Logical reasoning and programming
Complexity and decidability issues, SMT

Karel Chvalovský

CIIRC CTU

This presentation is significantly based on materials from recent SAT/SMT Summer Schools and SC² Summer School.
Limits of formal methods

Two famous fundamental theoretical problems are:

Incompleteness
Gödel’s first incompleteness theorem says that it is impossible to describe the basic arithmetic of the natural numbers by a set of axioms that is algorithmically recognizable.

Undecidability (unsolvability)
Church and Turing famously proved that there is no decision procedure (algorithm) for validity in first-order logic (FOL). For example the halting problem is expressible in FOL.

Turing 1950: Computing machinery and intelligence
“We may hope that machines will eventually compete with men in all purely intellectual fields.”
Decidable fragments of FOL

We can restrict the expressive power of FOL and obtain decidable (satisfiability) fragments, for example:

- monadic fragment
  - predicate and function symbols have at most one argument
- quantifier prefixes with arbitrary predicate symbols and no function symbols
  - $\exists^*\forall^*$,
  - $\exists^*\forall\exists^*$,
  - $\exists^*\forall\exists^*$ (without equality)
- quantifier prefixes with arbitrary predicate and function symbols
  - $\exists^*$,
  - $\exists^*\forall\exists^*$ (without equality)

There are other examples like we can restrict the number $k$ of variables in formulae, denoted $\text{FO}^k$, if $k \leq 2$, then decidable (without equality and function symbols).
Bernays–Schönfinkel–Ramsey class (EPR)

It is a fragment of FOL where satisfiability is decidable (\textsc{NEXPTIME}-complete).

They are formulae without function symbols and with a quantifier prefix

\[ \exists^* \forall^* \]

in a prenex normal form.

Because it is possible to easily obtain all possible instances (use constants), it is also called effectively propositional (EPR).
Interpretations and theories

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

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

- restricting interpretations — we allow only such classes of interpretations that correspond to our intended meaning
  - e.g., direct approach to equalities through paramodulation, saturation, ... 

Note that some theories do not have appropriate axiomatic systems.
(Un)decidable theories

If we have addition, multiplication, and equality, then the first-order theory of

- $\mathbb{N}$ is undecidable,
- $\mathbb{Q}$ is undecidable,
- $\mathbb{R}$ is decidable (if we add, e.g., $\sin$ or $\exp$, then it is not)

If we have only addition and equality, then $\mathbb{N}$ is decidable (Presburger arithmetic). It is also decidable, if we have only multiplication and equality.

Hilbert’s tenth problem is undecidable — Diophantine equations (polynomial equations with integer coefficients and only integer solutions).
Theory

A theory $\mathcal{T}$ is given by a 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 $\varphi$ 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 $\Gamma$ $\mathcal{T}$-entails a formula $\varphi$, denoted $\Gamma \models_\mathcal{T} \varphi$, if every $\mathcal{T}$-interpretation satisfying all formulae in $\Gamma$ satisfies also $\varphi$.

From now on, we are only interested in ground formulae as is common in SMT; hence quantifier-free. Note that an uninterpreted constant symbol behaves like a free variable.

Strictly speaking, we should talk about expansions of $\mathcal{T}$, because we have new constants in our language, however, we will happily ignore this formal problem (or we can treat them as free variables).
Satisfiability Modulo Theories (SMT)—example

We would like to know whether the following formula

\[ f(a) = b \land f(a) \neq b \]

is satisfiable; that is loosely speaking that there exists a solution.
We would like to know whether the following formula

\[ f(a) = b \land f(a) \neq b \]

is satisfiable; that is loosely speaking that there exists a solution.

Clearly, it is unsatisfiable regardless of the meaning of symbols in it, because it has a propositional form

\[ p \land \neg p, \]

where \( p \) is \( f(a) = b \).
We would like to know whether the following formula

\[(a \times (f(b) + f(c)) = d) \land (b \times (f(a) + f(c)) \neq d) \land (a = b)\]

is satisfiable.

