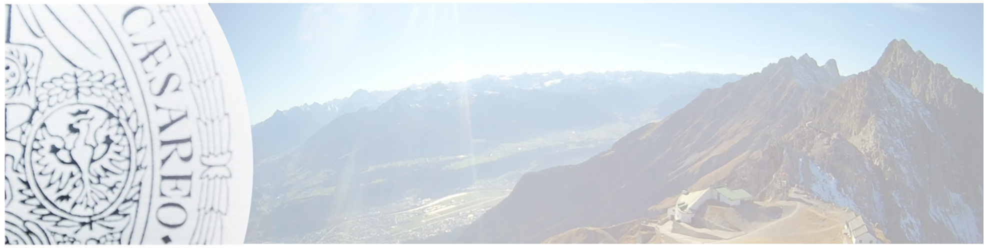
universität innsbruck



SAT and SMT Solving

Sarah Winkler

KRDB
Department of Computer Science
Free University of Bozen-Bolzano

lecture 6
WS 2022

Outline

First-Order Logic: Syntax

Definitions

- signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ consists of
  - set of function symbols $\mathcal{F}$
  - set of predicate symbols $\mathcal{P}$

  where each symbol is associated with fixed arity
- Σ-terms t are built according to grammar
$$t \quad ::= \quad x \mid c \mid f(\underbrace{t, \dots, t}_{n})$$
- Σ-formulas φ are built according to grammar
$$\varphi \quad ::= \quad Q \mid P(\underbrace{t, \dots, t}_{n}) \mid \bot \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \forall x.\, \varphi \mid \exists x.\, \varphi$$
- variable occurrence is free in φ if it is not bound by quantifier above
- formulas without free variables are sentences

First-Order Logic: Semantics

Definition

model $\mathcal{M}$ for signature $\Sigma = \langle \mathcal{F}, \mathcal{P} \rangle$ consists of

1. non-empty set A (universe of concrete values)
2. function $f^{\mathcal{M}} \colon A^n \to A$ for every n-ary $f \in \mathcal{F}$
3. set of n-tuples $P^{\mathcal{M}} \subseteq A^n$ for every n-ary $P \in \mathcal{P}$

Definitions

- environment for model $\mathcal{M}$ with universe A is mapping $I \colon X \to A$
- value $t^{\mathcal{M},I}$ of term t in model $\mathcal{M}$ wrt environment I:
  $t^{\mathcal{M},I} = I(t)$ if t is a variable, and $t^{\mathcal{M},I} = f^{\mathcal{M}}(t_n^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I})$ otherwise
- $$\mathcal{M} \models_I \varphi \quad \iff \quad \begin{cases} (t_n^{\mathcal{M},I}, \dots, t_n^{\mathcal{M},I}) \in P^{\mathcal{M}} & \text{if } \varphi = P(t_1, \dots, t_n) \\ \mathcal{M} \not\models_I \psi & \text{if } \varphi = \neg\psi \\ \mathcal{M} \models_I \varphi_1 \text{ and } \mathcal{M} \models_I \varphi_2 & \text{if } \varphi = \varphi_1 \wedge \varphi_2 \\ \mathcal{M} \models_I \varphi_1 \text{ or } \mathcal{M} \models_I \varphi_2 & \text{if } \varphi = \varphi_1 \vee \varphi_2 \\ \mathcal{M} \models_{I[x \mapsto a]} \psi \text{ for all } a \in A & \text{if } \varphi = \forall x.\, \psi \\ \mathcal{M} \models_{I[x \mapsto a]} \psi \text{ for some } a \in A & \text{if } \varphi = \exists x.\, \psi \end{cases}$$

Definition

- formula φ is satisfiable if $\mathcal{M} \models_I \varphi$ for some $\mathcal{M}$ and I
- set of formulas T is satisfiable if $\mathcal{M} \models_I \bigwedge_{\varphi \in T} \varphi$ for some $\mathcal{M}$ and I

Remark

if φ is sentence, $\mathcal{M} \models_I \varphi$ is independent of I

Definition (Theory)

Σ-theory T is set of Σ-sentences that is satisfiable

Definitions

for theory T, formulas F and G and list of literals M:

- F is T-consistent (or T-satisfiable) if $\{F\} \cup T$ is satisfiable
- F is T-inconsistent (or T-unsatisfiable) if not T-consistent
- F entails G in T (denoted $F \vDash_T G$) if $F \wedge \neg G$ is T-inconsistent
- F and G are T-equivalent (denoted $F \equiv_T G$) if $F \vDash_T G$ and $G \vDash_T F$

