Logical reasoning and programming
First-Order Logic (FOL) and
Satisfiability Modulo Theories (SMT)

Karel Chvalovský

CIIRC CTU

Parts of this presentation are significantly based on materials from recent 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, the set of all terms in L is denoted $Term_L$,
- formulae in L,
- semantics
    - a model for L, denoted $\mathcal{M} = (D, i)$, and
    - Tarski's definition of truth.

Tarski's definition of truth

Let $\mathcal{M} = (D, i)$ be a model for L, e be an evaluation in $\mathcal{M}$, then we say that a formula φ is satisfied in $\mathcal{M}$ by e, denoted $\mathcal{M} \models \varphi[e]$, or e satisfies φ in $\mathcal{M}$, if

- $\mathcal{M} \models p(t_1, \ldots, t_n)[e]$ iff $(t_1^{\mathcal{M}}[e], \ldots, t_n^{\mathcal{M}}[e]) \in i(p)$, where p is n-ary predicate symbol in L,
- $\mathcal{M} \models (t_1 = t_2)[e]$ iff $(t_1^{\mathcal{M}}[e], t_2^{\mathcal{M}}[e]) \in \mathrm{id}_D$, (in FOL with eq.)
- $\mathcal{M} \models (\neg\psi)[e]$ iff $\mathcal{M} \not\models \psi[e]$,
- $\mathcal{M} \models (\psi \to \chi)[e]$ iff $\mathcal{M} \not\models \psi[e]$ or $\mathcal{M} \models \chi[e]$,
- $\mathcal{M} \models (\psi \wedge \chi)[e]$ iff $\mathcal{M} \models \psi[e]$ and $\mathcal{M} \models \chi[e]$,
- $\mathcal{M} \models (\psi \vee \chi)[e]$ iff $\mathcal{M} \models \psi[e]$ or $\mathcal{M} \models \chi[e]$,
- $\mathcal{M} \models (\forall X\psi)[e]$ iff for every $a \in D$ holds $\mathcal{M} \models \psi[e(X \mapsto a)]$,
- $\mathcal{M} \models (\exists X\psi)[e]$ iff exists $a \in D$ s.t. $\mathcal{M} \models \psi[e(X \mapsto a)]$.

A formula φ is satisfiable, if there is $\mathcal{M}$ and e s.t. $\mathcal{M} \models \varphi[e]$. A set of formulae Γ is satisfiable, if there is $\mathcal{M}$ and e s.t. $\mathcal{M} \models \varphi[e]$, for every $\varphi \in \Gamma$.

Semantic consequence relation

A formula φ is valid (or holds) in $\mathcal{M}$, denoted $\mathcal{M} \models \varphi$, if φ is satisfied in $\mathcal{M}$ by any evaluation e.

A formula φ follows from (or is a consequence of) a set of formula Γ, denoted $\Gamma \models \varphi$, if and only if for any model $\mathcal{M}$ and evaluation e, if for every $\psi \in \Gamma$ holds $\mathcal{M} \models \psi[e]$, then $\mathcal{M} \models \varphi[e]$. We write $\models \varphi$, if $\Gamma = \emptyset$ and say that φ is valid (or holds).

$$\Gamma \models \varphi \quad \text{iff} \quad \forall \mathcal{M} \forall e (\forall \psi \in \Gamma (\mathcal{M} \models \psi[e]) \Rightarrow \mathcal{M} \models \varphi[e])$$

Note that

$$\Gamma \models \varphi \quad \text{iff} \quad \Gamma \cup \{\neg \varphi\} \text{ is unsatisfiable.}$$

We say that two formulae φ and ψ are (semantically) equivalent, denoted $\varphi \equiv \psi$, if $\{\varphi\} \models \psi$ and $\{\psi\} \models \varphi$.

SMT and FOL language

It is common in SMT that we are mainly interested in formulae containing no variables (and hence no quatifiers), they are called ground formulae.

Note that when it comes to satisfiability, uninterpreted constant symbols behave like free variables; they are not bounded by quantification (nor implicitly). However, this can be slightly misleading, because later on all variables will be implicitly universally quantified.

