SAT/SMT summer school 2015

Introduction to SMT

Alberto Griggio

Fondazione Bruno Kessler – Trento, Italy

Some material courtesy of Roberto Sebastiani and Leonardo de Moura

Outline

Introduction

The DPLL(T) architecture

Some relevant T-solvers

Combination of theories

Quantifiers in DPLL(T)

The SMT problem

- **Satisfiability Modulo Theories**
  - Given a (quantifier-free) FOL formula and a (decidable) combination of theories $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_m$, is there an assignment to the free variables $x_1, \ldots, x_n$ that makes the formula true?

- Example:

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

The SMT problem

- **Satisfiability Modulo Theories**
  - Given a (quantifier-free) FOL formula and a (decidable) combination of theories $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_m$, is there an assignment to the free variables $x_1, \ldots, x_n$ that makes the formula true?

- Example:

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

Linear Integer Arithmetic (LIA)

The SMT problem

- **Satisfiability Modulo Theories**
  - Given a (quantifier-free) FOL formula and a (decidable) combination of theories $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_m$, is there an assignment to the free variables $x_1, \ldots, x_n$ that makes the formula true?

- Example:

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

Linear Integer Arithmetic (LIA)

Equality (EUF)

The SMT problem

- **Satisfiability Modulo Theories**
  - Given a (quantifier-free) FOL formula and a (decidable) combination of theories $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_m$, is there an assignment to the free variables $x_1, \ldots, x_n$ that makes the formula true?

- Example:

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

Linear Integer Arithmetic (LIA)

Equality (EUF)

Arrays (A)

The SMT problem

- **Satisfiability Modulo Theories**
  - Given a (quantifier-free) FOL formula and a (decidable) combination of theories $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_m$, is there an assignment to the free variables $x_1, \ldots, x_n$ that makes the formula true?

- Example:

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

$$\mathrm{LIA} \models (x_1 = 0)$$
$$\mathrm{EUF} \models f(x_1) = f(0)$$
$$\mathrm{A} \models \mathsf{rd}(\mathsf{wr}(P, x_2, x_3), x_2) = x_3$$
$$\mathrm{Bool} \models \mathsf{rd}(\mathsf{wr}(P, x_2, x_3), x_2 + x_1) = x_3 + 1$$
$$\mathrm{LIA} \models \bot$$

SMT: some history

The “early days”

- The Simplify theorem prover [Detlefs, Nelson, Saxe]
  - The grandfather of SMT solvers
- **Efficient decision procedures**
  - Equality logic + extensions (Congruence Closure)
  - Linear arithmetic (Simplex)
  - Theory combination (Nelson-Oppen method)
  - Quantifiers (E-matching with triggers)
- **Inefficient boolean search**

SMT: some history - 2

The SAT breakthrough

- late '90s - early 2000: major progress in SAT solvers

- **CDCL paradigm: Conflict-Driven Clause-Learning DPLL**
  - Grasp, (z)Chaff, Berkmin, MiniSat, ...

- combine strengths of **model search** and **proof search** in a single procedure
  - Model search: efficient BCP and variable selection heuristics
  - Proof search: conflict analysis, non-chronological backtracking, clause learning

- **Smart ideas + clever engineering “tricks”**

SMT: some history - 3

From SAT to SMT

- **exploit advances in SAT solving for richer logics**
  - Boolean combinations of constraints over (combinations of) background theories

- **The Eager approach** (a.k.a. “bit-blasting”)
  - Encode an SMT formula into propositional logic
  - Solve with an off-the-shelf efficient SAT solver
  - Pioneered by UCLID
  - Still the dominant approach for bit-vector arithmetic

SMT: some history - 4

The Lazy approach and DPLL(T) (2002 – 2004)

- **(non-trivial) combination of SAT (CDCL) and T-solvers**
  - SAT-solver enumerates models of boolean skeleton of formula
  - Theory solvers check consistency in the theory
  - Most popular approach (e.g. Barcelogic, CVC4, MathSAT, SMTInterpol, Yices, Z3, VeriT, ...)

- **Yices 1.0 (2006)**
  - The first efficient "general-purpose" SMT solver

- **Z3 1.0 (2008)**
  - > 1600 citations, most influential tool paper at TACAS

Outline

Introduction

The DPLL(T) architecture

Some relevant T-solvers

Combination of theories

Quantifiers in DPLL(T)

The lazy approach to SMT

- A theory *T* is a set of structures *(D, I)* over a signature $\sum$:
  - *D* a **domain** for variables
  - *I* an **interpretation** for function symbols $I(f) : D^n \mapsto D$

The lazy approach to SMT

- A theory *T* is a set of structures *(D, I)* over a signature Σ:
  - *D* a **domain** for variables
  - *I* an **interpretation** for function symbols $I(f) : D^n \mapsto D$

- Deciding the satisfiability of φ modulo $\mathcal{T}$ can be reduced to deciding $\mathcal{T}$-satisfiability of **conjunctions (sets) of constraints**
  - Can exploit efficient decision procedures for sets of constraints, existing for many important theories
- **Naive approach:** convert φ to an equivalent φ' in disjunctive normal form (DNF), and check each conjunction separately

- Main idea of **lazy SMT**: use an efficient SAT solver to enumerate conjuncts without computing the DNF explicitly

A basic approach

- **Offline** lazy SMT

```
F = CNF_bool( φ )
while true:
    res, M = check_SAT(F)
    if res == true:
        M' = to_T(M)
        res = check_T(M')
        if res == true:
            return SAT
        else:
            F += !M
    else:
        return UNSAT
```

A basic approach

- **Offline** lazy SMT

```
F = CNF_bool( φ )
while true:
   res, M = check_SAT(F)
   if res == true:
      M' = to_T(M)
      res = check_T(M')
      if res == true:
         return SAT
      else:
         F += !M
   else:
      return UNSAT
```

Boolean reasoning

A basic approach

- **Offline** lazy SMT

```
F = CNF_bool( φ )
while true:
   res, M = check_SAT(F)
   if res == true:
      M' = to_T(M)
      res = check_T(M')
      if res == true:
         return SAT
      else:
         F += !M
   else:
      return UNSAT
```

Boolean reasoning

Theory reasoning

A basic approach

- **Offline** lazy SMT

```
F = CNF_bool( φ )
while true:
   res, M = check_SAT(F)
   if res == true:
      M' = to_T(M)
      res = check_T(M')
      if res == true:
         return SAT
      else:
         F += !M
   else:
      return UNSAT
```

Boolean reasoning

Theory reasoning

Block bad solutions

Example

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

Example

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

$$M = \{P_1, P_2, \neg A_1, A_2, \neg A_3, \neg A_4, A_5, A_6\}$$

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

Example

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

$$M = \{P_1, P_2, \neg A_1, A_2, \neg A_3, \neg A_4, A_5, A_6\}$$

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

$$\boxed{\neg(3x_1 - 3x_5 - 4 \leq 6) \mapsto \neg(x_1 - x_5 \leq 10/3) \mapsto (x_1 - x_5 > 10/3)}$$

UNSAT $\rightarrow$ add $\neg M$ and continue

DPLL(T)

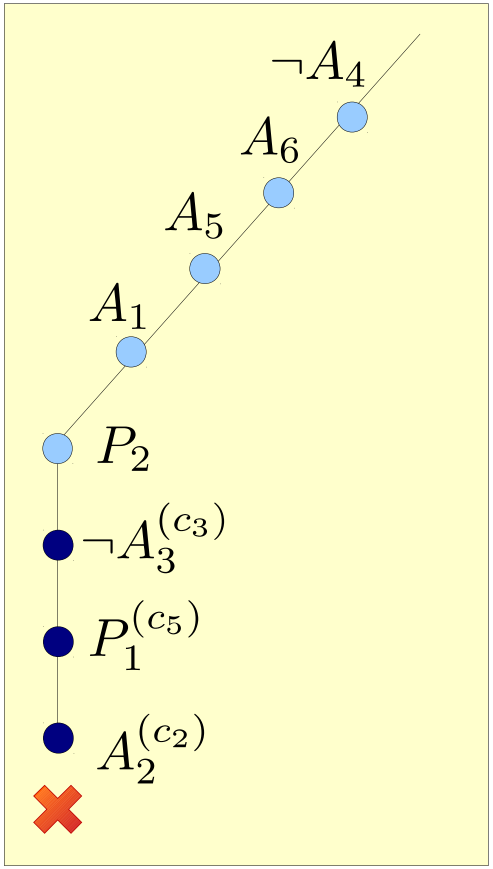
- **Online** approach to lazy SMT
- **Tight integration** between a CDCL-like SAT solver ("DPLL") and the decision procedure for *T* ("T-solver"), based on:
  - *T*-driven backjumping and learning
  - Early pruning
  - *T*-solver incrementality
  - *T*-propagation
  - Filtering of assignments to check
  - Creation of new *T*-atoms and *T*-lemmas "on-demand"
  - ...

T-backjumping and *T*-learning

- When unsat, *T*-solver can produce **reason for 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

Example

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

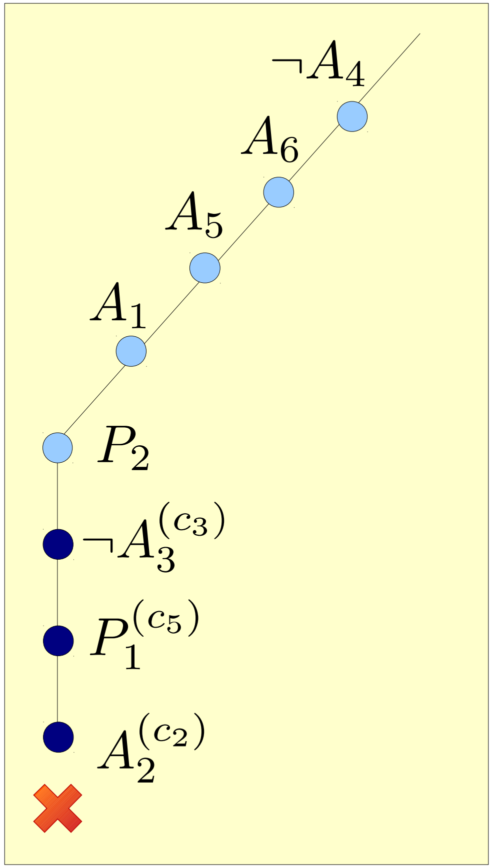


$$M = [\neg A_4, A_6, A_5, A_1, P_2, \neg A_3, P_1, A_2]$$

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

Example

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



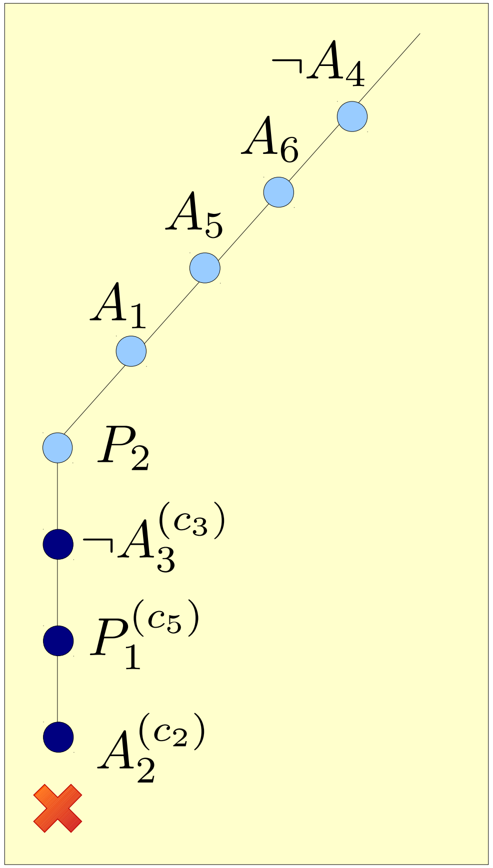
$$M = [\neg A_4, A_6, A_5, A_1, P_2, \neg A_3, P_1, A_2]$$

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

T-conflict set

Example

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

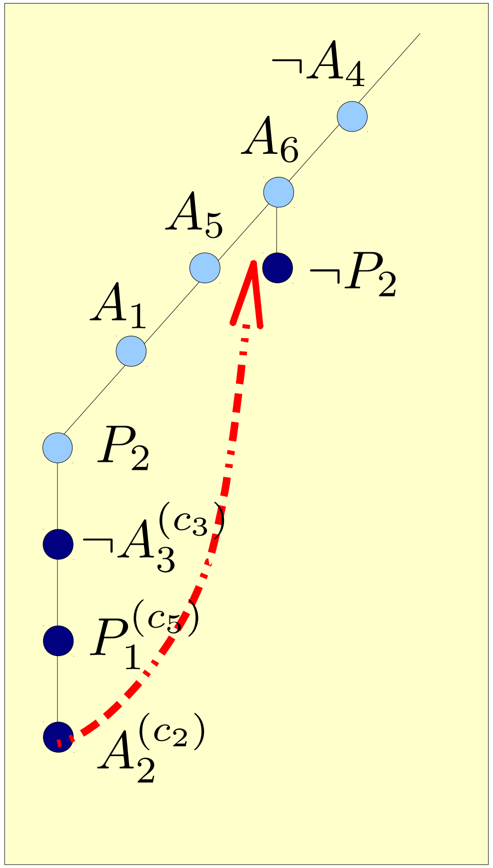


Conflict analysis:

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

Example

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

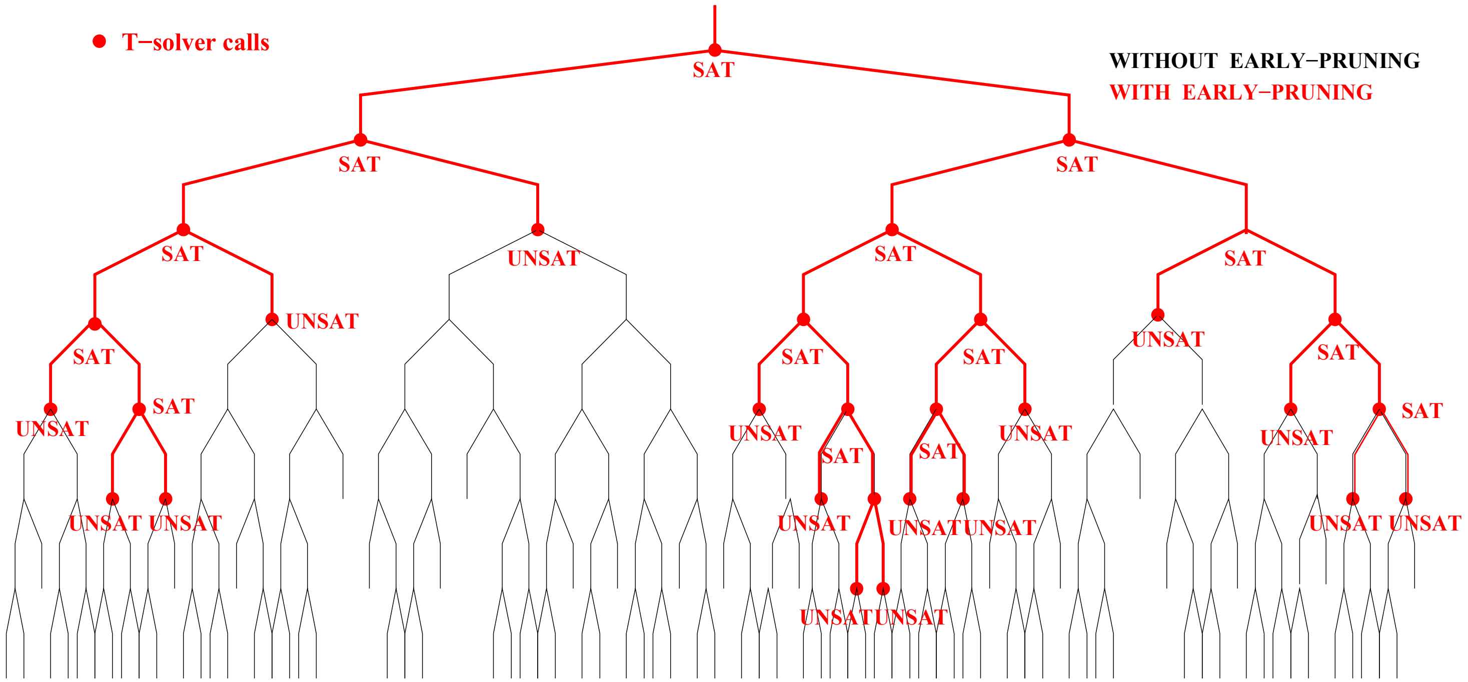


Conflict analysis:

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

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
  - Heuristically, based on some statistics (e.g. effectivenes, …)
- **No need of a conclusive answer** during early pruning calls
  - Can apply **approximate checks**
  - Trade effectiveness for efficiency

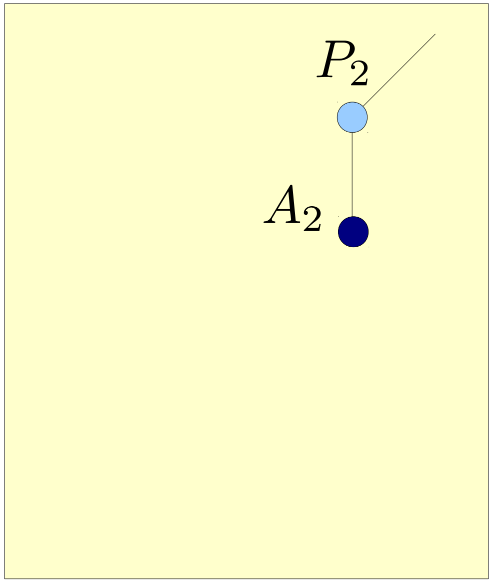
- Example: on linear integer arithmetic, solve only the real relaxation during early pruning calls

