector arithmetic
- **Encoding SMT formula into SAT**
  - EUF as an example: $f(a) = c \wedge f(b) \neq c \wedge a \neq b$
  - Introduce symbols to replace function applications
    - A for f(a), B for f(b), the formula becomes $A = c \wedge B \neq c \wedge a \neq b$
    - Add constraints based on EUF axioms: $a = b \rightarrow A = B$
    - Introduce bool variables to replace equations:
      - $P_{A=c} \wedge \neg P_{B=c} \wedge \neg P_{a=b}, P_{a=b} \rightarrow P_{A=B}$
    - Add constraints based on reflexivity: e.g., $P_{A=A}$ (not necessary here)
    - Add constraints for transitivity:
      - $P_{A=c} \wedge P_{B=c} \rightarrow P_{A=B}, P_{A=B} \wedge P_{B=c} \rightarrow P_{A=c}, P_{A=B} \wedge P_{A=c} \rightarrow P_{B=c}, \ldots$

Eager Approach

- **Why called "Eager"?**
  - Search uses all theory information from the beginning
- **Advantage:**
  - Can use best available SAT solver
- **Problems of Eager Approach:**
  - After encoding, SAT solver cannot utilize the dedicated solution algorithms of the original theories:
    - EUF can be solved by a fixed-point algorithm that continuously merges equivalence classes
  - Sophisticated encodings are needed for each theory
  - Not all SMT can be encoded into SAT

Lazy Approach



- **Lazy SMT Solvers: CVC4, MathSat, Yices, Z3, …**
  - Yices 1.0 (2006): the first efficient "general-purpose" SMT solver
  - Z3 1.0 (2008): > 10000 citations, most influential tool paper at TACAS
- **Lazy approach: Combines SAT and theory solvers**
  - SAT-solvers enumerate models for the boolean part
  - Theory solvers check satisfiability in the theory
- **Example 1: Arithmetic theory**

$$
\begin{array}{lcl}
\psi & \triangleq & \\
c_1 & : & \neg(2x_2 - x_3 > 2) \vee (x_1 + x_3 \leq 5) \\
c_2 & : & \neg(x_1 - x_3 \leq 5) \vee (x_1 - x_5 \leq 1) \\
c_3 & : & \neg(3x_1 - 2x_2 \leq 3) \vee \neg(x_1 - x_3 \leq 5) \\
c_4 & : & \neg(3x_1 - x_3 \leq 6) \vee \neg(x_1 + x_3 \leq 5) \\
c_5 & : & (x_1 + x_3 \leq 5) \vee (3x_1 - 2x_2 \leq 3) \\
c_6 & : & (x_2 - x_4 \leq 6) \vee \neg(x_1 + x_3 \leq 5) \\
c_7 & : & (x_1 + x_3 \leq 5) \vee (x_3 = 3x_5 + 4) \vee \neg(x_1 - x_3 \leq 5)
\end{array}
\qquad
\begin{array}{l}
\psi_{\mathbb{B}} \triangleq \\
\overline{A_{11}} \vee A_{12} \\
\overline{A_{21}} \vee A_{22} \\
\overline{A_{31}} \vee \overline{A_{32}} \\
\overline{A_{41}} \vee \overline{A_{42}} \\
A_{51} \vee A_{31} \\
A_{61} \vee \overline{A_{62}} \\
A_{71} \vee A_{72} \vee \overline{A_{73}}
\end{array}
$$

  - Sat-solver gives $\{A_{12},\ A_{22},\ \overline{A_{31}},\ \overline{A_{41}},\ A_{51},\ A_{61},\ A_{72}\}$,
  - Then the theory solver says the following is UNSAT
    - $\{(x_1 + x_3 \leq 5), (x_1 - x_5 \leq 1), \neg(3x_1 - 2x_2 \leq 3), \neg(3x_1 - x_3 \leq 6), (x_2 - x_4 \leq 6), (x_3 = 3x_5 + 4)\}$
  - Add $(\overline{\boldsymbol{A_{12}}} \vee \overline{A_{22}} \vee \overline{\boldsymbol{A_{31}}} \vee \overline{\boldsymbol{A_{41}}} \vee \overline{A_{51}} \vee \overline{A_{61}} \vee \overline{A_{72}})$ to $\boldsymbol{\psi}_{\mathbb{B}}$

Lazy Approach

❑Example 2: EUF theory

$$\underbrace{g(a)=c}_{1} \wedge (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \wedge \underbrace{c\neq d}_{\overline{4}}$$