Definition (Theory of Equality EQ)

- signature: no function symbols, binary predicate $=$
- axioms:

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

Definition (Theory of Equality With Uninterpreted Functions EUF)

- signature: function symbols $\mathcal{F}$, predicate symbols $\mathcal{P}$ including binary $=$
- axioms:

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

plus for all $f/n \in \mathcal{F}$ and $P/n \in \mathcal{P}$ functional consistency axioms:

$$\forall x_1 y_1 \ \dots\ x_n y_n.\ (x_1 = y_1 \wedge \cdots \wedge x_n = y_n \ \rightarrow\ f(x_1, \dots, x_n) = f(y_1, \dots, y_n))$$

$$\forall x_1\, y_1 \ \dots\ x_n\, y_n.\ (x_1 = y_1 \wedge \cdots \wedge x_n = y_n \ \rightarrow\ (P(x_1, \dots, x_n) \rightarrow P(y_1, \dots, y_n)))$$

Definition

DPLL(T) consists of DPLL rules unit propagate, decide, fail, and restart plus

- T-backjump $\qquad M\, l^d\, N \parallel F, C \implies M\, l' \parallel F, C$
  if $M\, l^d\, N \vDash \neg C$ and $\exists$ clause $C' \vee l'$ such that
  - $F, C \vDash_T C' \vee l'$
  - $M \vDash \neg C'$ and l' is undefined in M, and l' or l'^c occurs in F or in $M\, l^d\, N$
- T-learn $\qquad M \parallel F \implies M \parallel F,\ C$
  if $F \vDash_T C$ and all atoms of C occur in M or F
- T-forget $\qquad M \parallel F,\ C \implies M \parallel F$
  if $F \vDash_T C$
- T-propagate $\qquad M \parallel F \implies M\, l \parallel F$
  if $M \vDash_T l$, literal l or l^c occurs in F, and l is undefined in M

Naive Lazy Approach in DPLL(T)

- whenever state $M \parallel F$ is final wrt unit propagate, decide, fail, T-backjump:
  check T-consistency of M with T-solver
- if M is T-consistent then satisfiability is proven
- otherwise $\exists l_1, \dots, l_k$ subset of M such that $\vDash_T \neg(l_1 \wedge \cdots \wedge l_k)$
- use T-learn to add $\neg l_1 \vee \cdots \vee \neg l_k$
- apply restart

Improvement 1: Incremental T-Solver

- T-solver checks T-consistency of model M whenever literal is added to M

Improvement 2: On-Line SAT solver

- after T-learn added clause, apply fail or T-backjump instead of restart

Improvement 3: Eager Theory Propagation

- apply T-propagate before decide

Outline

- Summary of Last Week
- Deciding EQ: Equality Graphs
- Deciding EUF: Congruence Closure
- Correctness of DPLL(T)
- Some More Practical SMT

Equality Graph

Aim

build theory solver for theory of equality (EQ)

Definition

- equality logic formula φ_{EQ} is set of equations and inequalities between variables
- write $\mathcal{V}ar(\varphi_{\mathsf{EQ}})$ for set of variables occurring in φ_{EQ}

Definition

equality graph for φ_{EQ} is undirected graph $(V, E_=, E_{\neq})$ with two kinds of edges

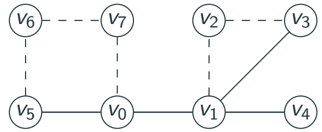
- nodes $V = \mathcal{V}ar(\varphi_{\mathsf{EQ}})$
- $(x, y) \in E_=$ iff $x = y$ in φ_{EQ}
- $(x, y) \in E_{\neq}$ iff $x \neq y$ in φ_{EQ}

equality edge

edges $E_=$ are drawn dashed, $E_{\neq}$ are drawn solid

Example

$v_0 \neq v_1 \quad v_0 \neq v_5 \quad v_1 = v_2 \quad v_1 \neq v_4 \quad v_1 \neq v_3 \quad v_2 = v_3 \quad v_5 = v_6 \quad v_6 = v_7 \quad v_7 = v_0$



Definition (Contradictory cycle)

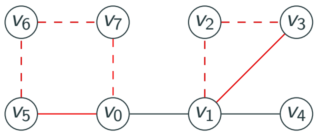
contradictory cycle is simple cycle in equality graph with one $E_{\neq}$ edge and all others $E_=$ edges

