Logical reasoning and programming

Lecture 5: Satisfiability Modulo Theories (SMT)

Karel Chvalovský

CIIRC CTU

October 20, 2025

Parts of this presentation are based on materials from SAT/SMT Summer Schools and SC^2 Summer School.

First-Order Logic (recap)

We moved to First-Order Logic (FOL) and introduced some basic notions

- we fix a language L (function and predicate symbols),
- terms in L,
- formulae in L,
- semantics
  - a model for L, denoted $\mathcal{M} = (D, i)$,
  - an evaluation of variables,
  - Tarski's definition of truth,
  - semantic consequence relation $\Gamma \models \varphi$ (iff $\Gamma \cup \{\neg\varphi\}$ is unsat.)

Theory $\mathcal{T}$ (given by L)

We say that an interpretation $\mathcal{M} = (D, i)$ for L is a $\mathcal{T}$-interpretation if $\mathcal{M}$ satisfies all axioms of $\mathcal{T}$, or$i$ admits only intended interpret ations of $\mathcal{T}$.

We say that a formula φ is

- $\mathcal{T}$-satisfiable, if $\mathcal{M} \models \varphi$ for a $\mathcal{T}$-interpretation $\mathcal{M}$;
- $\mathcal{T}$-valid, if $\mathcal{M} \models \varphi$ for every $\mathcal{T}$-interpretation $\mathcal{M}$.

A set of formulae Γ $\mathcal{T}$-entails a formula φ, denoted $\Gamma \models_{\mathcal{T}} \varphi$, if every $\mathcal{T}$-interpretation satisfying all formulae in Γ satisfies also φ.

Satisfiability Modulo Theories (SMT)

We have a formula that has a propositional structure, but propositional variables are expressions in a theory $\mathcal{T}$.

Example

From

$$(\underbrace{x = 0}_{p} \vee \underbrace{x = 1}_{q}) \wedge (\underbrace{x + y + z \neq 0}_{\neg r}) \wedge (\underbrace{f(y) > f(z)}_{s}),$$

we obtain

$$(p \vee q) \wedge \neg r \wedge s,$$

by so-called propositional abstraction (r is $x + y + z = 0$).

Solving SMT

There are two basic approaches how to solve a satisfiability of a formula φ modulo a theory $\mathcal{T}$:

Eager (encode an SMT problem in SAT)

- we translate the problem over the theory into an equisatisfiable propositional formula and use a SAT solver,
- it is eager, because the SAT solver has access to complete theory information from the beginning,
- it requires sophisticated encodings.

Lazy (combine a SAT solver with a decision procedure)

- It is very common that we have theory solvers for problems that are conjunctions of literals.

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—theory view

$$\neg\, a = b$$
$$x = a \;\vee\; x = b$$
$$y = a \;\vee\; y = b$$
$$z = a \;\vee\; z = b$$
$$\neg\, x = y$$

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—propositional view

$$
\begin{array}{ccc}
 & \neg & p_1 \\
p_2 & \vee & p_3 \\
p_4 & \vee & p_5 \\
p_6 & \vee & p_7 \\
 & \neg & p_8
\end{array}
$$

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—propositional view

$$
\begin{array}{ccc}
 & \neg & p_1 \\
p_2 & \vee & p_3 \\
p_4 & \vee & p_5 \\
p_6 & \vee & p_7 \\
 & \neg & p_8
\end{array}
$$

SAT solver

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—propositional view

$$
\begin{array}{ccc}
\neg & p_1 & \\
p_2 & \vee & p_3 \\
p_4 & \vee & p_5 \\
p_6 & \vee & p_7 \\
\neg & p_8 &
\end{array}
$$

SAT solver

$$\neg\ p_1 \quad \wedge\ \neg\ p_8 \quad \wedge\ p_2 \quad \wedge\ \neg\ p_3 \quad \wedge\ p_4 \quad \wedge\ \neg\ p_5 \quad \wedge\ p_6 \quad \wedge\ \neg\ p_7$$

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—theory view

$$\neg\, a = b$$
$$x = a \;\vee\; x = b$$
$$y = a \;\vee\; y = b$$
$$z = a \;\vee\; z = b$$
$$\neg\, x = y$$

Theory solver

$$\neg a = b \wedge \neg x = y \wedge x = a \wedge \neg x = b \wedge y = a \wedge \neg y = b \wedge z = a \wedge \neg z = b$$

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—theory view

$$\neg\, a = b$$
$$x = a \;\vee\; x = b$$
$$y = a \;\vee\; y = b$$
$$z = a \;\vee\; z = b$$
$$\neg\, x = y$$

Theory solver

$$\neg a = b \wedge \neg x = y \wedge x = a \wedge \neg x = b \wedge y = a \wedge \neg y = b \wedge z = a \wedge \neg z = b$$

Theory solver says unsatisfiable. Block this (minimal) solution by adding a clause.

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—theory view

$$\neg\, a = b$$
$$x = a \;\vee\; x = b$$
$$y = a \;\vee\; y = b$$
$$z = a \;\vee\; z = b$$
$$\neg\, x = y$$
$$x = y \;\vee \neg\, x = a \;\vee \neg\, y = a$$

Theory solver

$$\neg a = b \;\wedge\; \neg x = y \;\wedge\; x = a \;\wedge\; \neg x = b \;\wedge\; y = a \;\wedge\; \neg y = b \;\wedge\; z = a \;\wedge\; \neg z = b$$

Theory solver says unsatisfiable. Block this (minimal) solution by adding a clause.

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—propositional view

$$
\begin{array}{rrrrr}
 & \neg & p_1 & & \\
 & p_2 & \vee & p_3 & \\
 & p_4 & \vee & p_5 & \\
 & p_6 & \vee & p_7 & \\
 & \neg & p_8 & & \\
p_8 & \vee\neg & p_2 & \vee\neg & p_4
\end{array}
$$

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—propositional view

$$
\begin{array}{ccccc}
 & \neg & p_1 & & \\
 & p_2 & \vee & p_3 & \\
 & p_4 & \vee & p_5 & \\
 & p_6 & \vee & p_7 & \\
 & \neg & p_8 & & \\
p_8 & \vee\neg & p_2 & \vee\neg & p_4
\end{array}
$$

SAT solver

$$\neg p_1 \wedge \neg p_8 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge p_5 \wedge p_6 \wedge \neg p_7$$

Very lazy SMT—example

Is the following conjunction of clauses satisfiable?

Problem—theory view

$$\neg\, a = b$$
$$x = a \;\vee\; x = b$$
$$y = a \;\vee\; y = b$$
$$z = a \;\vee\; z = b$$
$$\neg\, x = y$$
$$x = y \;\vee \neg\, x = a \;\vee \neg\, y = a$$

Theory solver

$$\neg a = b \;\wedge\; \neg x = y \wedge\; x = a \wedge\; \neg x = b \wedge\; \neg y = a \wedge\; y = b \wedge\; z = a \wedge\; \neg z = b$$

Satisfiable by taking two constants $c_1 \neq c_2$, where $c_1 = a = x = z$ and $c_2 = b = y$.

Lazy approach

Let φ be a formula and we want to know whether φ is $\mathcal{T}$-satisfiable. Let φ' be a propositional abstraction of φ.