- ➢SAT solver gives $\{1, \overline{2}, \overline{4}\}$,
- ➢Theory solver says the following is **T-inconsistent**:
  - $\{(g(a) = c), (f(g(a)) \neq f(c)), (c \neq d)\}$
- ➢Add $\overline{1} \vee 2 \vee \overline{4}$ to the formula and send to SAT solver
- ➢SAT solver gives $\{1,2,3, \overline{4}\}$
- ➢Theory solver says **T-inconsistent** to the followings:
  - $\{(g(a) = c), (f(g(a)) = f(c)), (g(a) = d), (c \neq d)\}$
- ➢Add $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{4}$ to the formula
- ➢Finally, SAT-solver declares UNSAT to the original formula

Basic Lazy Approach

- **Convert to DNF and use SAT to enumerate conjuncts**
  - Check one conjunction at a time
  - Restrict the theory solver to conjunctions of constraints
- **Why "lazy"?**
  - Theory information is used lazily when checking T-consistency of propositional models
  - Theory information does not guide the search (too lazy), only validating it
    - DPLL(T) will improve it

```
1  ψ = to_cnf(φ);
2  while(true){
3    res , M = check_SAT(ψ);
4    if( res){
5      M_T = to_theory(M);
6      res = check_theory(M_T);
7      if(res)
8        return SAT;
9      else
10       ψ∧ = ¬M;
11   }else
12     return UNSAT;
13 }
```

Lazy approach: Optimizations

- **Several optimizations for enhancing efficiency:**
  - ~~Check **T-consistency** only of full propositional models~~
  - Check **T-consistency** of partial assignment while being built
  - ~~Given a **T-inconsistent** assignment M, add $\neg M$ as a clause~~
  - Given a **T-inconsistent** assignment M, identify a **T-inconsistent** subset $M_0 \subseteq M$ and add $\neg M_0$ as a clause
  - ~~Upon a **T-inconsistency**, add clause and restart~~
  - Upon a **T-inconsistency**, backtrack to some point where the assignment was still **T-consistent**
  - **T-propagation: exploited in conflict analysis**
    - **Search guided by T-Solver by finding T-consequences**

DPLL(T) (also known as CDCL(T))

- **DPLL(T) = DPLL(X) + T-Solver**
  - DPLL(X): Very similar to a SAT solver, enumerates Boolean models
  - T-Solver: Should be incremental and backtrackable
    - Checks consistency of conjunctions of literals
    - Computes theory propagations
    - Produces explanations of inconsistency/T-propagation
- **Example: EUF and formula** $\underbrace{g(a)=c}_{1} \wedge (\underbrace{f(g(a))\neq f(c)}_{\overline{2}} \vee \underbrace{g(a)=d}_{3}) \wedge \underbrace{c\neq d}_{\overline{4}}$

$$
\begin{array}{rcll}
\emptyset & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\
1 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{UnitPropagate}) \\
1\,\overline{4} & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \\
1\,\overline{4}\,2 & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{T-Propagate}) \\
1\,\overline{4}\,2\,\overline{3} & \| & 1,\ \overline{2}\vee 3,\ \overline{4} & \Rightarrow \quad (\text{Fail})
\end{array}
$$

UNSAT

DPLL(T) (also known as CDCL(T))

❑Preprocessing

- ➢Main goal: from arbitrary formulas to sets of clauses (CNFs)
- ➢Also: (cheap) rewriting steps to simplify the input problem
  - Normalization of constraints: $(x + x \geq 3) \rightarrow (0 \leq 2x - 3)$
  - Elimination of variables: $(x = y) \Lambda \varphi(x, y) \rightarrow \varphi(x, x)$
- ➢Encoding of constraints not supported by the core engine
  - E.g., Term-ITEs: $\varphi(ite(b, x, y)) \rightarrow \varphi(i) \Lambda (b \rightarrow i = x) \Lambda (\neg b \rightarrow i = y)$
  - Division: $\varphi(x / y) \rightarrow \varphi(d) \Lambda (x = d * y) \Lambda (d \geq 0) \Lambda (d < y)$
  - Max/Min functions:

$$\varphi(\max(x, y)) \rightarrow \varphi(m) \wedge (m \geq x) \wedge (m \geq y) \wedge ((m = x) \vee (m = y))$$