Do we need to know the interpretations of \(\times, +,\) and \(f\)?
Satisfiability Modulo Theories (SMT)—example

We would like to know whether the following formula

$$(a \times (f(b) + f(c)) = d) \land (b \times (f(a) + f(c)) \neq d) \land (a = b)$$

is satisfiable.

Do we need to know the interpretations of $\times$, $+$, and $f$? No, it is unsatisfiable modulo the theory of equality.
We would like to know whether the following formula

\[(x = 0 \lor x = 1) \land (x + y + z \neq 0) \land (f(y) > f(z))\]

is satisfiable.
Satisfiability Modulo Theories (SMT)—example

We would like to know whether the following formula

\[(x = 0 \lor x = 1) \land (x + y + z \neq 0) \land (f(y) > f(z))\]

is satisfiable.

It is satisfiable in a union of theories called unquantified linear real arithmetic with uninterpreted sort and function symbols (QF_UFLRA):

- quantifier free (QF),
- linear real arithmetic (LRA),
- uninterpreted functions (UF).
Satisfiability Modulo Theories (SMT)

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

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

It is very common that we have theory solvers for problems that are conjunctions of literals. Hence there are two approaches how to solve a satisfiability of a formula $\varphi$ modulo a theory $\mathcal{T}$.

**Eager**

We can transform $\varphi$ into a DNF, and then call a solver for $\mathcal{T}$ on every conjunction of literals we obtain. If any of them is satisfiable, then the whole formula is satisfiable. Unfortunately, like in SAT, a DNF can be big and hence requires many calls to the solver for $\mathcal{T}$.

**Lazy**

Hence we usually use a different approach. Let $\varphi'$ be a propositional abstraction of $\varphi$. We call a SAT solver on $\varphi'$ and obtain a set of literals $l'_1, \ldots, l'_n$ in $\varphi'$ that satisfy the formula (or $\varphi'$ is unsatisfiable). We can then ask the solver for $\mathcal{T}$, whether the set of corresponding literals $l_1, \ldots, l_n$ in $\varphi$ is satisfiable in $\mathcal{T}$. If not, then we can add a new propositional clause $\overline{l'_1} \lor \cdots \lor \overline{l'_n}$ to $\varphi'$ and repeat the whole process with a new propositional formula.
\neg a = b
\rightarrow
(x = a \lor x = b)
(y = a \lor y = b)
(z = a \lor z = b)
\neg x = y

Check with SAT solver
View Theory
¬ B₁
( B₂ ∨ B₃ )
( B₄ ∨ B₅ )
( B₆ ∨ B₇ )
¬ B₈
\[ \neg B_1 \]
\[ \neg B_2 \lor B_3 \]
\[ \neg B_4 \lor B_5 \]
\[ \neg B_6 \lor B_7 \]
\[ \neg B_8 \]

Check with SAT solver
Very Lazy SMT

every
d

¬ B₁

( B₂ ∨ B₃ )

( B₄ ∨ B₅ )

( B₆ ∨ B₇ )

¬ B₈

Check with SAT solver

[ ¬ B₁, ¬ B₈, ¬ B₃, B₂, ¬ B₅, B₄, ¬ B₇, B₆ ]
\[ \neg a = b \]
\[ (x = a \lor x = b) \]
\[ (y = a \lor y = b) \]
\[ (z = a \lor z = b) \]
\[ \neg x = y \]

Check with SAT solver

\[ [ \neg a = b, \neg x = y, \neg x = b, x = a, \neg y = b, y = a, \neg z = b, z = a ] \]
Very Lazy SMT

eexample

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

\[
\neg a = b \\
(x = a \lor x = b) \\
(y = a \lor y = b) \\
(z = a \lor z = b) \\
\neg x = y \\
(x = y \lor \neg x = a \lor \neg y = a)
\]

Check with SAT solver

\[
[ \neg a = b, \neg x = y, \neg x = b, x = a, \neg y = b, y = a, \neg z = b, z = a ]
\]
\[
\neg B_1 \\
(B_2 \lor B_3) \\
(B_4 \lor B_5) \\
(B_6 \lor B_7) \\
\neg B_8 \\
(B_8 \lor \neg B_2 \lor \neg B_4)
\]
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 \\
( B_8 \lor \neg B_2 \lor \neg B_4 )