Strictly speaking, we should talk about expansions of a theory, because we add new constants into our language, however, we will happily ignore this formal problem (or we can treat them as free variables).

Example

$\varphi = p(X, f(a, Y)) \land q(X, Z, c)$ is not ground, but $\varphi' = p(x, f(a, y)) \land q(x, z, c)$ is ground. Moreover, φ and φ' are equisatisfiable.

Why are we interested in this fragment of FOL?

For example, say that a compiler produced from

$$z = (x_1 + y_1) \cdot (x_2 + y_2);$$

the following code

$$u_1 = x_1 + y_1;$$
$$u_2 = x_2 + y_2;$$
$$z = u_1 \cdot u_2;$$

So we want to show that this translation is correct by proving

$$(u_1 = x_1 + y_1) \wedge (u_2 = x_2 + y_2) \wedge (z = u_1 \cdot u_2)$$
$$\rightarrow (z = (x_1 + y_1) \cdot (x_2 + y_2)).$$

Clearly, our fragment is sufficient for that.

Interpretations and theories

When we speak about theories, it means that we want to restrict the interpretation (meaning) of some symbols in the language.

There are two main approaches how to do that

▶ axiomatic — we restrict the interpretations indirectly by providing axioms that have to be satisfied,
  ▶ e.g., the equality axioms,
  ▶ axioms usually require quantifiers,
▶ restricting interpretations — we allow only such classes of interpretations that correspond to our intended meaning,
  ▶ e.g., we say that all the variables range over integers.

Note that some theories do not have appropriate axiomatic systems.

Theory $\mathcal{T}$

A theory $\mathcal{T}$ is given by a first-order language 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 interpretations 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.

Example

If $\mathcal{T}$ is real arithmetic, then D are real numbers and $i(\leq)$, $i(+)$,
. . . have their standard meanings.

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

$$(x = 0 \lor x = 1) \land (x + y + z \neq 0) \land (f(y) > f(z)),$$

we obtain

$$(p \lor q) \land \neg r \land s,$$

by so-called propositional abstraction, where p is $x = 0$, q is $x = 1$, r is $x + y + z = 0$, and s is $f(y) > f(z)$.

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

$$\neg a = b$$
$$(x = a \vee x = b)$$
$$(y = a \vee y = b)$$
$$(z = a \vee z = b)$$
$$\neg x = y$$

Very Lazy SMT
example

$$\neg B_1$$
$$(\ B_2\ \lor\ B_3\ )$$
$$(\ B_4\ \lor\ B_5\ )$$
$$(\ B_6\ \lor\ B_7\ )$$
$$\neg B_8$$

Very Lazy SMT
example

$$\neg \ B_1$$
$$(\ B_2 \ \lor \ B_3 \ )$$
$$(\ B_4 \ \lor \ B_5 \ )$$
$$(\ B_6 \ \lor \ B_7 \ )$$
$$\neg \ B_8$$

Check with SAT solver

Very Lazy SMT
example

> **View**
> Boolean

$$\neg\ B_1$$
$$(\ B_2\ \vee\ B_3\ )$$
$$(\ B_4\ \vee\ B_5\ )$$
$$(\ B_6\ \vee\ B_7\ )$$
$$\neg\ B_8$$

Check with SAT solver

$[\![\ \neg\ B_1\ ,\ \neg\ B_8\ ,\ \neg\ B_3\ ,\ B_2\ ,\ \neg\ B_5\ ,\ B_4\ ,\ \neg\ B_7\ ,\ B_6\ ]\!]$

Very Lazy SMT
example

$$\neg a = b$$
$$(x = a \lor x = b)$$
$$(y = a \lor y = b)$$
$$(z = a \lor z = b)$$
$$\neg x = y$$

Check with SAT solver

$\llbracket \neg a = b,\ \neg x = y,\ \neg x = b,\ x = a,\ \neg y = b,\ y = a,\ \neg z = b,\ z = a \rrbracket$