- we call a SAT solver on φ'
- if $\varphi' \in \mathtt{SAT}$, then we obtain a set of literals $l'_1, \ldots, l'_n$ in φ' that satisfies the formula φ',
  - we can then ask a $\mathcal{T}$-solver whether the set of corresponding literals $l_1, \ldots, l_n$ in φ is satisfiable in $\mathcal{T}$
    - if it is, then return φ is satisfiable and provide a model,
    - if it is not, then we can add a new propositional clause $\overline{l'_1} \vee \cdots \vee \overline{l'_n}$ to φ' (or something better) and repeat the whole process with a new propositional formula,
- if $\varphi' \notin \mathtt{SAT}$, then return φ is unsatisfiable.

DPLL($\mathcal{T}$)—lazy approach + theory propagations

It "effectively" transforms a satisfiability of an arbitrary quantifier-free formula over $\mathcal{T}$ to a satisfiability of a conjunction of literals over $\mathcal{T}$. In basic lazy approach there is no theory guidance, here $\mathcal{T}$-solver guides the search by producing $\mathcal{T}$-consequences.

For efficiency reasons we want several things from a solver for $\mathcal{T}$:

- checks consistency of conjunctions of literals,
- computes $\mathcal{T}$-propagations ($\mathcal{T}$-consequences),
- pruduces explanations (ideally minimal) of $\mathcal{T}$-inconsistencies and $\mathcal{T}$-propagations,
- should be incremental and backtrackable,
- (generate $\mathcal{T}$-atoms and $\mathcal{T}$-lemmata).

Although it is called DPLL($\mathcal{T}$), in practice, CDCL is usually used and hence we want a support for backjumping and conflicts. Moreover, we want to test already partial assignments not only full propositional models for $\mathcal{T}$-consistency.

DPLL($\mathcal{T}$)- –$\mathcal{T}$-propagations example

We have

$$\underbrace{g(a) = c}_{p} \wedge \big(\underbrace{f(g(a)) \neq f(c)}_{\neg q} \vee \underbrace{g(a) = d}_{r} \big) \wedge \underbrace{c \neq d}_{\neg s}.$$

DPLL($\mathcal{T}$)- –$\mathcal{T}$-propagations example

We have

$$\underbrace{g(a) = c}_{p} \wedge \big(\underbrace{f(g(a)) \neq f(c)}_{\neg q} \vee \underbrace{g(a) = d}_{r} \big) \wedge \underbrace{c \neq d}_{\neg s}.$$

SAT solver

$\mathcal{T}$-solver

p

$\neg s$

By unit propagations.

DPLL($\mathcal{T}$)- –$\mathcal{T}$-propagations example

We have

$$\underbrace{g(a) = c}_{p} \wedge \big(\underbrace{f(g(a)) \neq f(c)}_{\neg q} \vee \underbrace{g(a) = d}_{r} \big) \wedge \underbrace{c \neq d}_{\neg s}.$$

SAT solver	$\mathcal{T}$-solver
p	$g(a) = c$
$\neg s$	$c \neq d$

DPLL($\mathcal{T}$)- –$\mathcal{T}$-propagations example

We have

$$\underbrace{g(a) = c}_{p} \wedge (\underbrace{f(g(a)) \neq f(c)}_{\neg q} \vee \underbrace{g(a) = d}_{r}) \wedge \underbrace{c \neq d}_{\neg s} .$$

SAT solver	$\mathcal{T}$-solver
p	$g(a) = c$
$\neg s$	$c \neq d$
	$f(g(a)) = f(c)$
	$g(a) \neq d$

By $\mathcal{T}$-propagations.

DPLL($\mathcal{T}$)- –$\mathcal{T}$-propagations example

We have

$$\underbrace{g(a) = c}_{p} \wedge \big(\underbrace{f(g(a)) \neq f(c)}_{\neg q} \vee \underbrace{g(a) = d}_{r} \big) \wedge \underbrace{c \neq d}_{\neg s}.$$

SAT solver	$\mathcal{T}$-solver
p	$g(a) = c$
$\neg s$	$c \neq d$
q	$f(g(a)) = f(c)$
$\neg r$	$g(a) \neq d$

DPLL($\mathcal{T}$)- –$\mathcal{T}$-propagations example

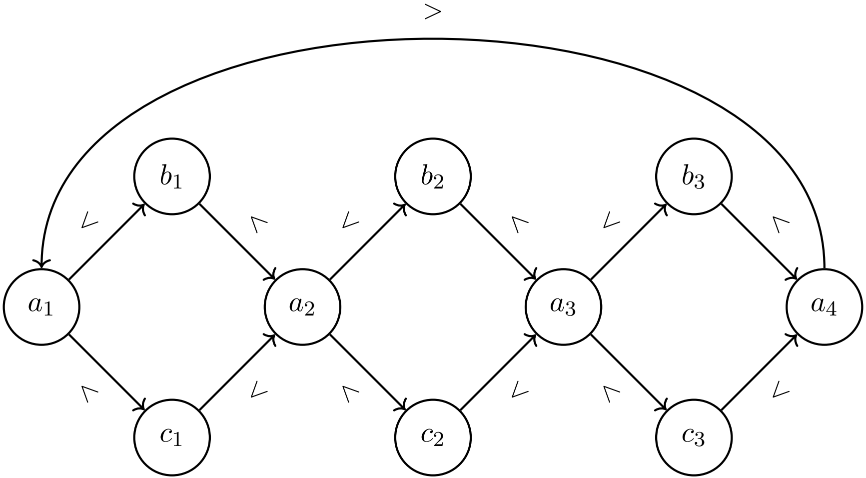
We have

$$\underbrace{g(a) = c}_{p} \wedge \big(\underbrace{f(g(a)) \neq f(c)}_{\neg q} \vee \underbrace{g(a) = d}_{r} \big) \wedge \underbrace{c \neq d}_{\neg s}.$$

SAT solver	$\mathcal{T}$-solver
p	$g(a) = c$
$\neg s$	$c \neq d$
q	$f(g(a)) = f(c)$
$\neg r$	$g(a) \neq d$
UNSAT	

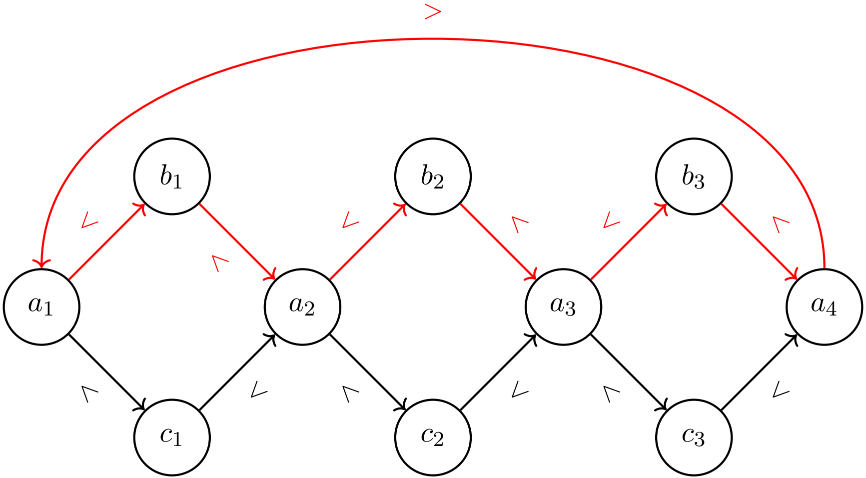
DPLL($\mathcal{T}$) issue—diamond example

Is $a_1 > a_n \wedge \bigwedge_{k=1}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$ satisfiable?



