EMBEDDED SYSTEMS
FONDAZIONE BRUNO KESSLER

CAV Verification Mentoring Workshop 2017

SMT Solving

Alberto Griggio
Fondazione Bruno Kessler – Trento, Italy

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:

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

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:

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

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

Why SMT?

- logic as language for various applications in formal methods (and more)
  - Modeling
  - Verification
  - Planning / scheduling
  - Synthesis
  - …
- Need **efficient, automated** reasoning techniques
  - SMT is a “sweet spot” between expressiveness and efficiency
  - SMT solvers as backend “workhorse” engines of many (verification) techniques and tools

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)**
  - > 3000 citations, most influential tool paper at TACAS

SAT with CDCL (aka DPLL)

```
CDCL(F)
  A = [], dl = 0
  while (true)
    if (unit_propagation(F, A))
      if (!all_assigned(F, A))
        lit = pick_lit(F, A)
        dl++
        A = A + (lit, -)
      else return SAT
    else
      lvl, cls = analyze(F, A)
      if (lvl < 0) return UNSAT
      else
        backtrack(F, A, lvl)
        learn(cls)
        dl = lvl
```

Trail of assignments *(lit, reason)*

Model Search

Proof Search

SAT with CDCL (aka DPLL)

```
CDCL(F)
  A = [], dl = 0
  while (true)
    if (unit_propagation(F, A))
      if (!all_assigned(F, A))
        lit = pick_lit(F, A)
        dl++
        A = A + (lit, -)
      else return SAT
    else
      lvl, cls = analyze(F, A)
      if (lvl < 0) return UNSAT

              ack(F, A, lvl)
              cls)
              vl
```

Trail of assignments *(lit, reason)*

Model Search

Proof Search

Propositional resolution

$$\frac{C_1 \vee p \qquad C_2 \vee \neg p}{C_1 \vee C_2}$$

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

Efficient SMT

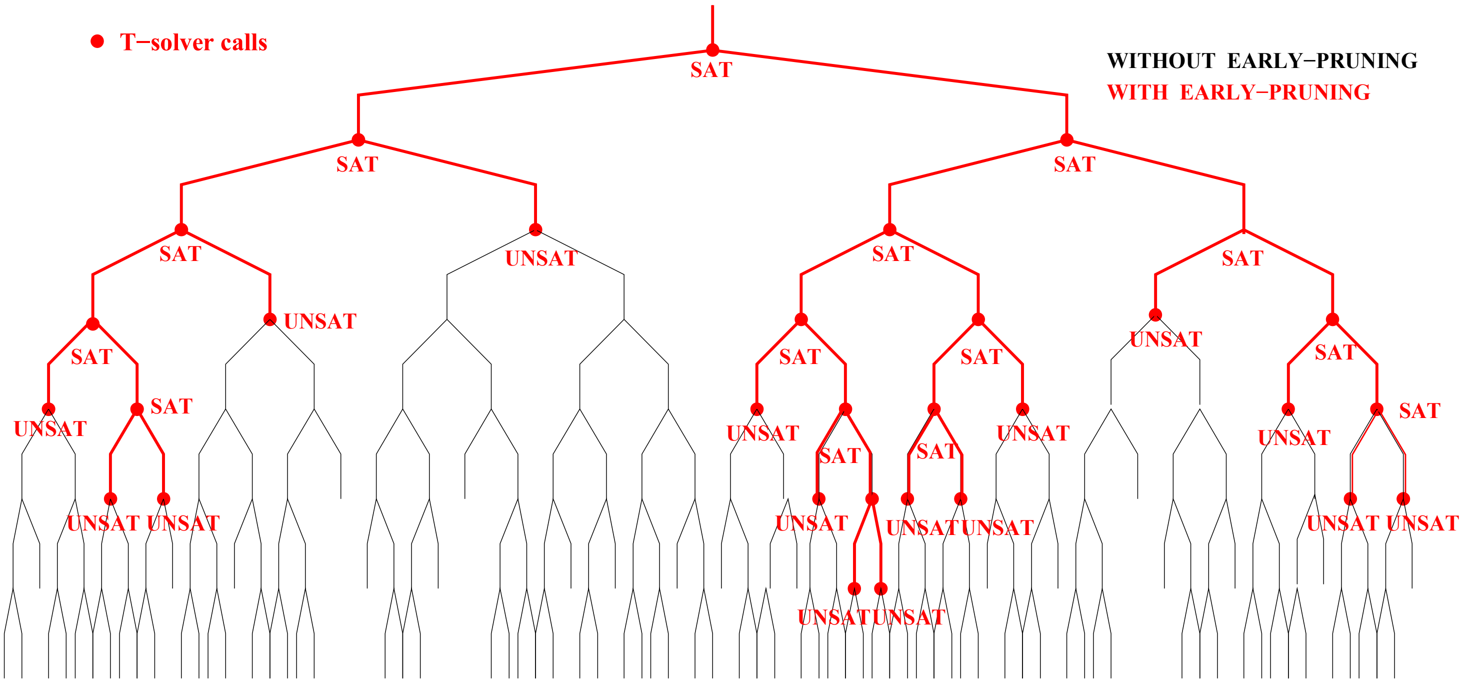
- **DPLL(T): Online** approach to lazy SMT
- **Tight integration** between a CDCL-like SAT solver ("DPLL") and the decision procedure for *T* ("T-solver"), based on:
  - Early pruning
  - *T*-driven backjumping and learning
  - *T*-solver incrementality
  - *T*-propagation
- **Separation of concerns**
  - efficient boolean reasoning via CDCL
  - only conjunctions of constraints in T-solvers
- **Modular architecture**
  - reasonably easy to change SAT solver or add other theories

DPLL(T)

```
DPLL-T(F)
  A = [], dl = 0
  while (true)
    conflict = FALSE
    if (unit_propagation(F, A) &&
        theory_propagation(F, A))
       if (!all_assigned(F, A))
         lit = pick_lit(F, A), dl++
         A = A + (lit, -)
       else if (theory_check(F, A))
         return SAT
       else conflict = TRUE
    else conflict = TRUE
    if (conflict)
      lvl, cls = theory_analyze(F, A)
      if (lvl < 0) return UNSAT
      else
         backtrack(F, A, lvl)
         learn(cls)
         dl = lvl
```

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



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

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)

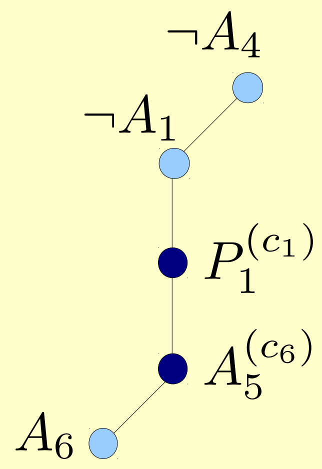
T-propagation

- 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

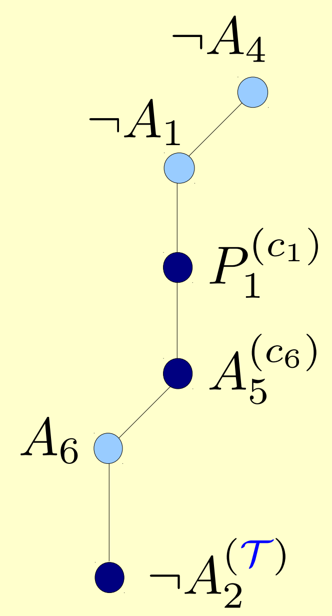
$$
\begin{array}{lll}
\varphi & \stackrel{\text{def}}{=} & \\
c_1: & (2x_2 - x_3 > 2) \vee P_1 & \\
c_2: & \neg P_2 \vee (x_1 - x_5 \leq 1) & \\
c_3: & \neg(3x_1 - 2x_2 \leq 3) \vee \neg P_2 & \\
c_4: & \neg(3x_1 - x_3 \leq 6) \vee \neg P_1 & \\
c_5: & P_1 \vee (3x_1 - 2x_2 \leq 3) & \\
c_6: & (x_2 -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

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



Example

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



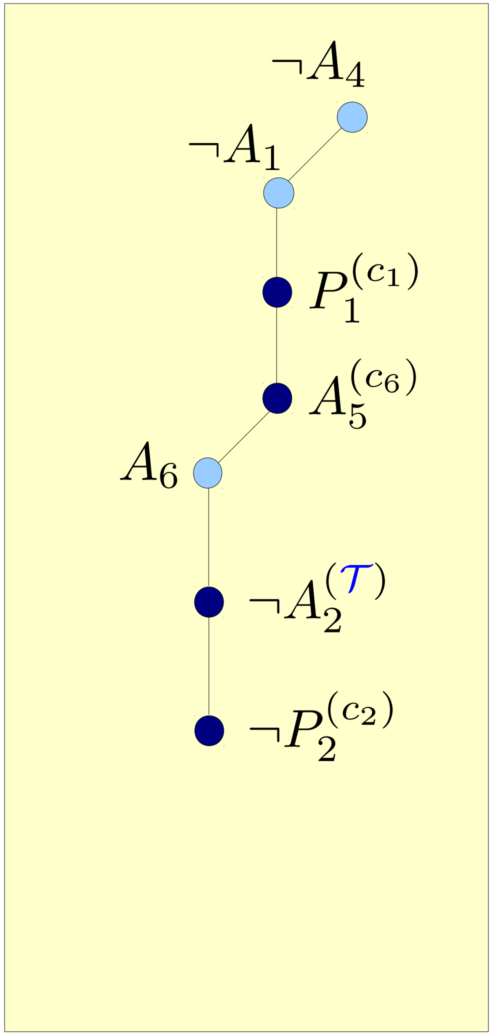
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6]$$