Very Lazy SMT
example

View

Theory

$$\neg a = b$$
$$(x = a \lor x = b)$$
$$(y = a \lor y = b)$$
$$(z = a \lor z = b)$$
$$\neg x = y$$

Check with T-solver
$x = a \land y = a \Rightarrow x = y$

Check with SAT solver
$[\![\neg a = b,\ \neg x = y,\ \neg x = b,\ x = a,\ \neg y = b,\ y = a,\ \neg z = b,\ z = a]\!]$

Very Lazy SMT
example

$$\neg a = b$$
$$(x = a \lor x = b)$$
$$(y = a \lor y = b)$$
$$(z = a \lor z = b)$$
$$\neg x = y$$
$$(x = y \lor \neg x = a \lor \neg y = a)$$

Block

Add clause

Check with SAT solver

$[\![\neg a = b, \ \neg x = y, \ \neg x = b, \ x = a, \ \neg y = b, \ y = a, \ \neg z = b, \ z = a]\!]$

Very Lazy SMT
example

$$\neg\ B_1$$
$$(\ B_2\ \vee\ B_3\ )$$
$$(\ B_4\ \vee\ B_5\ )$$
$$(\ B_6\ \vee\ B_7\ )$$
$$\neg\ B_8$$
$$(\ B_8\ \vee\neg\ B_2\ \vee\neg\ B_4\ )$$

Check with SAT solver

Very Lazy SMT
example

$$\neg B_1$$
$$(\ B_2\ \vee\ B_3\ )$$
$$(\ B_4\ \vee\ B_5\ )$$
$$(\ B_6\ \vee\ B_7\ )$$
$$\neg B_8$$
$$(\ B_8\ \vee \neg\ B_2\ \vee \neg\ B_4\ )$$

Check with SAT solver

$$[\![\ \neg\ B_1\ ,\ \neg\ B_8\ ,\ \neg\ B_3\ ,\ B_2\ ,\ \neg\ B_4\ ,\ B_5\ ,\ \neg\ B_7\ ,\ B_6\ ]\!]$$

Very Lazy SMT
example

$$\neg a = b$$
$$(x = a \vee x = b)$$
$$(y = a \vee y = b)$$
$$(z = a \vee z = b)$$
$$\neg x = y$$
$$(x = y \vee \neg x = a \vee \neg y = a)$$

Check with SAT solver

$\llbracket \neg a = b,\ \neg x = y,\ \neg x = b,\ x = a,\ \neg y = a,\ y = b,\ \neg z = b,\ z = a \rrbracket$

Very Lazy SMT
example

Check with T-solver

Satisfiable

$a, x, z \mapsto c_1$

$b, y \mapsto c_2$

$$\neg a = b$$
$$(x = a \vee x = b)$$
$$(y = a \vee y = b)$$
$$(z = a \vee z = b)$$
$$\neg x = y$$
$$(x = y \vee \neg x = a \vee \neg y = a)$$

Check with SAT solver

$[\![\neg a = b, \ \neg x = y, \ \neg x = b, \ x = a, \ \neg y = a, \ y = b, \ \neg z = b, \ z = a]\!]$

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(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\parallel\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad \Rightarrow \quad (\mathsf{UnitPropagate})$$

DPLL(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\parallel\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\textsf{UnitPropagate})$$

$$1 \;\parallel\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\textsf{UnitPropagate})$$

DPLL(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{UnitPropagate})$$

$$1 \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{UnitPropagate})$$

$$1\,\overline{4} \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{T-Propagate})$$

DPLL(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\|\; 1,\; \overline{2} \vee 3,\; \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate})$$

$$1 \;\|\; 1,\; \overline{2} \vee 3,\; \overline{4} \quad \Rightarrow \quad (\text{UnitPropagate})$$

$$1\,\overline{4} \;\|\; 1,\; \overline{2} \vee 3,\; \overline{4} \quad \Rightarrow \quad (\text{T-Propagate})$$

$$1\,\overline{4}\,2 \;\|\; 1,\; \overline{2} \vee 3,\; \overline{4} \quad \Rightarrow \quad (\text{T-Propagate})$$