Example

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

Example

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

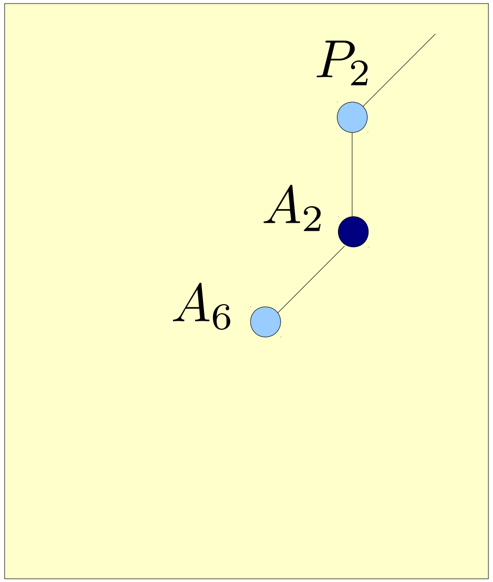


$$M = [P_2, A_2]$$

SAT

Example

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

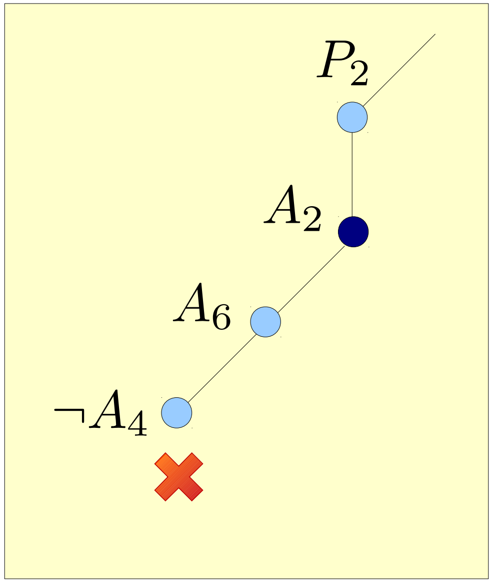


$$M = [P_2, A_2, A_6]$$

SAT

Example

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



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

UNSAT

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

T-solver incrementality

- With early pruning, *T*-solvers invoked **very frequently** on **similar** problems
  - **Stack** of constraints (the assignment stack of CDCL) **incrementally updated**
- **Incrementality:** when a new constraint is added, no need to redo all the computation “from scratch”
- **Backtrackability:** support cheap (stack-based) removal of constraints without “resetting” the internal state
- **Crucial for efficiency**
  - Distinguishing feature for effective integration in DPLL(T)

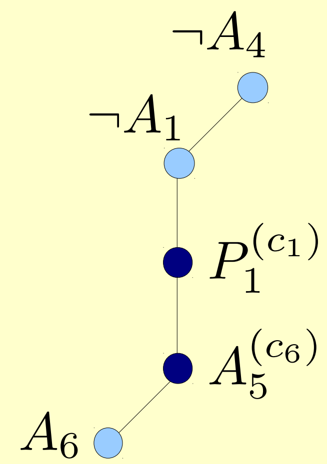
T-propagation

- T-solvers might support **deduction** of unassigned constraints
  - If early pruning check on M returns sat, *T*-solver might also return a set D of unassigned atoms such that $M \models_{\mathcal{T}} l$ 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 $\neg M \vee l$ (*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

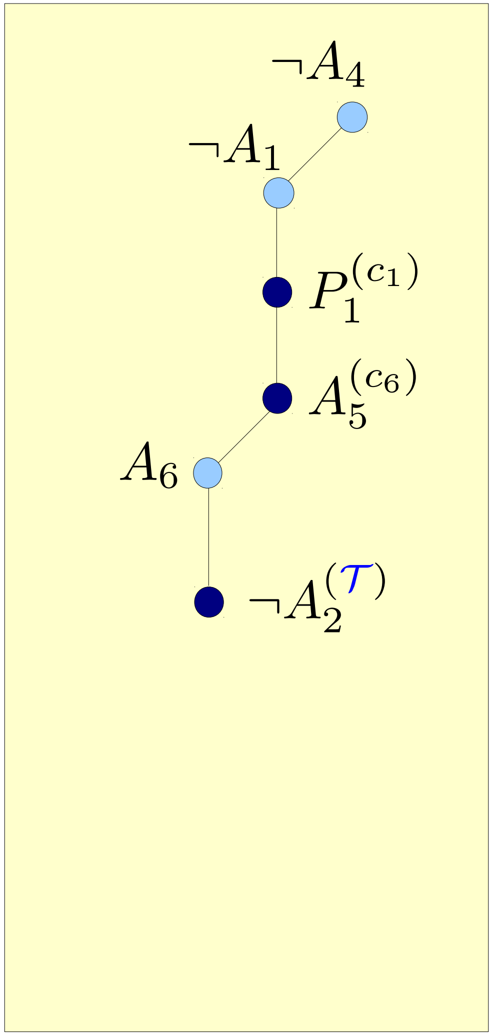
$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \varphi^{\text{Bool}} \stackrel{\text{def}}{=} \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & A_1 \vee P_1 \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \neg P_2 \vee A_2 \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \neg A_3 \vee \neg P_2 \\
c_4: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

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



Example

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



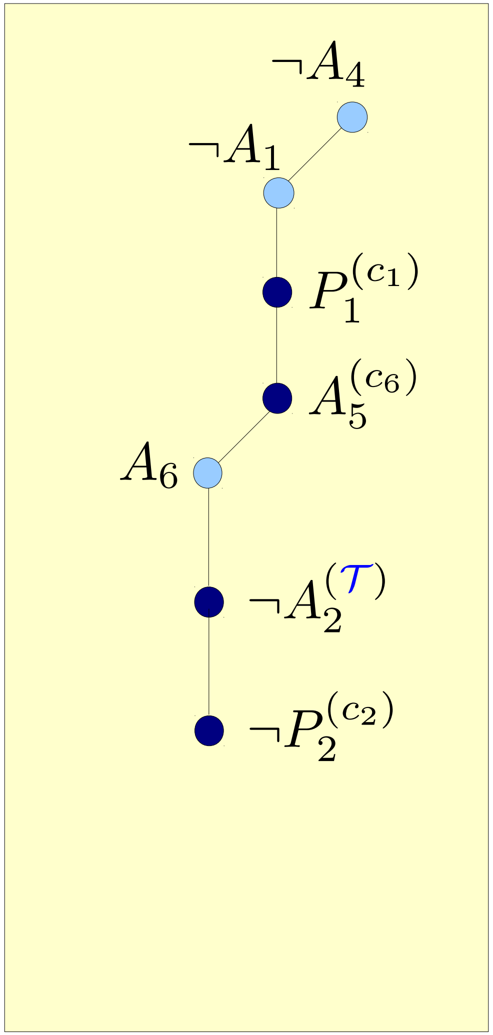
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6]$$

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

Example

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

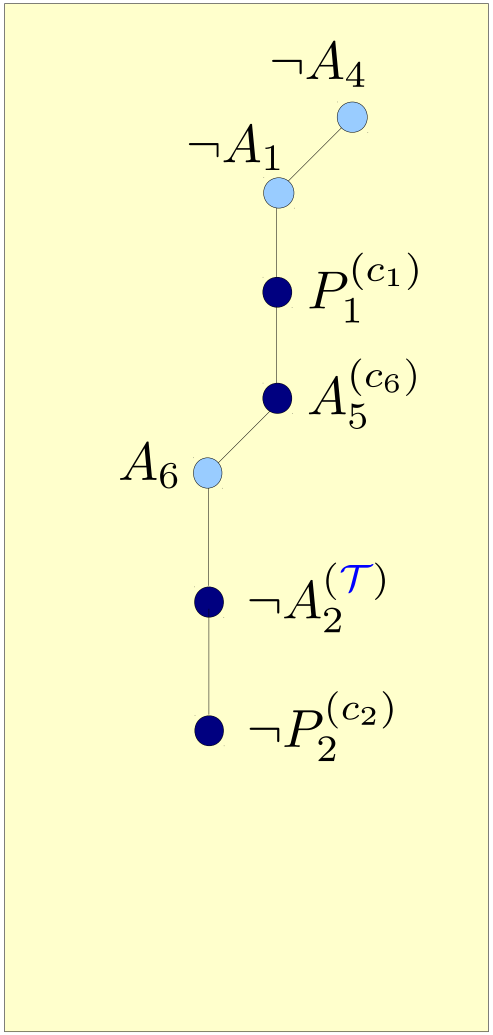
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2]$$



Example

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

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

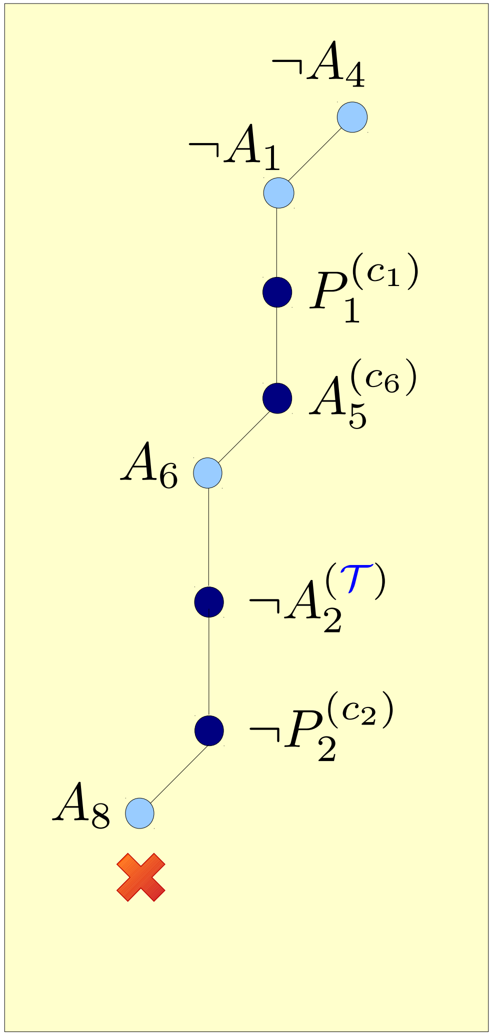


Example

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

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

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

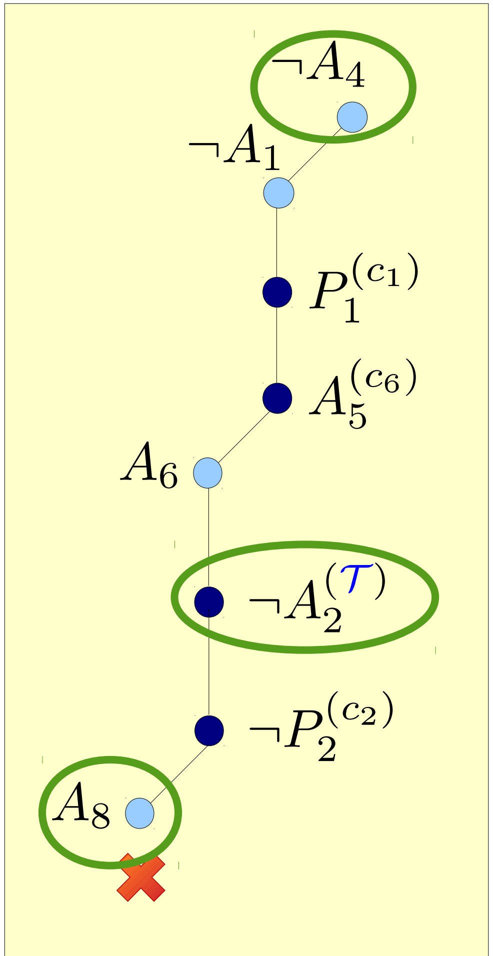


Example

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

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

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



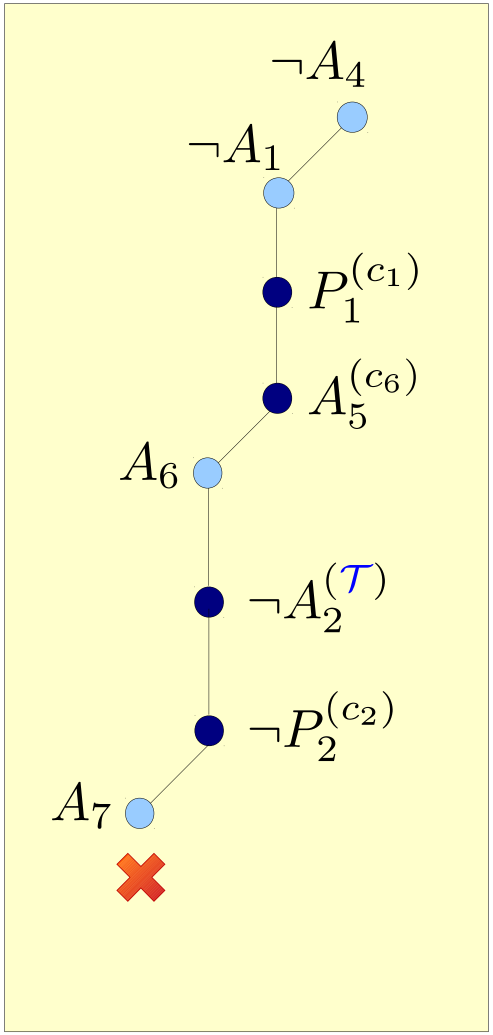
Conflict analysis → compute T-reason for $\neg A_2$

Example

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

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

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

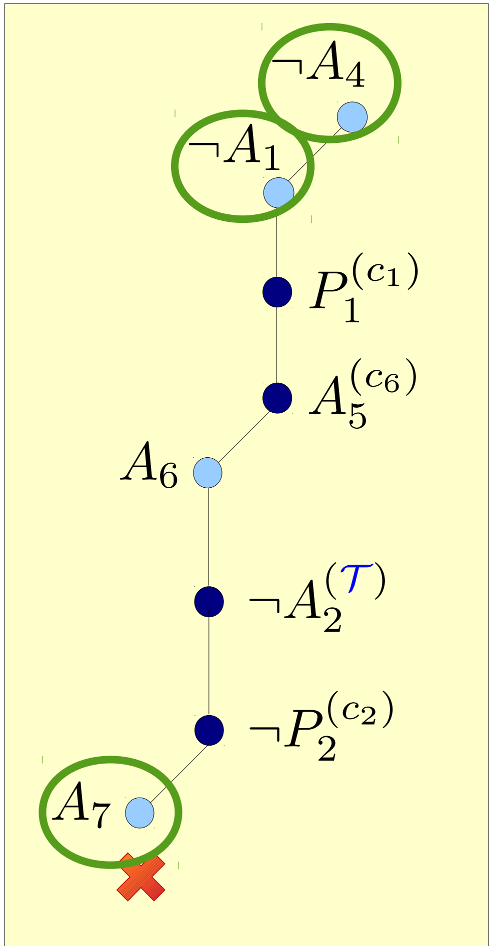


Example

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

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

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



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

Filtering of assignments

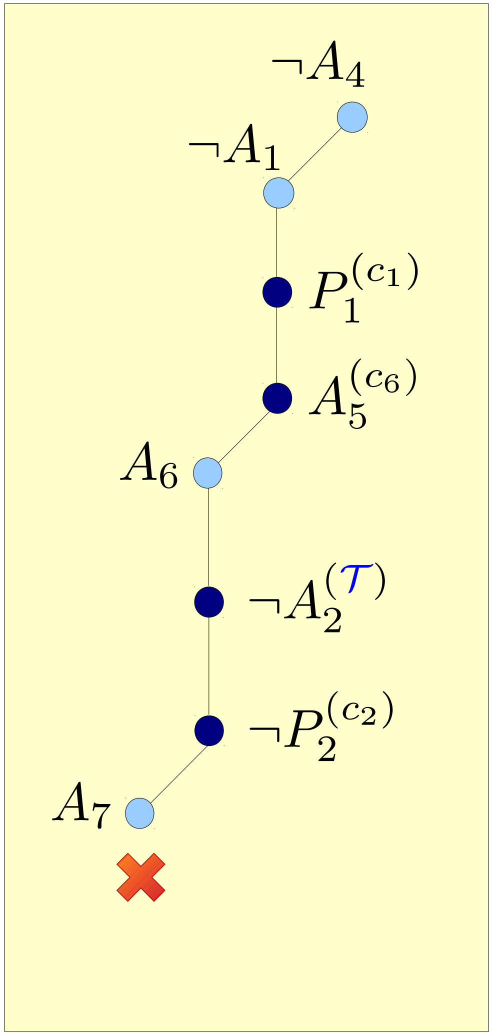
- Remove **unnecessary literals** from current assignment M
  - **Irrelevant** literals: l s.t. $M \setminus \{l\} \models \varphi$ (φ arbitary, not CNF)
  - **Ghost** literals: l occurs only in clauses satisfied by $M \setminus \{l\}$
  - **Pure** literals: $\neg l \in M$ and l occurs only positively in φ
    - Note: this is **not** the pure-literal rule of SAT!
- **Pros:**
  - reduce effort for *T*-solver
  - increases the chances of finding a solution
- **Cons:**
  - may weaken the effect of early pruning (esp. with *T*-propagation)
  - may introduce overhead in SAT search
- Typically used for expensive theories

Example

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

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

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



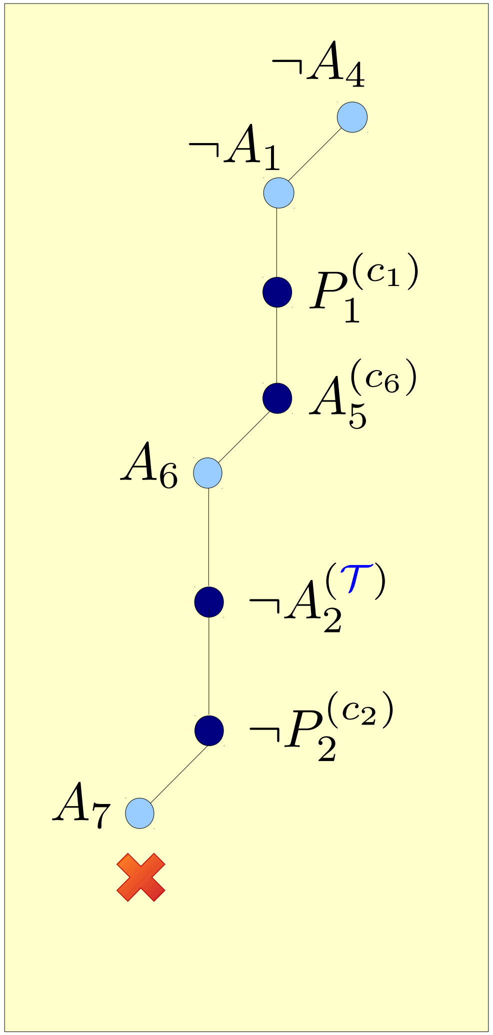
Example

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

Ghost!