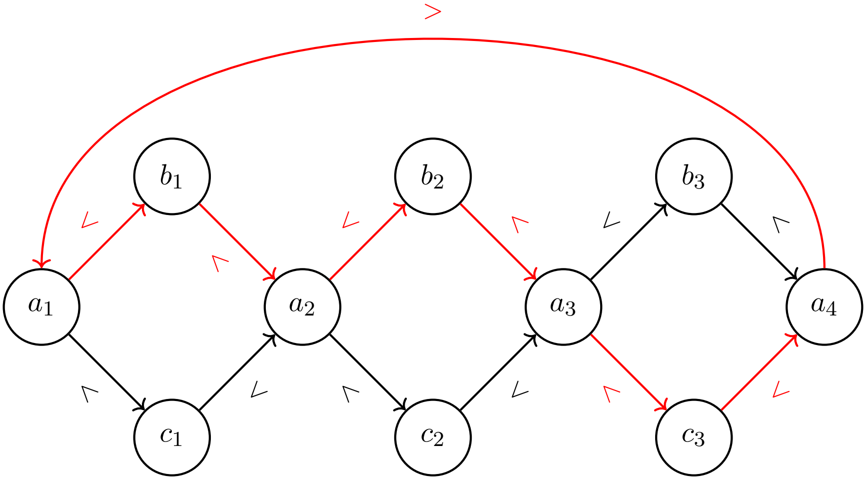
DPLL($\mathcal{T}$) issue—diamond example

$$\text{Is } a_1 > a_n \wedge \bigwedge_{k=1}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1})) \text{ satisfiable?}$$



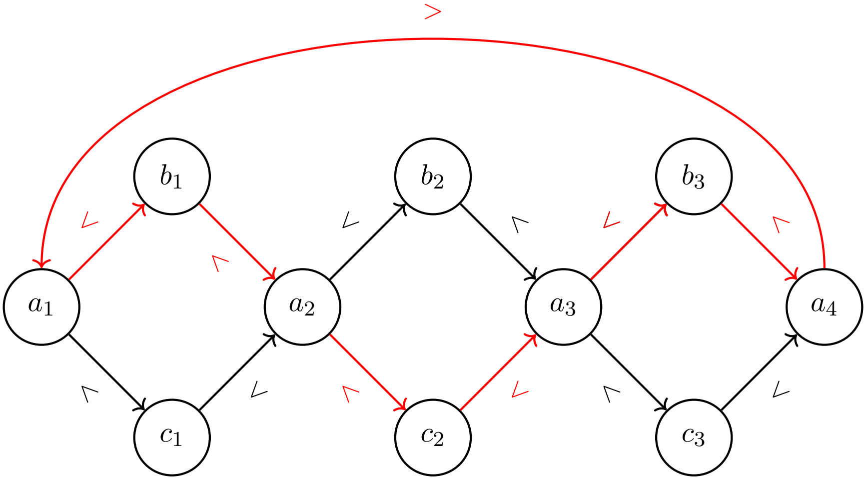
DPLL($\mathcal{T}$) issue—diamond example

Is $a_1 > a_n \wedge \bigwedge_{k=1}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$ satisfiable?



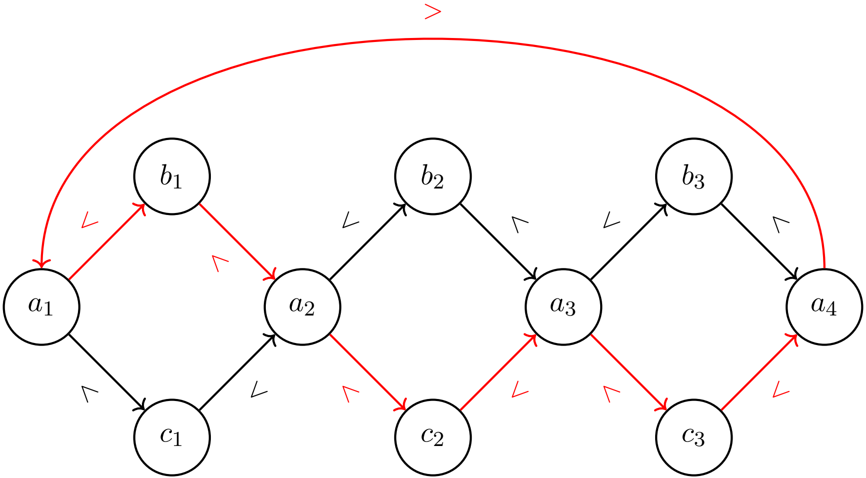
DPLL($\mathcal{T}$) issue—diamond example

Is $a_1 > a_n \wedge \bigwedge_{k=1}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$ satisfiable?



DPLL($\mathcal{T}$) issue—diamond example

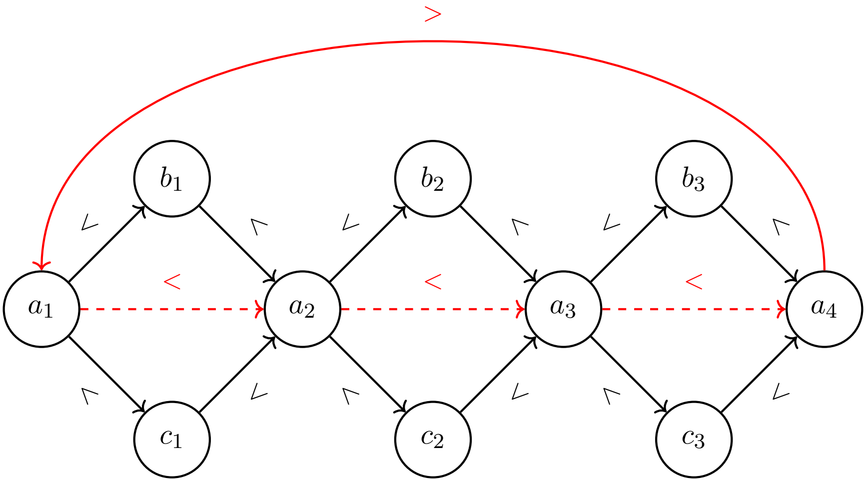
Is $a_1 > a_n \wedge \bigwedge_{k=1}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$ satisfiable?



This leads to an exponential enumeration.

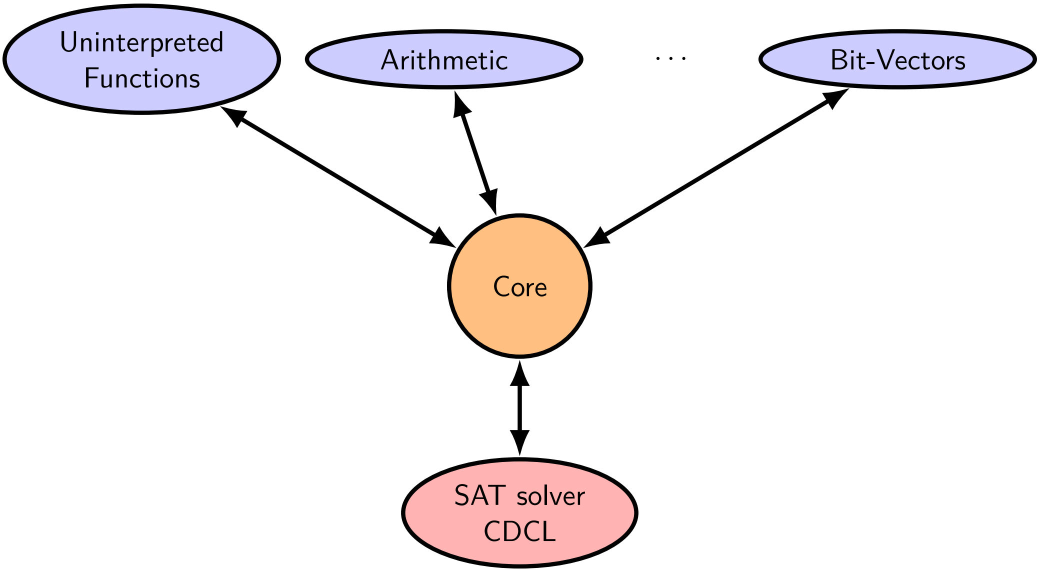
DPLL($\mathcal{T}$) issue—diamond example

Is $a_1 > a_n \wedge \bigwedge_{k=1}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$ satisfiable?