DPLL(T) - Example

Consider again EUF and the formula:

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

$$\emptyset \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{UnitPropagate})$$

$$1 \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{UnitPropagate})$$

$$1\,\overline{4} \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{T-Propagate})$$

$$1\,\overline{4}\,2 \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{T-Propagate})$$

$$1\,\overline{4}\,2\,\overline{3} \;\|\; 1,\; \overline{2}\vee 3,\; \overline{4} \quad\Rightarrow\quad (\text{Fail})$$

DPLL(T) - Example

Consider again EUF and the formula:

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

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

UNSAT

DPLL(T) Framework
great but not perfect

Example (Diamonds)

$$a_0 > a_n \wedge \bigwedge_{k=0}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$$

DPLL(T) Framework

great but not perfect

Example (Diamonds)

$$a_0 > a_n \wedge \bigwedge_{k=0}^{n-1} \left((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}) \right)$$

DPLL(T) Framework

great but not perfect

Example (Diamonds)

$$a_0 > a_n \wedge \bigwedge_{k=0}^{n-1} \left((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}) \right)$$

DPLL(T) Framework

great but not perfect

Example (Diamonds)

$$a_0 > a_n \wedge \bigwedge_{k=0}^{n-1} ((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}))$$

DPLL(T) Framework
great but not perfect

And so on...

Exponential enumeration of paths.

Example (D

$$a_0 > a_n \wedge \bigwedge_{k=0}^{n-1} \left((a_k < b_k \wedge b_k < a_{k+1}) \vee (a_k < c_k \wedge c_k < a_{k+1}) \right)$$

DPLL(T) Framework
great but not perfect
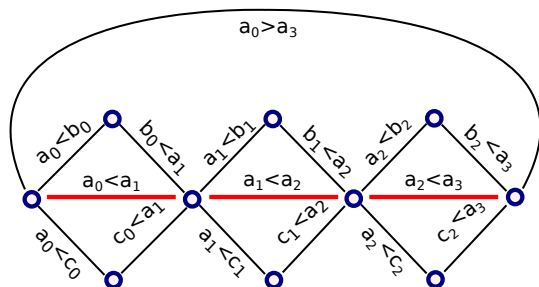
Example (Diamonds)

$$a_0 > a_n \land \bigwedge_{k=0}^{n-1} ((a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1}))$$
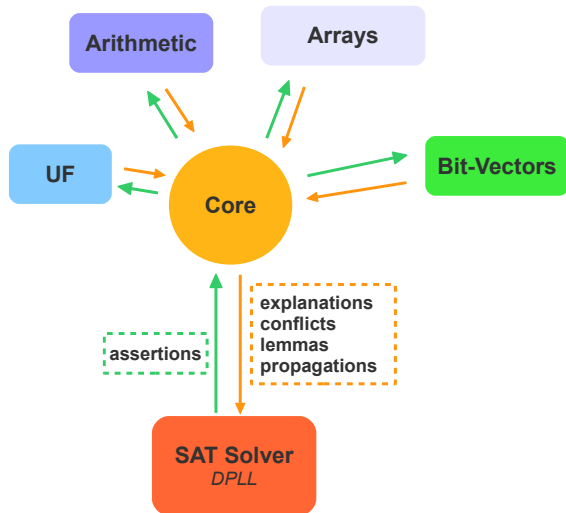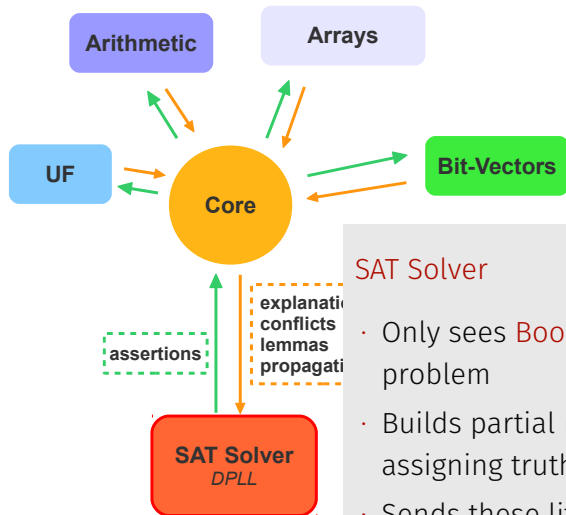
SAT Solver