Check with SAT solver

\llbracket \neg B_1 , \neg B_8 , \neg B_3 , B_2 , \neg B_4 , B_5 , \neg B_7 , B_6 \rrbracket
Very Lazy SMT

deexample

\[ \neg a = b \]
\[ (x = a \lor x = b) \]
\[ (y = a \lor y = b) \]
\[ (z = a \lor z = b) \]
\[ \neg x = y \]
\[ (x = y \lor \neg x = a \lor \neg y = a) \]

Check with SAT solver

\[ [ \neg a = b, \neg x = y, \neg x = b, x = a, \neg y = a, y = b, \neg z = b, z = a ] \]
Very Lazy SMT
example

Check with $T$-solver

Satisfiable

$a, x, z \mapsto c_1$
$b, y \mapsto c_2$

\[
\neg a = b \\
(x = a \lor x = b) \\
(y = a \lor y = b) \\
(z = a \lor z = b) \\
\neg x = y \\
(x = y \lor \neg x = a \lor \neg y = a)
\]

Check with SAT solver

\[
[\neg a = b, \neg x = y, \neg x = b, x = a, \neg y = a, y = b, \neg z = b, z = a]
\]
DPLL(\(\mathcal{T}\))

It effectively transforms a satisfiability of an arbitrary quantifier-free formula over \(\mathcal{T}\) to a satisfiability of a conjunction of literals over \(\mathcal{T}\).

For efficiency reasons we want several things from a solver for \(\mathcal{T}\):
- incremental,
- backtrackable,
- produces explanations (ideally minimal),
- (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.
Example (Diamonds)

\[ a_0 > a_n \land \bigwedge_{k=0}^{n-1} ((a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1})) \]
DPLL(T) Framework

great but not perfect

Example (Diamonds)

\[ a_0 > a_n \land \bigwedge_{k=0}^{n-1} ((a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1})) \]
Example (Diamonds)

\[ a_0 > a_n \land \bigwedge_{k=0}^{n-1} \left( (a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1}) \right) \]
DPLL(T) Framework

great but not perfect

Example (Diamonds)

\[ a_0 > a_n \land \bigwedge_{k=0}^{n-1} ((a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1})) \]
DPLL(T) Framework

great but not perfect

Example (Diamonds)

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

And so on...

Exponential enumeration of paths.
DPLL(T) Framework

great but not perfect

Example (Diamonds)

\[ a_0 > a_n \land \bigwedge_{k=0}^{n-1} ((a_k < b_k \land b_k < a_{k+1}) \lor (a_k < c_k \land c_k < a_{k+1})) \]
We already know that the only predicate symbol we need is equality, because every other uninterpreted predicate symbol can be expressed by a fresh uninterpreted function

\[
p(t_1, \ldots, t_n) \text{ becomes } f_p(t_1, \ldots, t_n) = \top, \\
\neg p(t_1, \ldots, t_n) \text{ becomes } f_p(t_1, \ldots, t_n) \neq \top,
\]

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

**Example**

\[
p(a) \lor \neg q(a, g(a, b)) \text{ becomes } f_p(a) = \top \lor f_q(a, g(a, b)) \neq \top.
\]
DPLL(T) Framework

typical architecture
Uninterpreted functions (UF)

We have literals of the form

\[ s = t \text{ and } s \neq t, \]

where equality is reflexive, symmetric, transitive, and satisfies congruence axioms

\[ \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n (x_1 = y_1 \land \cdots \land x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)), \]

\[ \forall x_1 \ldots \forall x_m \forall y_1 \ldots \forall y_m (x_1 = y_1 \land \cdots \land x_m = y_m \rightarrow (p(x_1, \ldots, x_m) \rightarrow p(y_m, \ldots, y_m))). \]

for every \( n \)-ary function symbol \( f \) and \( m \)-ary predicate symbol \( p \).