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

Example

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

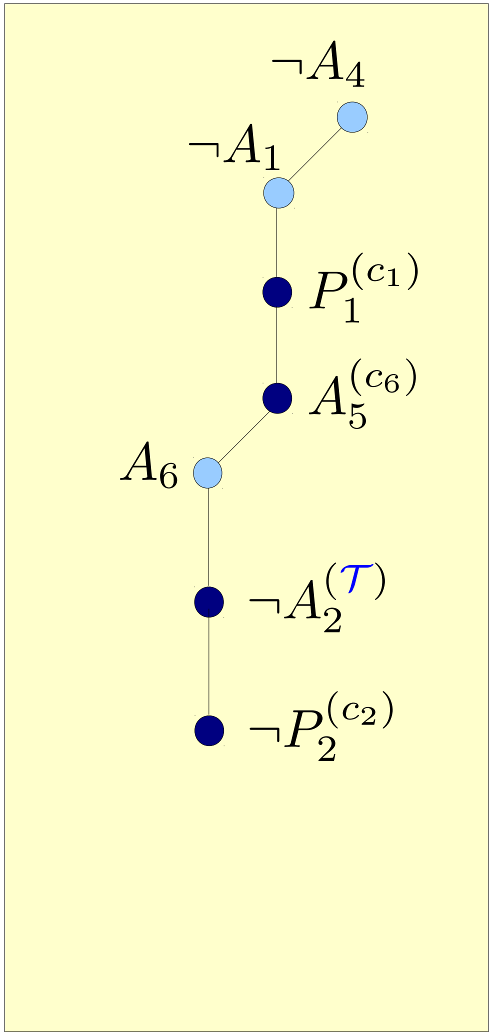
$$M = [\neg A_4, \neg A_1, P_1, A_5, A_6, \neg A_2, \neg P_2]$$



Example

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

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

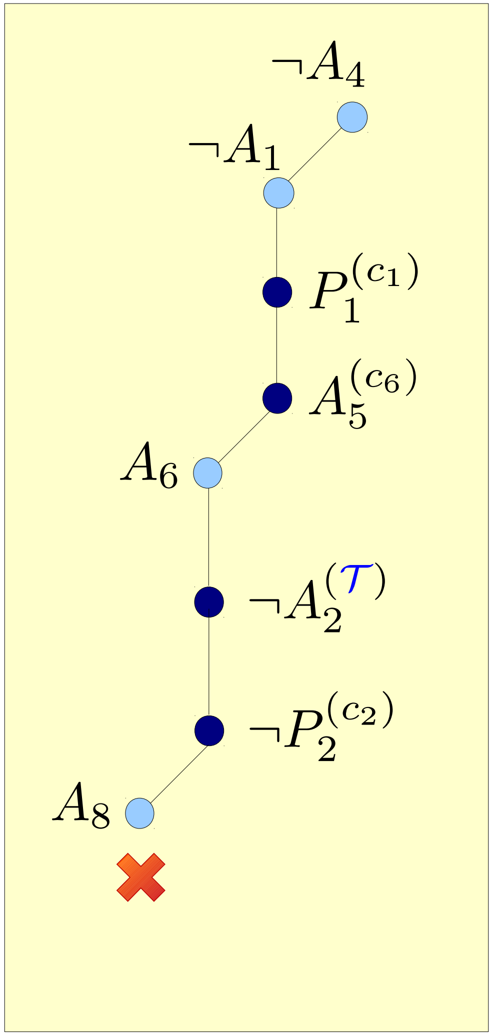


Example

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

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

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

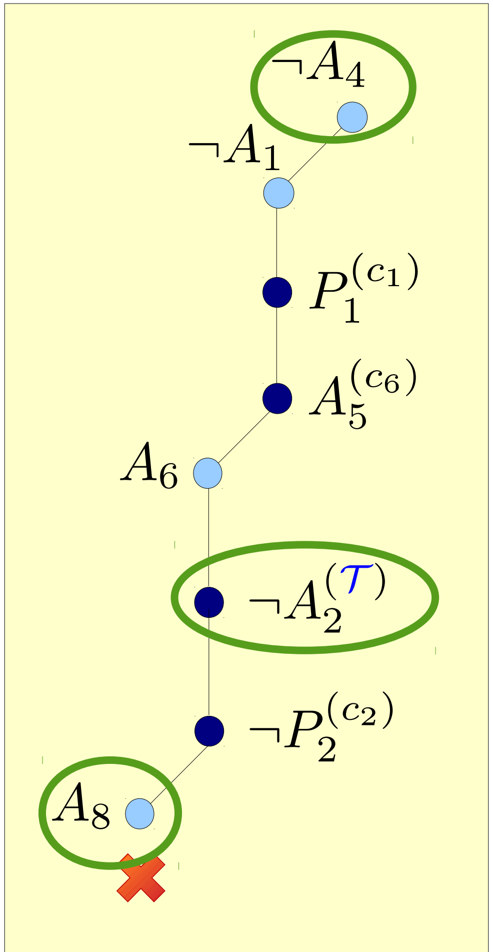


Example

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

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

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



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

Modern SMT functionalities

Many built-in theories and combinations

- Equality, arithmetic (linear, some non-linear), bit-vectors, arrays, floats, datatypes, …
- Quantifiers

Much more than just satisfiability checking

- Model generation (less obvious than it seems)
- Incremental interface (push/pop, assumptions)
- Model enumeration
- Quantifier elimination
- Proofs, unsat cores, interpolants

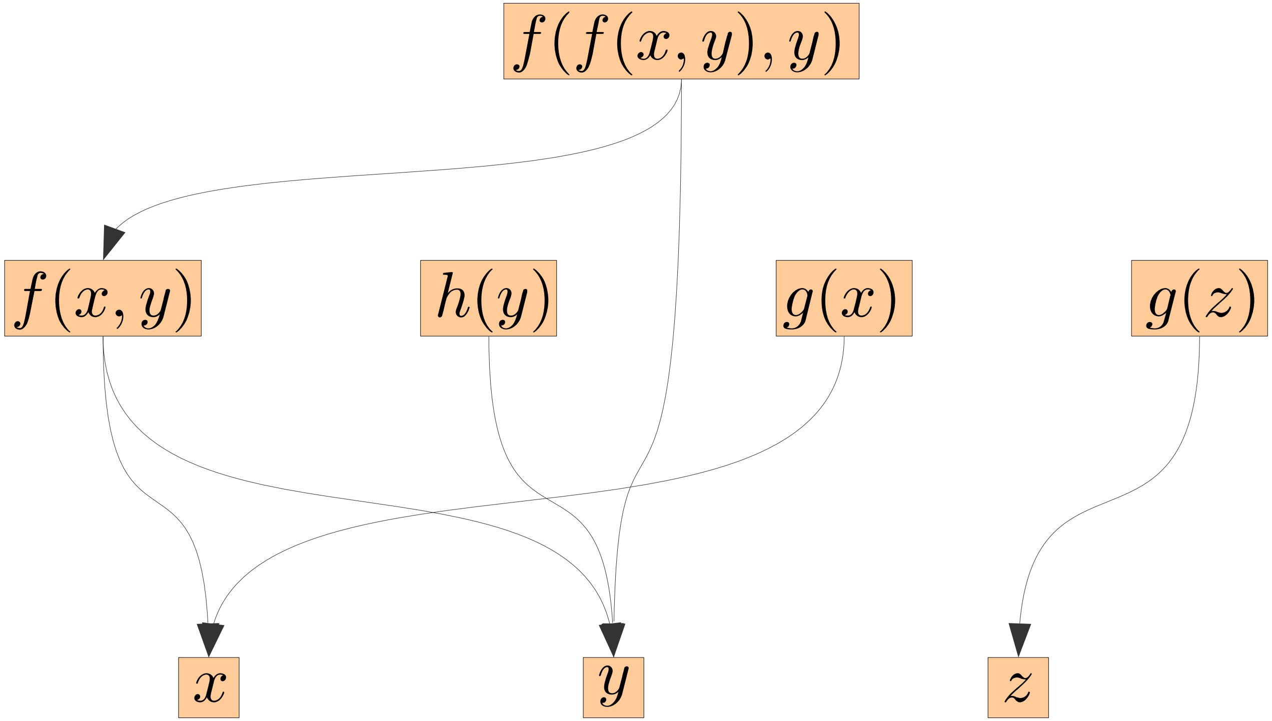
T-solver for Equality (EUF)