· Only sees Boolean skeleton of problem
· Builds partial model by assigning truth values to literals
· Sends these literals to the core as assertions

Core

· Sends each assertion to the appropriate theory

· Sends deduced literals to other theories/SAT solver

· Handles theory combination
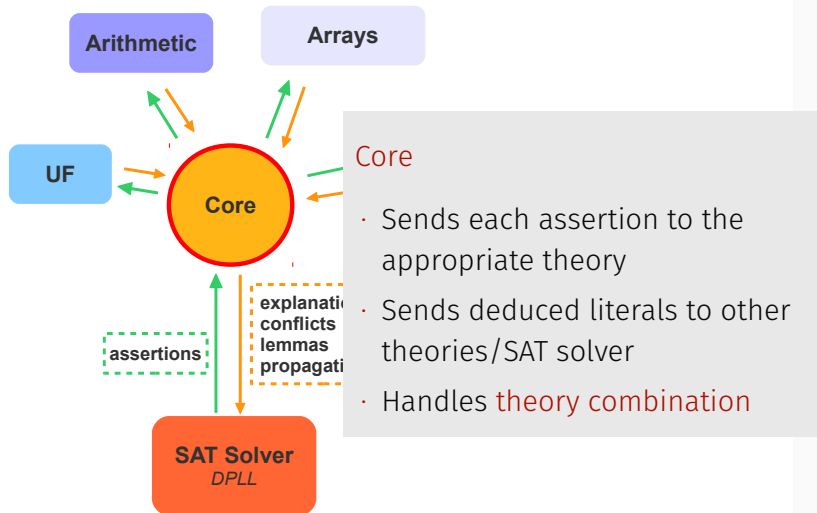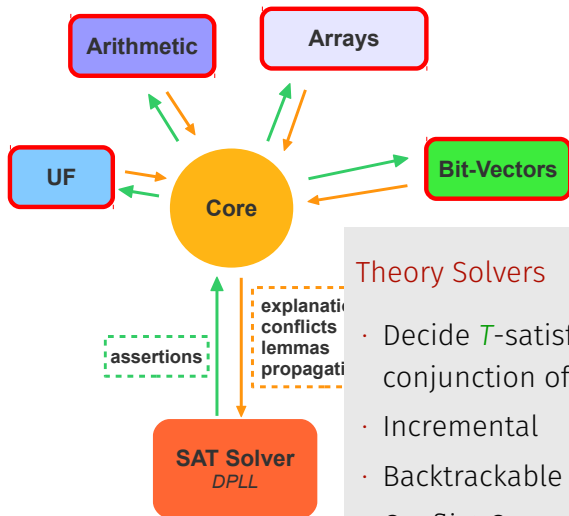
Theory Solvers

· Decide T-satisfiability of a conjunction of theory literals
· Incremental
· Backtrackable
· Conflict Generation
· Theory Propagation

Various theories and their combinations

It is possible to combine various theories, called logics here, and give them "canonical" names.
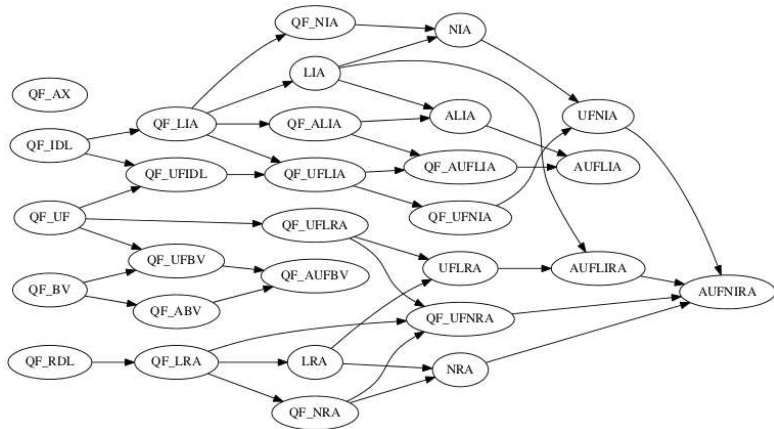
source: SMT-LIB