Theorem

φ_{EQ} is satisfiable iff its equality graph has no contradictory cycle

Example

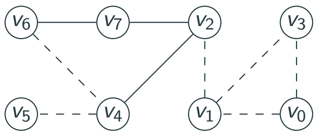
$v_0 \neq v_1 \quad v_0 \neq v_5 \quad v_1 = v_2 \quad v_1 \neq v_4 \quad v_1 \neq v_3 \quad v_2 = v_3 \quad v_5 = v_6 \quad v_6 = v_7 \quad v_7 = v_0$



unsatisfiable

Example

$v_0 = v_1 \quad v_0 = v_2 \quad v_1 = v_2 \quad v_1 = v_3 \quad v_2 \neq v_4 \quad v_4 = v_5 \quad v_4 = v_6 \quad v_6 \neq v_7 \quad v_7 \neq v_2$



satisfiable

Outline

- Summary of Last Week
- Deciding EQ: Equality Graphs
- Deciding EUF: Congruence Closure
- Correctness of DPLL(T)
- Some More Practical SMT

Aim

build theory solver for theory of equality with uninterpreted functions (EUF)

Definitions (Terms)

number of arguments

- set of function symbols $\mathcal{F}$ with fixed arity
- set of variables V
- terms $\mathcal{T}(\mathcal{F}, V)$ are built according to grammar

$$t \quad ::= \quad x \mid c \mid f(\underbrace{t, \dots, t}_{n})$$

  if $x \in V$, c is constant, and $f \in$ F has arity n

- subterms

$$\mathcal{S}ub(t) = \begin{cases} \{t\} & \text{if } t \in V \\ \{t\} \cup \bigcup_i \mathcal{S}ub(t_i) & \text{if } t = f(t_1, \dots, t_n) \end{cases}$$

Example

- for $\mathcal{F} = \{\mathsf{f}/1, \mathsf{g}/2, \mathsf{a}/0\}$ and $x, y \in V$ have terms a, $\mathsf{f}(x)$, $\mathsf{f}(\mathsf{a})$, $\mathsf{g}(x, \mathsf{f}(y))$, ...
- for $t = \mathsf{g}(\mathsf{g}(x, x), \mathsf{f}(\mathsf{f}(\mathsf{a})))$ have $\mathcal{S}ub(t) = \{t,\ \mathsf{g}(x, x),\ x,\ \mathsf{f}(\mathsf{f}(\mathsf{a})),\ \mathsf{f}(\mathsf{a}),\ \mathsf{a}\}$

Congruence Closure

Input: set of equations E and equation $s = t$ (without variables, only constants)

Output: $s = t$ is *implied* ($E \vDash_{EUF} s = t$) or *not implied* ($E \nvDash_{EUF} s = t$)

1. build congruence classes
   (a) collect all subterms of terms in $E \cup \{s = t\}$
   (b) put different subterms of $E \cup \{s = t\}$ in separate sets
   (c) merge sets $\{\dots, t_1, \dots\}$ and $\{\dots, t_2, \dots\}$ for all $t_1 = t_2$ in E
   (d) merge sets $\{\dots, f(t_1, \dots, t_n), \dots\}$ and $\{\dots, f(u_1, \dots, u_n), \dots\}$ if t_i and u_i belong to same set for all $1 \leqslant i \leqslant n$
   (e) repeat (d) until no change
2. if s and t belong to same set then return *implied* else return *not implied*

Example (1)

- given set of equations E

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))) = \mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{b})))) \quad \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{b}))) = \mathsf{f}(\mathsf{a}) \quad \mathsf{g}(\mathsf{g}(\mathsf{b})) = \mathsf{g}(\mathsf{f}(\mathsf{a})) \quad \mathsf{g}(\mathsf{a}) = \mathsf{b}$$

  and test equation $\mathsf{f}(\mathsf{a}) = \mathsf{g}(\mathsf{a})$
- sets

1. $\{\,\mathsf{a}\,\}$
2. $\{\,\mathsf{f}(\mathsf{a}),\ \mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{b})))\,\}$
3. $\{\,\mathsf{b},\ \mathsf{g}(\mathsf{a})\,\}$
4. $\{\,\mathsf{g}(\mathsf{b})\,\}$
5. $\{\,\mathsf{f}(\mathsf{f}(\mathsf{a}))\,\}$
6. $\{\,\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))),\ \mathsf{g}(\mathsf{f}(\mathsf{g}(\mathsf{f}(\mathsf{b})))),\ \mathsf{g}(\mathsf{g}(\mathsf{b})),\ \mathsf{g}(\mathsf{f}(\mathsf{a}))\,\}$
7. $\{\,\mathsf{f}(\mathsf{b})\,\}$
8. $\{\,\mathsf{g}(\mathsf{f}(\mathsf{b}))\,\}$