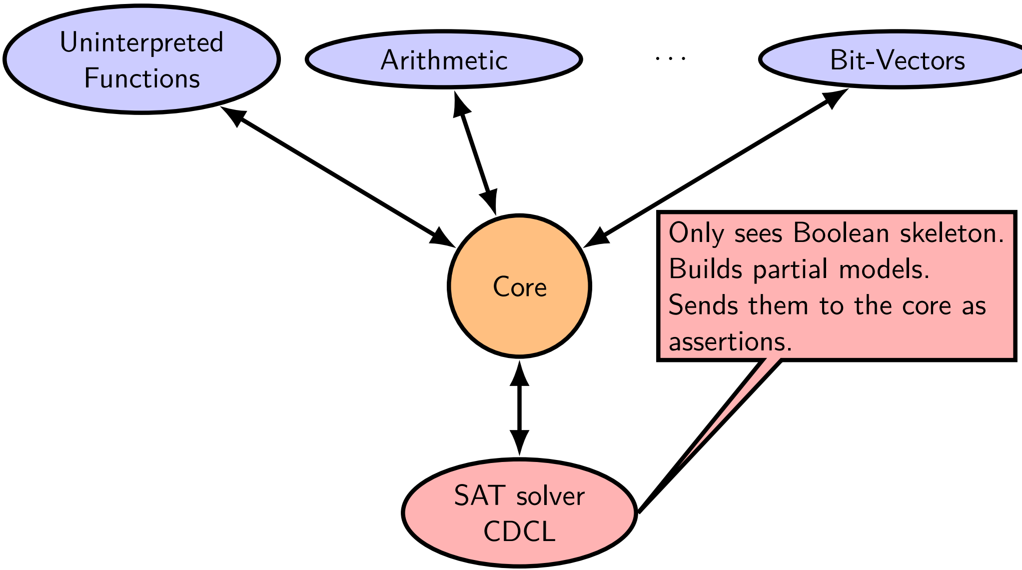


Can be solved by deducing lemmata $a_i < a_{i+1}$.

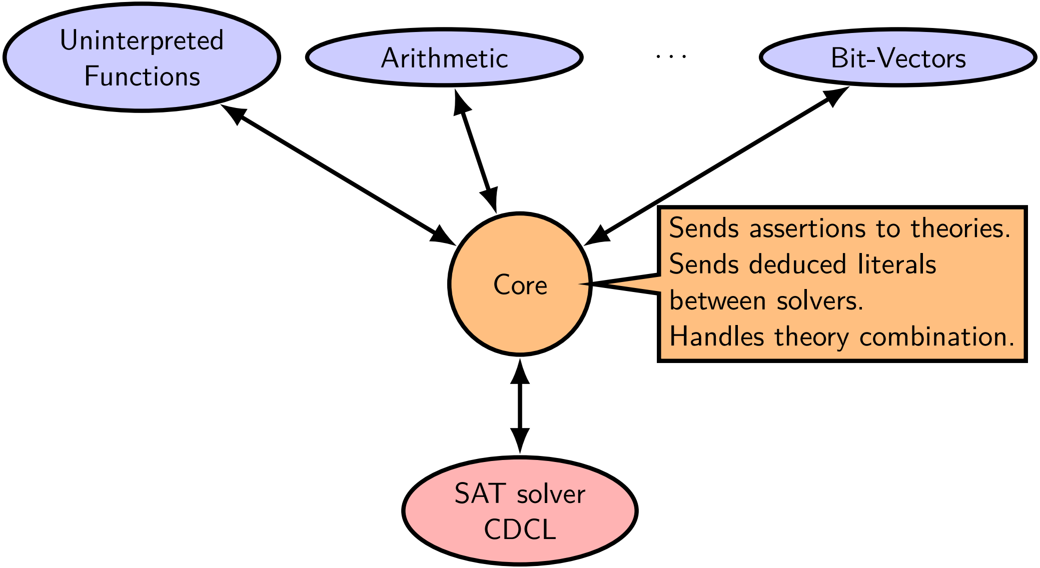
DPLL($\mathcal{T}$) modular architecture



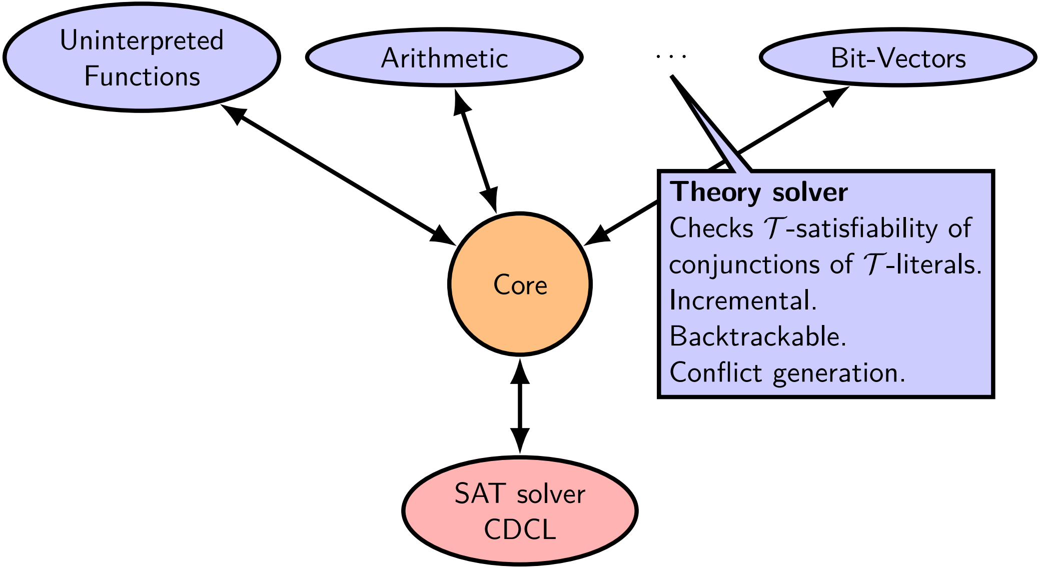
DPLL($\mathcal{T}$) modular architecture