- 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

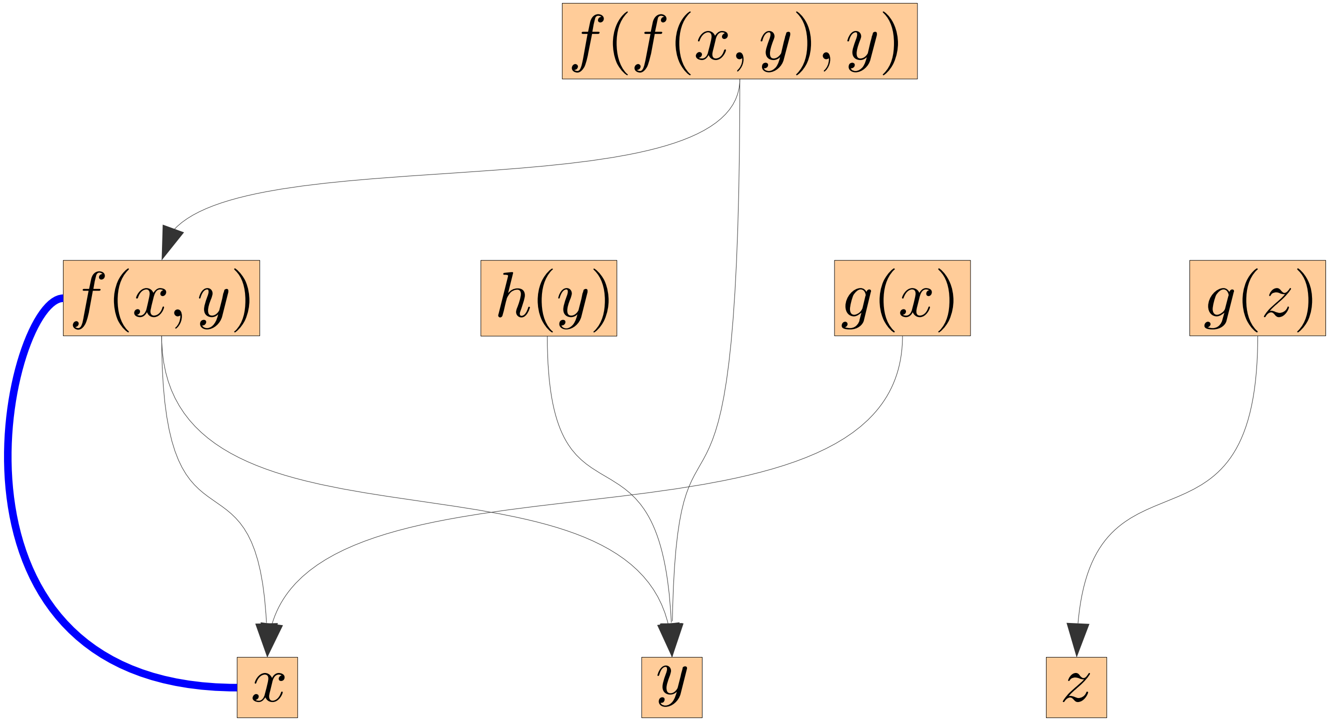
Example

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



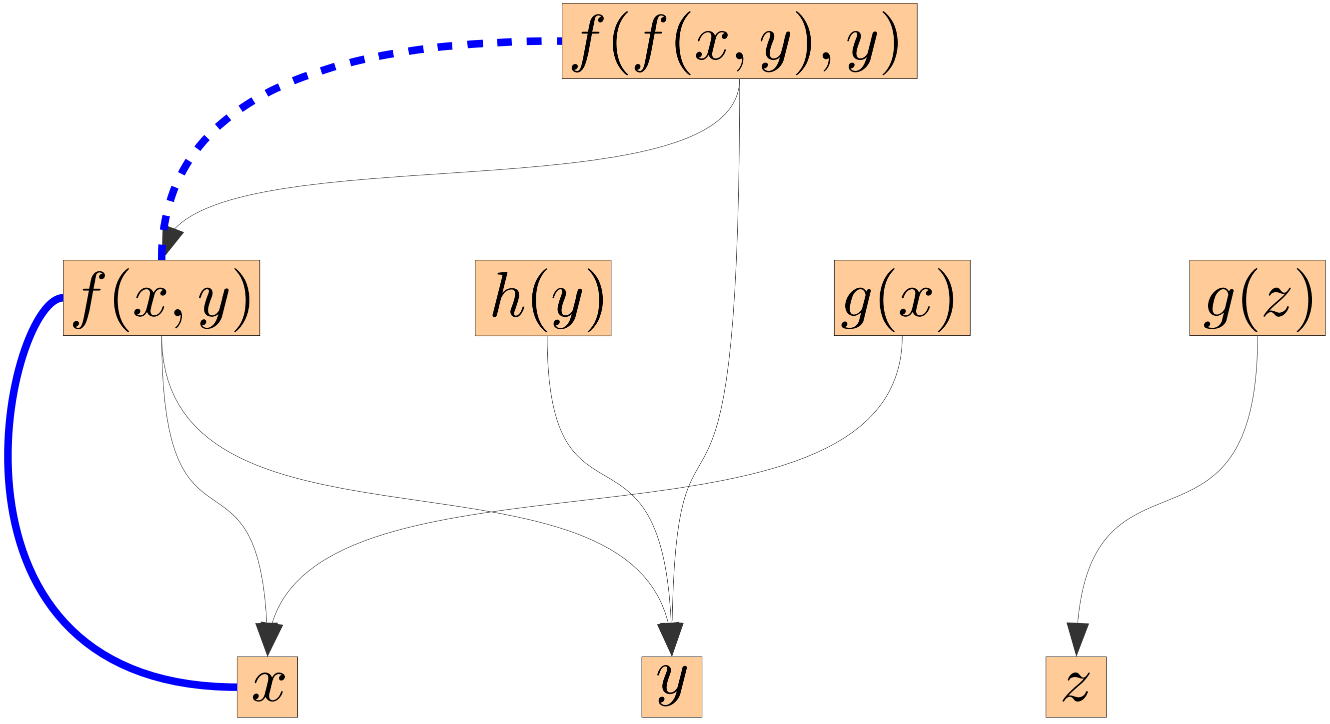
Example

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



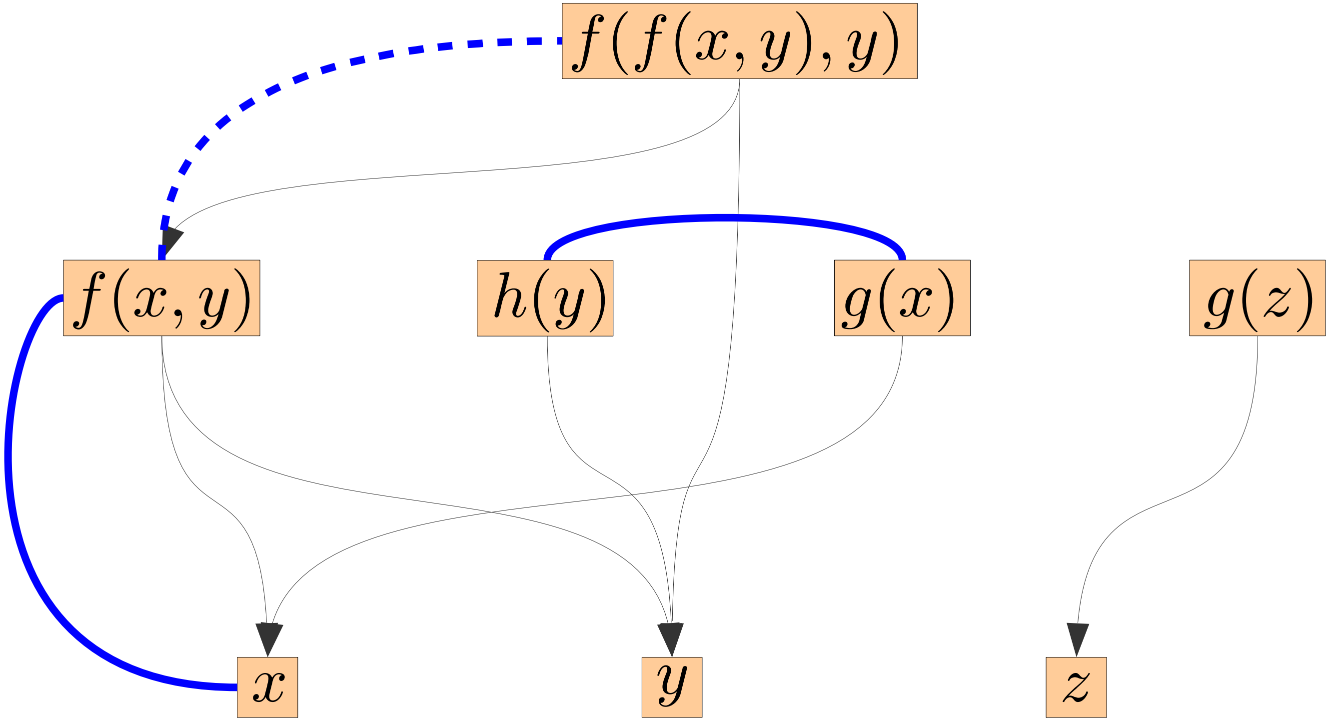
Example

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



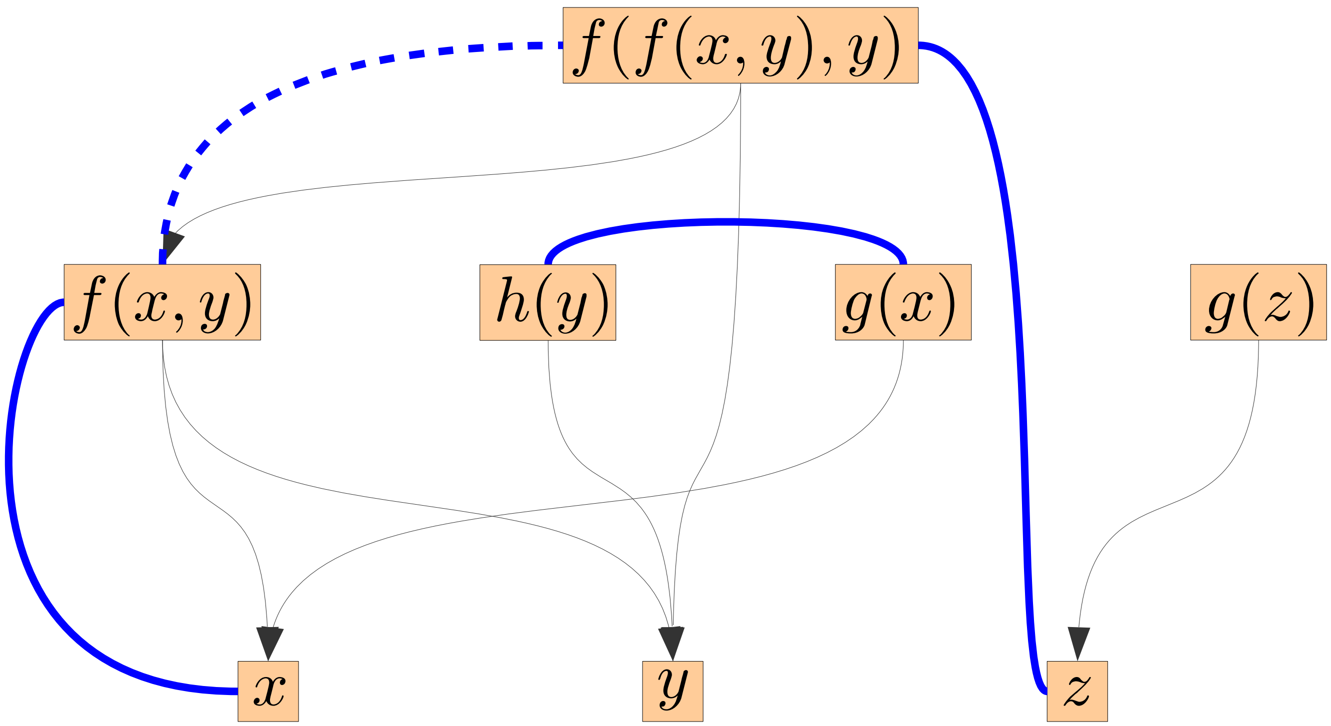
Example

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



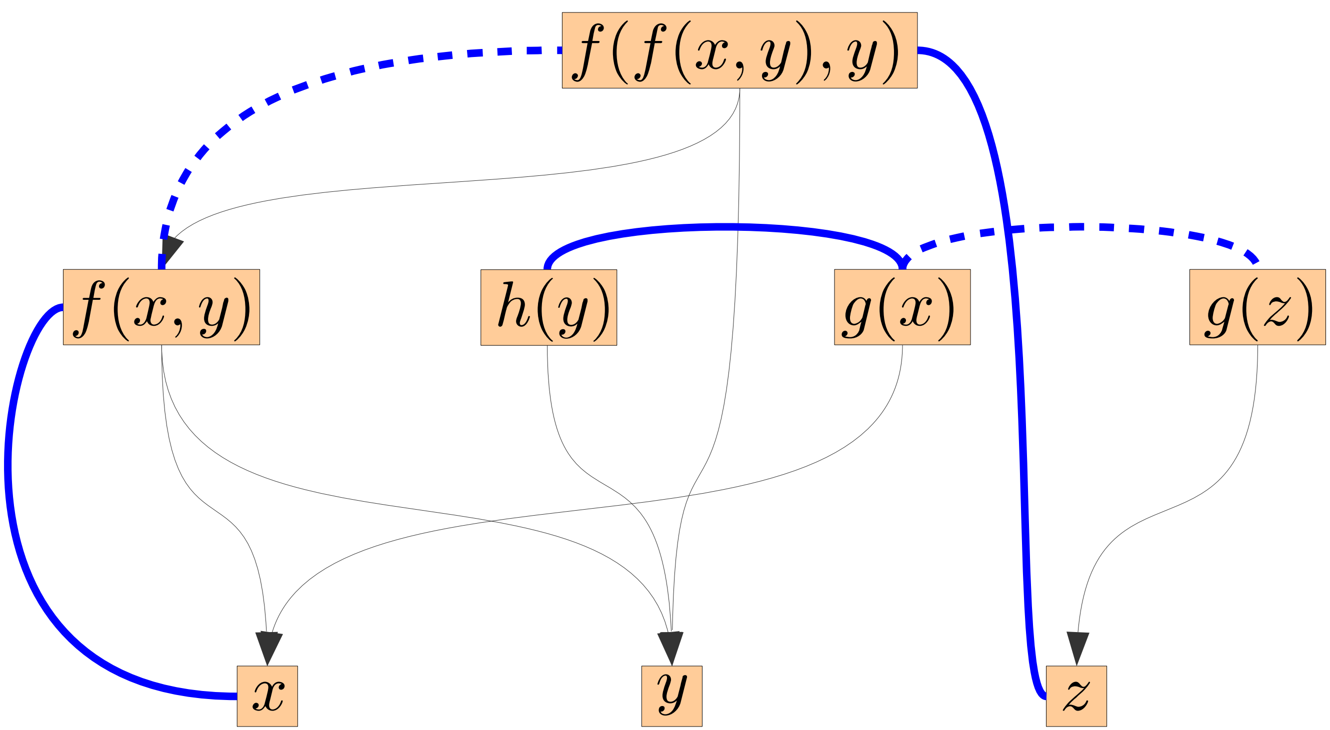
Example

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



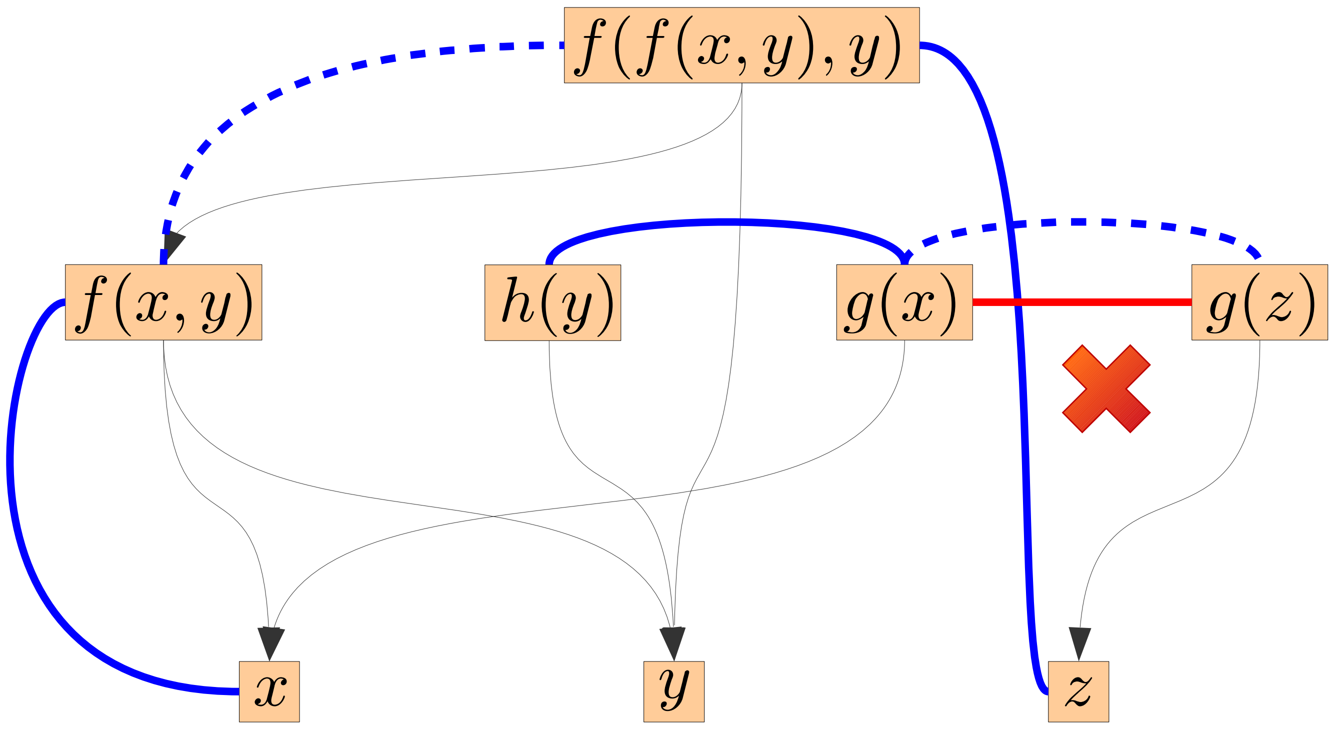
Example

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



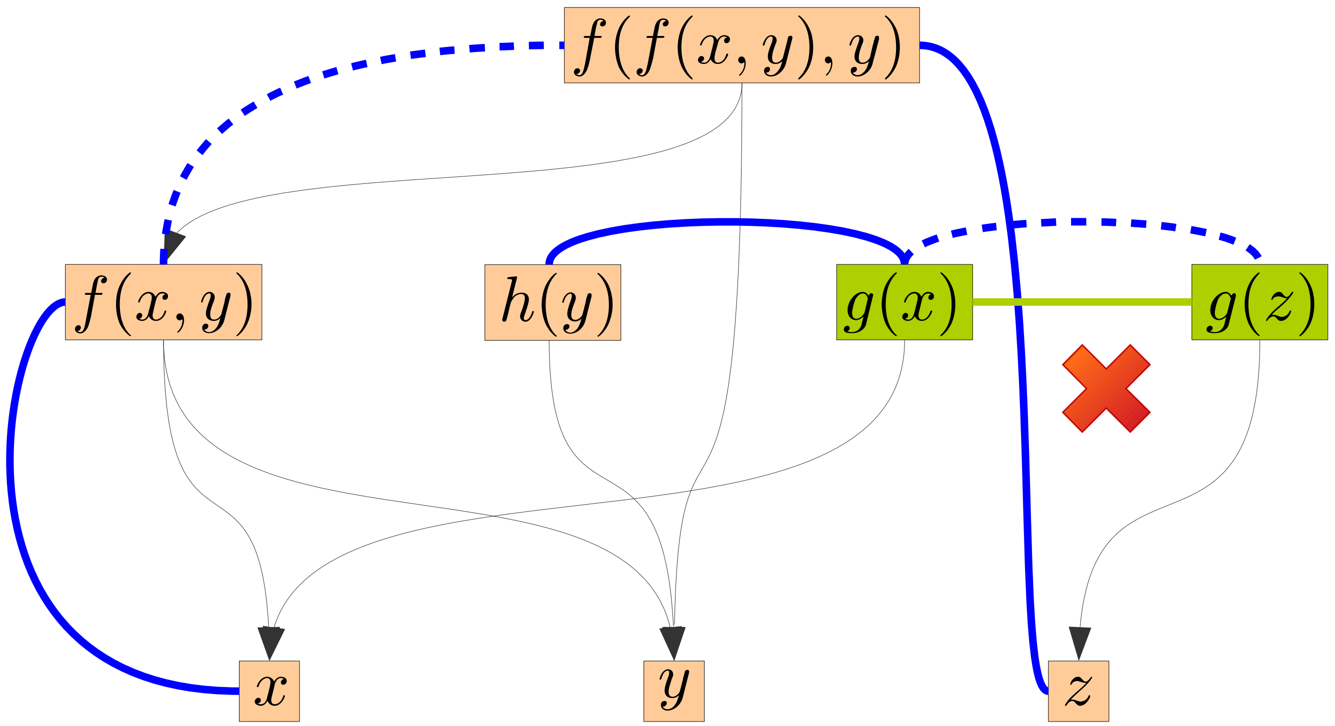
Example

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