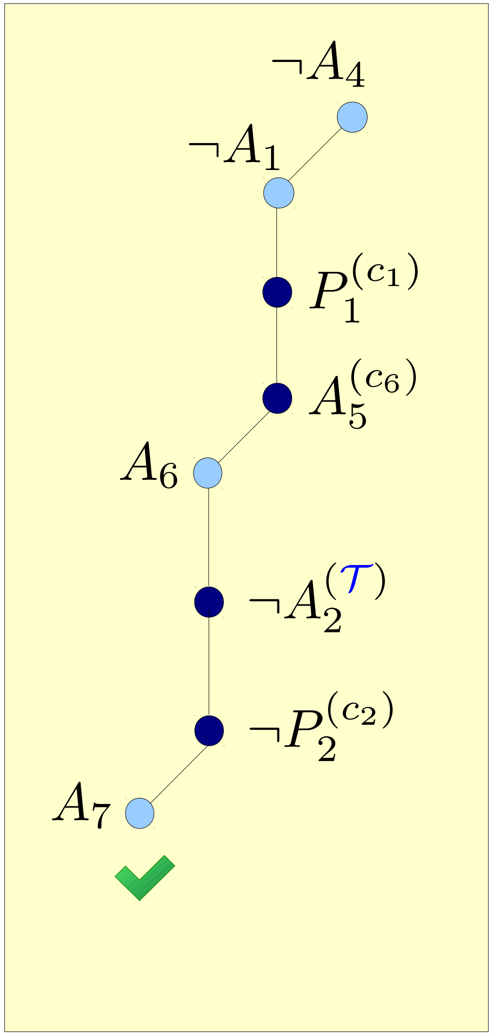
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2, A_7]$$

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



Example

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



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

SAT

T-atoms and *T*-lemmas on demand

- Some *T*-solvers might need to perform **internal case splits** to decide satisfiability
  - Example: linear integer arithmetic

$$(x - 3y \leq 0), (y - 2x \leq 0), (x + 3y \leq 3) \mapsto \begin{cases} \text{case } (y \leq 0) \mapsto \bot \\ \text{case } (y \geq 1) \mapsto \bot \end{cases}$$

- **Splitting on-demand**: use the SAT solver for case splits
  - Encode splits as *T*-valid clauses (*T*-lemmas) with fresh *T*-atoms
  - Generated on-the-fly during search, when needed
  - **Benefits:** reuse the efficient SAT search
    - simplify the implementation
    - exploit advanced search-space exploration techniques (backjumping, learning, restarts, ...)
  - **Potential drawback:** “pollute” the SAT search

T-atoms and *T*-lemmas on demand

- *T*-solver can now return **unknown** also for complete checks
  - In this case, it must also produce one or more *T*-lemmas
  - SAT solver learns the lemmas and continues searching
  - eventually, *T*-solver **can decide** sat/unsat

- **Termination** issues
  - If SAT solver drops lemmas, might get into an infinite loop
    - similar to the Boolean case (and the “basic” SMT case), similar solution (e.g. monotonically increase # of kept lemmas)
  - *T*-solver can generate an infinite number of new *T*-atoms!
    - For several theories (e.g. linear integer arithmetic, arrays) enough to draw new *T*-atoms **from a finite set** (dependent on the input problem)

T-solver interface example

```
class TheorySolver {
    bool tell_atom(Var boolatom, Expr tatom);

    void new_decision_level();
    void backtrack(int level);

    void assume(Lit l);
    lbool check(bool approx);

    void get_conflict(LitList &out);

    Lit get_next_implied();
    bool get_explanation(Lit implied, LitList &out);

    bool get_lemma(LitList &out);

    Expr get_value(Expr term);
};
```

DPLL(T) example

```
def DPLL-T():
   while True:
      conflict = False
      if unit_propagation():
         res = T.check(!all_assigned())
         if res == False: conflict = True
         elif res == True: conflict = theory_propagation()
         elif learn_T_lemmas(): continue
         elif !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)
```

DPLL(T) example

```
def DPLL-T():
    while True:
        conflict = False
        if unit_propagation():
            res = T.check(!all_assigned())
            if res == False: conflict = True
            elif res == True: conflict = theory_propagation()
            elif learn_T_lemmas(): continue
            elif !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)
```

call T.assume(lit)

call T.get_next_implied()

call T.get_lemma()

call T.new_decision_level()
T.assume(lit)

call T.get_value(e)

call T.get_conflict(c)
T.get_explanation(l, e)

call T.backtrack(lvl)

Outline

Introduction

The DPLL(T) architecture

Some relevant T-solvers

Combination of theories

Quantifiers in DPLL(T)

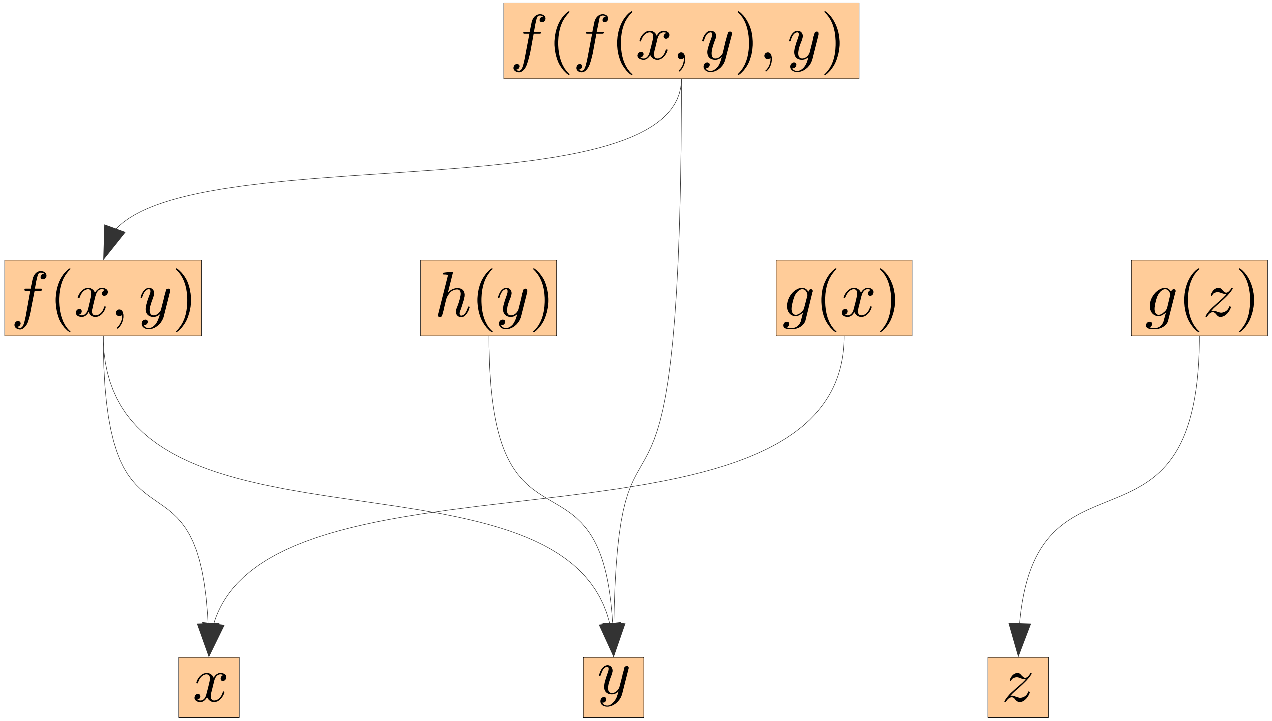
Equality (EUF)

- Polynomial time $O(n \log n)$ **congruence closure** procedure
- Fully incremental and backtrackable (stack-based)
- Supports **efficient *T*-propagation**
  - Exhaustive for positive equalities
  - Incomplete for disequalities
- Lazy explanations and conflict generation

- Typically used as a **“core”** *T*-solver
- Supports efficient extensions, e.g.
  - Integer offsets
  - Bit-vector slicing and concatenation