DPLL($\mathcal{T}$) modular architecture



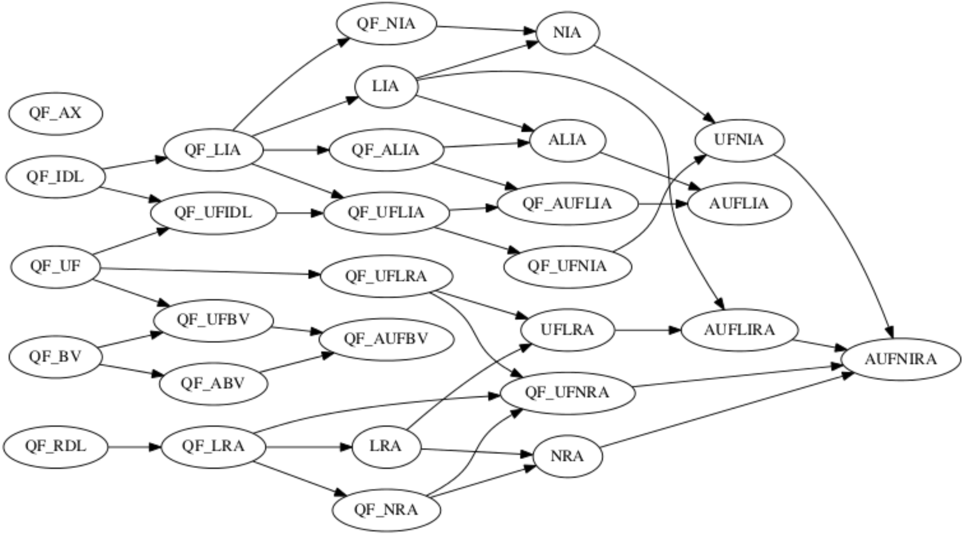
DPLL($\mathcal{T}$) modular architecture



Theory solver
Checks $\mathcal{T}$-satisfiability of conjunctions of $\mathcal{T}$-literals.
Incremental.
Backtrackable.
Conflict generation.

Various theories and their combinations

It is possible to combine various theories, called logics here, and give them "canonical" names.



source: SMT-LIB

Uninterpreted functions (UF)

We have literals of the form

$$s = t \quad \text{and} \quad s \neq t,$$

where s and t may contain constants (variables) and function symbols. Equality is reflexive, symmetric, transitive, and satisfies congruence axioms

$$\forall X_1 \ldots \forall X_m \forall Y_1 \ldots \forall Y_m (X_1 = Y_1 \wedge \cdots \wedge X_m = Y_m \rightarrow$$
$$f(X_1, \ldots, X_m) = f(Y_1, \ldots, Y_m))$$

for every m-ary function symbol f. It is sometimes called functional consistency in this context.

(QF_UF) is usually the core of an SMT solver, which is used in other theories. It is decidable in $\mathcal{O}(n \log n)$.

Why no uninterpreted predicate symbols?

The only predicate symbol allowed in (QF_UF) is equality, because every other uninterpreted predicate symbol can be expressed by a fresh uninterpreted function

$$p(t_1, \ldots, t_m) \text{ becomes } f_p(t_1, \ldots, t_m) = \top,$$
$$\neg p(t_1, \ldots, t_m) \text{ becomes } f_p(t_1, \ldots, t_m) \neq \top,$$

where $\top$ is a new constant and f_p is a new function symbol for every predicate p in our original language. Note that f_p and $\top$ are not valid arguments of other terms.

Example

$p(a) \vee \neg q(a, g(a, b))$ becomes $f_p(a) = \top \vee f_q(a, g(a, b)) \neq \top$.

Producing congruence closure

We have a formula φ

$$s_1 = t_1 \wedge \cdots \wedge s_k = t_k \wedge s_{k+1} \neq t_{k+1} \wedge \cdots \wedge s_{k+l} \neq t_{k+l}.$$

The idea is to produce the congruence closure of

$$s_1 = t_1 \wedge \cdots \wedge s_k = t_k$$

that is we apply reflexivity, symmetry, transitivity, and congruence axioms as many times as possible.

Then we just check whether any of

$$s_{k+1} = t_{k+1}, \ldots, s_{k+l} = t_{k+l}$$

is among them. If this is the case, the problem is unsatisfiable. Otherwise, it is satisfiable.

Example

We want to check $a = b \wedge f(g(a)) \neq f(g(b))$.

Producing congruence closure II.

We have a formula φ

$$s_1 = t_1 \wedge \dots \wedge s_k = t_k \wedge s_{k+1} \neq t_{k+1} \wedge \dots \wedge s_{k+l} \neq t_{k+l}.$$

Although the congruence closure is in general infinite, it is sufficient to check only finitely many equalities here, namely the equalities produced from all the subterms occurring in φ.

Example

We want to check $a = b \wedge f(g(a)) \neq f(g(b))$. Hence it is sufficient to produce only equalities containing a, b, $g(a)$, $g(b)$, $f(g(a))$, and $f(g(b))$. It means we get

$$\{a = a, b = b, a = b, g(a) = g(a), g(b) = g(b), g(a) = g(b),$$
$$f(g(a)) = f(g(a)), f(g(b)) = f(g(b)), f(g(a)) = f(g(b))\}.$$

Hence it is unsatisfiable, because it contains $f(g(a)) = f(g(b))$.

Producing congruence closure III.

We have a formula φ

$$s_1 = t_1 \wedge \cdots \wedge s_k = t_k \wedge s_{k+1} \neq t_{k+1} \wedge \cdots \wedge s_{k+l} \neq t_{k+l}.$$

It is convenient to represent the congruence closure by the equivalence classes of the terms occurring in φ:

- each subterm occurring in φ forms an equivalence class,
- for every $s_i = t_i \in \varphi$
  - we merge the equivalance classes containing s_i and t_i,
  - we apply the congruence axioms (at least one argument is from the merged class containing s_i and t_i).

    If this leads to new merges, we propagate congruences further (at least one argument is from a newly merged class) as long as possible.
- return unsatisfiable if s_j and t_j are in the same equivalence class for $s_j \neq t_j \in \varphi$, otherwise return satisfiable.

It is possible to produce an even better representation using DAGs.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

Write down all the subterms.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

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

$$f(x,y) \qquad g(x) \qquad h(y) \qquad g(z)$$

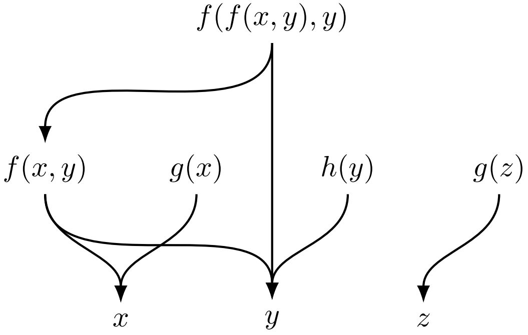
$$x \qquad y \qquad z$$

Including the immediate subterm relations.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

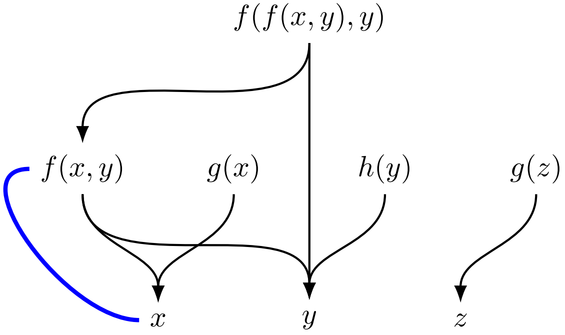


Merge the equivalence classes containing $f(x,y)$ and x.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

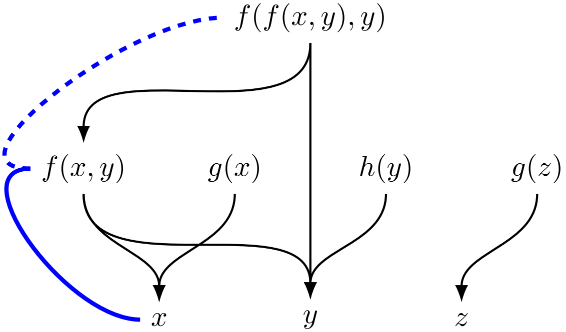


Apply the congruence closure axioms.