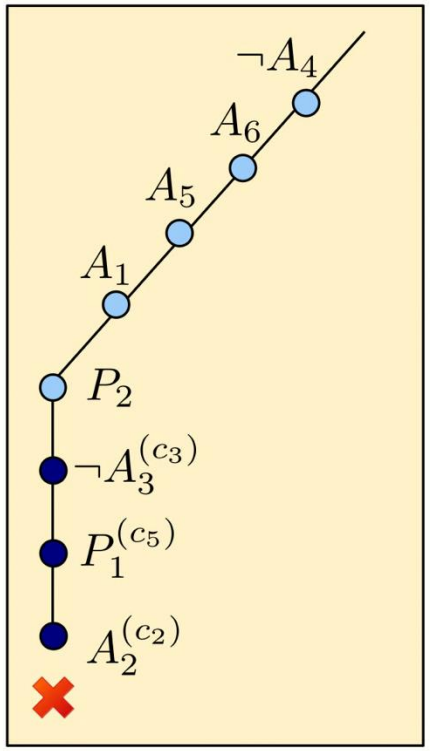
DPLL(T) Algorithm

- **T-backjumping and T-learning**
  - When unsat, T-solver can produce a reason for the inconsistency
  - T-conflict set: inconsistent subset of the input constraints
  - T-conflict clause given as input to the CDCL conflict analysis
    - Drives non-chronological backtracking (backjumping)
    - Can be learned by the SAT solver
  - The less redundant the T-conflict set, the more search is saved
    - Ideally, should be minimal (irredundant)
      - Removing any element makes the set consistent
    - But for some theories might be expensive to achieve
      - Trade-off between size and cost

```
def DPLL_T():
    while True:
        conflict = False
        if unit_propagation():
            res = theory_check(not all_assigned())
            if res == False: conflict = True
            elif res == True: conflict = theory_propagation()
            elif learn_theory_lemmas(): continue
            elif not all_assigned(): decide()
            else:
                build_model()
                return SAT
        else: conflict = True
        if conflict:
            lvl, cls = conflict_analysis()
            if lvl < 0: return UNSAT
            else:
                backtrack(lvl)
                learn(cls)
```

Example

$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \varphi^{\text{Bool}} \quad \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \neg A_3 \vee \neg P_2 \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & \neg A_4 \vee \neg P_1 \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & P_1 \vee A_3 \\
c_6: & (x_2 - x_4 \leq 6) \vee \neg P_1 & A_5 \vee \neg P_1 \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & P_1 \vee A_6 \vee \neg P_2
\end{array}
$$



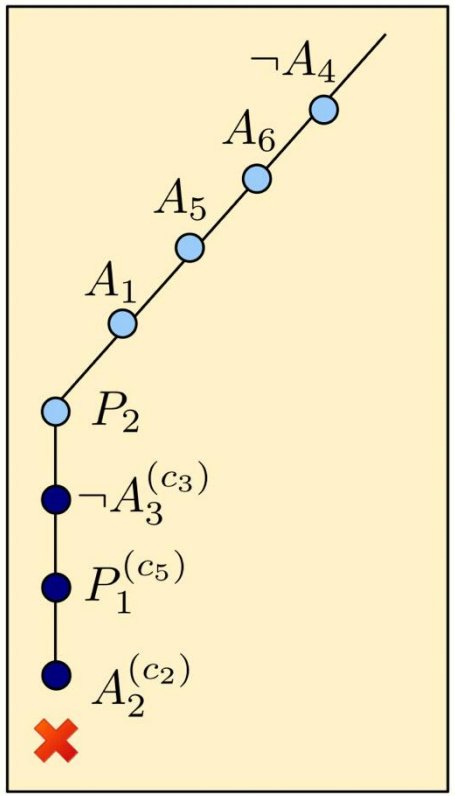
$$M = [\neg A_4, A_6, A_5, A_1, P_2, \neg A_3, P_1, A_2]$$

$$M' = \{\neg(3x_1 - x_3 \leq 6), (x_3 = 3x_5 + 4), (x_2 - x_4 \leq 6),$$
$$\neg(2x_2 - x_3 > 2), \neg(3x_1 - 2x_2 \leq 3), (x_1 - x_5 \leq 1)\}$$

T-conflict set

Example

φ	$\stackrel{\text{def}}{=}$	φ^{Bool}	$\stackrel{\text{def}}{=}$
c_1 :	$(2x_2 - x_3 > 2) \vee P_1$		$A_1 \vee P_1$
c_2 :	$\neg P_2 \vee (x_1 - x_5 \leq 1)$		$\neg P_2 \vee A_2$
c_3 :	$\neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2$		$\neg A_3 \vee \neg P_2$
c_4 :	$\neg(3x_1 - x_3 \leq 6) \vee \neg P_1$		$\neg A_4 \vee \neg P_1$
c_5 :	$P_1 \vee (3x_1 - 2x_2 \leq 3)$		$P_1 \vee A_3$
c_6 :	$(x_2 - x_4 \leq 6) \vee \neg P_1$		$A_5 \vee \neg P_1$
c_7 :	$P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2$		$P_1 \vee A_6 \vee \neg P_2$