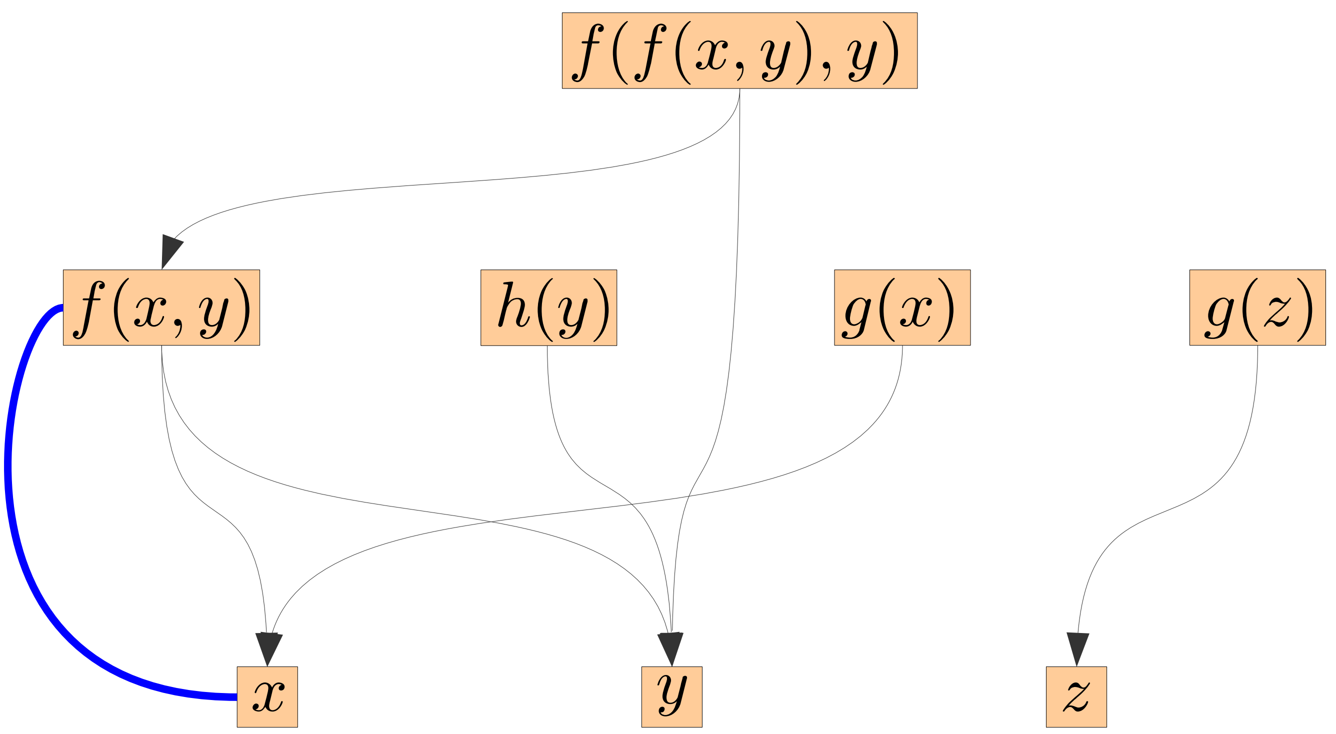
Example

$$[(f(x,y) = x), (h(y) = g(x)), (f(f(x,y),y) = z), \neg(g(x) = g(z))]$$



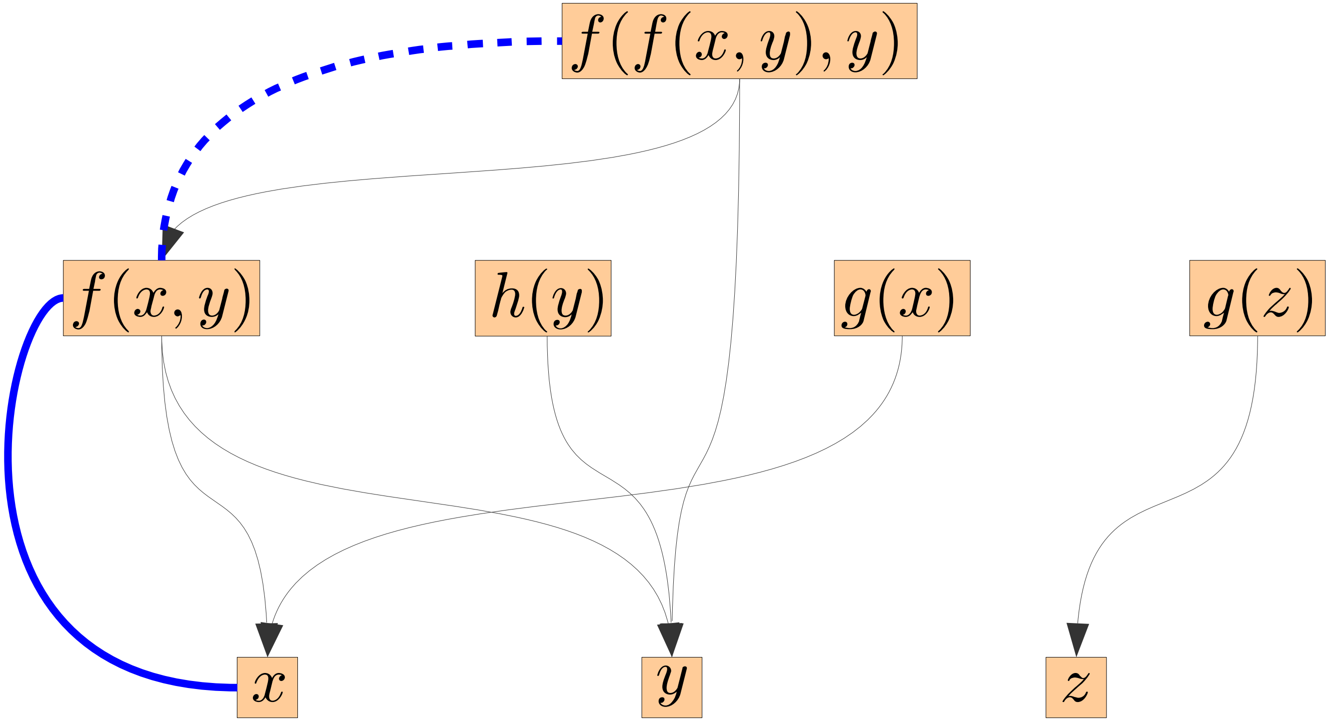
Example

$$[\boxed{(f(x,y) = x)}, (h(y) = g(x)), (f(f(x,y),y) = z), \neg(g(x) = g(z))]$$



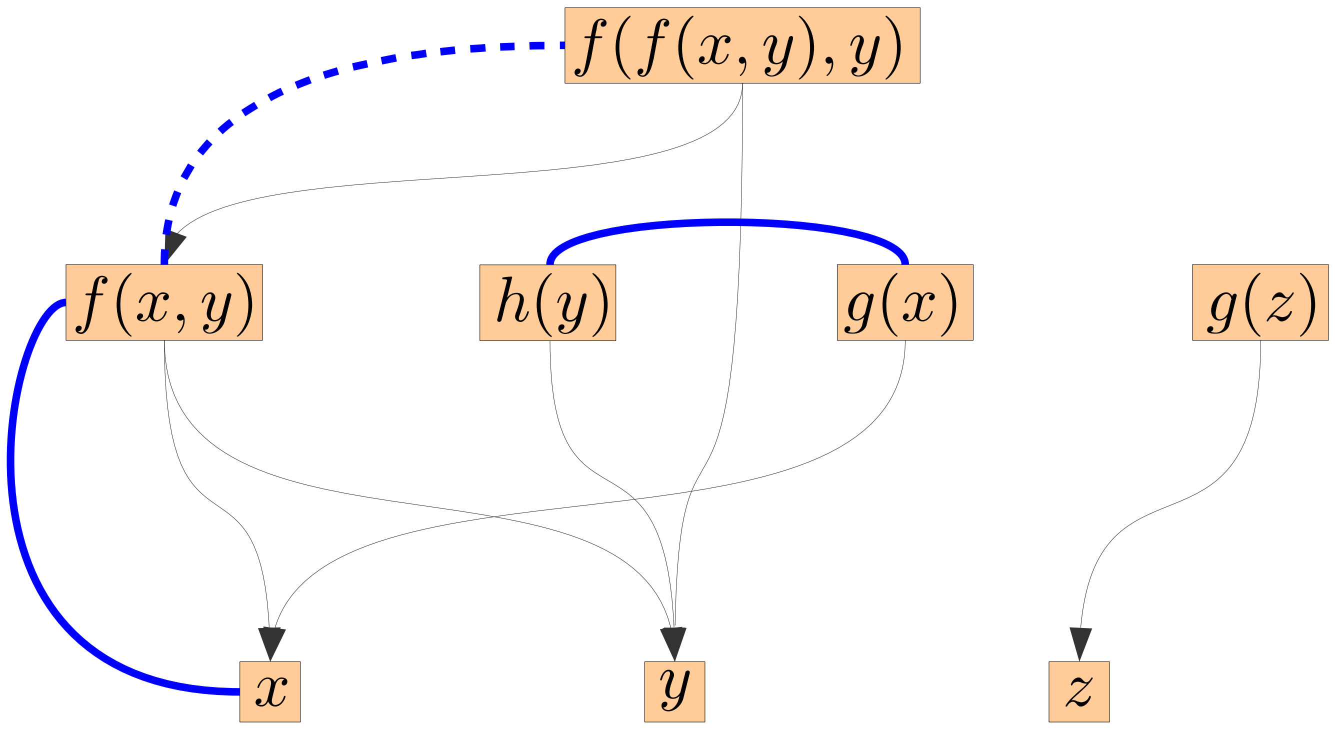
Example

$$[\boxed{(f(x,y) = x)}, (h(y) = g(x)), (f(f(x,y),y) = z), \neg(g(x) = g(z))]$$



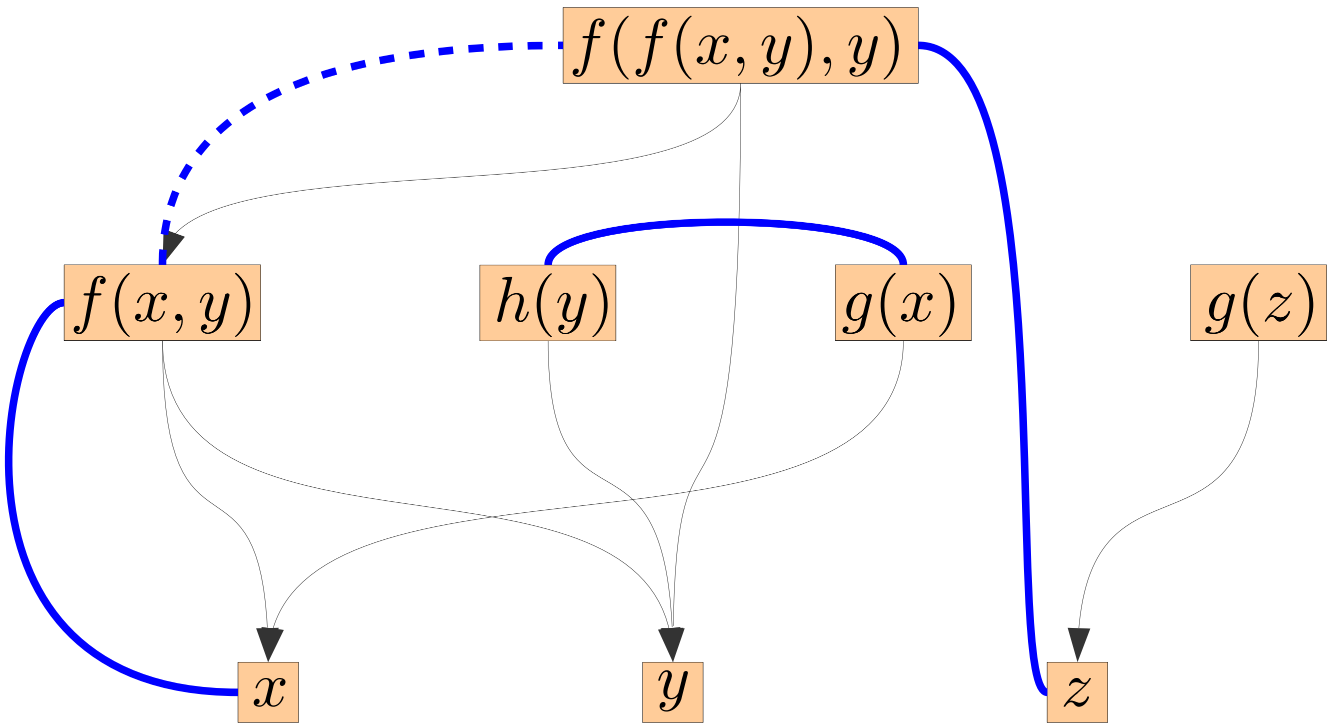
Example

$$[(f(x,y) = x), \boxed{(h(y) = g(x))}, (f(f(x,y),y) = z), \neg(g(x) = g(z))]$$



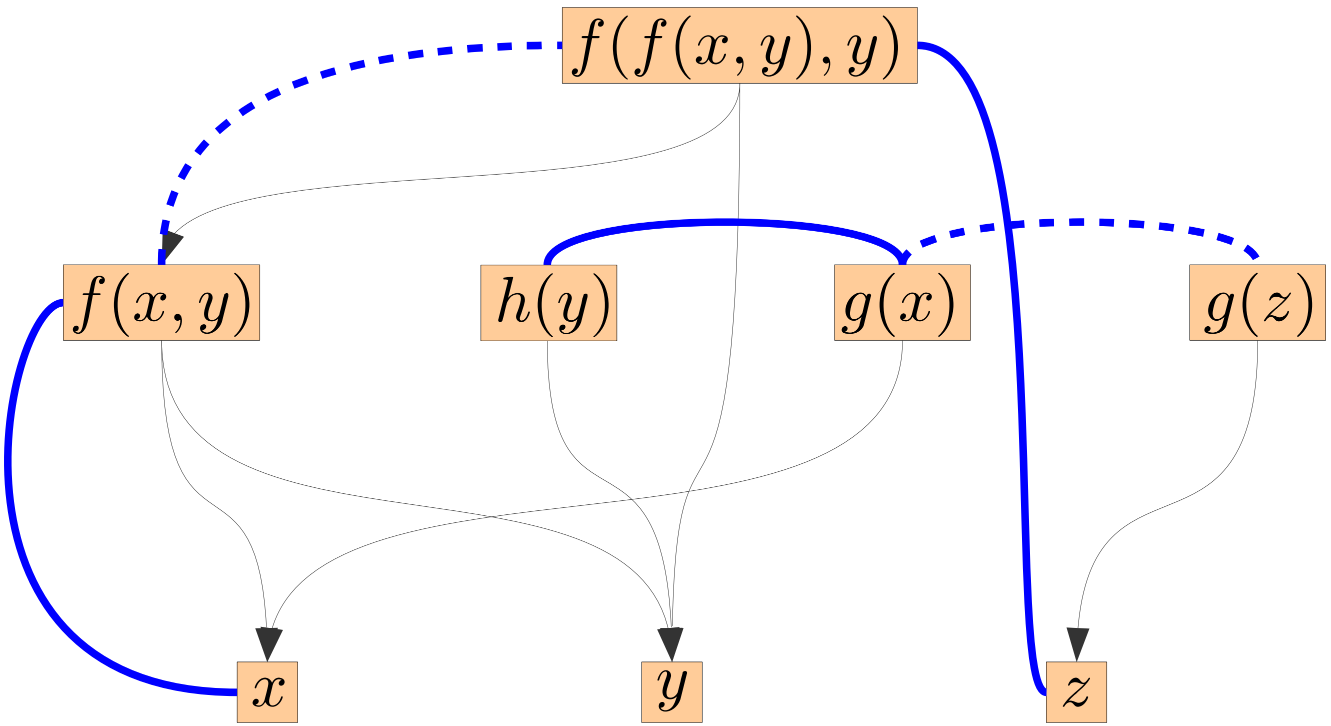
Example

$$[(f(x,y)=x),(h(y)=g(x)),\boxed{(f(f(x,y),y)=z)},\neg(g(x)=g(z))]$$



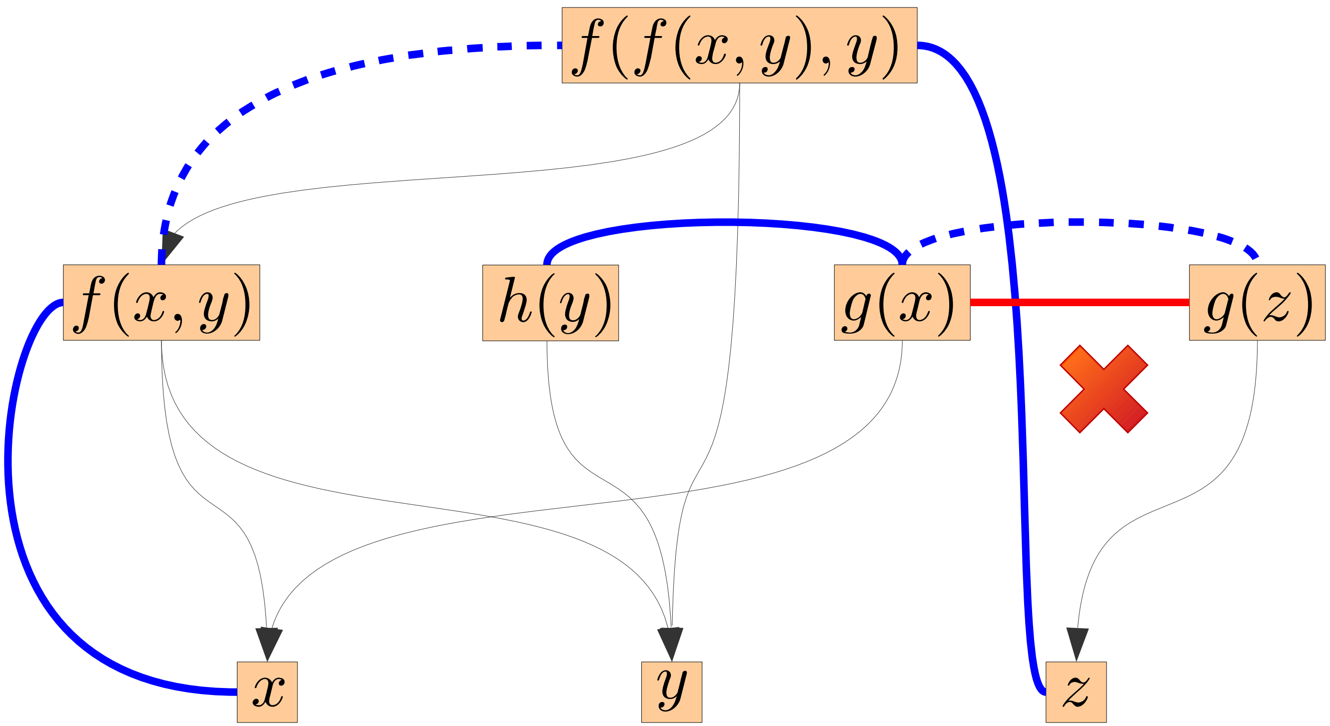
Example

$$[(f(x,y)=x),(h(y)=g(x)),\boxed{(f(f(x,y),y)=z)},\neg(g(x)=g(z))]$$



Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$

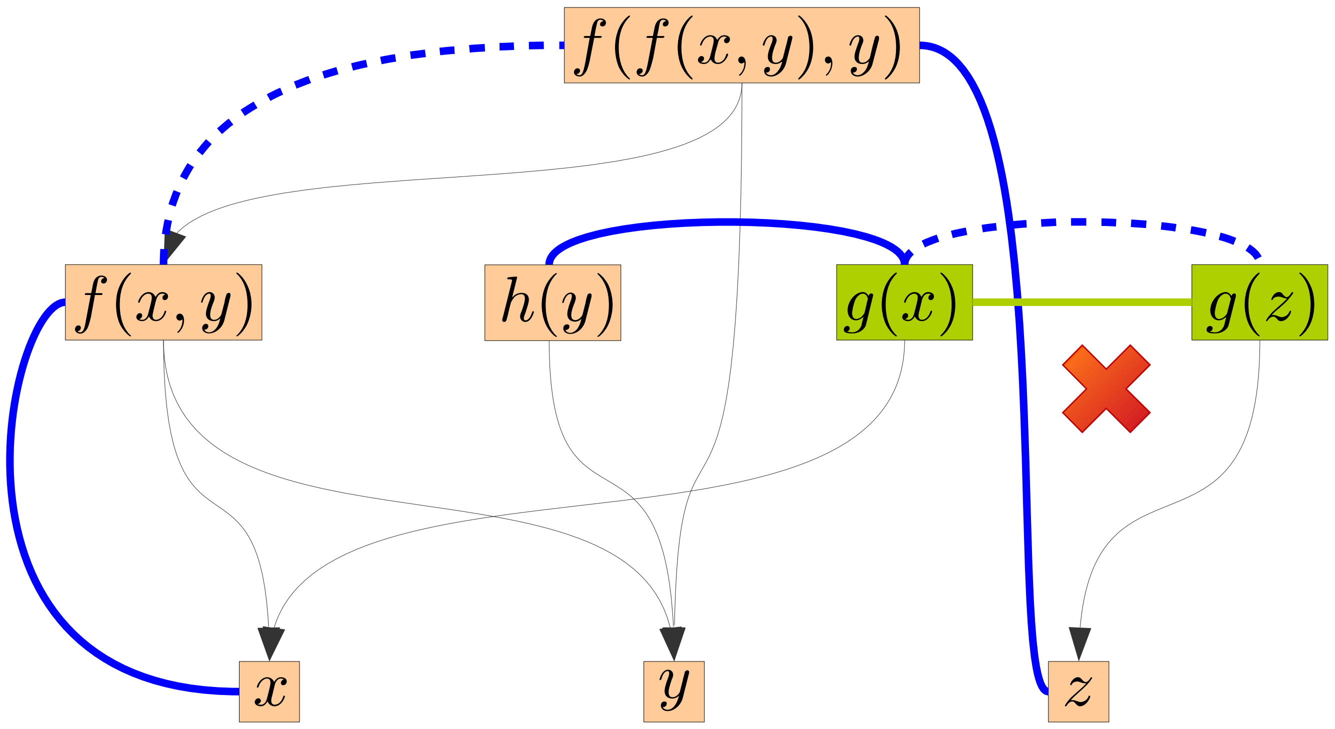


Example

$$[(f(x,y) = x), (h(y) = g(x)), (f(f(x,y),y) = z), \boxed{\neg(g(x) = g(z))}]$$

`get_conflict():`

$$\neg(g(x) = g(z))$$

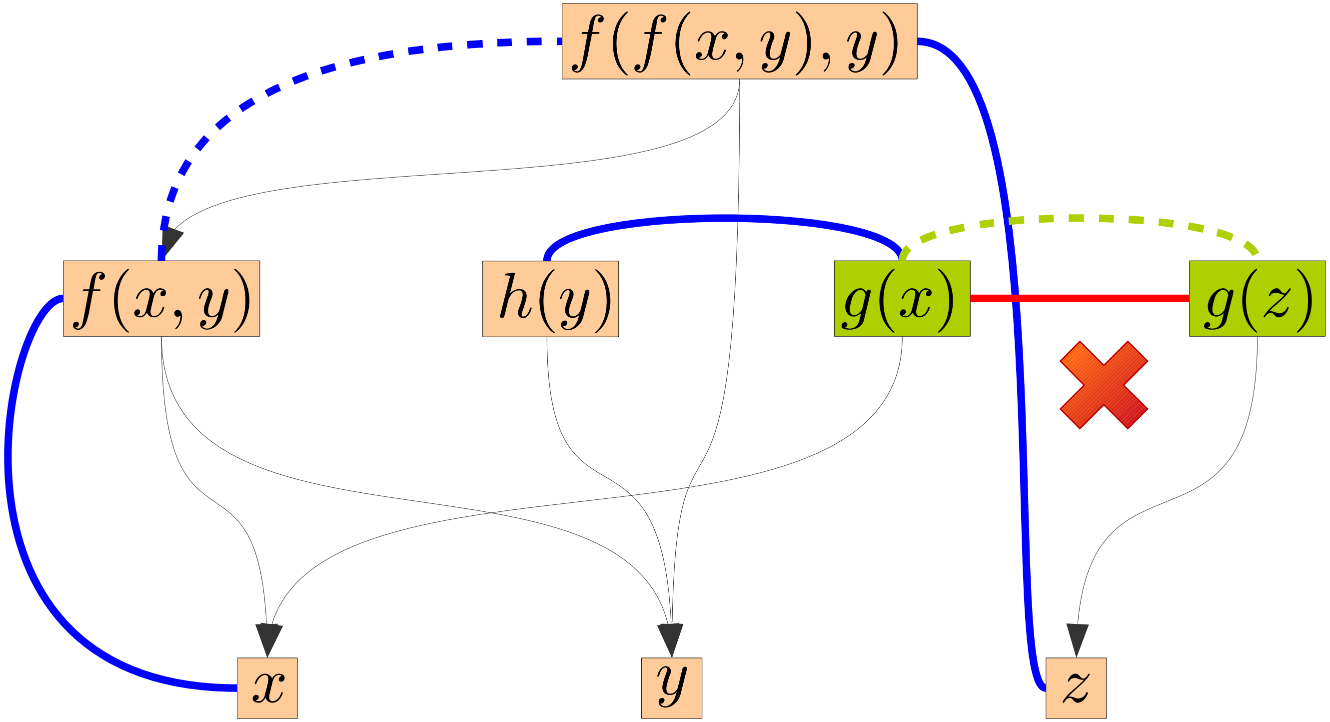


Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$