- conclusion: $E \nvDash_{EUF} \mathsf{f}(\mathsf{a}) = \mathsf{g}(\mathsf{a})$

Example (2)

- given set of equations E

$$\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))) = \mathsf{a} \qquad \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))))) = \mathsf{a}$$

  and test equaton $\mathsf{f}(\mathsf{a}) = \mathsf{a}$

- $\{\,\mathsf{a},\ \mathsf{f}(\mathsf{a}),\ \mathsf{f}(\mathsf{f}(\mathsf{a})),\ \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a}))),\ \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a})))),\ \mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{f}(\mathsf{a})))))\,\}$
- conclusion: $E \vDash_{EUF} \mathsf{f}(\mathsf{a}) = \mathsf{a}$

Ok, But How About a Solver for EUF?

Assume conjunction of EUF literals φ with free variables $x_1, \dots, x_n$.

Definition (Skolemization)

$\widehat{\varphi} = \varphi[x_1 \mapsto c_1, \dots, x_n \mapsto c_n]$ where $c_1, \dots, c_n$ are distinct fresh constants

Lemma

φ is EUF-satisfiable iff $\widehat{\varphi}$ is EUF-satisfiable

Assumption

assume that $=$ is the only predicate in φ

Remark

if φ contains n-ary predicate P different from equality:

- add new constant *true* and n-ary function f_P
- replace $P(t_1, \dots, t_n)$ by $f_P(t_1, \dots, t_n) = true$
- replace $P(t_1, \dots, t_n)$ by $f_P(t_1, \dots, t_n) \neq true$

Assume conjunction of equations and inequalities φ with free variables $x_1, \dots, x_n$.

P is set of equations,
N is set of inequalities

Deciding satisfiability of set of EUF literals

split $\varphi = (\bigwedge P) \wedge (\bigwedge N)$ into positive literals P and negative literals N

$\varphi = (\bigwedge P) \wedge (\bigwedge N)$	EUF-unsatisfiable	
$\iff (\bigwedge \widehat{P}) \wedge (\bigwedge \widehat{N})$	EUF-unsatisfiable	skolemization
$\iff \neg\big((\bigwedge \widehat{P}) \wedge (\bigwedge \widehat{N})\big)$	EUF-valid	φ unsat iff $\neg\varphi$ valid
$\iff \bigwedge \widehat{P} \to \bigvee_{l \in \widehat{N}} \neg l$	EUF-valid	
$\iff \exists\, s \neq t$ in $\widehat{N}$ such that $\bigwedge \widehat{P} \to s = t$ is EUF-valid		semantics of $\vee$
$\iff \exists\, s \neq t$ in $\widehat{N}$ such that $\bigwedge \widehat{P} \vDash_{EUF} s = t$		semantics of $\vDash_{EUF}$

Obtained Satisfiability Check

$$(\bigwedge P) \wedge (\bigwedge N) \text{ unsatisfiable} \quad \iff \quad \exists\, s \neq t \text{ in } \widehat{N} \text{ such that } \bigwedge \widehat{P} \vDash_T s = t$$

Example

1. $g(a) = c \wedge f(g(a)) \neq f(c) \wedge c \neq d$
   - split into $P = \{g(a) = c\}$ and $N = \{f(g(a)) \neq f(c),\ c \neq d\}$
   - have $g(a) = c \vDash_T f(g(a)) = f(c)$, so unsatisfiable
2. $g(a) = c \wedge f(g(a)) = f(c) \wedge g(a) = d \wedge c \neq d$
   - split into $P = \{g(a) = c,\ f(g(a)) = f(c),\ g(a) = d\}$ and $N = \{c \neq d\}$
   - have $g(a) = c, f(g(a)) = f(c), g(a) = d \vDash_T c = d$, so unsatisfiable
3. $g(a) = c \wedge c = d \wedge f(x) = x \wedge d \neq g(x) \wedge f(x) \neq d$
   - $P = \{g(a) = c,\ c = d,\ f(x) = x\}$ and $N = \{d \neq g(x),\ f(x) \neq d\}$
   - skolemize $P = \{g(a) = c,\ c = d,\ f(e) = e\}$, $N = \{d \neq g(e),\ f(e) \neq d\}$
     - $g(a) = c,\ c = d,\ f(e) = e \nvDash_T d = g(e)$
     - $g(a) = c,\ c = d,\ f(e) = e \nvDash_T f(e) = d$

   so satisfiable

Outline

- Summary of Last Week
- Deciding EQ: Equality Graphs
- Deciding EUF: Congruence Closure
- Correctness of DPLL(T)
- Some More Practical SMT