(QF_UF) is usually the core of a SMT solver, which is used in other theories. It is decidable by the congruence closure in \( \mathcal{O}(n \log n) \).
Example

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

\[(f(x, y) = x), (h(y) = g(x)), (f(f(x, y), y) = z), \neg(g(x) = g(z))]\]
\[ \left( f(x, y) = x \right), \left( h(y) = g(x) \right), \left( f(f(x, y), y) = z \right), \neg (g(x) = g(z)) \]
Example

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

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

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

get_conflict():

\[\neg(g(x) = g(z))\]
Example

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

get_conflict():
\[\neg(g(x) = g(z))\]
Example

\[(f(x, y) = x), (h(y) = g(x)), (f(f(x, y), y) = z), \neg(g(x) = g(z))\]\n
get_conflict():

\[\neg(g(x) = g(z))\]
Example

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

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

get_conflict():

\[\neg(g(x) = g(z))\]

\[(f(f(x, y), y) = z)\]
Example

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

get_conflict():

\[-(g(x) = g(z))

(f(f(x, y), y) = z)\]
Example

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

get\_conflict():

\[\neg (g(x) = g(z))\]

\[(f(f(x, y), y) = z)\]

\[(f(x, y) = x)\]
Example

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

get\_conflict():

\[\neg (g(x) = g(z))\]
\[(f(f(x, y), y) = z)\]
\[(f(x, y) = x)\]
Difference logic

We have

\[ x - y \triangleright k \]

where \( \triangleright \in \{\leq, <, =, \neq, >, \geq\} \), \( x \) and \( y \) are variables and \( k \) a constant over integers (QF_IDL) or reals (QF_RDL).

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

\[ x - y = k \text{ using } x - y \leq k \land y - x \leq -k \]
\[ x - y < k \text{ using } x - y \leq k - 1 \quad \text{for integers} \]
\[ x - y \leq k - \delta \quad \text{for reals} \]

where \( \delta \) is treated on symbolic level.
1. Construct a graph from literals
2. Check if there is a negative path

**Theorem** *Literals unsatisfiable $\iff \exists$ negative path*

**Example**

$$[x \leq 1, \ x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6]$$
1. Construct a graph from literals
2. Check if there is a negative path

Theorem *Literals unsatisfiable* $\iff \exists$ *negative path*

Example

$$[x \leq 1, \ x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6]$$
1. Construct a graph from literals
2. Check if there is a negative path

**Theorem** \( \text{Literals unsatisfiable} \iff \exists \text{ negative path} \)

**Example**

\[
[ x \leq 1, \ x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6 ]
\]
1. Construct a graph from literals
2. Check if there is a negative path

**Theorem** *Literals unsatisfiable* \(\iff\exists\) *negative path*

**Example**

\[
\begin{align*}
&x \leq 1, \ x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6
\end{align*}
\]
1. Construct a graph from literals
2. Check if there is a negative path

**Theorem** *Literals unsatisfiable* $\iff \exists$ negative path

**Example**

\[
[x \leq 1, \ x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6]
\]
1. Construct a graph from literals
2. Check if there is a negative path

**Theorem** *Literal unsatisfiable* $\iff \exists$ *negative path*

**Example**

$$[ x \leq 1, \, x - y \leq 2, \, y - z \leq 3, \, z - x \leq -6 ]$$
1. Construct a graph from literals
2. Check if there is a negative path

**Theorem** *Literals unsatisfiable* ⇔ ∃ negative path

**Example**

\[
[ x \leq 1, \ x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6 ]
\]

**Conflict Set:**

\[
x - y \leq 2, \ y - z \leq 3, \ z - x \leq -6
\]
Linear arithmetic

We have

\[ a_1 x_1 + \cdots + a_n x_n \triangleleft b \]

where \( \triangleleft \in \{\leq, <, =, \neq, >, \geq\} \), \( a_1 \) positive, and \( x_1, \ldots, x_n \) are variables over integers (QF_LIA) or reals (QF_LRA). We can again assume that we have only \( \leq \).