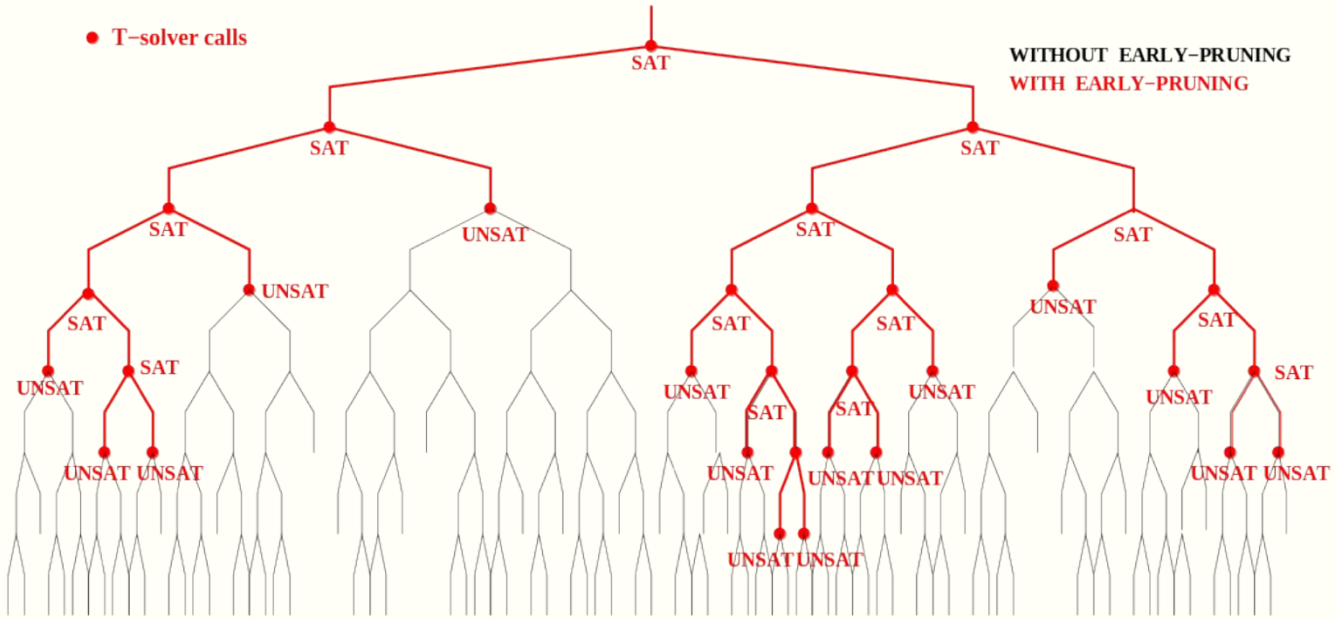


Conflict analysis:

$$\frac{\overbrace{A_4 \vee \neg A_6 \vee \neg A_2}^{T\text{-conflict clause}} \qquad \overbrace{\neg P_2 \vee A_2}^{c_2}}{A_4 \vee \neg A_6 \vee \neg P_2}$$

Early Pruning

- **Invoke T-solver on intermediate assignments, during the CDCL search**
  - If unsat is returned, can backtrack immediately
- **Advantage: can drastically prune the search tree**
- **Drawback: possibly many useless (expensive) T-solver calls**



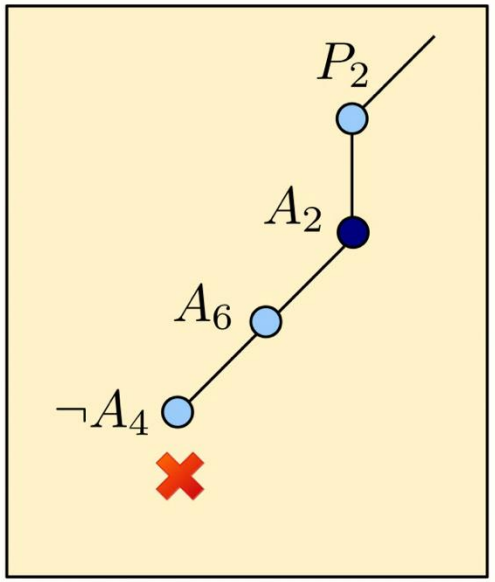
Early Pruning

- **Different strategies to call T-solver**
  - Eagerly, every time a new atom is assigned
  - After every round of **BCP** (Boolean Constraint Propagation)
  - Heuristically, based on some statistics (e.g. effectiveness, …)
- **No need for a conclusive answer during early pruning calls**
  - Can apply approximate checks
  - Trade effectiveness for efficiency

Example: on linear integer arithmetic, solve only the real relaxation during early pruning calls