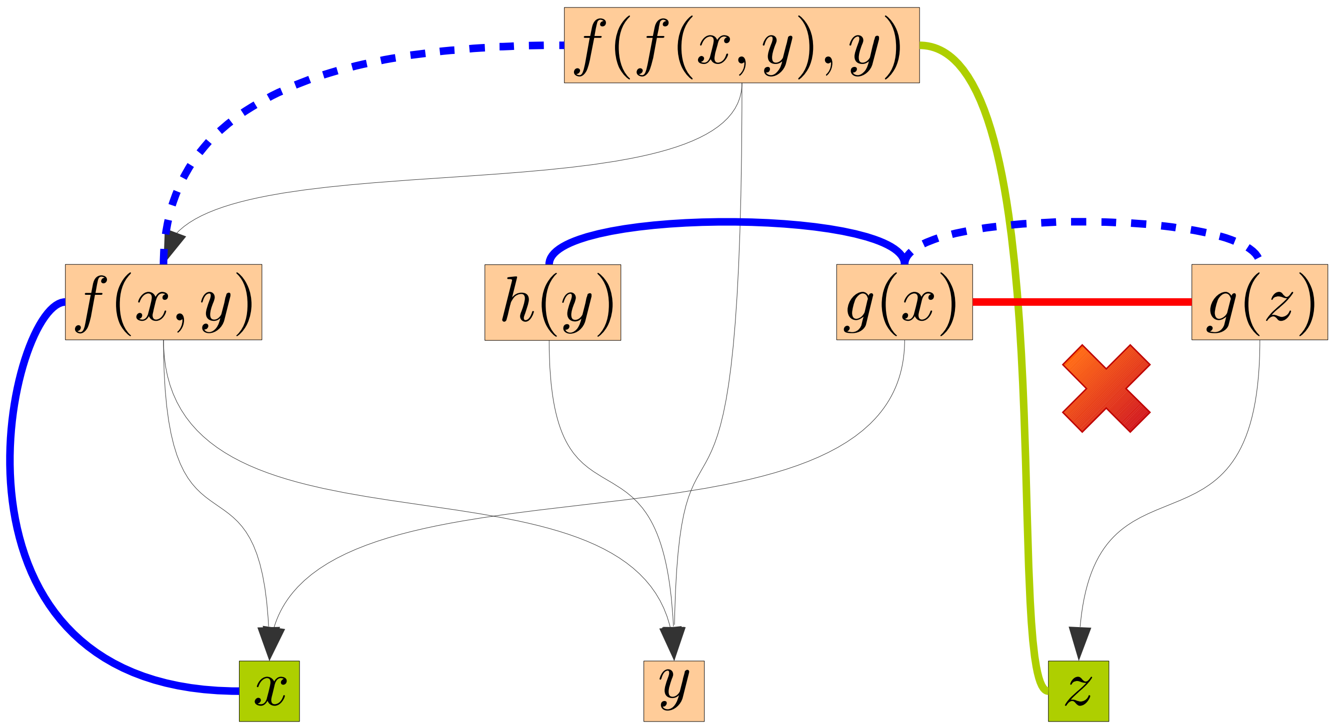
`get_conflict():`

$$\neg(g(x)=g(z))$$



Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$

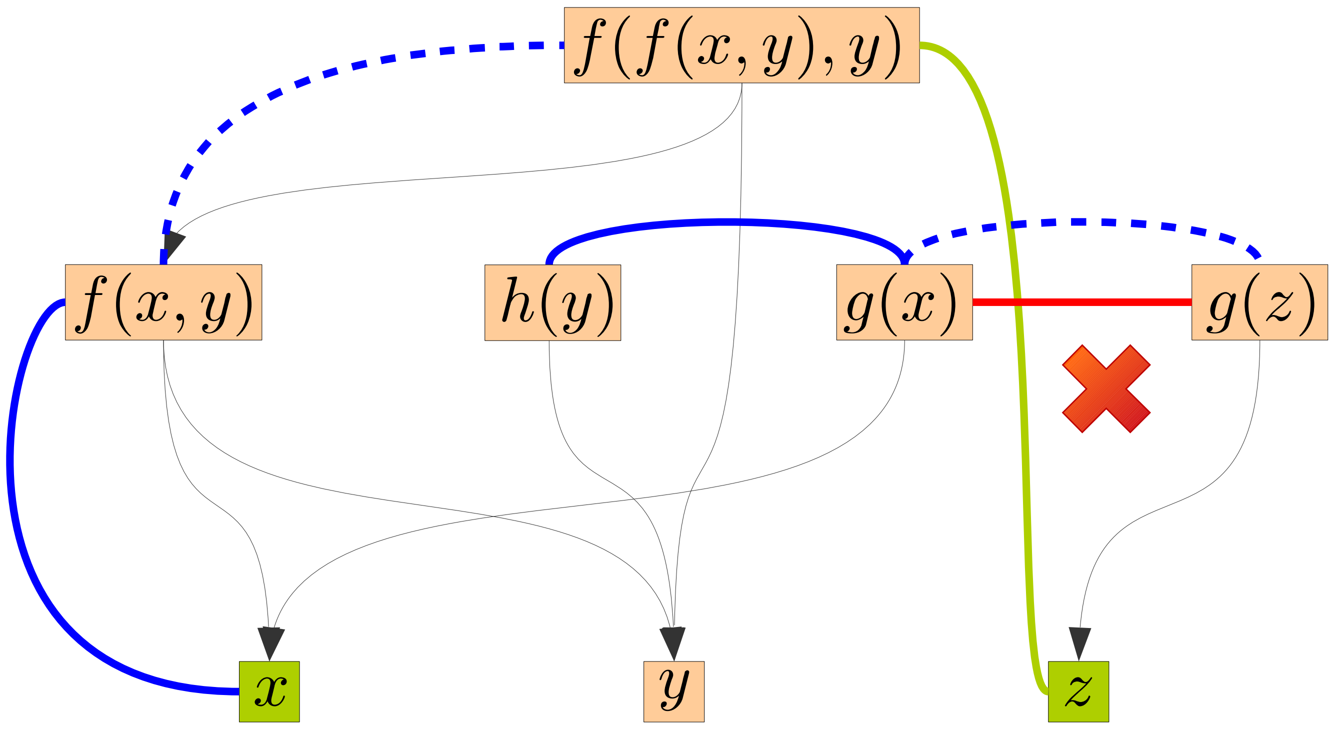


`get_conflict():`

$$\neg(g(x)=g(z))$$

Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$



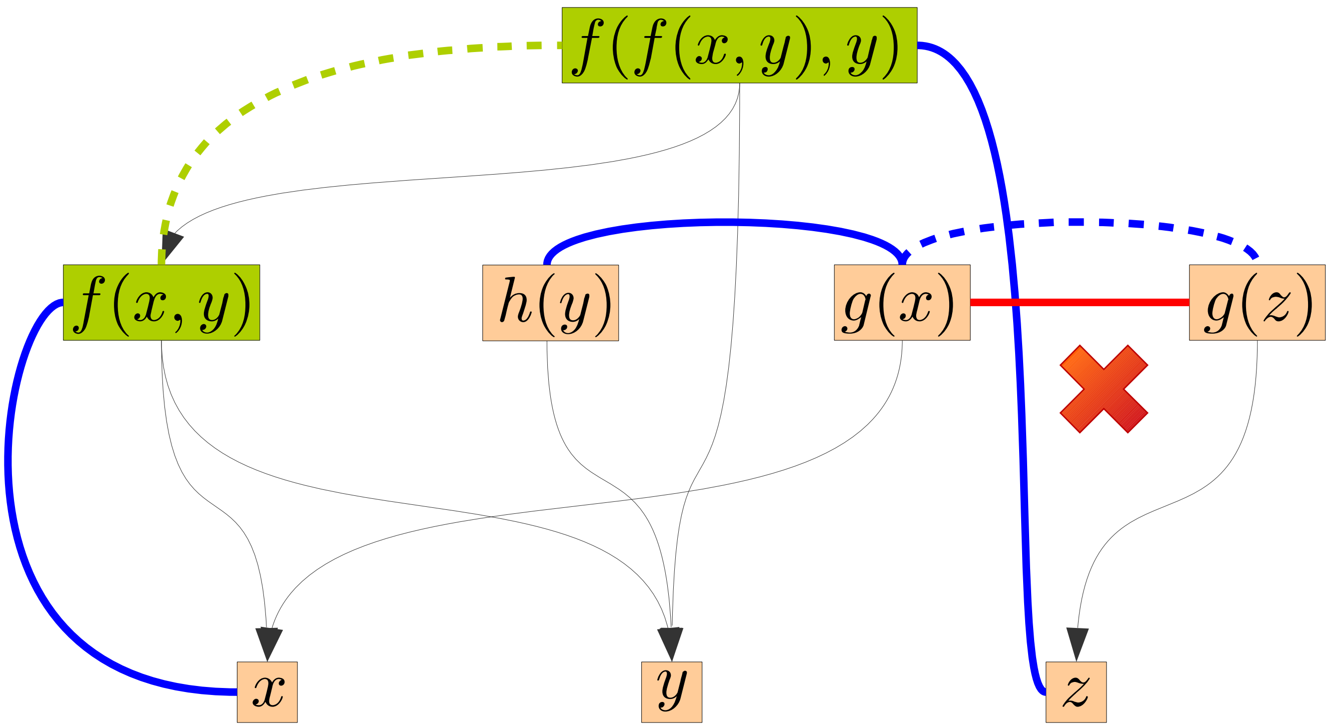
`get_conflict():`

$$\neg(g(x)=g(z))$$

$$(f(f(x,y),y)=z)$$

Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$




```
get_conflict():
```

$$\neg(g(x)=g(z))$$

$$(f(f(x,y),y)=z)$$

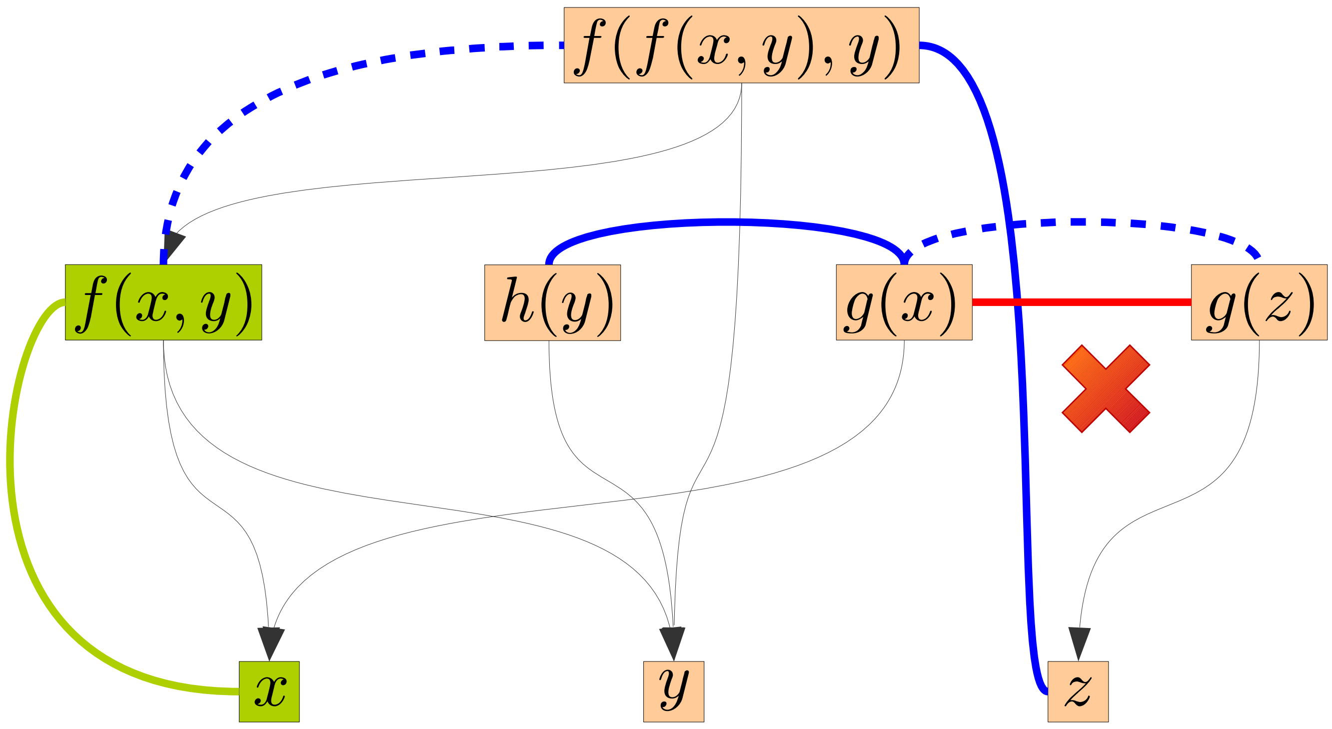
Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$

get_conflict():

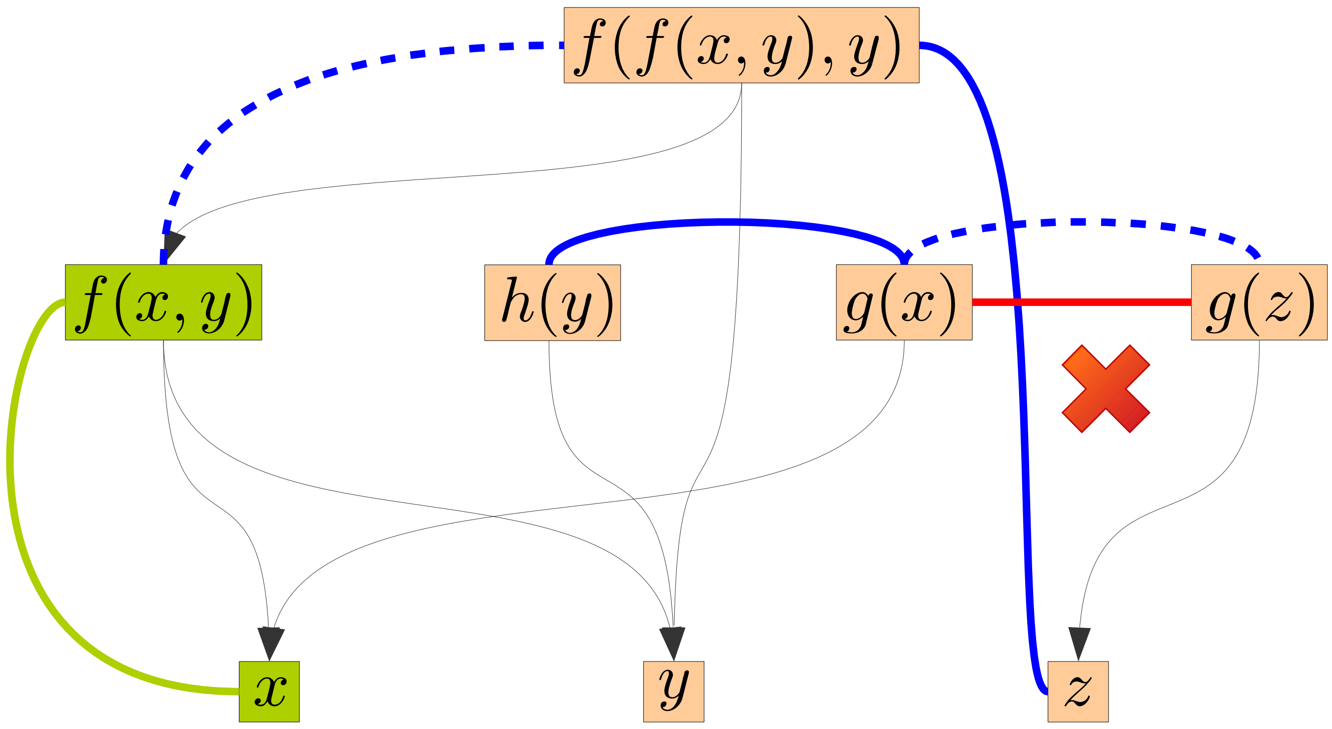
$$\neg(g(x)=g(z))$$

$$(f(f(x,y),y)=z)$$



Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$



`get_conflict():`

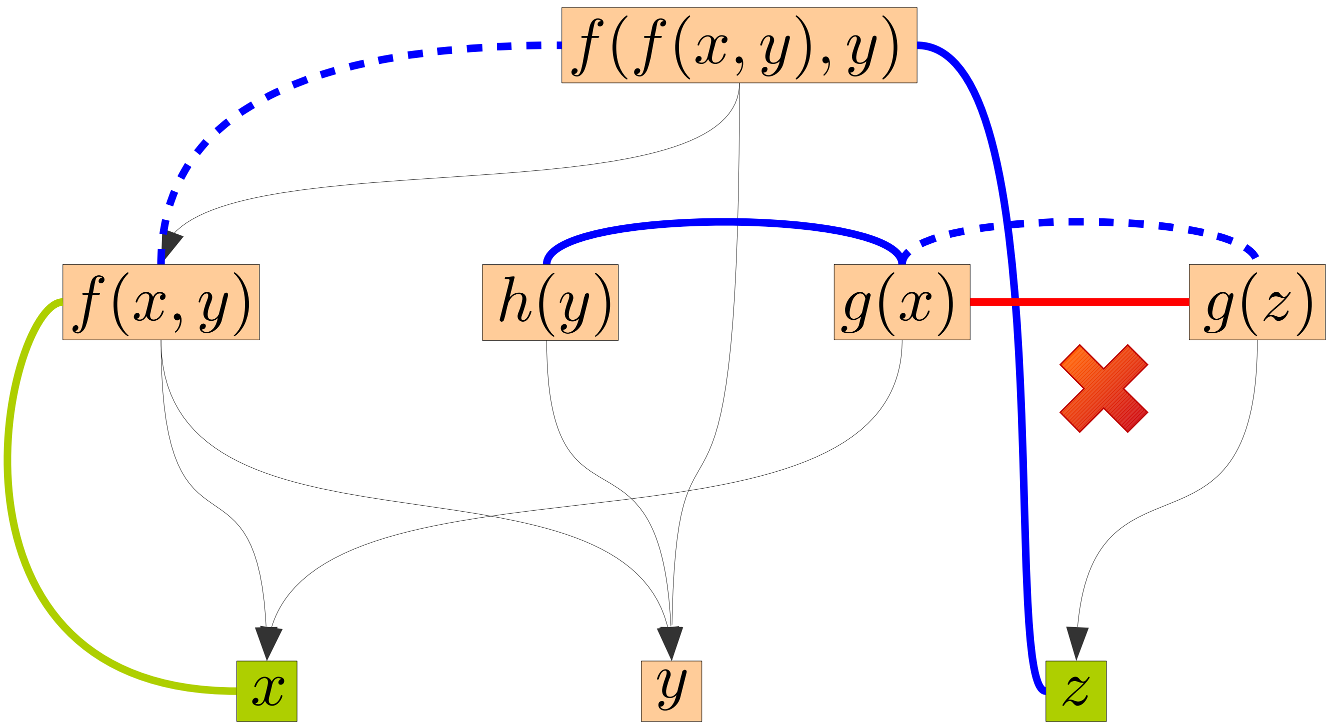
$$\neg(g(x)=g(z))$$

$$(f(f(x,y),y)=z)$$

$$(f(x,y)=x)$$

Example

$$[(f(x,y)=x),(h(y)=g(x)),(f(f(x,y),y)=z),\boxed{\neg(g(x)=g(z))}]$$



`get_conflict():`

$$\neg(g(x)=g(z))$$
$$(f(f(x,y),y)=z)$$
$$(f(x,y)=x)$$

Linear Rational Arithmetic (LRA)