Producing congruence closure—example

We want to satisfy

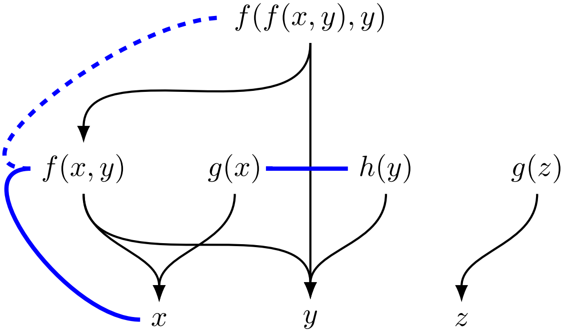
$$f(x, y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x, y), y) = z \quad \wedge \quad g(x) \neq g(z).$$



Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

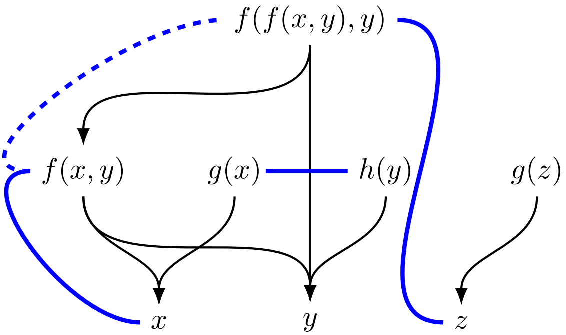


Merge the equivalence classes for $h(y)$ and $g(x)$.

Producing congruence closure—example

We want to satisfy

$$f(x,y)=x \quad \wedge \quad h(y)=g(x) \quad \wedge \quad f(f(x,y),y)=z \quad \wedge \quad g(x)\neq g(z).$$

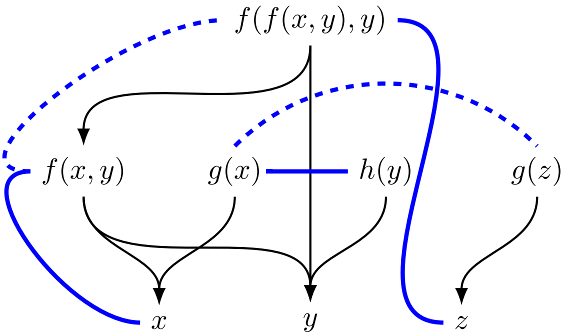


Merge the equivalence classes for $f(f(x,y),y)$ and z.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

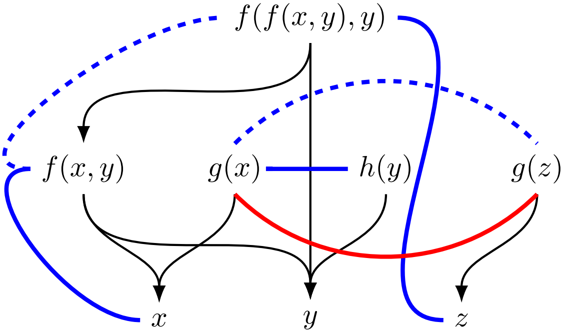


Apply the congruence closure axioms.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad \color{red}{g(x) \neq g(z)}.$$

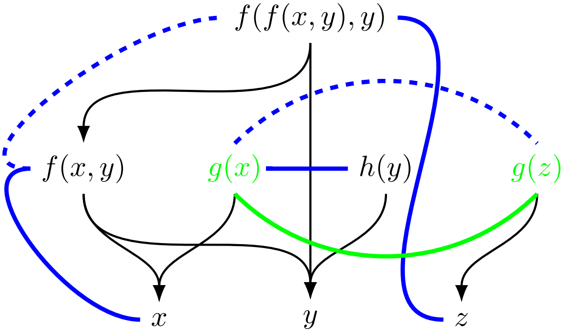


Unsatisfiable because $g(x)$ and $g(z)$ are in the same equivalence class.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

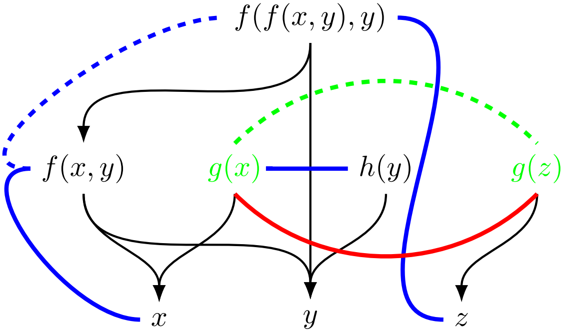


The conflict involves $g(x) \neq g(z)$.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

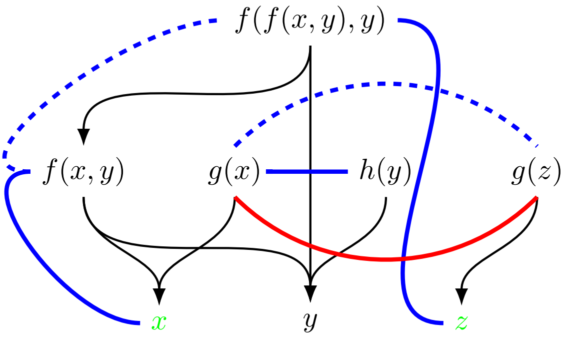


The conflict involves $g(x) \neq g(z)$.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

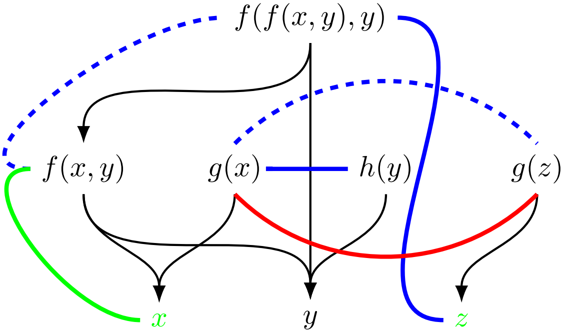


The conflict involves $g(x) \neq g(z)$.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

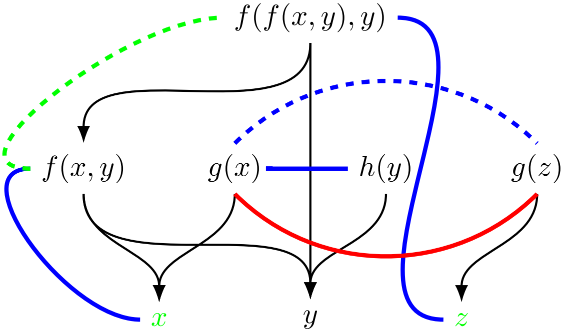


The conflict involves $g(x) \neq g(z)$, $x = f(x,y)$.