Example

$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \varphi^{\text{Bool}} \quad \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \neg A_3 \vee \neg P_2 \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & \neg A_4 \vee \neg P_1 \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & P_1 \vee A_3 \\
c_6: & (x_2 - x_4 \leq 6) \vee \neg P_1 & A_5 \vee \neg P_1 \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & P_1 \vee A_6 \vee \neg P_2
\end{array}
$$



$$M = [P_2, A_2, A_6, \neg A_4]$$

UNSAT

T-conflict = $\{\neg(3x_1 - x_3 \leq 6), (x_3 = 3x_5 + 4), (x_1 - x_5 \leq 1)\}$

T-propagation

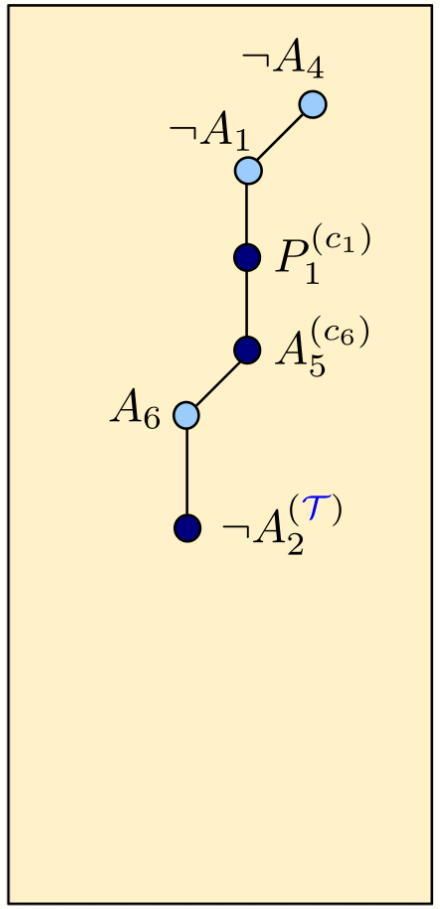
- **T-solvers might support deduction of unassigned constraints**
  - If early pruning check on M returns sat, T-solver might also return a set of unassigned atoms D such that $\mathrm{M} \models_{\mathcal{T}} l$ for all $l \in D$
- **T-propagation: add each such l to the CDCL stack**
  - As if BCP was applied to the (T-valid) clause $\neg\mathrm{M} \vee l$ (T-reason)
  - But do not compute the T-reason clause explicitly yet
- **Lazy explanation: compute T-reason clause only if needed during conflict analysis**
  - Like T-conflicts, the less redundant the better

Example

φ	$\stackrel{\text{def}}{=}$	φ^{Bool}	$\stackrel{\text{def}}{=}$
c_1 :	$(2x_2 - x_3 > 2) \vee P_1$		$A_1 \vee P_1$
c_2 :	$\neg P_2 \vee (x_1 - x_5 \leq 1)$		$\neg P_2 \vee A_2$
c_3 :	$\neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2$		$\neg A_3 \vee \neg P_2$
c_4 :	$\neg(3x_1 - x_3 \leq 6) \vee \neg P_1$		$\neg A_4 \vee \neg P_1$
c_5 :	$P_1 \vee (3x_1 - 2x_2 \leq 3)$		$P_1 \vee A_3$
c_6 :	$(x_2 - x_4 \leq 6) \vee \neg P_1$		$A_5 \vee \neg P_1$
c_7 :	$P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2$		$P_1 \vee A_6 \vee \neg P_2$
c_8 :	$P_2 \vee (2x_2 - 3x_1 \geq 5) \vee$ $(x_3 + x_5 - 4x_1 \geq 0)$		$P_2 \vee A_7 \vee A_8$

$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6]$$

$$\frac{\neg(3x_1 - x_3 \leq 6) \qquad (x_3 = 3x_5 + 4)}{\dfrac{\neg(3x_1 - 3x_5 \leq 10)}{\neg(x_1 - x_5 \leq 1) \equiv \neg A_2}}$$