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

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

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

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

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

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

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

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

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

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

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.

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

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

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

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

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

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

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

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

get_conflict():

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

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

get_conflict():

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

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

get_conflict():

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

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

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

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

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

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

get_conflict():

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

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

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

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

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

get_conflict():

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

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

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

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:

$$1 \leq s_a,$$
$$s_a \leq 10,$$
$$s_a + 5 \leq s_b,$$
$$s_b \leq 10,$$

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 $G = (V, E)$:

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

How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem

Satisfiable iff there is no negative cycle.

Example
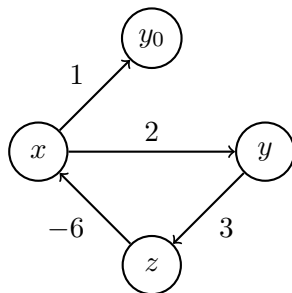
$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$
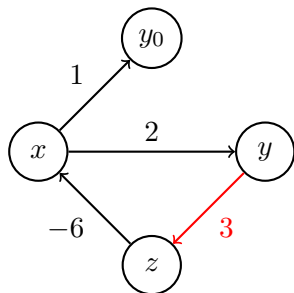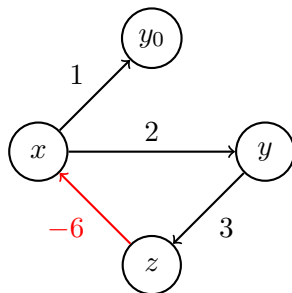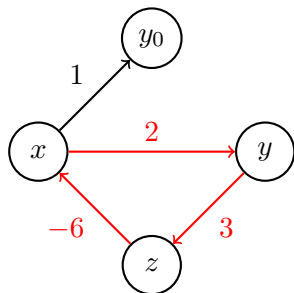
How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example
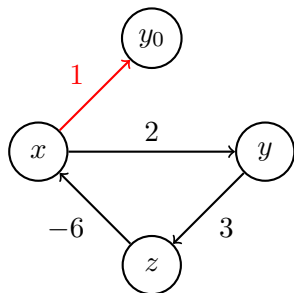
$$x \leq 1 \land x - y \leq 2 \land y - z \leq 3 \land z - x \leq -6$$

y_0 should be equal to 0!

How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example

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

How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example

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

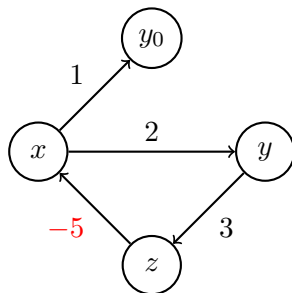# How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \wedge x - y \leq 2 \wedge y - z \leq 3 \wedge z - x \leq -6$$
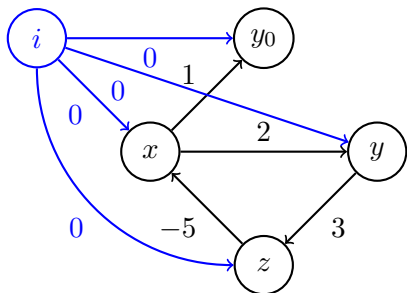
How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example

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

This conflict set is communicated back to the SAT solver!