Example

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



`get_conflict():`

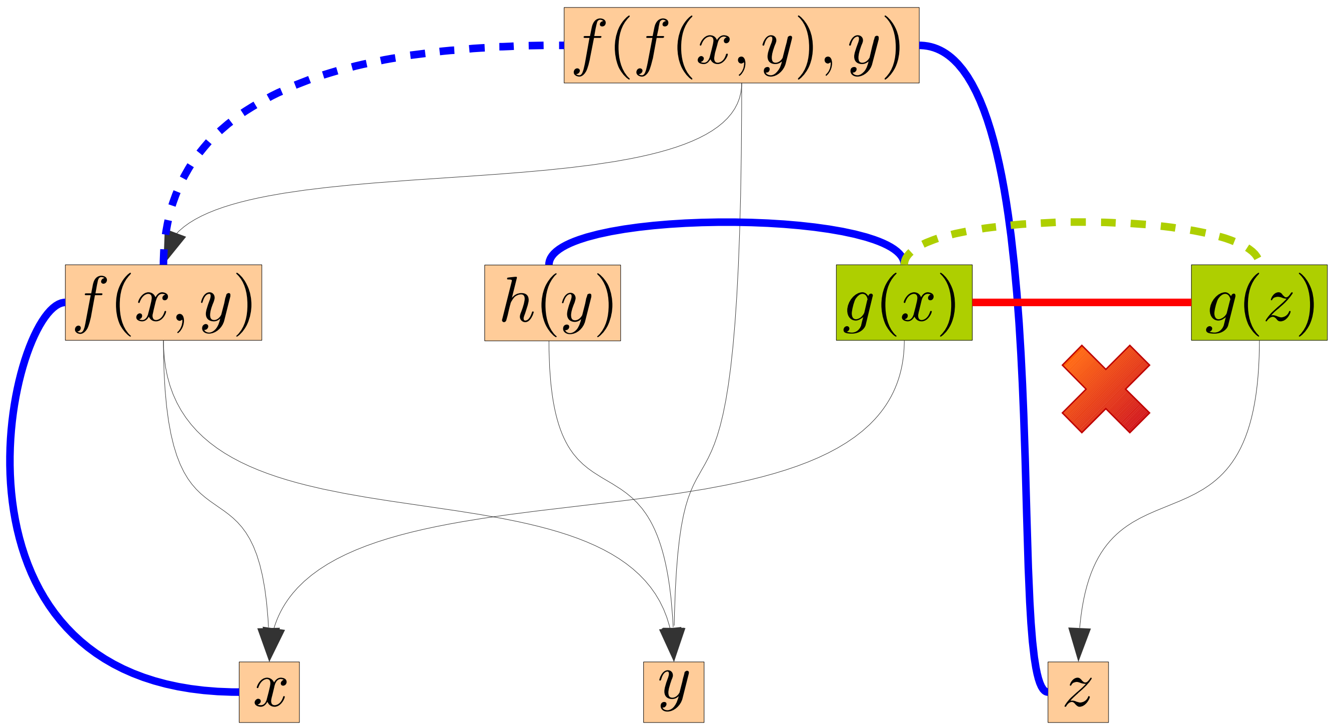
$$\neg(g(x)=g(z))$$

Example

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

`get_conflict():`

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

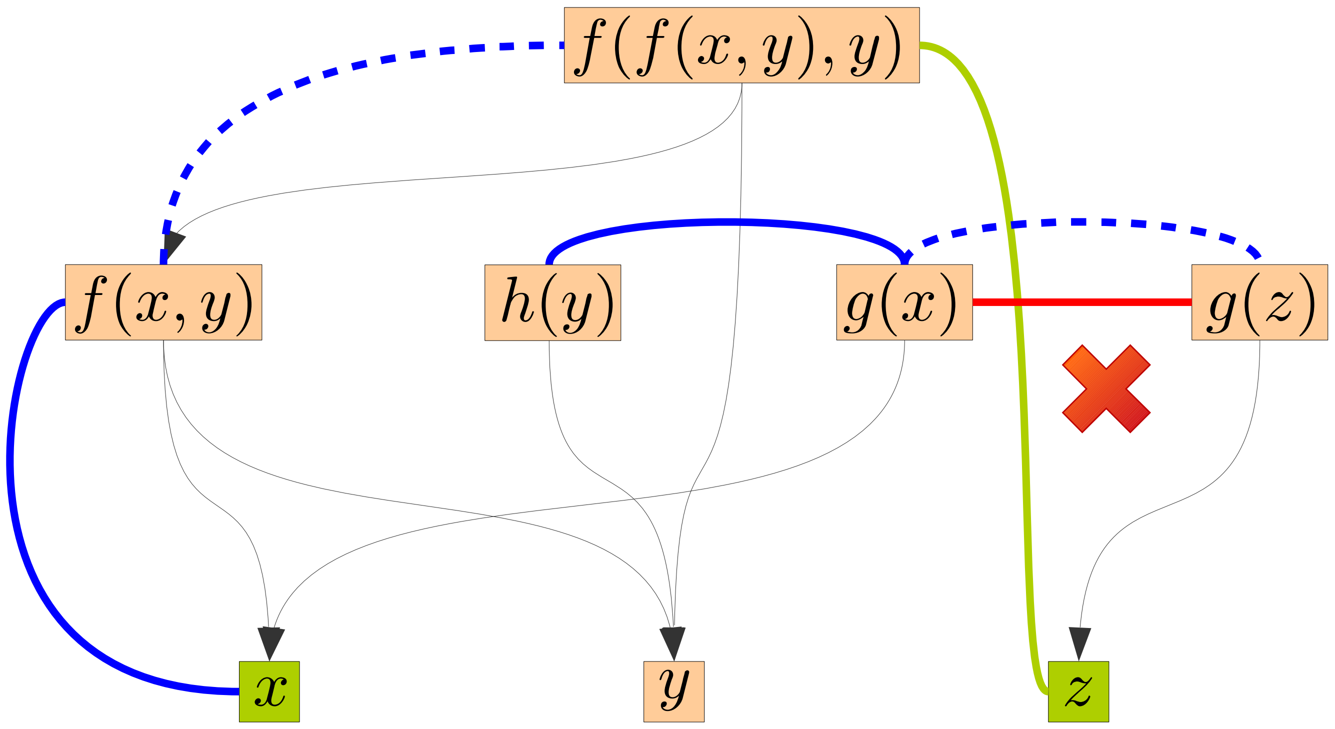


Example

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

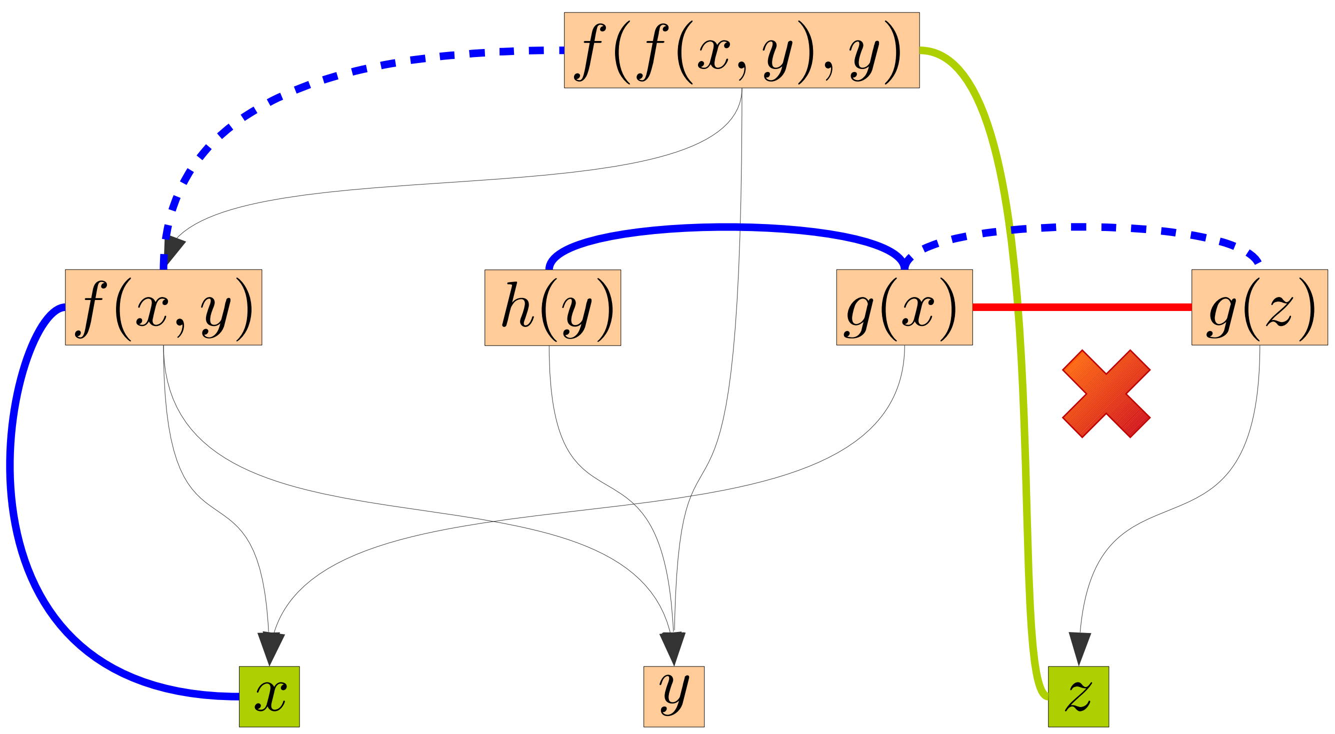
`get_conflict():`

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



Example

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



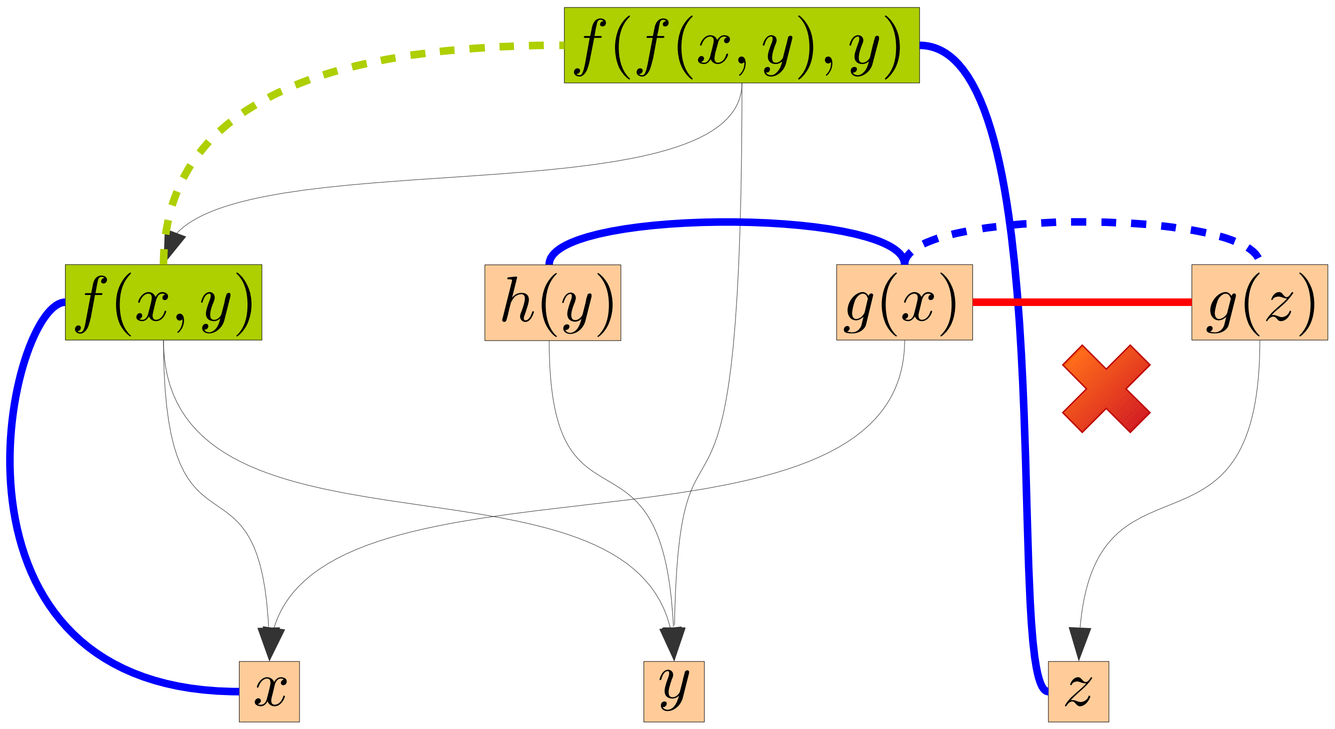
`get_conflict():`

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

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

Example

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