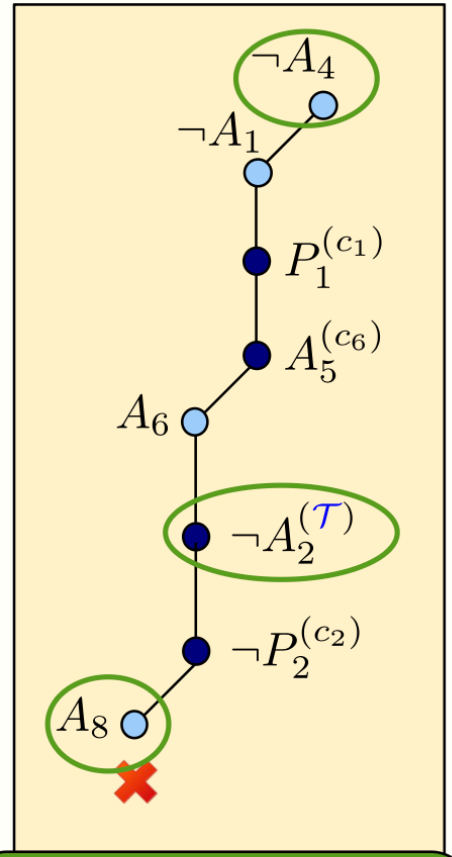


Example

$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \varphi^{\text{Bool}} \quad \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \neg A_3 \vee \neg P_2 \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & \neg A_4 \vee \neg P_1 \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & P_1 \vee A_3 \\
c_6: & (x_2 - x_4 \leq 6) \vee \neg P_1 & A_5 \vee \neg P_1 \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & P_1 \vee A_6 \vee \neg P_2 \\
c_8: & P_2 \vee (2x_2 - 3x_1 \geq 5) \vee & P_2 \vee A_7 \vee A_8 \\
 & (x_3 + x_5 - 4x_1 \geq 0) &
\end{array}
$$

$$
M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2, A_8]
$$

$$
\dfrac{\dfrac{\neg(3x_1 - x_3 \leq 6) \qquad \neg(x_1 - x_5 \leq 1)}{\neg(-x_3 + 3x_5 \leq 3)} \qquad (x_3 + x_5 - 4x_1 \geq 0)}{\bot}
$$



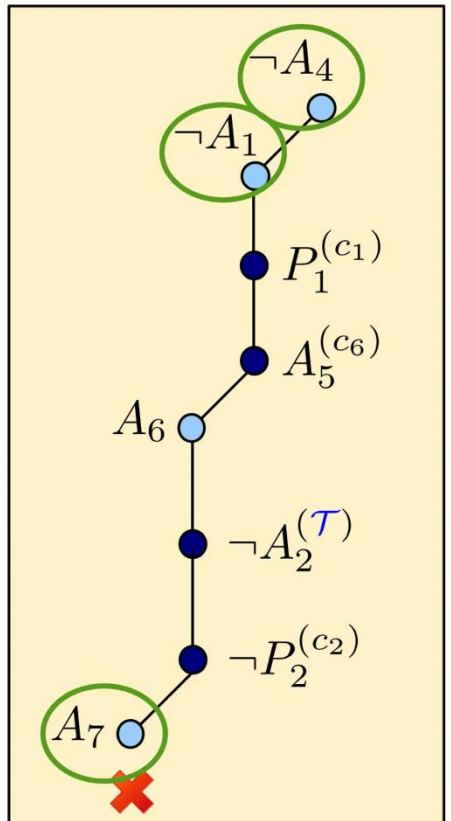
Conflict analysis → compute *T*-reason for $\neg A_2$

Example

$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \varphi^{\text{Bool}} \quad \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \neg A_3 \vee \neg P_2 \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & \neg A_4 \vee \neg P_1 \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & P_1 \vee A_3 \\
c_6: & (x_2 - x_4 \leq 6) \vee \neg P_1 & A_5 \vee \neg P_1 \\
c_7: & P_1 \vee (x_3 = 3x_5 + 4) \vee \neg P_2 & P_1 \vee A_6 \vee \neg P_2 \\
c_8: & P_2 \vee (2x_2 - 3x_1 \geq 5) \vee & P_2 \vee A_7 \vee A_8 \\
 & (x_3 + x_5 - 4x_1 \geq 0) &
\end{array}
$$

$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2, A_7]$$

$$\dfrac{\dfrac{\neg(3x_1 - x_3 \leq 6) \qquad \neg(2x_2 - x_3 > 2)}{\neg(3x_1 - 2x_2 \leq 4)} \qquad (2x_2 - 3x_1 \geq 5)}{\bot}$$



$\neg A_2$ not involved in conflict analysis → *no need to compute T-reason*

Theories of Interest: EUF

- **Equality (i.e., "=") with Uninterpreted Functions**
  - Used to abstract non-supported constructions,
    - e.g: Non-linear multiplication and ALUs in circuits
  - Example 1: $a * (f(b) + f(c)) = d \ \wedge b * (f(a) + f(c)) \neq d \ \wedge a = b$ is UNSAT
    - $a = b \ \Rightarrow \ f(a) = f(b)$ by EUF, no need to compute f function
  - Example 2: we abstract the above formula into:
    - $h(a, g(f(b), f(c))) = d \wedge h(b, g(f(a), f(c))) \neq d \wedge a \ = \ b$
    - EUF allows to know the formula is **UNSAT** without knowing h, g, and f