Producing congruence closure—example

We want to satisfy

$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$

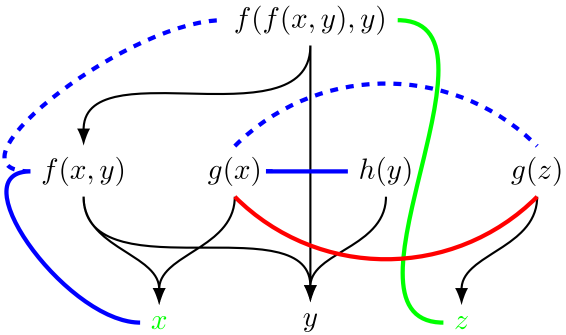


The conflict involves $g(x) \neq g(z)$, $x = f(x,y)$.

Producing congruence closure—example

We want to satisfy

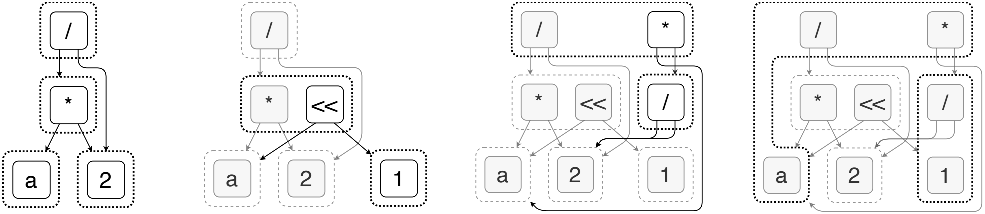
$$f(x,y) = x \quad \wedge \quad h(y) = g(x) \quad \wedge \quad f(f(x,y),y) = z \quad \wedge \quad g(x) \neq g(z).$$



The conflict involves $g(x) \neq g(z)$, $x = f(x,y)$, and $f(f(x,y),y) = z$.

Equality graphs (e-graphs)

Congruence closures in the form of equality graphs (e-graphs) do not appear only in theorem provers, but are used, for example, in rewrite-driven compiler optimizations and program synthesizers (using equality saturation). For example, the egg library provides high-performance, flexible e-graphs implemented in Rust.



(a) Initial e-graph contains $(a \times 2)/2$.

(b) After applying rewrite $x \times 2 \to x \ll 1$.

(c) After applying rewrite $(x \times y)/z \to x \times (y/z)$.

(d) After applying rewrites $x/x \to 1$ and $1 \times x \to x$.

Fig. 2. An e-graph consists of e-classes (dashed boxes) containing equivalent e-nodes (solid boxes). Edges connect e-nodes to their child e-classes. Additions and modifications are emphasized in black. Applying rewrites to an e-graph adds new e-nodes and edges, but nothing is removed. Expressions added by rewrites are merged with the matched e-class. In Figure 2d, the rewrites do not add any new nodes, only merge e-classes. The resulting e-graph has a cycle, representing infinitely many expressions: a, $a \times 1$, $a \times 1 \times 1$, and so on.

source: Willsey et al. 2021, p. 5

Difference logic

We have

$$x - y \bowtie k$$

where $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, x, y, and k (number) are over integers (QF_IDL) or reals (QF_RDL).

We can assume that all are of the form $x - y \leq k$, because

$$\begin{array}{lll} x - y \geq k & \text{is} & y - x \leq -k, \\ x - y = k & \text{is} & x - y \leq k \wedge y - x \leq -k, \\ x - y \neq k & \text{is} & x - y < k \vee y - x < -k, \\ x - y < k & \text{is} & x - y \leq k - 1, \qquad \text{for integers} \\ & & x - y \leq k - \delta, \qquad \text{for reals} \end{array}$$

where δ is treated on symbolic level (or a sufficiently small real).

Moreover, every solution can be shifted. Hence we can introduce a fresh variable y_0, replace all $x \leq k$ by $x - y_0 \leq k$ and shift a solution in such a way that y_0 becomes 0.

Why is difference logic interesting?

It is an important fragment of arithmetic. We can, for example, express a simple scheduling problem:

$$\begin{aligned} 1 &\leq s_a, \\ s_a &\leq 10, \\ s_a + 5 &\leq s_b, \\ s_b &\leq 10, \end{aligned}$$

where s_a and s_b express when tasks a and b start.

Clearly, using $\neq$ (and hence $\vee$) and integers, we can encode NP problems like a k-coloring of a graph $G = (V, E)$:

$$\begin{aligned} &1 \leq c_v \leq k && \text{for } v \in V, \\ &c_v \neq c_w && \text{for } (v, w) \in E. \end{aligned}$$

How to decide difference logic?

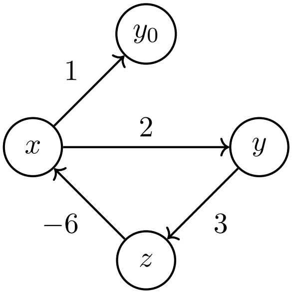
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$



How to decide difference logic?

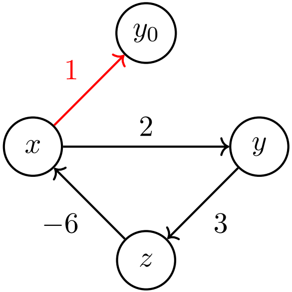
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$



We write $x - y_0 \leq 1$ instead, where y_0 should be equal to 0!

How to decide difference logic?

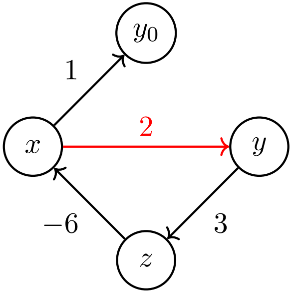
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \le 1 \land x - y \le 2 \land y - z \le 3 \land z - x \le -6$$



How to decide difference logic?

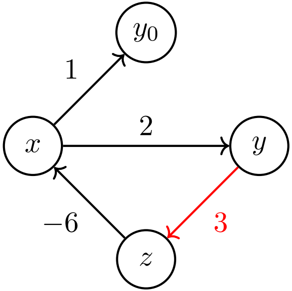
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$



How to decide difference logic?

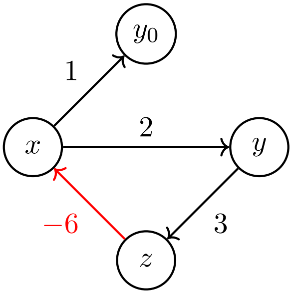
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$



How to decide difference logic?

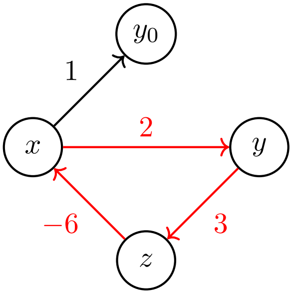
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$



This conflict set is communicated back to the SAT solver!

How to decide difference logic?

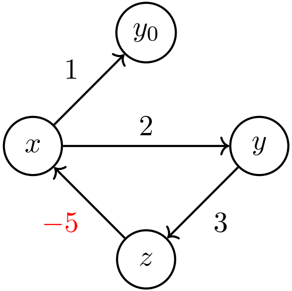
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -5$$



How to decide difference logic?

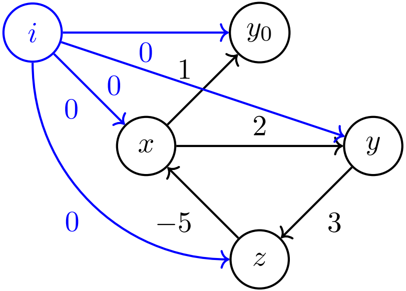
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \land x - y \leq 2 \land y - z \leq 3 \land z - x \leq -5$$



Solution$= -(\min$ path from $i)$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

How to decide difference logic?