Although simplex is exponential (and LRA is in \( \mathbb{P} \)), it is fast in practice.

For LIA (is \( \mathbb{NP} \)-complete) simplex with branch-and-bound (cutting planes) is used.
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).
Arrays (A)

We have two basic operations on arrays

- \( \text{select}(a, i) \) is the value of array \( a \) at the position \( i \)
- \( \text{store}(a, i, v) \) is the array \( a \) with \( v \) at the position \( i \)

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).
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.
Combining theories

Instead of solving satisfiability of a formula $\varphi$ in a theory $\mathcal{T}$, we can also try to solve the same problem for a theory that is created as a union of more theories $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$, where we already have solvers for all individual $\mathcal{T}_i$, for $1 \leq i \leq n$. For example, we can combine $\mathcal{T}_{LRA}$ and $\mathcal{T}_{UF}$.

We could develop a special solver for the new theory $\mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$, or attempt to combine individual solvers together in a uniform way by

- separating reasoning for individual theories,
- exchange equalities between solvers.
Consider the following set of literals over $T_{\text{LRA}} \cup T_{\text{UF}}$ ($T_{\text{LRA}}$, linear real arithmetic):

\[
\begin{align*}
  f(f(x) - f(y)) &= a \\
  f(0) &> a + 2 \\
  x &= y
\end{align*}
\]
Consider the following set of literals over $T_{\text{LRA}} \cup T_{\text{UF}}$ ($T_{\text{LRA}}$, linear real arithmetic):

\[
\begin{align*}
    f(f(x) - f(y)) &= a \\
    f(0) &> a + 2 \\
    x &= y
\end{align*}
\]

**First step:** *purify* literals so that each belongs to a single theory.
Consider the following set of literals over $T_{LRA} \cup T_{UF}$ ($T_{LRA}$, linear \textit{real} arithmetic):

\[
\begin{align*}
  f(f(x) - f(y)) & = a \\
  f(0) & > a + 2 \\
  x & = y
\end{align*}
\]

\textbf{First step:} \textit{purify} literals so that each belongs to a single theory

\[
\begin{align*}
  f(f(x) - f(y)) = a & \implies f(e_1) = a \\
  e_1 = f(x) - f(y) & \implies f(e_1) = a \\
  e_1 = e_2 - e_3 & \\
  e_2 = f(x) & \\
  e_3 = f(y)
\end{align*}
\]
Consider the following set of literals over $T_{\text{LRA}} \cup T_{\text{UF}}$ ($T_{\text{LRA}}$, linear real arithmetic):

\[
\begin{align*}
    f(f(x) - f(y)) &= a \\
    f(0) &> a + 2 \\
    x &= y
\end{align*}
\]

First step: **purify** literals so that each belongs to a single theory

\[
\begin{align*}
    f(0) > a + 2 &\implies f(e_4) > a + 2 &\implies f(e_4) &= e_5 \\
    e_4 &= 0 &\implies e_4 &= 0 \\
    e_5 &> a + 2
\end{align*}
\]
**Second step:** exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

\[
\begin{array}{c|c}
L_1 & L_2 \\
\hline
f(e_1) = a & e_2 - e_3 = e_1 \\
f(x) = e_2 & e_4 = 0 \\
f(y) = e_3 & e_5 > a + 2 \\
f(e_4) = e_5 \\
x = y
\end{array}
\]
**Second step:** exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

\[
\begin{array}{ccc}
| & L_1 & L_2 |\\
\hline
f(e_1) = a & e_2 - e_3 = e_1 \\
f(x) = e_2 & e_4 = 0 \\
f(y) = e_3 & e_5 > a + 2 \\
f(e_4) = e_5 & \\
x = y & \\
\end{array}
\]

$L_1 \models_{UF} e_2 = e_3$
**Second step:** exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
</tbody>
</table>
Second step: exchange entailed interface equalities, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
</tbody>
</table>

$L_2 \models_{LRA} e_1 = e_4$
Second step: exchange entailed interface equalities, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>
**Second step:** exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td></td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