Theories of Interest: Arithmetic

- **Very useful for obvious reasons**
- **Restricted fragments support more efficient methods:**
  - Bounds: $x \bowtie k$ with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - Difference logic: $x - y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - UTVPI: $\pm x \pm y \bowtie k$, with $\bowtie \in \{<, >, \leq, \geq, =\}$
  - Linear arithmetic, e.g: $2x - 3y + 4z \leq 5$
  - Non-linear arithmetic, e.g: $2xy + 4xz^2 - 5y \leq 10$
  - Variables are either reals or integers
- **Machine-inspired arithmetic: floating-point arithmetic**

Theories of Interest: Arrays

- **Two interpreted function symbols read and write**
- **Theory is axiomatized by:**
  - $\forall a \forall i \forall v\ (\mathrm{read}(\mathrm{write}(a, i, v), i) = v)$
  - $\forall a \forall i \forall j \forall v\ (i \neq j \rightarrow \mathrm{read}(\mathrm{write}(a, i, v), j) = \mathrm{read}(a, j))$
- **Sometimes extensionality is added:**
  - $\forall a \forall b\ ((\forall i(read(a, i) = read(b, i))) \rightarrow a = b$
- **Is the following set of literals satisfiable?**
  - $write(a, i, x) \neq b \quad read(b, i) = y \quad read(write(b, i, x), j) = y$

    $a = b \qquad\qquad i = j$
- **Used for:**
  - Software verification (arrays)
  - Hardware verification (memories)

Theories of Interest: Bit vectors

- **Constants represent vectors of bits**
- **Useful both for hardware and software verification**
- **Different type of operations:**
  - String-like operations: concatenation, extraction, …
  - Logical operations: bit-wise not, or, and, …
  - Arithmetic operations: add, substract, multiply, …
- **Example:** Assume bit-vectors have size 3. Is the formula SAT?
  - $a[0:1] \neq b[0:1] \wedge (a|b) = c \wedge c[0] = 0 \wedge a[1] + b[1] = 0$

Combination of theories

- **Very often in practice more than one theory is needed**
  - Example:

$$\varphi \stackrel{\text{def}}{=} (x_1 \geq 0) \wedge (x_1 < 1) \wedge$$
$$((\mathrm{f}(x_1) = \mathrm{f}(0)) \rightarrow (\mathrm{rd}(\mathrm{wr}(P, x_2, x_3), x_2 + x_1) = x_3 + 1))$$

  - Use theories in Arrays, EUF, Arithmetic
- **How to build solvers for $SMT(T_1 \cdots T_n)$ that are both efficient and modular?**
  - Can we reuse Ti-solvers and combine them?
  - Under what conditions?
- **How do we go from $DPLL(T)$ to $DPLL(T_1 \cdots T_n)$?**