- Constraints of the form $\sum_i a_i x_i \le c$
- Variant of **simplex** specifically designed for DPLL(T)
  - **Very efficient** backtracking
  - Incremental checks
  - **Cheap deduction** of unsassigned literals
  - **Minimal explanations** generation
  - Can handle efficiently also strict inequalities
    - Rewrite $(t < 0)$ to $(t + \varepsilon \le 0)$, treat ε symbolically
  - Worst-case exponential (although LRA is polynomial), but fast in practice

Simplex for DPLL(T)

Preprocessing: $\sum a_h x_h \leq u \mapsto x_{\text{slack}} = \sum a_h x_h \wedge x_{\text{slack}} \leq u$

Tableau of equations (fixed) + bounds (added/removed)

Candidate solution β always consistent with the tableau

$$
\begin{array}{rl}
x_{\text{slack } 1} = & \\
x_{\text{slack } 2} = & \\
\vdots & \\
x_{\text{slack } i} = & \boxed{a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{im}x_m} \\
\vdots & \\
x_{\text{slack } n} = &
\end{array}
\qquad
\boxed{\begin{array}{c} -\infty \\ l_2 \\ \vdots \\ l_i \\ \vdots \\ l_n \end{array}}
\leq
\boxed{\begin{array}{c} x_{\text{slack } 1} \\ x_{\text{slack } 2} \\ \vdots \\ x_{\text{slack } i} \\ \vdots \\ x_{\text{slack } n} \end{array}}
\leq
\boxed{\begin{array}{c} u_1 \\ +\infty \\ \vdots \\ u_i \\ \vdots \\ u_n \end{array}}
$$

Simplex for DPLL(T)

Preprocessing: $\sum a_h x_h \leq u \mapsto x_{\text{slack}} = \sum a_h x_h \wedge x_{\text{slack}} \leq u$

Tableau of equations (fixed) + bounds (added/removed)

Candidate solution β always consistent with the tableau

$x_j =$

$x_{\text{slack } 2} =$

$\vdots$

$x_{\text{slack } i} = \sum_{a'_{ij}>0} a'_{ij} x_{\text{slack } j} + \sum_{a'_{ik}<0} a'_{ik} x_{\text{slack } k}$

$\vdots$

$x_h =$

Pivoting steps to make β satisfy the bounds

$$\begin{pmatrix} -\infty \\ l_2 \\ \vdots \\ l_i \\ \vdots \\ l_n \end{pmatrix} \leq \begin{pmatrix} x_{\text{slack } 1} \\ x_{\text{slack } 2} \\ \vdots \\ x_{\text{slack } i} \\ \vdots \\ x_{\text{slack } n} \end{pmatrix} \leq \begin{pmatrix} u_1 \\ +\infty \\ \vdots \\ u_i \\ \vdots \\ u_n \end{pmatrix}$$

Simplex for DPLL(T)

Preprocessing: $\sum a_h x_h \leq u \mapsto x_{\text{slack}} = \sum a_h x_h \wedge x_{\text{slack}} \leq u$

Tableau of equations (fixed) + bounds (added/removed)

Candidate solution β always consistent with the tableau

$\beta(x_{\text{slack } i}) < l_i$

and for the others β can not change $\Rightarrow$ conflict!

$$
\begin{aligned}
x_j &= \\
x_{\text{slack } 2} &= \\
&\vdots \\
x_{\text{slack } i} &= \sum_{a'_{ij}>0} a'_{ij} x_{\text{slack } j} + \sum_{a'_{ik}<0} a'_{ik} x_{\text{slack } k} \\
&\vdots \\
x_h &=
\end{aligned}
$$

$-\infty$		$x_{\text{slack } 1}$		u_1
l_2		$x_{\text{slack } 2}$		$+\infty$
$\vdots$	$\leq$	$\vdots$	$\leq$	$\vdots$
l_i	$>$	$x_{\text{slack } i}$		u_i
$\vdots$		$\vdots$		$\vdots$
l_n		$x_{\text{slack } n}$		u_n

Simplex for DPLL(T)

Preprocessing: $\sum a_h x_h \le u \mapsto x_{\text{slack}} = \sum a_h x_h \wedge x_{\text{slack}} \le u$

Tableau of equations (fixed) + bounds (added/removed)

Candidate solution β always consistent with the tableau

$\beta(x_{\text{slack } i}) < l_i$
and for the others β can not change $\Rightarrow$ conflict!

$x_j =$

$x_{\text{slack } 2} =$

$\vdots$

$$x_{\text{slack } i} = \sum_{a'_{ij}>0} a'_{ij} x_{\text{slack } j} + \sum_{a'_{ik}<0} a'_{ik} x_{\text{slack } k}$$

$\vdots$

$x_h =$

```
get_conflict():
```

$\{(\sum a_h x_h \le u)\}_j$ for $x_{\text{slack } j}$ $\cup$

$\{(\sum a_h x_h \ge l)\}_k$ for $x_{\text{slack } k}$ $\cup$

$\{(\sum a_h x_h \ge l)\}_i$ for $x_{\text{slack } i}$

Linear Integer Arithmetic (LIA)

- **NP-complete** problem
- Popular approach: simplex + **branch and bound**
  - Approximate checks solve only over the rationals
  - In complete checks, force integrality of variables by adding either:
    - **Branch and bound** lemmas $(x \leq \lfloor c \rfloor) \vee (x \geq \lceil c \rceil)$
    - **Cutting plane** lemmas
      - Inequalities entailed by the current constraints, excluding only non-integer solutions
      - Gomory cuts commonly used
    - Using splitting on-demand
  - Might also include other specialized sub-solvers for tractable fragments
    - E.g. specialized equational reasoning

Arrays (A)

- **Read** (rd) and **write** (wr) operations over arrays
- **Equality** over array variables (extensionality)
- Example: $\mathsf{wr}(a, i, x) = \mathsf{wr}(b, i, \mathsf{rd}(a, j, y)) \wedge \neg(a = b)$
- Common approach: reduction to EUF via **lazy axiom instantiation**
  - read-over-write: $\forall a.\forall i.\forall x.(\mathsf{rd}(\mathsf{wr}(a, i, x), i) = x)$
    $\forall a.\forall i.\forall j.\forall x.((i \neq j) \rightarrow \mathsf{rd}(\mathsf{wr}(a, i, x), j) = \mathsf{rd}(a, j))$
  - extensionality: $\forall a.\forall b.((a \neq b) \rightarrow \exists i.(\mathsf{rd}(a, i) \neq \mathsf{rd}(b, i))$
  - Add **lemmas on-demand** by instantiating the quantified variables with terms occurring in the input formula
    - Using smart "frugal" strategies: **check** candidate solution, instantiate only **(potentially) violated axioms**

Example

$$\neg(j = k), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), j) = \mathsf{rd}(a, j)), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$$

EUF solution (equivalence classes):

$\{a, \mathsf{wr}(a, i, x))\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), j)\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), k)\},$
$\{x, i, j\}, \{k\}, \{\mathsf{rd}(a, j)\}, \{\mathsf{rd}(a, k)\}$

Example

$$\neg(j = k), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), j) = \mathsf{rd}(a, j)), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$$

EUF solution (equivalence classes):

$\{a, \mathsf{wr}(a, i, x)\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), j)\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), k)\},$
$\{x, i, j\}, \{k\}, \{\mathsf{rd}(a, j)\}, \{\mathsf{rd}(a, k)\}$

Add violated lemma: $(i \neq k) \rightarrow (\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$

Example

$$\neg(j = k), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), j) = \mathsf{rd}(a, j)), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$$

EUF solution (equivalence classes):

$$\{a, \mathsf{wr}(a, i, x))\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), j)\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), k)\},$$
$$\{x, i, j\}, \{k\}, \{\mathsf{rd}(a, j)\}, \{\mathsf{rd}(a, k)\}$$

Add violated lemma: $(i \neq k) \rightarrow (\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$

EUF solution (equivalence classes):

$$\{a, \mathsf{wr}(a, i, x))\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), j)\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), k)\},$$
$$\{x, j\}, \{i, k\}, \{\mathsf{rd}(a, j)\}, \{\mathsf{rd}(a, k)\}$$

Example

$$\neg(j = k), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), j) = \mathsf{rd}(a, j)), \neg(\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$$

EUF solution (equivalence classes):

$\{a, \mathsf{wr}(a, i, x))\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), j)\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), k)\},$
$\{x, i, j\}, \{k\}, \{\mathsf{rd}(a, j)\}, \{\mathsf{rd}(a, k)\}$

Add violated lemma: $(i \neq k) \rightarrow (\mathsf{rd}(\mathsf{wr}(a, i, x), k) = \mathsf{rd}(a, k))$

EUF solution (equivalence classes):

$\{a, \mathsf{wr}(a, i, x))\}, \{\boxed{\mathsf{rd}(\mathsf{wr}(a, i, x), j)}\}, \{\mathsf{rd}(\mathsf{wr}(a, i, x), k)\},$
$\{x, \boxed{j}\}, \{\boxed{i}, k\}, \{\boxed{\mathsf{rd}(a, j)}\}, \{\mathsf{rd}(a, k)\}$

Add violated lemma: $(i \neq j) \rightarrow (\mathsf{rd}(\mathsf{wr}(a, i, x), j) = \mathsf{rd}(a, j))$

Example

$$\neg(j = k), \neg(\mathsf{rd}(\mathsf{wr}(a,i,x),j) = \mathsf{rd}(a,j)), \neg(\mathsf{rd}(\mathsf{wr}(a,i,x),k) = \mathsf{rd}(a,k))$$

EUF solution (equivalence classes):

$\{a, \mathsf{wr}(a,i,x))\}, \{\mathsf{rd}(\mathsf{wr}(a,i,x),j)\}, \{\mathsf{rd}(\mathsf{wr}(a,i,x),k)\},$
$\{x,i,j\}, \{k\}, \{\mathsf{rd}(a,j)\}, \{\mathsf{rd}(a,k)\}$

Add violated lemma: $(i \neq k) \rightarrow (\mathsf{rd}(\mathsf{wr}(a,i,x),k) = \mathsf{rd}(a,k))$

EUF solution (equivalence classes):

$\{a, \mathsf{wr}(a,i,x))\}, \{\mathsf{rd}(\mathsf{wr}(a,i,x),j)\}, \{\mathsf{rd}(\mathsf{wr}(a,i,x),k)\},$
$\{x,j\}, \{i,k\}, \{\mathsf{rd}(a,j)\}, \{\mathsf{rd}(a,k)\}$

Add violated lemma: $(i \neq j) \rightarrow (\mathsf{rd}(\mathsf{wr}(a,i,x),j) = \mathsf{rd}(a,j))$

EUF solver returns UNSAT

Bit-vectors (BV)

- Most solvers use an **eager approach** for BV, not DPLL(T)
  - **Heavy preprocessing** based on rewriting rules + bit-blasting
  - Example:
$$(x_{[1]} \neq 0_{[1]}) \wedge (y_{[31]} :: x_{[1]} \% 2_{[32]} = 0_{[32]}) \mapsto$$
$$(x_{[1]} = 1_{[1]}) \wedge (y_{[31]} :: x_{[1]} \% 2_{[32]} = 0_{[32]}) \mapsto$$
$$(y_{[31]} :: 1_{[1]} \% 2_{[32]} = 0_{[32]}) \mapsto \perp$$
- Alternative: **lazy bit-blasting**, compatible with DPLL(T)
  - Use a second SAT solver as T-solver for BV
    - bit-blast only BV-atoms, not the whole formula
  - Boolean skeleton of the formula handled by the main SAT solver
  - Easier integration with other theories and functionalities based on a DPLL(T) architecture
  - Can integrate additional specialized sub-solvers
- Eager still better performance-wise

Lazy bit-blasting: implementation

- **For each BV-atom** α occurring in the input formula, create a fresh Boolean "label" variable l_α , and bit-blast to SAT-BV the formula $(l_\alpha \leftrightarrow \alpha)$

- **Exploit SAT solving under assumptions**
  - When the main solver generates the BV-assignment $\alpha_1 \ldots \alpha_n$
  - Invoke SAT-BV with the assumptions $l_{\alpha_1} \ldots l_{\alpha_n}$
  - If unsat, generate an unsat core of the assumptions $l_{\alpha_i} \ldots l_{\alpha_j}$
    - From its negation, generate a BV-lemma $\neg\alpha_i \vee \ldots \vee \neg\alpha_j$ and send it to the main solver as a blocking clause, like in standard DPLL(T)

Outline

Introduction

The DPLL(T) architecture

Some relevant T-solvers

Combination of theories

Quantifiers in DPLL(T)

Combination of theories

- Very often in practice more than one theory is needed
  - Example (from intro):

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

- How to build solvers for SMT(T_1 … T_n) that are both **efficient** and **modular**?
  - Can we **reuse** T_i-solvers and **combine** them?
  - Under what conditions?
  - How do we go from DPLL(T) to DPLL(T_1 … 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 (other than $=$)
  - 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
    - Examples: EUF, LIA, LRA, A
    - Counterexample: BV
    - *(Extensions exist to deal with some non-stably-infinite theories)*

The Nelson-Oppen method

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
  - Example:

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

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

$$(l_3+4 \leq l_2) \wedge (l_1 = x+3y) \wedge (l_2 = g(w)) \wedge (f(l_1) = l_3)$$

The Nelson-Oppen method

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
  - Example:

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

- T_i-solvers cooperate by **exchanging (disjunctions of) entailed interface equalities**
  - I.e., equalities between shared variables

The Nelson-Oppen method

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
  - Example:

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

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

$$(\boxed{l_3} + 4 \leq \boxed{l_2}) \wedge (\boxed{l_1} = x + 3y) \wedge (\boxed{l_2} = g(w)) \wedge (f(\boxed{l_1}) = \boxed{l_3})$$

Interface variables

- T_i-solvers cooperate by **exchanging (disjunctions of) entailed interface equalities**
  - I.e., equalities between shared variables

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

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
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

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
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

$$(x_1 = x_3)$$

$$\models$$

$$(x_5 = x_6)$$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

$$\models$$

$$(x_1 = x_3) \vee (x_1 = x_4)$$

$$(x_1 = x_3)$$

$$\models$$

$$(x_5 = x_6)$$

$$(x_5 = x_6)$$

$$\models$$

$$(x_2 = x_3) \vee (x_2 = x_4)$$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

$$\models$$

$$(x_1 = x_3) \vee (x_1 = x_4) \Rightarrow (x_1 = x_3)$$

$$(x_1 = x_3) \models (x_5 = x_6)$$

$$(x_5 = x_6) \Leftarrow (x_5 = x_6)$$

$$\models$$

$$(x_2 = x_3) \vee (x_2 = x_4) \Rightarrow (x_2 = x_3)$$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

$$\models$$

$$(x_1 = x_3) \vee (x_1 = x_4) \Rightarrow (x_1 = x_3)$$

$$(x_1 = x_3) \models (x_5 = x_6)$$

$$(x_5 = x_6) \Leftarrow (x_5 = x_6)$$

$$\models$$

$$(x_2 = x_3) \vee (x_2 = x_4) \Rightarrow (x_2 = x_3) \quad (x_2 = x_4)$$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\models$

$(x_1 = x_3) \vee (x_1 = x_4)$

$(x_5 = x_6)$

$\models$

$(x_2 = x_3) \vee (x_2 = x_4)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