How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example

$$x \leq 1 \land x - y \leq 2 \land y - z \leq 3 \land z - x \leq -5$$
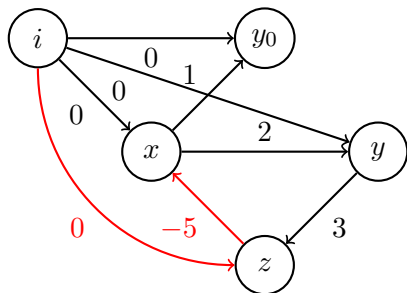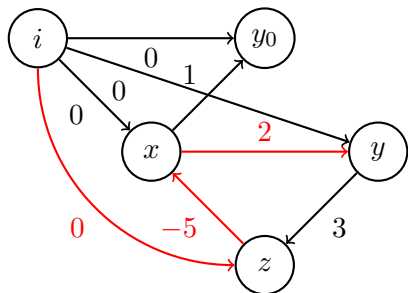
How to decide difference logic?

We represent a problem (conjunction) 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 $i)$

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

How to decide difference logic?

We represent a problem (conjunction) by a graph.

Theorem
Satisfiable iff there is no negative cycle.

Example

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

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

$x = 5$

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$
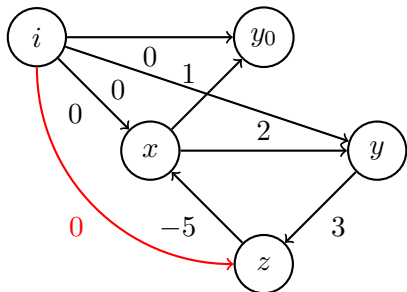
How to decide difference logic?

We represent a problem (conjunction) 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 $\mathcal{O}(|V| \cdot |E|)$
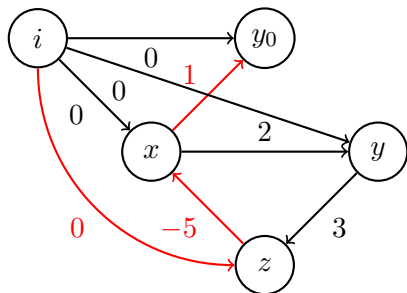
How to decide difference logic?

We represent a problem (conjunction) 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 (conjunction) 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 $i)$

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

Bellman–Ford in $\mathcal{O}(|V| \cdot |E|)$
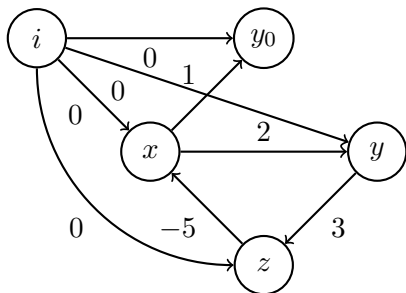
How to decide difference logic?

We represent a problem (conjunction) 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 $\mathcal{O}(|V| \cdot |E|)$
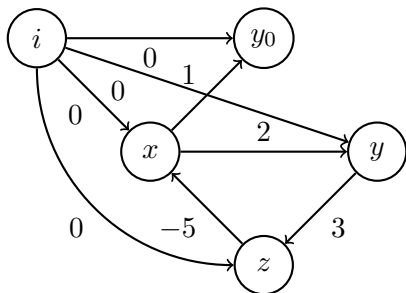
How to decide difference logic?

We represent a problem (conjunction) 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

$$a_1 x_1 + \cdots + a_n x_n \bowtie b$$

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

▶ $\text{select}(a, i)$ is the value of array a at the possition i

▶ $\text{store}(a, i, v)$ 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:

$$\text{select}(\text{store}(a, i, v), i) = v$$
$$i \neq j \rightarrow \text{select}(\text{store}(a, i, v), j) = \text{select}(a, j)$$

If we add the axiom of extensionality

$$\forall i (\text{select}(a, i) = \text{select}(b, i)) \rightarrow a = b,$$

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)
```

Used presentations

The following slides are taken from

- ▶ 10 and 14 are from Jovanović 2016,
- ▶ 13 is from Oliveras 2019,
- ▶ 15 is from Tinelli 2017, and
- ▶ 22 is from Griggio 2015.

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.

Bibliography II

📄 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.