Definition (Basic DPLL(T))
system $\mathcal{B}$ consists of unit propagate, decide, fail, T-backjump, and T-propagate

Definition (Full DPLL(T))
system $\mathcal{D}$ extends $\mathcal{B}$ by T-learn, T-forget, and restart

Lemma
if $\| F \Longrightarrow_{\mathcal{D}}^{*} M \| G$ *then*

- *all atoms in M and G are atoms in F*
- *M does not contain complementary literals, and every literal at most once*
- *G is T-equivalent to F ($F \equiv_T G$)*
- *if $M = M_0\ l_1^d\ M_1\ l_2^d\ M_2 \dots l_k^d\ M_k$ with $l_1, \dots, l_k$ all the decision literals then $F, l_1, \dots, l_i \vDash_T M_i$ for all $0 \leqslant i \leqslant k$*

Consider derivation with final state S_n:

$$\| F \Longrightarrow_{\mathcal{D}} S_1 \Longrightarrow_{\mathcal{D}} S_2 \Longrightarrow_{\mathcal{D}} \dots \Longrightarrow_{\mathcal{D}} S_n$$

Theorem
if S_n = FailState then F is T-unsatisfiable

Proof.

- must have $\| F \Longrightarrow_{\mathcal{D}}^{*} M \| F' \overset{\text{fail}}{\Longrightarrow}_{\mathcal{D}}$ FailState, so $M \vDash \neg C$ for some C in F'
- M cannot contain decision literals (otherwise T-backjump applicable)
- by Lemma before, $F' \vDash_T M$, so $F' \vDash_T \neg C$
- also have $F' \vDash_T C$ because C is in F' and $F \equiv_T F'$ so T-inconsistent ∎

Theorem
if $S_n = M \| F'$ and M is T-consistent then F is T-satisfiable and $M \vDash_T F$

Proof.

- S_n is final, so all literals of F' are defined in M (otherwise decide applicable)
- $\nexists$ clause C in F' such that $M \vDash \neg C$ (otherwise backjump or fail applicable)
- so $M \vDash F'$ and by T-consistency $M \vDash_T F'$
- have $F \equiv_T F'$ so M also T-satisfies F ∎

Theorem (Termination)

$$\Gamma: \quad \| F \Longrightarrow_{\mathcal{D}}^{*} S_0 \Longrightarrow_{\mathcal{D}}^{*} S_1 \Longrightarrow_{\mathcal{D}}^{*} \dots$$

is finite if

- *there is no infinite sub-derivation of only T-learn and T-forget steps, and*
- *for every sub-derivation*

$$S_i \overset{\text{restart}}{\Longrightarrow}_{\mathcal{D}} S_{i+1} \Longrightarrow_{\mathcal{D}}^{*} S_j \overset{\text{restart}}{\Longrightarrow}_{\mathcal{D}} S_{j+1} \Longrightarrow_{\mathcal{D}}^{*} S_k \overset{\text{restart}}{\Longrightarrow}_{\mathcal{D}} S_{k+1}$$

with no restart steps in $S_{i+1} \Longrightarrow_{\mathcal{D}}^{*} S_j$ and $S_{j+1} \Longrightarrow_{\mathcal{D}}^{*} S_k$:
  - *there are more $\mathcal{B}$-steps in $S_j \Longrightarrow_{\mathcal{D}}^{*} S_k$ than in $S_i \Longrightarrow_{\mathcal{D}}^{*} S_j$, or*
  - *a clause is learned in $S_j \Longrightarrow_{\mathcal{D}}^{*} S_k$ that is never forgotten in Γ*

Proof.
similar as for DPLL:

- restart is applied with increasing periodicity, or
- otherwise clause is learned (and there are only finitely many clauses) ∎

Integer Arithmetic in `python/z3`

```
from z3 import *
a = Int('a') # create integer variables
b = Int('b')
c = Int('c')

phi = And(c > 0, b >= 0, a < -1) # some comparisons
psi = (a == If (b == c, b - 2, c - 4)) # if-then-else expression
print(phi)
solver = Solver()
solver.add(phi, psi) # assert constraints
solver.add(a + b < 2 * c) # arithmetic

result = solver.check() # check for satisfiability
if result == z3.sat:
  model = solver.model() # get valuation
  print model[a], model[b], model[c] # -3 0 1
```