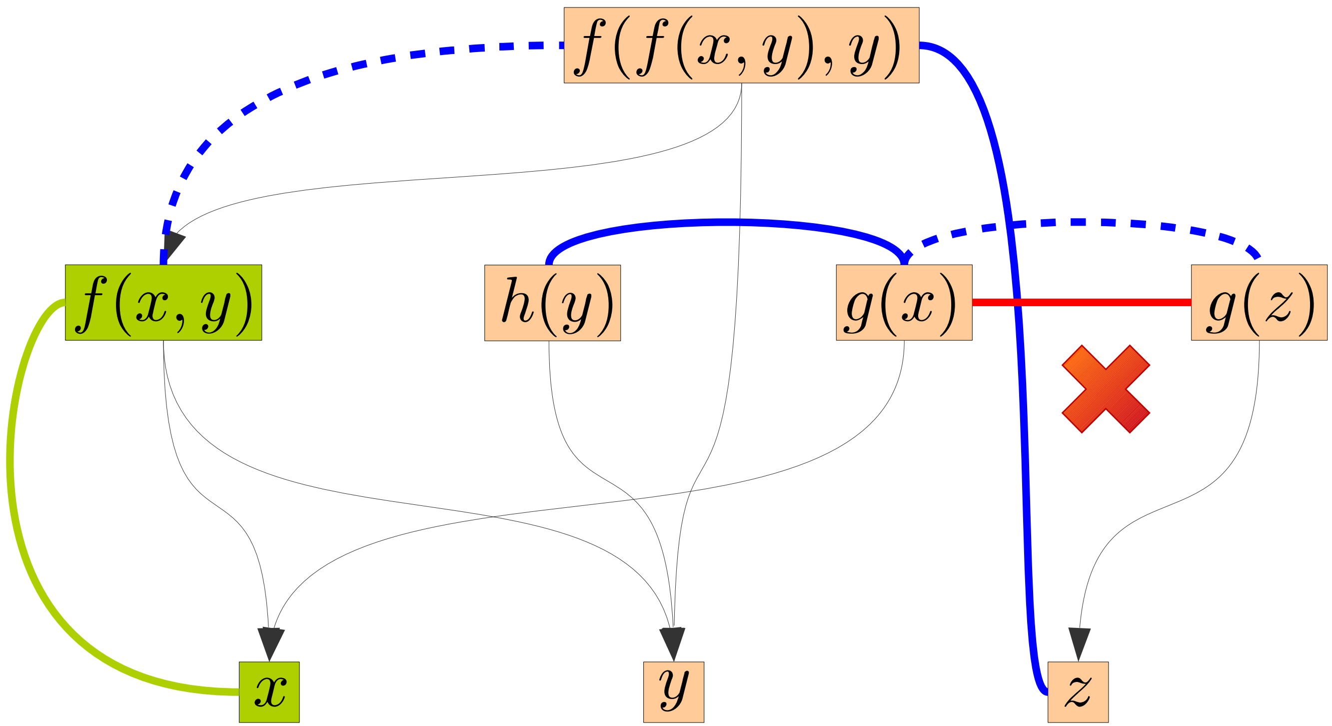
`get_conflict():`

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

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

Example

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



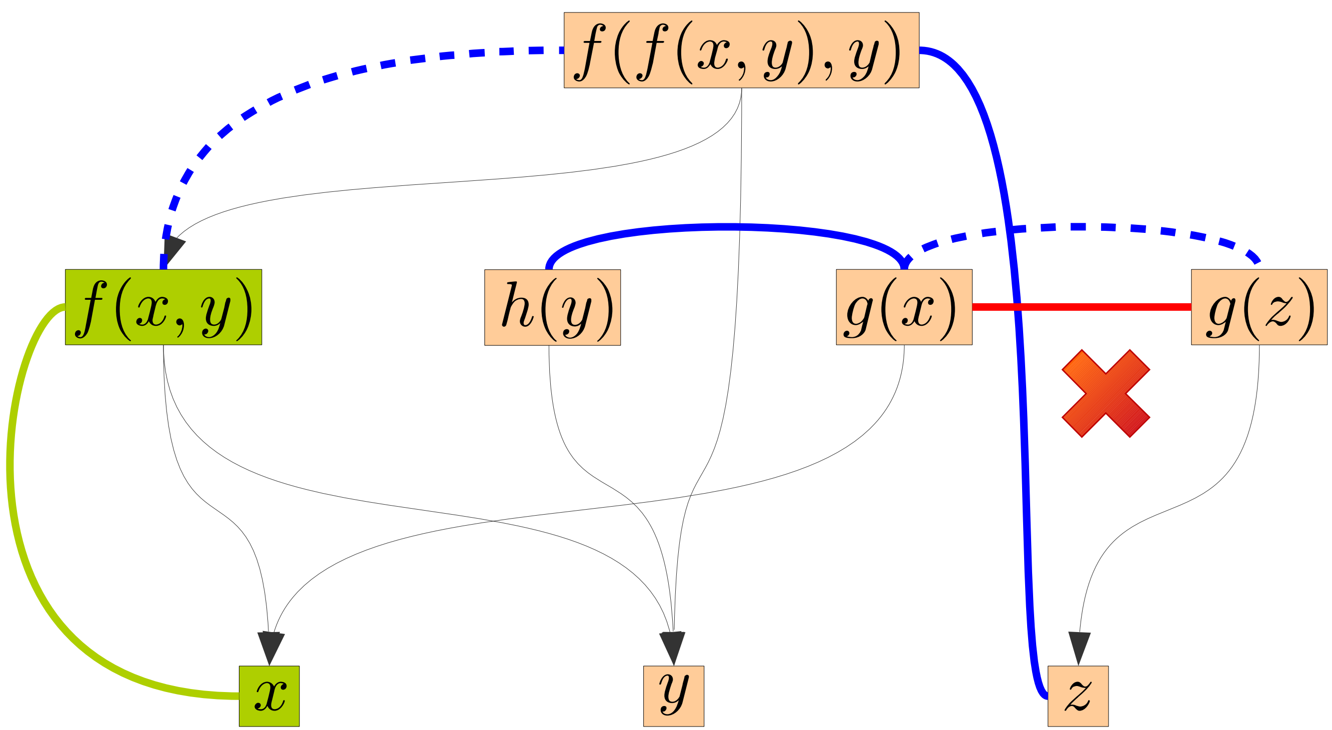
`get_conflict():`

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

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

Example

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



`get_conflict():`

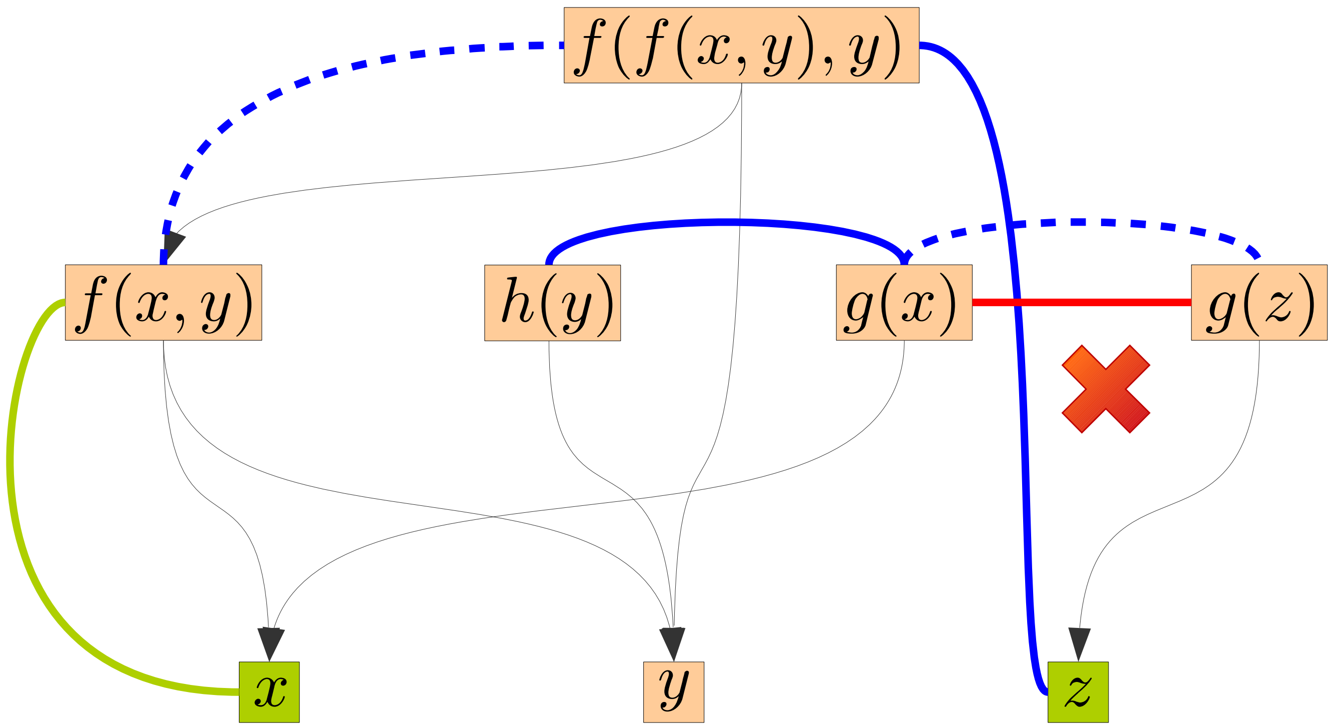
$$\neg(g(x) = g(z))$$

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

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

Example

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



`get_conflict():`