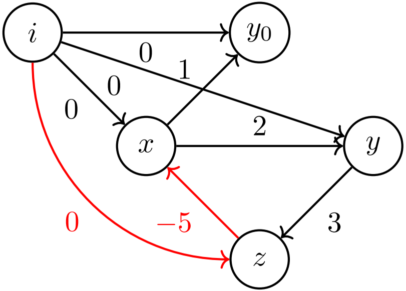
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -5$$



Solution$= -(\min$ path from $i)$

$x = 5$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

How to decide difference logic?

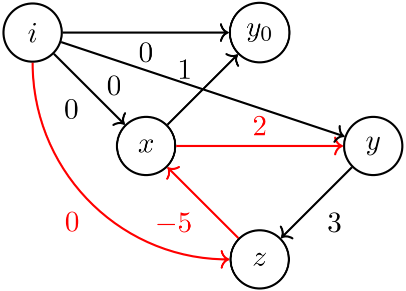
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -5$$



Solution$= -(\min$ path from $i)$

$x = 5$, $y = 3$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

How to decide difference logic?

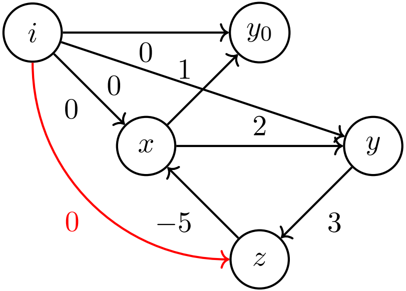
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -5$$



Solution$= -(\min$ path from $i)$

$x = 5$, $y = 3$, $z = 0$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

How to decide difference logic?

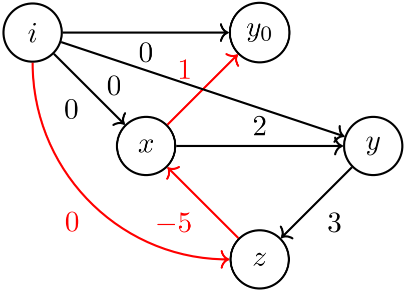
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \land x - y \leq 2 \land y - z \leq 3 \land z - x \leq -5$$



Solution$= -(\min$ path from $i)$

$x = 5$, $y = 3$, $z = 0$, $y_0 = 4$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

How to decide difference logic?

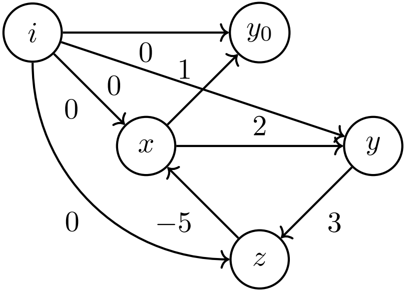
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -5$$



Solution$= -(\min$ path from $i)$

$x = 5,\ y = 3,\ z = 0,\ y_0 = 4$

Shift solution!

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

How to decide difference logic?

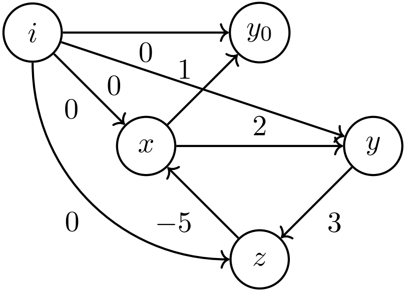
We represent a problem (i.e. a conjunction of literals) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -5$$



Solution$= -(\min$ path from $i)$

$x = 5$, $y = 3$, $z = 0$, $y_0 = 4$

Shift solution!

$x = 1$, $y = -1$, $z = -4$, $y_0 = 0$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$

Linear arithmetic

We have

$$a_1x_1 + \cdots + a_nx_n \bowtie b$$

where $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$, a_1 positive, and $x_1, \ldots, x_n$ are over integers (QF_LIA) or reals (QF_LRA).

We can again assume that we have only $\leq$ (if a_1 positive, then also $\geq$).

Although simplex is exponential (and LRA is in P), it is fast in practice.

For LIA (is NP-complete) simplex with branch-and-bound (cutting planes) is usually used.

For further details, see Kroening and Strichman 2016.

Bit-vectors

We have fixed-sized vectors of bits (QF_BV).

Various types of operations

- logical (bit-wise), e.g., and
- arithmetic, e.g., add
- comparisons, e.g., $<$
- string-like, e.g., concat

Note that we can eagerly translate them into propositional logic and use directly SAT (bit-blasting). Moreover, in many cases this produces better results. However, some operations, e.g., multiplication, produce hard SAT instances and other approaches are needed.

Floating Point numbers are bit-vectors based on the IEEE standard.

Arrays (A)

We have two basic operations on arrays

- $\mathrm{select}(a, i)$ is the value of array a at the possition i
- $\mathrm{store}(a, i, v)$ is the array a with v at the possition i

and equality is also a part of the language.

It satisfies the following read-over-write axioms:

$$\mathrm{select}(\mathrm{store}(a, i, v), i) = v$$
$$i \neq j \to \mathrm{select}(\mathrm{store}(a, i, v), j) = \mathrm{select}(a, j)$$

If we add the axiom of extensionality

$$\forall i(\mathrm{select}(a, i) = \mathrm{select}(b, i)) \to a = b,$$

then we obtain (QF_AX).

Arrays properties

Note that the size of a model depends on the number of accesses to memory and not on the size of the memory we model.

Computing $\mathcal{T}_A$ is NP-complete and hence in practice we usually treat select and store as uninterpreted functions and add instances of violated array axioms on demand.

SMT-LIB

Among other things it is a common input and output language for SMT solvers that is described here.

```
; Integer arithmetic
(set-logic QF_LIA)
(declare-fun x () Int)
(declare-fun y () Int)
(assert (= (- x y) (+ x (- y) 1)))
(check-sat)
; unsat
(exit)
```

Bibliography I

- Griggio, Alberto (2015). "Introduction to SMT". SAT/SMT Summer School 2015. URL: http://www.cs.nyu.edu/~barrett/summerschool/griggio.pdf.
- Jovanović, Dejan (2016). "Introduction to Satisfiability Modulo Theories". SAT/SMT/AR Summer School 2016. URL: http://ssa-school-2016.it.uu.se/wp-content/uploads/2016/06/jovanovic.pdf.
- Kroening, Daniel and Ofer Strichman (2016). *Decision Procedures - An Algorithmic Point of View, Second Edition*. Texts in Theoretical Computer Science. An EATCS Series. Springer. ISBN: 978-3-662-50496-3. DOI: 10.1007/978-3-662-50497-0.
- Oliveras, Albert (2019). "Introduction to SMT". SAT/SMT/AR Summer School 2019. URL: https://alexeyignatiev.github.io/ssa-school-2019/slides/ao-satsmtar19-slides.pdf.
- Tinelli, Cesare (2017). "Foundations of Satisfiability Modulo Theories". SC^2 Summer School 2017. URL: http://www.sc-square.org/CSA/school/lectures/SCSC-Tinelli.pdf.
- Willsey, Max et al. (2021). "Egg: Fast and Extensible Equality Saturation". In: *Proceedings of the ACM on Programming Languages* 5.POPL, pp. 1–29. DOI: 10.1145/3434304.