$(x_1 = x_3)$ $(x_1 = x_4)$

$\models$

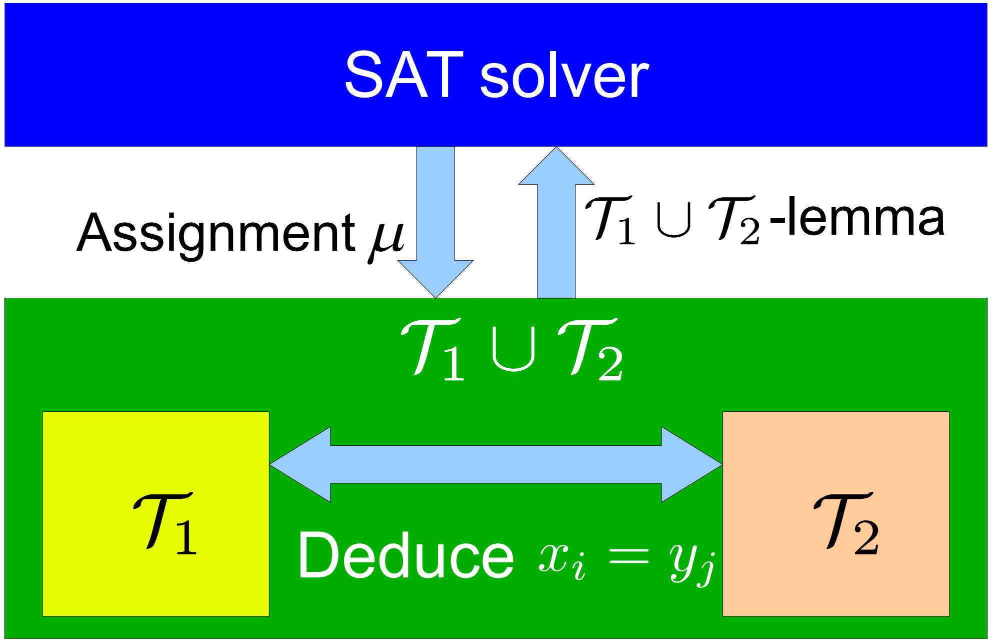
$(x_5 = x_6)$

No more deductions possible

$(x_2 = x_3)$ $(x_2 = x_4)$

DPLL(T) for combined theories

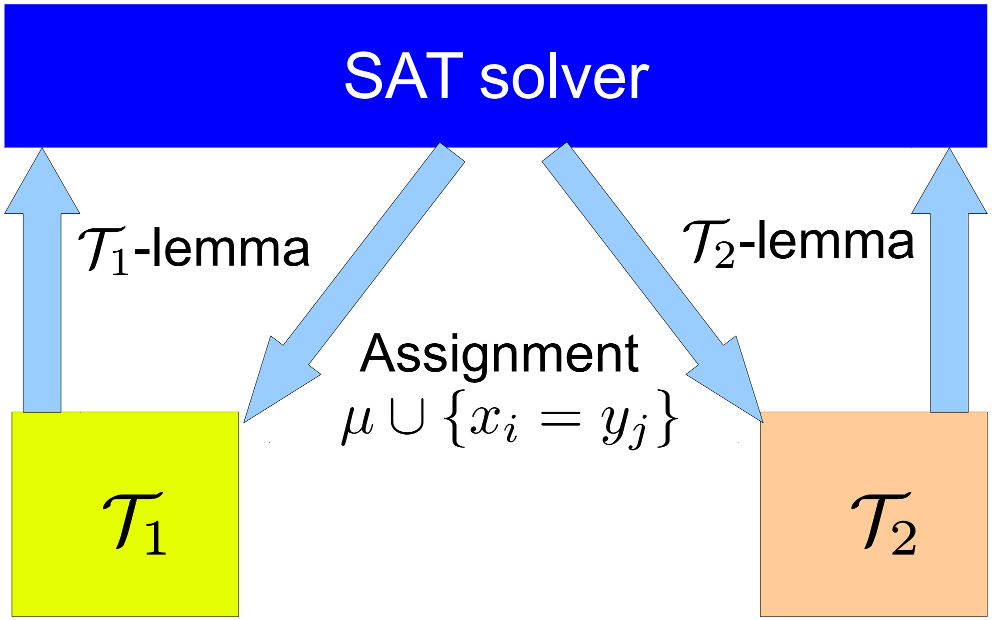
- Traditional approach: a **single** combined Nelson-Oppen *T*-solver
  - T_i-solvers exchange (disjunctions of) implied interface equalities **internally**
    - Interface equalities invisible to the SAT solver



  - Drawbacks: T_i-solvers need to:
    - be deduction complete for interface equalities
    - be able to perform case splits internally

Delayed Theory Combination

- Alternative to traditional approach
  - Each T_i-solver interacts directly and only with the SAT solver
  - SAT solver takes care of (all or part of) the combination
    - Augment the Boolean search space with the possible interface equalities $(x_i = y_j)$
- Advantages:
  - No need of complete deduction of interface equalities
  - Case analysis via splitting on-demand



Delayed theory combination in practice

- **Model-based** heuristic:
  - Initially, no interface equalities generated
  - When a solution is found, check against all the possible interface equalities
    - If T_1 and T_2 agree on the implied equalities, return **SAT**
    - Otherwise, branch on equalities implied by T_1-model but not by T_2-model
  - Optimistic approach, similar to axiom instantiation

- Still allow T_i-solvers to **exchange equalities internally**
  - But no requirement of completeness
  - Avoids “polluting” the SAT space with equality deductions leading to conflicts

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

LIA-model:
$$\begin{array}{ll} x_1 \mapsto 1 & x_2 \mapsto 2 \\ x_3 \mapsto 0 & x_4 \mapsto 1 \\ x_5 \mapsto 0 & x_6 \mapsto 1 \end{array}$$

$$\models$$

$$(x_1 = x_4)$$

EUF-model:
$$\begin{array}{l} \{x_1\}\ \{x_2\}\ \{x_3\}\ \{x_4\} \\ \{x_5, f(x_3)\}\ \{x_6, f(x_1)\} \\ \{f(x_2)\}\ \{f(x_4)\} \end{array}$$

$$\not\models$$

$$(x_1 = x_4)$$

Branch on $(x_1 = x_4)$

$$(x_1 = x_4)$$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

$\neg(f(x_1) = f(x_2)),$ EUF
$\neg(f(x_2) = f(x_4)),$
$(f(x_3) = x_5), (f(x_1) = x_6)$

LIA-model:
$$\begin{array}{ll} x_1 \mapsto 1 & x_2 \mapsto 2 \\ x_3 \mapsto 0 & x_4 \mapsto 1 \\ x_5 \mapsto 0 & x_6 \mapsto 1 \end{array}$$

$$\models$$

$$(x_3 = x_5)$$
$$(x_4 = x_6)$$

EUF-model:
$$\{x_1, x_4\}\ \{x_2\}\ \{x_3\}$$
$$\{x_5, f(x_3)\}$$
$$\{x_6, f(x_1), f(x_4)\}\{f(x_2)\}$$

$$\not\models$$

$$(x_3 = x_5)$$
$$(x_4 = x_6)$$

...

$$(x_1 = x_4)$$
$$(x_3 = x_5)$$
$$(x_4 = x_6)$$

Example

LIA $(x_1 \geq 0), (x_1 \leq 1), (x_2 \geq x_6)$
$(x_2 \leq x_6 + 1), (x_5 = x_4 - 1)$
$(x_3 = 0), (x_4 = 1)$

LIA-model: $x_1 \mapsto 1 \quad x_2 \mapsto 2$
$x_3 \mapsto 0 \quad x_4 \mapsto 1$
$x_5 \mapsto 0 \quad x_6 \mapsto 1$

$\neg(f(x_1) = f(x_2))$, EUF
$\neg(f(x_2) = f(x_4))$,
$(f(x_3) = x_5), (f(x_1) = x_6)$

EUF-model: $\{x_1, x_4, x_6, f(x_1), f(x_4)\}$
$\{x_3, x_5, f(x_3)\}$
$\{f(x_2)\}\{x_2\}$

...

$(x_1 = x_4)$

$(x_3 = x_5)$

$(x_4 = x_6)$

Outline

Introduction

The DPLL(T) architecture

Some relevant T-solvers

Combination of theories

Quantifiers in DPLL(T)

Motivations

- SMT solvers mostly deal with **quantifier-free** problems
  - Often good compromise between expressiveness and efficiency
    - *A key factor for the success of SMT*

- Yet, in practice it is useful to incorporate *some* support for **quantifiers**
  - Examples:
    - Support user-provided axioms/assertions

      $$\forall i, j. (i \leq j) \rightarrow (\mathsf{rd}(a, i) \leq \mathsf{rd}(a, j)) \qquad \text{“}a \text{ is sorted”}$$

    - Axiomatisation of extra theories w/o built-in support

      $$\forall x. p(x, x) \qquad \forall x, y, z. p(x, y) \wedge p(y, z) \rightarrow p(x, z)$$
      $$\forall x, y. p(x, y) \wedge p(y, x) \rightarrow x = y$$

Quantifiers in DPLL(T)

- Assumption: formulas of the form $\psi \wedge \bigwedge_j \forall \vec{x}.D_j(\vec{x})$
  ψ quantifier-free
  - Can always remove existentials by **Skolemization**

$$\forall x.\exists y.\varphi(x,y) \mapsto \forall x.\varphi(f_y(x)), \quad f_y \text{ fresh}$$

- Main idea: handle quantifiers via **axiom instantiation**
  - Pick a quantified clause $\forall \vec{x}.D(\vec{x})$, heuristically instantiate its variables with quantifier-free terms $\vec{t_1} \ldots \vec{t_k}$, and add the generated clauses $\{D(\vec{t_1}) \ldots D(\vec{t_k})\}$ to the SAT solver
  - terminate when **unsat** is detected

Quantifiers in DPLL(T)

- Assumption: formulas of the form $\psi \wedge \bigwedge_j \forall \vec{x}.D_j(\vec{x})$
  ψ quantifier-free
  - Can always remove existentials by **Skolemization**

$$\forall x.\exists y.\varphi(x,y) \mapsto \forall x.\varphi(f_y(x)), \quad f_y \text{ fresh}$$

- Main idea: handle quantifiers via **axiom instantiation**
  - Pick a quantified clause $\forall \vec{x}.D(\vec{x})$, heuristically instantiate its variables with quantifier-free terms $\vec{t_1} \ldots \vec{t_k}$, and add the generated clauses $\{D(\vec{t_1}) \ldots D(\vec{t_k})\}$ to the SAT solver
  - terminate when **unsat** is detected
- Problems:
  - how to choose the **relevant instances** to add?
  - how to detect **satisfiable formulas**?

E-matching

- Discover relevant instances using the **EUF congruence closure graph** (E-graph)
- Given an axiom $\forall \vec{x}.D(\vec{x})$, an E-graph E, a *trigger* $p(\vec{x})$ and a *substitution* θ from vars to ground terms:
  - $D(\vec{x})\theta$ is relevant $\Leftrightarrow$ exists $t \in E$ such that $E \models (t = p(\vec{x})\theta)$
- E-matching: for each axiom $\forall \vec{x}.D_i(\vec{x})$ with trigger $p_i(\vec{x})$
  - generate all substitutions θ_i^j s.t. $E \models (t = p_i(\vec{x})\theta_i^j), t \in E$
  - generate the axiom instances $D_i(\vec{x})\theta_i^j$
  - reason modulo equivalence classes in E
    - discard substitutions that are equivalent modulo E

E-matching

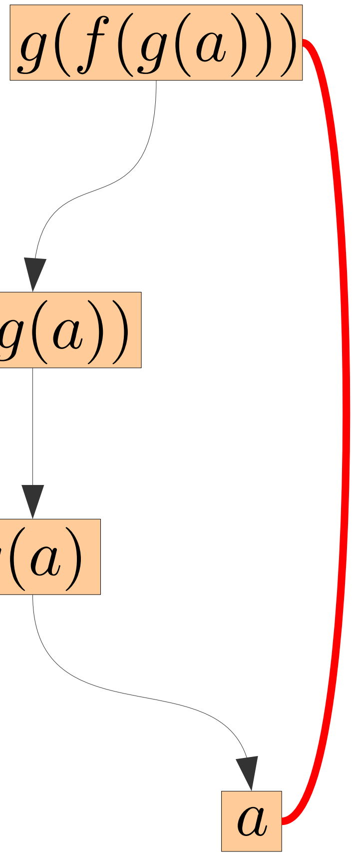
- Discover relevant instances using the **EUF congruence closure graph** (E-graph)
- Given an axiom $\forall \vec{x}. D(\vec{x})$, an E-graph E, a *trigger* $p(\vec{x})$ and a *substitution* θ from vars to ground terms:
  - $D(\vec{x})\theta$ is relevant $\Leftrightarrow$ exists $t \in E$ such that $E \models (t = p(\vec{x})\theta)$
- E-matching: for each axiom $\forall \vec{x}. D_i(\vec{x})$ with trigger $\boxed{p_i(\vec{x})}$
  - generate all substitutions θ_i^j s.t. $E \models (t = p_i(\vec{x})\theta_i^j), t \in E$
  - generate the axiom instances $D_i(\vec{x})\theta_i^j$
  - reason modulo equivalence classes in E
    - discard substitutions that are equivalent modulo E

user-provided or syntactically determined from $D_i(\vec{x})$

Example

$$\varphi \stackrel{\text{def}}{=} \neg(g(f(g(a))) = a) \wedge \underbrace{\forall x.(f(x) = x)}_{\text{trigger } f(x)} \wedge \underbrace{\forall x.(g(g(x)) = x)}_{\text{trigger } g(g(x))}$$

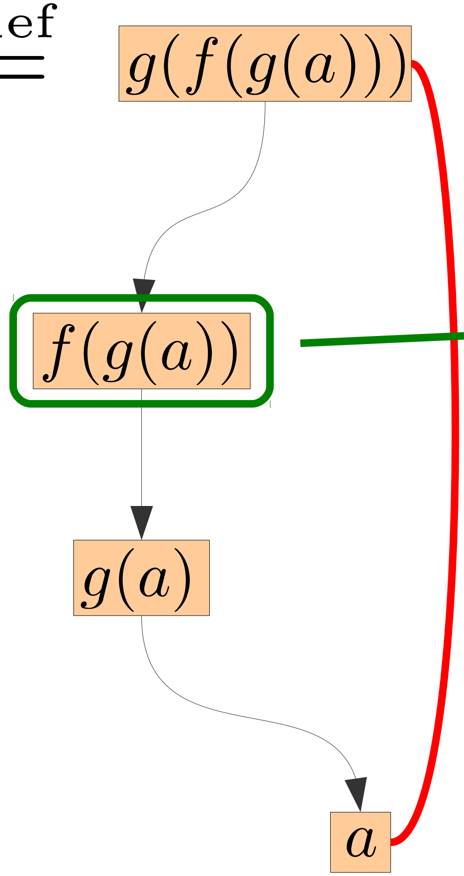
$E \stackrel{\text{def}}{=}$



Example

$$\varphi \stackrel{\text{def}}{=} \neg(g(f(g(a))) = a) \wedge \boxed{\underbrace{\forall x.(f(x) = x)}_{\text{trigger } f(x)}} \wedge \underbrace{\forall x.(g(g(x)) = x)}_{\text{trigger } g(g(x))}$$

$E \stackrel{\text{def}}{=}$



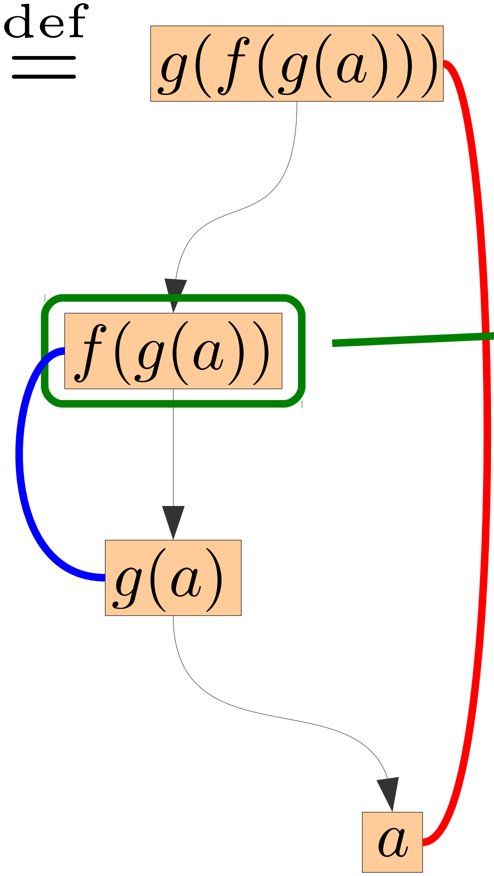
Match with $\theta \stackrel{\text{def}}{=} \{x \mapsto g(a)\}$

Add $f(g(a)) = g(a)$

Example

$$\varphi \overset{\text{def}}{=} \neg(g(f(g(a))) = a) \wedge \boxed{\underbrace{\forall x.(f(x) = x)}_{\text{trigger } f(x)}} \wedge \underbrace{\forall x.(g(g(x)) = x)}_{\text{trigger } g(g(x))}$$

$E \overset{\text{def}}{=}$



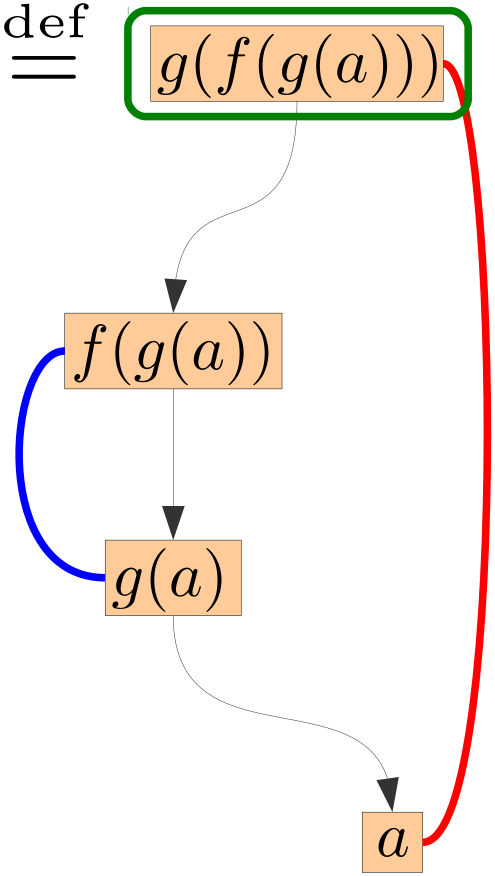
Match with $\theta \overset{\text{def}}{=} \{x \mapsto g(a)\}$