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

Quantifiers in DPLL(T)

- 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 $\vec{t_1} \dots \vec{t_k}$, and add the generated clauses $\{D(\vec{t_1}) \dots D(\vec{t_k})\}$ 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 $D_i(\vec{x})$

E-matching

- 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**
  - Instance generation might get out of control
  - ...

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: $\neg\forall x.(f(x) < x + a)[M(c)/c] \mapsto$
      $$\exists x.\neg(\text{ite}(x = 0, 3, \text{ite}(x = 1, 1, 0)) < x + 1)$$

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

Current trends and future challenges

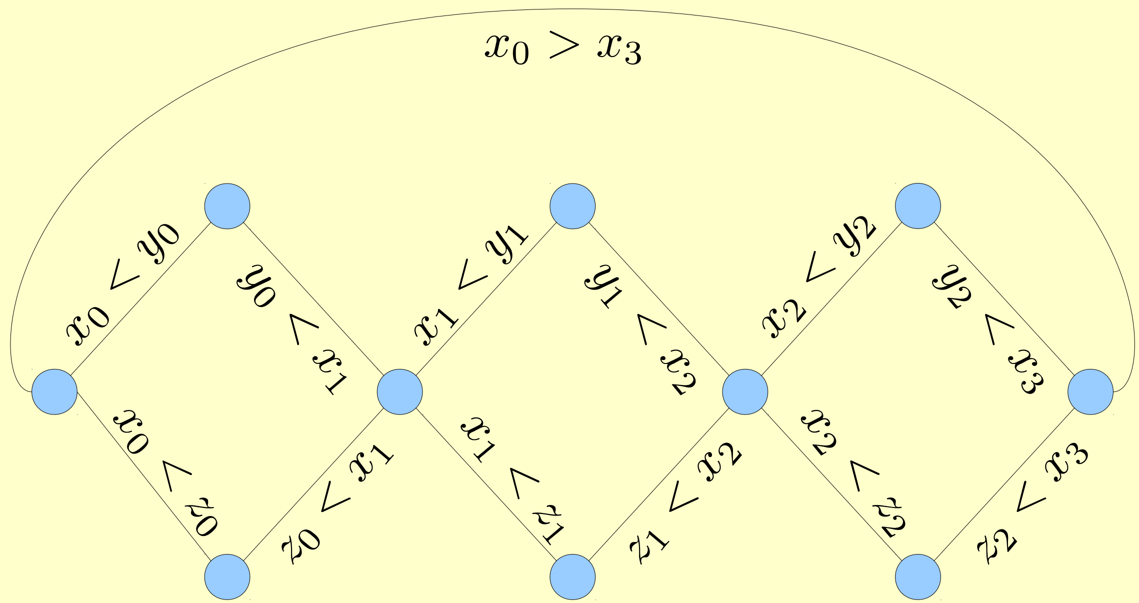
Beyond solving: Optimization Modulo T

- Find a model for φ that is **optimal** wrt. some cost function c
- Boolean cost functions $c \stackrel{\text{def}}{=} \sum_i w_i \cdot \text{ite}(P_i, 1, 0)$
  - DPLL(T) with “increasingly strong” theories
    - Make c part of the theory, strengthen with $(c < ub)$ when an upper bound is found
    - Can encode MaxSMT problems
  - DPLL(T + Costs)
    - A *T*-solver for the “theory of costs”
    - Can encode MaxSMT and Pseudo-Boolean modulo Theories
- Linear cost functions $c \stackrel{\text{def}}{=} \sum_i w_i \cdot x_i$
  - DPLL(T + LP optimization)
    - Optimization via linear programming (simplex)
  - cost minimization embedded inside the CDCL search

Beyond DPLL(T)

- Modular integration of DPLL(T) can be harmful sometimes
  - “Rigid” interface between theory and boolean
  - Restricted by syntax of the input formula