$L_1 \models_{UF} a = e_5$
**Second step:** exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$a = e_5$</td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>
Second step: exchange entailed interface equalities, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

\[
\begin{array}{ll}
L_1 & L_2 \\
\hline
f(e_1) = a & e_2 = e_3 = e_1 \\
f(x) = e_2 & e_4 = 0 \\
f(y) = e_3 & e_5 > a + 2 \\
f(e_4) = e_5 & e_2 = e_3 \\
x = y & a = e_5 \\
e_1 = e_4 &
\end{array}
\]

Third step: check for satisfiability locally

$L_1 \not\models_{UF} \bot$

$L_2 \models_{LRA} \bot$
**Second step:** exchange entailed *interface equalities*, equalities over shared constants $e_1, e_2, e_3, e_4, e_5, a$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(e_1) = a$</td>
<td>$e_2 - e_3 = e_1$</td>
</tr>
<tr>
<td>$f(x) = e_2$</td>
<td>$e_4 = 0$</td>
</tr>
<tr>
<td>$f(y) = e_3$</td>
<td>$e_5 &gt; a + 2$</td>
</tr>
<tr>
<td>$f(e_4) = e_5$</td>
<td>$e_2 = e_3$</td>
</tr>
<tr>
<td>$x = y$</td>
<td>$a = e_5$</td>
</tr>
<tr>
<td>$e_1 = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

**Third step:** check for satisfiability locally

$L_1 \not\models_{UF} \perp$

$L_2 \models_{LRA} \perp$

Report *unsatisfiable*
Nelson–Oppen procedure

It possible to combine theories that have disjoint signatures (equalities are shared) and are stably infinite, meaning that every \( \mathcal{T} \)-satisfiable ground formula is \( \mathcal{T} \)-satisfied by an infinite \( \mathcal{T} \)-interpretation. For example, finite structures like bit-vectors are not stably infinite.

A theory \( \mathcal{T} \) is convex, if \( \Gamma \models_{\mathcal{T}} \varphi_1 \lor \cdots \lor \varphi_n \), then \( \Gamma \models_{\mathcal{T}} \varphi_i \) for some \( i \in \{1, \ldots, n\} \).
Consider the following unsatisfiable set of literals over $T_{LIA} \cup T_{UF}$ ($T_{LIA}$, linear integer arithmetic):

\[
\begin{align*}
1 \leq x & \leq 2 \\
 f(1) &= a \\
 f(2) &= f(1) + 3 \\
 a &= b + 2
\end{align*}
\]
Consider the following unsatisfiable set of literals over $T_{\text{LIA}} \cup T_{\text{UF}}$ ($T_{\text{LIA}}$, linear integer arithmetic):

\[
1 \leq x \leq 2 \\
 f(1) = a \\
 f(2) = f(1) + 3 \\
 a = b + 2
\]

**First step:** *purify* literals so that each belongs to a single theory
Consider the following unsatisfiable set of literals over $T_{LIA} \cup T_{UF}$ ($T_{LIA}$, linear integer arithmetic):

$$1 \leq x \leq 2$$
$$f(1) = a$$
$$f(2) = f(1) + 3$$
$$a = b + 2$$

**First step:** purify literals so that each belongs to a single theory

$$f(1) = a \implies f(e_1) = a$$
$$e_1 = 1$$
Consider the following unsatisfiable set of literals over $T_{\text{LIA}} \cup T_{\text{UF}}$ ($T_{\text{LIA}}$, linear integer arithmetic):

\begin{align*}
1 \leq x & \leq 2 \\
f(1) &= a \\
f(2) &= f(1) + 3 \\
a &= b + 2
\end{align*}

**First step:** *purify* literals so that each belongs to a single theory

\begin{align*}
f(2) &= f(1) + 3 \implies e_2 = 2 \\
f(e_2) &= e_3 \\
f(e_1) &= e_4 \\
e_3 &= e_4 + 3
\end{align*}
**Second step:** exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>
Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th></th>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>2</td>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>3</td>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>4</td>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>5</td>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

No more entailed equalities, but $L_1 \models_{LIA} x = e_1 \lor x = e_2$
Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