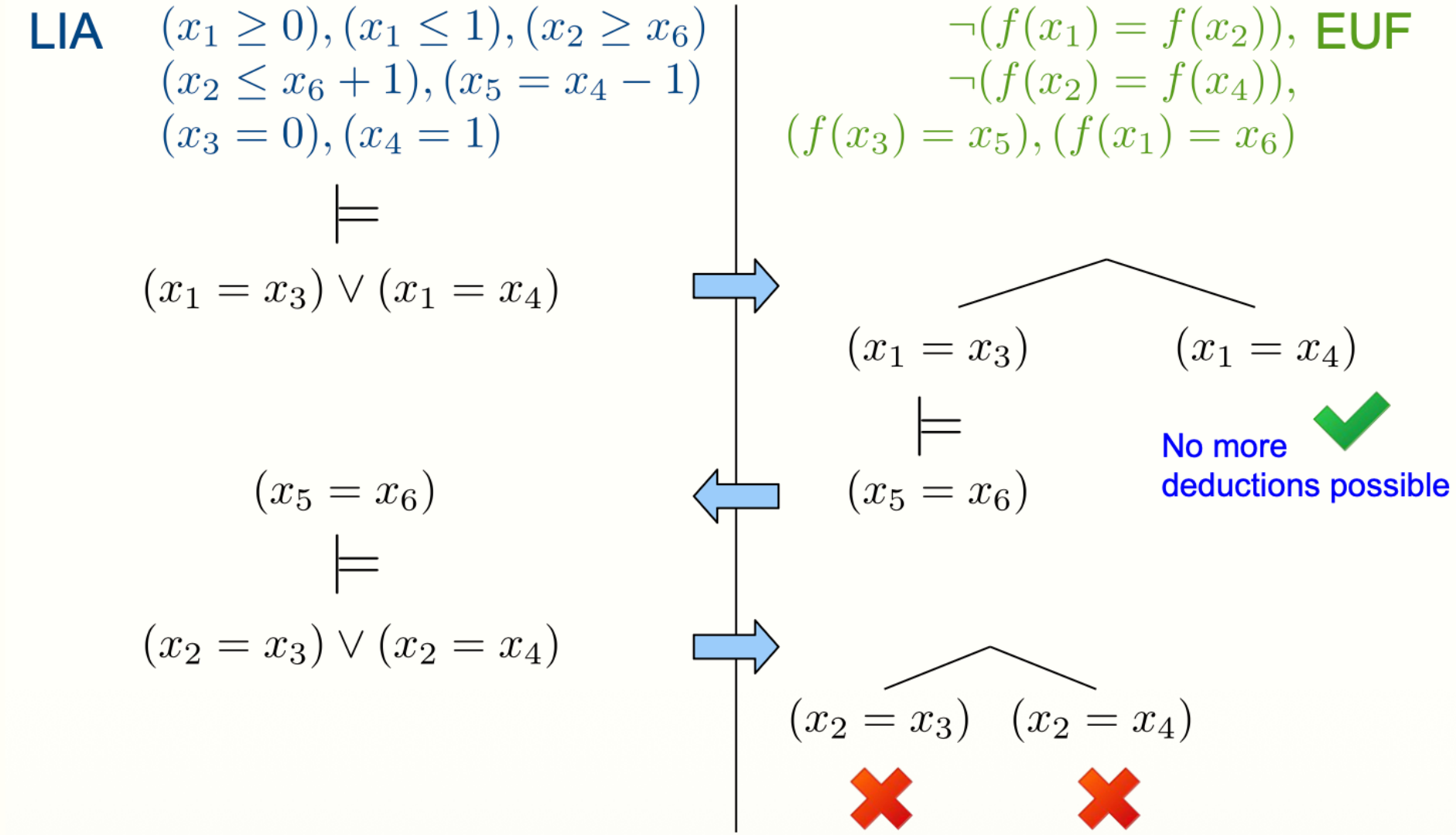
The Nelson-Oppen method

- **A general technique for combining T_i-solvers**
- **Requires:**
  - T_i's to have disjoint signatures, i.e. no symbols in common (except =)
  - T_i's to be stably-infinite, i.e, every quantifier-free T_i-satisfiable formula is satisfiable in an infinite model of T_i
- **How it works (for $T_1 \cup T_2$)**
  - Preprocessing purification step on the input formula φ
    - Pure formula: no atom containing symbols of different T_i's (except =)
      - By labeling subterms

$$\varphi \stackrel{\text{def}}{=} f(\overbrace{x+3y}^{l_1})+4 \leq \overbrace{g(w)}^{l_2} \mapsto$$
$$(\overbrace{f(l_1)}^{l_3} +4 \leq l_2) \wedge (l_1 = x+3y) \wedge (l_2 = g(w)) \mapsto$$
$$(l_3+4 \leq l_2) \wedge (l_1 = x+3y) \wedge (l_2 = g(w)) \wedge (f(l_1) = l_3)$$

- **T_i-solvers cooperate by exchanging entailed interface equalities**
  - i.e., equalities between shared variables

Example


LIA
$(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$
$\models$
$(x_1 = x_3) \vee (x_1 = x_4)$
$\neg(f(x_1) = f(x_2))$, EUF
$\neg(f(x_2) = f(x_4))$,
$(f(x_3) = x_5), (f(x_1) = x_6)$
$(x_1 = x_3)$
$(x_1 = x_4)$
$\models$
$(x_5 = x_6)$
No more
deductions possible
$(x_5 = x_6)$
$\models$
$(x_2 = x_3) \vee (x_2 = x_4)$
$(x_2 = x_3)$
$(x_2 = x_4)$

(Optional)项目：SMT求解器学习

❑主流SMT求解器

- Z3：微软研究院开发，支持多种理论，Python、C++等多种编程语言，广泛应用于软件验证、程序分析、测试生成等领域
- Yices 2：斯坦福国际研究院（SRI）开发，专注于命题逻辑的SMT求解器，在硬件设计和验证领域有广泛应用
- CVC5: 斯坦福大学等开发， 在字符串理论实现上更为成熟，处理复杂字符串操作时效率更高，在软件安全领域发挥重要作用
- Bitwuzla： 专注于位向量和数组理论
- 。。。

❑选择一款SMT求解器学习其使用方法，背后原理以及支持理论

- 撰写不少于1000字的学习报告