Add $f(g(a)) = g(a)$

Example

$$\varphi \stackrel{\text{def}}{=} \neg(g(f(g(a))) = a) \wedge \underbrace{\forall x.(f(x) = x)}_{\text{trigger } f(x)} \wedge \boxed{\underbrace{\forall x.(g(g(x)) = x)}_{\text{trigger } g(g(x))}}$$

$E \stackrel{\text{def}}{=}$



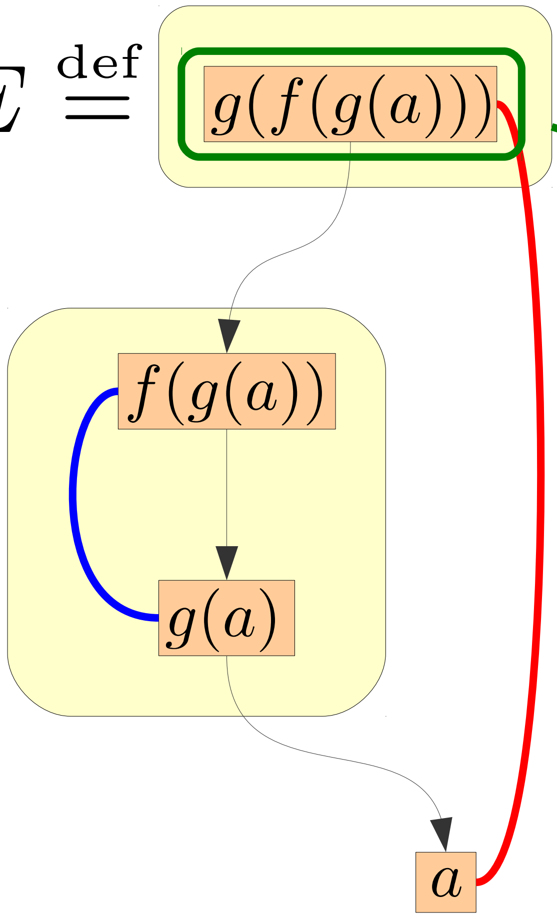
Match with $\theta \stackrel{\text{def}}{=} \{x \mapsto a\}$

Add $g(g(a)) = a$

Example

$$\varphi \stackrel{\text{def}}{=} \neg(g(f(g(a))) = a) \wedge \underbrace{\forall x.(f(x) = x)}_{\text{trigger } f(x)} \wedge \boxed{\underbrace{\forall x.(g(g(x)) = x)}_{\text{trigger } g(g(x))}}$$

$E \stackrel{\text{def}}{=}$



Match with $\theta \stackrel{\text{def}}{=} \{x \mapsto a\}$

Add $g(g(a)) = a$

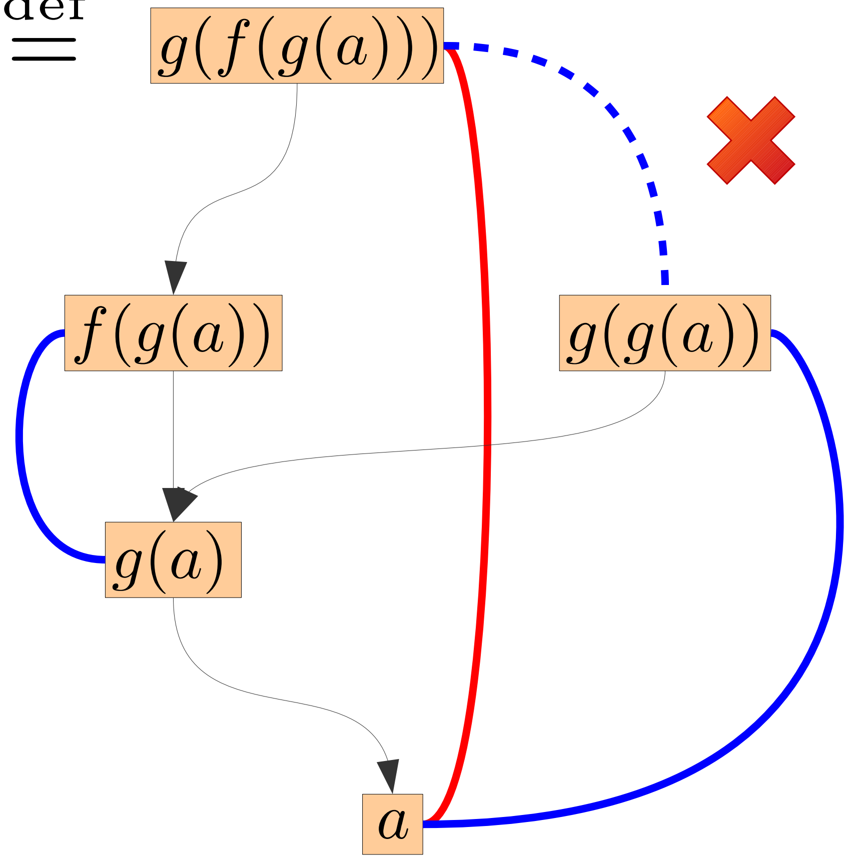
Because

$$E \models g(f(g(a))) = g(g(a))$$

Example

$$\varphi \stackrel{\text{def}}{=} \neg(g(f(g(a))) = a) \wedge \underbrace{\forall x.(f(x) = x)}_{\text{trigger } f(x)} \wedge \underbrace{\forall x.(g(g(x)) = x)}_{\text{trigger } g(g(x))}$$

$E \stackrel{\text{def}}{=}$



E-matching: discussion

- Advantages:
  - Integrates smoothly with DPLL(T)
  - Fast and efficient at finding "shallow" proofs in big formulas
    - A typical scenario in SMT-based verification

- However, various drawbacks:
  - Can never say **sat,** but is **not** even **refutationally complete**
  - Needs **ground seeds**
    - Example: $(\forall x. P(x)) \wedge (\forall x. \neg P(x))$
  - Sensitive to **bad triggers**
    - Example: $(\forall x. f(g(x)) = x)$ [with trigger $f(g(x))$]
      $(g(a) = c) \wedge (g(b) = c) \wedge \neg(a = b)$

Model-based Instantiation

- Idea: $\varphi \stackrel{\text{def}}{=} \psi \wedge \bigwedge_i (\forall \vec{x}. D_i(\vec{x}))$
  - build a **model** M for ψ
  - **check** if M satisfies the quantified axioms $\bigwedge_i (\forall \vec{x}. D_i \vec{x})$
    - If yes, return **sat**
otherwise, generate an **instance** that **blocks** the bad model

Model-based Instantiation

- Idea: $\varphi \stackrel{\text{def}}{=} \psi \wedge \bigwedge_i (\forall \vec{x}. D_i(\vec{x}))$
  - build a **model** M for ψ
  - **check** if M satisfies the quantified axioms $\bigwedge_i (\forall \vec{x}. D_i \vec{x})$
    - If yes, return **sat**
      otherwise, generate an **instance** that **blocks** the bad model

- How:
  - Use a **symbolic representation** for M, using lambda-terms
    - Example: $(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)$

      $$M \stackrel{\text{def}}{=} \{a \mapsto 1, b \mapsto 0, f \mapsto \boxed{\lambda x.\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0))}\}$$

Model-based Instantiation

- Idea: $\varphi \stackrel{\text{def}}{=} \psi \wedge \bigwedge_i (\forall \vec{x}. D_i(\vec{x}))$
  - build a **model** M for ψ
  - **check** if M satisfies the quantified axioms $\bigwedge_i (\forall \vec{x}. D_i \vec{x})$
    - If yes, return **sat**
      otherwise, generate an **instance** that **blocks** the bad model

- How:
  - Use a **symbolic representation** for M, using lambda-terms
    - Example: $(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)$
      $M \stackrel{\text{def}}{=} \{a \mapsto 1, b \mapsto 0, f \mapsto \boxed{\lambda x.\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0))}\}$
  - **Check unsatisfiability** of $\neg \forall \vec{x}. D_i(\vec{x})[M(c)/c]$ with SMT
    - Example: $\neg \forall x.(f(x) < x + a)[M(c)/c] \mapsto$
      $\exists x. \neg(\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0)) < x + 1)$

Example

$$\varphi \stackrel{\text{def}}{=} \overbrace{(f(a) = 1) \land (a > b) \land (f(b) > f(a) + 1)}^{\psi} \land$$

$$\forall x.(f(x) \geq b) \land (f(x) < a + b)$$

Example

$$\varphi \stackrel{\text{def}}{=} \overbrace{(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)}^{\psi} \wedge$$
$$\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$$

- Check ψ ✔

$$M \stackrel{\text{def}}{=} \{a \mapsto 1, b \mapsto 0, f \mapsto \lambda x.\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0))\}$$

Example

$$\varphi \stackrel{\text{def}}{=} \overbrace{(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)}^{\psi} \wedge$$
$$\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$$

- Check ψ

$$M \stackrel{\text{def}}{=} \{a \mapsto 1, b \mapsto 0, f \mapsto \lambda x.\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0))\}$$

- Check $M \models (\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$, i.e.

$$\neg((\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0)) \geq 0) \wedge$$
$$(\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0)) < 1 + 0)) \models \bot$$

Example

$$\varphi \stackrel{\text{def}}{=} \overbrace{(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)}^{\psi} \wedge$$
$$\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$$

- Check ψ

$$M \stackrel{\text{def}}{=} \{a \mapsto 1, b \mapsto 0, f \mapsto \lambda x.\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0))\}$$

- Check $M \models (\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$, i.e.

$$\neg((\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0)) \geq 0) \wedge$$
$$(\mathrm{ite}(x = 0, 3, \mathrm{ite}(x = 1, 1, 0)) < 1 + 0)) \models \bot$$

  - Counterexample: $\{x \mapsto 0\}$

Example

$$\varphi \stackrel{\text{def}}{=} \overbrace{(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)}^{\psi} \wedge$$
$$\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$$

- Check ψ

$$M \stackrel{\text{def}}{=} \{a \mapsto 1, \boxed{b \mapsto 0,} f \mapsto \lambda x.\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0))\}$$

- Check $M \models (\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$, i.e.

$$\neg((\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0)) \geq 0) \wedge$$
$$(\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0)) < 1 + 0)) \models \bot$$

  - Counterexample: $\{\boxed{x \mapsto 0}\}$
  - Generated instance: $(f(\boxed{b}) \geq b) \wedge (f(\boxed{b}) < a + b)$

Example

$$\varphi \stackrel{\text{def}}{=} \overbrace{(f(a) = 1) \wedge (a > b) \wedge (f(b) > f(a) + 1)}^{\psi} \wedge$$
$$\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$$

- Check $\psi \wedge (f(b) \geq b) \wedge (f(b) < a + b)$

  $$M \stackrel{\text{def}}{=} \{a \mapsto 3, b \mapsto 1, f \mapsto \lambda x.\text{ite}(x = 1, 3, 1)\}$$

- Check $M \models (\forall x.(f(x) \geq b) \wedge (f(x) < a + b)$, i.e.

  $$\neg((\text{ite}(x = 1, 3, 1) \geq 1) \wedge (\text{ite}(x = 1, 3, 1) < 3 + 1) \models \perp$$

SAT

Complete Instantiation

- No hope for a complete procedure in general
  - FOL without theories is only **semi-decidable**...
  - ...and in fact **undecidable** with (some) theories (e.g. LIA)

- However, many decidable fragments exist
  - With suitable instantiation strategies, model-based techniques can be applied effectively

Finite Model Finding

- Idea: search for models interpreting quantified variables over **finite domains**
  - with finite domain, complete instantiation is possible
  - if the domains are small (and the instantiation smart), might also be **practical**
  - Applicable when quantified vars range over uninterpreted sorts

Finite Model Finding

- Idea: search for models interpreting quantified variables over **finite domains**
  - with finite domain, complete instantiation is possible
  - if the domains are small (and the instantiation smart), might also be **practical**
  - Applicable when quantified vars range over uninterpreted sorts
- How:
  - Add a *T*-solver for **cardinality constraints** on uninterpreted sorts
    - Use splitting on-demand with card. lemmas $(|S| \leq k) \vee (|S| > k)$
    - Tightly integrated with EUF solver
  - When "finite" model is found, instantiate exhaustively the axioms
    - But avoid redundant instances
  - Return **sat** if a model is found

Example

$$\varphi \stackrel{\text{def}}{=} \underbrace{\neg(f(a) = g(b))}_{\psi} \wedge \forall x. \neg(f(x) = f(g(x))$$

Example

$$\varphi \stackrel{\text{def}}{=} \underbrace{\neg(f(a) = g(b))}_{\psi} \wedge \forall x. \neg(f(x) = f(g(x)))$$

- Find model for ψ: $M \stackrel{\text{def}}{=} \begin{array}{l} \{a, f(a)\} \\ \{b, g(b)\} \end{array}$

Example

$$\varphi \stackrel{\text{def}}{=} \underbrace{\neg(f(a) = g(b))}_{\psi} \wedge \forall x. \neg(f(x) = f(g(x)))$$

- Find model for ψ: $M \stackrel{\text{def}}{=} \begin{matrix} \{a, f(a)\} \\ \{b, g(b)\} \end{matrix}$
- Try cardinality $(|S| \leq 1)$ ✗

Example

$$\varphi \stackrel{\text{def}}{=} \underbrace{\neg(f(a) = g(b))}_{\psi} \wedge \forall x. \neg(f(x) = f(g(x)))$$

- Find model for ψ : $M \stackrel{\text{def}}{=} \begin{array}{l} \{a, f(a)\} \\ \{b, g(b)\} \end{array}$
- Try cardinality $(|S| \leq 1)$
- Try cardinality $(|S| \leq 2)$

Example

$$\varphi \stackrel{\text{def}}{=} \underbrace{\neg(f(a) = g(b))}_{\psi} \wedge \forall x. \neg(f(x) = f(g(x)))$$

- Find model for ψ: $M \stackrel{\text{def}}{=} \begin{matrix} \{a, f(a)\} \\ \{b, g(b)\} \end{matrix}$

- Try cardinality $(|S| \leq 1)$ ✗
- Try cardinality $(|S| \leq 2)$ ✓
- Generate instances using representatives of equiv. classes

$$I_1 \stackrel{\text{def}}{=} \neg(f(a) = f(g(a))) \qquad I_2 \stackrel{\text{def}}{=} \neg(f(b) = f(g(b)))$$

- Check satisfiability of $\psi \wedge (|S| \leq 2) \wedge I_1 \wedge I_2$

Example

$$\varphi \stackrel{\text{def}}{=} \underbrace{\neg(f(a) = g(b))}_{\psi} \wedge \forall x. \neg(f(x) = f(g(x)))$$

- Find model for ψ: $M \stackrel{\text{def}}{=} \begin{array}{l} \{a, f(a)\} \\ \{b, g(b)\} \end{array}$

- Try cardinality $(|S| \leq 1)$ ✗
- Try cardinality $(|S| \leq 2)$ ✓
- Generate instances using representatives of equiv. classes

$$I_1 \stackrel{\text{def}}{=} \neg(f(a) = f(g(a))) \qquad I_2 \stackrel{\text{def}}{=} \neg(f(b) = f(g(b)))$$

- Check satisfiability of $\psi \wedge (|S| \leq 2) \wedge I_1 \wedge I_2$

$$M \stackrel{\text{def}}{=} \begin{array}{l} \{a, f(a), f(g(b))\} \\ \{b, g(b), f(g(a)), f(b)\} \end{array}$$ ✓

SAT

Selected bibliography

DISCLAIMER: this is not meant to be complete, just a starting point. Apologies to missing authors/works

- SMT in general and DPLL(T)

  - Nieuwenhuis, Oliveras, Tinelli. **Solving SAT and SAT Modulo Theories: From an abstract Davis--Putnam--Logemann--Loveland procedure to DPLL(T)**. J. ACM 2006
  - Sebastiani. **Lazy Satisfiability Modulo Theories.** JSAT 2007
  - Barrett, Sebastiani, Seshia, Tinelli. **Satisfiability Modulo Theories.** SAT handbook 2009

- Theory solvers

  - Detlefs, Nelson, Saxe. **Simplify: a theorem prover for program checking**. J. ACM 2005
  - Nieuwenhuis, Oliveras. **Fast congruence closure and extensions**. Inf. Comput. 2007

Selected bibliography

- Theory solvers (cont'd)

  - Dutertre, de Moura. **A Fast Linear-Arithmetic Solver for DPLL(T)**. CAV 2006
  - de Moura, Bjørner. **Model-based Theory Combination**. Electr. Notes Theor. Comput. Sci. 2008
  - Brummayer, Biere. **Lemmas on Demand for the Extensional Theory of Arrays**. JSAT 2009
  - de Moura, Bjørner. **Generalized, efficient array decision procedures**. FMCAD 2009
  - Jovanovic, de Moura. **Cutting to the Chase - Solving Linear Integer Arithmetic**. J. Autom. Reasoning 2013
  - Hadarean, Bansal, Jovanovic, Barrett, Tinelli. **A Tale of Two Solvers: Eager and Lazy Approaches to Bit-Vectors**. CAV 2014

Selected bibliography

- Quantifiers
  - de Moura, Bjørner. **Efficient E-Matching for SMT Solvers**. CADE 2007
  - Ge, de Moura. **Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories**. CAV 2009
  - Reynolds, Tinelli, Goel, Krstic. **Finite Model Finding in SMT**. CAV 2013

Thank You