Consider each case of $x = e_1 \lor x = e_2$ separately
**Second step:** exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>

Case 1) $x = e_1$
**Second step:** exchange entailed interface equalities over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_1$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td>$a = e_4$</td>
</tr>
<tr>
<td>$x = e_1$</td>
<td>$x = e_1$</td>
</tr>
</tbody>
</table>
Second step: exchange entailed *interface equalities* over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_1$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_1$</td>
<td></td>
</tr>
</tbody>
</table>

$L_2 \vdash_{UF} a = b$, which entails $\bot$ when sent to $L_1$
Second step: exchange entailed interface equalities over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>L₁</th>
<th>L₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td></td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
</tbody>
</table>
**MOTIVATING EXAMPLE (NON-CONVEX CASE)**

**Second step:** exchange entailed *interface equalities* over shared constants \(x, e_1, a, b, e_2, e_3, e_4\)

<table>
<thead>
<tr>
<th>(L_1)</th>
<th>(L_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1 \leq x)</td>
<td>(f(e_1) = a)</td>
</tr>
<tr>
<td>(x \leq 2)</td>
<td>(f(x) = b)</td>
</tr>
<tr>
<td>(e_1 = 1)</td>
<td>(f(e_2) = e_3)</td>
</tr>
<tr>
<td>(a = b + 2)</td>
<td>(f(e_1) = e_4)</td>
</tr>
<tr>
<td>(e_2 = 2)</td>
<td></td>
</tr>
<tr>
<td>(e_3 = e_4 + 3)</td>
<td></td>
</tr>
<tr>
<td>(a = e_4)</td>
<td></td>
</tr>
</tbody>
</table>

Case 2) \(x = e_2\)
Second step: exchange entailed interface equalities over shared constants $x, e_1, a, b, e_2, e_3, e_4$

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 \leq x$</td>
<td>$f(e_1) = a$</td>
</tr>
<tr>
<td>$x \leq 2$</td>
<td>$f(x) = b$</td>
</tr>
<tr>
<td>$e_1 = 1$</td>
<td>$f(e_2) = e_3$</td>
</tr>
<tr>
<td>$a = b + 2$</td>
<td>$f(e_1) = e_4$</td>
</tr>
<tr>
<td>$e_2 = 2$</td>
<td>$x = e_2$</td>
</tr>
<tr>
<td>$e_3 = e_4 + 3$</td>
<td></td>
</tr>
<tr>
<td>$a = e_4$</td>
<td></td>
</tr>
<tr>
<td>$x = e_2$</td>
<td></td>
</tr>
</tbody>
</table>
Second step: exchange entailed *interface equalities* over shared constants \( x, e_1, a, b, e_2, e_3, e_4 \)

<table>
<thead>
<tr>
<th>( L_1 )</th>
<th>( L_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 \leq x )</td>
<td>( f(e_1) = a )</td>
</tr>
<tr>
<td>( x \leq 2 )</td>
<td>( f(x) = b )</td>
</tr>
<tr>
<td>( e_1 = 1 )</td>
<td>( f(e_2) = e_3 )</td>
</tr>
<tr>
<td>( a = b + 2 )</td>
<td>( f(e_1) = e_4 )</td>
</tr>
<tr>
<td>( e_2 = 2 )</td>
<td>( x = e_2 )</td>
</tr>
<tr>
<td>( e_3 = e_4 + 3 )</td>
<td></td>
</tr>
<tr>
<td>( a = e_4 )</td>
<td></td>
</tr>
<tr>
<td>( x = e_2 )</td>
<td></td>
</tr>
</tbody>
</table>

\( L_2 \models_{UF} e_3 = b \), which entails \( \bot \) when sent to \( L_1 \)
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-const x Int)
(declare-const y Int)
(assert (= (- x y) (+ x (- y) 1)))
(check-sat)
; unsat
(exit)
The following slides are taken from
- 12, 14, and 16 are from Jovanović 2016,
- 20 and 28 are from Tinelli 2017,
- 18, is from Griggio 2015.