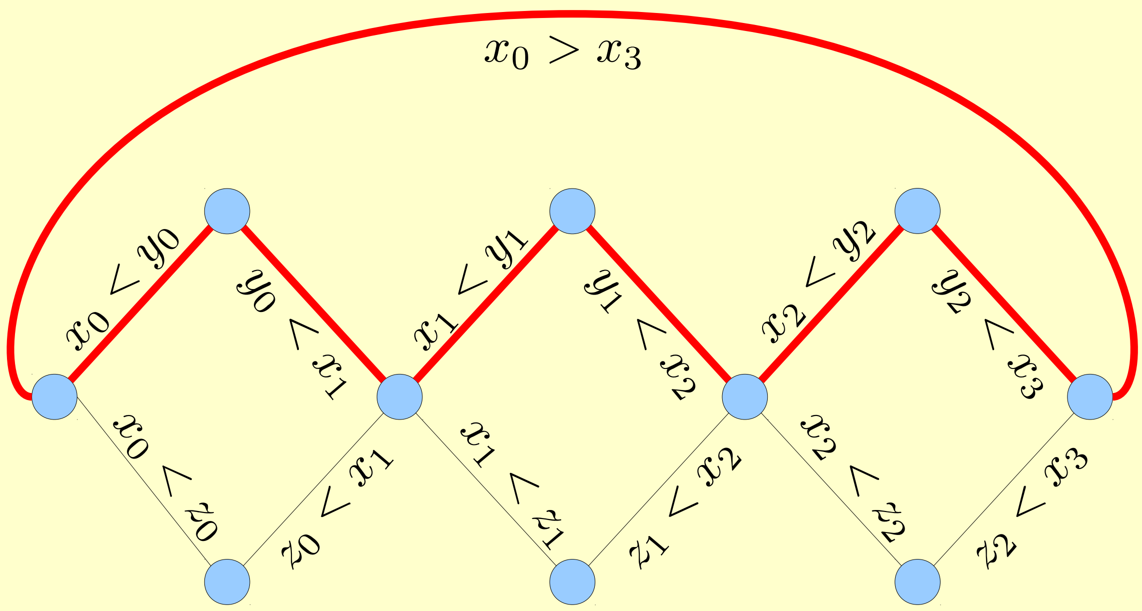
- Example [Jovanovic]:



Beyond DPLL(T)

- Modular integration of DPLL(T) can be harmful sometimes
  - “Rigid” interface between theory and boolean
  - Restricted by syntax of the input formula

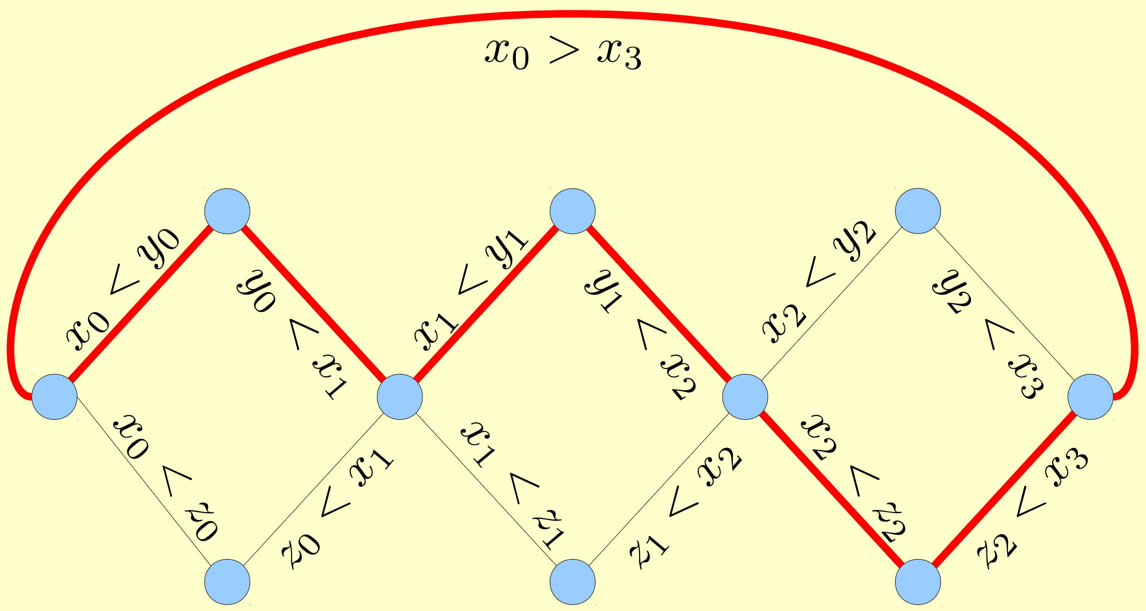
- Example [Jovanovic]:



Beyond DPLL(T)

- Modular integration of DPLL(T) can be harmful sometimes
  - “Rigid” interface between theory and boolean
  - Restricted by syntax of the input formula

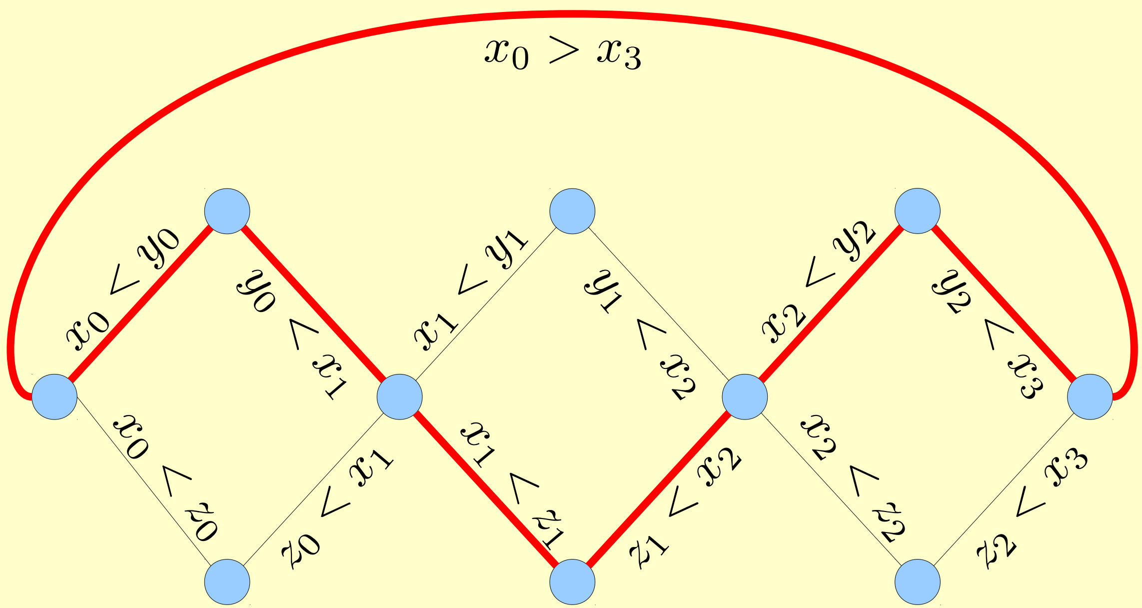
- Example [Jovanovic]:



Beyond DPLL(T)

- Modular integration of DPLL(T) can be harmful sometimes
  - "Rigid" interface between theory and boolean
  - Restricted by syntax of the input formula

- Example [Jovanovic]:



Beyond DPLL(T)

- Model constructing approaches
  - Lift CDCL architecture to operate **directly over the theory**

```
MCSAT(F)
  A = [], dl = 0
  while (true)
    if (theory_unit_rule(F, A))
       if (!all_assigned(F, A))
         var, value = pick_assignment(F, A)
         dl++
         A = A + (var = value, -)
       else return SAT
    else
      lvl, cls = theory_analyze(F, A)
      if (lvl < 0) return UNSAT
      else
         backtrack(F, A, lvl)
         learn(cls)
         dl = lvl
```

Beyond DPLL(T)

- Model constructing approaches
  - Lift CDCL architecture to operate **directly over the theory**

Trail of variable assignments

```
MCSAT(F)
  A = [], dl = 0
  while (true)
    if (theory_unit_rule(F, A))
       if (!all_assigned(F, A))
         var, value = pick_assignment(F, A)
         dl++
         A = A + (var = value, -)
       else return SAT
    else
      lvl, cls = theory_analyze(F, A)
      if (lvl < 0) return UNSAT
      else
         backtrack(F, A, lvl)
         learn(cls)
         dl = lvl
```

Theory reasoning

$$\dfrac{\dfrac{2x + 3y - z \geq 1}{6x + 9y - 3z \geq -3} \qquad \dfrac{-3x - 2y + 4z \geq 2}{-6x - 4y + 8z \geq 4}}{5y + 5z \geq 1}$$

Abstract CD(C)L

- Can we go further?

- Abstract CD(C)L
  - CDCL-like search over abstract domains
  - Based on fixpoint characterization of model search and conflict analysis
  - Applicable to any abstract domain (satisfying some conditions)
    - Not just formulas
    - E.g. CDCL-like analysis of programs

SMT in automated reasoning

- **SC²: SMT Checking meets Symbolic Computation**
  - EU project to make the two communities talk to each other
    - Focus on hard arithmetic theories
- **Integration with first-order theorem provers**
  - E.g. the Avatar architecture
- **Integration with higher-order theorem provers**
  - Incorporate higher-order features, induction
  - E.g. the Matryoshka project
- **Parallelization / exploiting multi cores and clusters**

SMT in verification

- **Provide more than just a yes/no answer**
  - Models, proofs, interpolants, incremental interface, ...
    - Good support for “easy” theories, not so much for “harder” ones
- **Synthesis via SMT**
  - SMT-based quantifier elimination
  - Other special-purpose techniques for handling quantifiers
    - E.g. EF-SMT
- **Constrained Horn Clauses**
  - Model checking as a (quantified) SMT problem

Conclusions

- SMT is a key technology with many important applications
  - Verification (of course)
  - But also more (e.g. planning, scheduling, synthesis, optimization)

- Well-estabilished core, but still many open research directions
  - Relatively few people working on it!
    - $\Rightarrow$ lots of good opporunities

Thank You