



## SAT and SMT Solving

Sarah Winkler

Computational Logic Group  
Department of Computer Science  
University of Innsbruck

lecture 6  
SS 2019

### Definitions

for formulas  $F$  and  $G$  and list of literals  $M$ :

- ▶ **theory  $T$**  is set of first-order logic formulas without free variables
- ▶  $F$  is  **$T$ -consistent** (or  **$T$ -satisfiable**) if  $F \wedge T$  is satisfiable in first-order sense
- ▶  $F$  is  **$T$ -inconsistent** (or  **$T$ -unsatisfiable**) if not  $T$ -consistent
- ▶  $M = l_1, \dots, l_k$  is  **$T$ -consistent** if  $l_1 \wedge \dots \wedge l_k$  is
- ▶  $M$  is  **$T$ -model** of  $F$  if  $M \models F$  and  $M$  is  $T$ -consistent
- ▶  $F$  **entails  $G$  in  $T$**  (denoted  $F \models_T G$ ) if  $F \wedge \neg G$  is  $T$ -inconsistent
- ▶  $F$  and  $G$  are  **$T$ -equivalent** (denoted  $F \equiv_T G$ ) if  $F \models_T G$  and  $G \models_T F$

## Outline

- Summary of Last Week
- Correctness of DPLL( $T$ )
- Congruence Closure
- Some More Practical SMT

1

### Definition (Theory of Equality)

**theory of equality (EQ)** uses binary predicate  $\approx$  and consists of axioms

$$\forall x (x \approx x) \quad \forall x y (x \approx y \rightarrow y \approx x) \quad \forall x y z (x \approx y \wedge y \approx z \rightarrow x \approx z)$$

### Definition (Theory of Equality With Uninterpreted Functions)

**EUf** over set of function symbols  $\mathcal{F}$  consists of equality axioms:

$$\forall x (x \approx x) \quad \forall x y (x \approx y \rightarrow y \approx x) \quad \forall x y z (x \approx y \wedge y \approx z \rightarrow x \approx z)$$

plus for all  $f/n \in \mathcal{F}$  {the **functional consistency axiom**:

$$\forall x_1 y_1 \dots x_n y_n (x_1 \approx y_1 \wedge \dots \wedge x_n \approx y_n \rightarrow f(x_1, \dots, x_n) \approx f(y_1, \dots, y_n))$$

## Definition (DPLL( $T$ ) Transition Rules)

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

- ▶  **$T$ -backjump**  $M \text{ I}^d N \parallel F, C \implies M \text{ I}' \parallel F, C$   
if  $M \text{ I}^d N \models \neg C$  and  $\exists$  clause  $C' \vee I'$  such that
  - ▶  $F, C \models_T C' \vee I'$
  - ▶  $M \models \neg C'$  and  $I'$  is undefined in  $M$ , and  $I'$  or  $I'^c$  occurs in  $F$  or in  $M \text{ I}^d N$
- ▶  **$T$ -learn**  $M \parallel F \implies M \parallel F, C$   
if  $F \models_T C$  and all atoms of  $C$  occur in  $M$  or  $F$
- ▶  **$T$ -forget**  $M \parallel F, C \implies M \parallel F$   
if  $F \models_T C$
- ▶  **$T$ -propagate**  $M \parallel F \implies M \text{ I} \parallel F$   
if  $M \models_T I$ , literal  $I$  or  $I^c$  occurs in  $F$ , and  $I$  is undefined in  $M$

4

## Outline

- Summary of Last Week
- Correctness of DPLL( $T$ )
- Congruence Closure
- Some More Practical SMT

6

## 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  $\exists I_1, \dots, I_k$  subset of  $M$  such that  $\models_T \neg(I_1 \wedge \dots \wedge I_k)$
- ▶ use  $T$ -learn to add  $\neg I_1 \vee \dots \vee \neg I_k$
- ▶ 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

## Remark

all three improvements can be combined

5

## Definition (Basic DPLL( $T$ ))

system  $\mathcal{B}$  consists of unit propagate, decide, fail,  $T$ -backjump, and  $T$ -propagate

## Definition (Full DPLL( $T$ ))

system  $\mathcal{F}$  extends  $\mathcal{B}$  by  $T$ -learn,  $T$ -forget, and restart

## Lemma

if  $\parallel F \implies_F^* M \parallel G$  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  $M = M_0 \text{ I}_1^d M_1 \text{ I}_2^d M_2 \dots \text{ I}_k^d M_k$  with  $I_1, \dots, I_k$  all the decision literals then  $F, I_1, \dots, I_i \models_T M_i$  for all  $0 \leq i \leq k$

7

Consider derivation with final state  $S_n$ :

$$\| F \Rightarrow_{\mathcal{F}} S_1 \Rightarrow_{\mathcal{F}} S_2 \Rightarrow_{\mathcal{F}} \dots \Rightarrow_{\mathcal{F}} S_n$$

### Theorem

if  $S_n = \text{FailState}$  then  $F$  is  $T$ -unsatisfiable

### Proof.

- ▶ must have  $\| F \Rightarrow_{\mathcal{F}}^* M \parallel F' \xrightarrow{\text{fail}}_{\mathcal{F}} \text{FailState}$ , so  $M \models \neg C$  for some  $C$  in  $F'$
- ▶  $M$  cannot contain decision literals (otherwise  $T$ -backjump applicable)
- ▶ by Lemma before,  $F' \models_T M$ , so  $F' \models_T \neg C$
- ▶ also have  $F' \models_T C$  because  $C$  is in  $F'$ , so  $F \equiv_T F'$  is  $T$ -inconsistent ■

### Theorem

if  $S_n = M \parallel F'$  and  $M$  is  $T$ -consistent then  $F$  is  $T$ -satisfiable and  $M \models_T F$

### Proof.

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

8

## Outline

- Summary of Last Week
- Correctness of DPLL( $T$ )
- Congruence Closure
- Some More Practical SMT

## Theorem (Termination)

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

is finite if

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

$$S_i \xrightarrow{\text{restart}}_{\mathcal{F}} S_{i+1} \Rightarrow_{\mathcal{F}}^* S_j \xrightarrow{\text{restart}}_{\mathcal{F}} S_{j+1} \Rightarrow_{\mathcal{F}}^* S_k \xrightarrow{\text{restart}}_{\mathcal{F}} S_{k+1}$$

with no restart steps in  $S_{i+1} \Rightarrow_{\mathcal{F}}^* S_j$  and  $S_{j+1} \Rightarrow_{\mathcal{F}}^* S_k$ :

- ▶ there are more  $\mathcal{B}$ -steps in  $S_j \Rightarrow_{\mathcal{F}}^* S_k$  than in  $S_i \Rightarrow_{\mathcal{F}}^* S_j$ , or
- ▶ a clause is learned in  $S_j \Rightarrow_{\mathcal{F}}^* S_k$  that is never forgotten in  $\Gamma$

### Proof.

similar as for DPLL:

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

9

## Aim

build  $T$ -solver for EUF using congruence closure

## Definitions (Terms)

- ▶ signature  $\mathcal{F}$  function symbols with fixed arity
- ▶ variables  $\mathcal{V}$   $\mathcal{F} \cap \mathcal{V} = \emptyset$
- ▶ terms  $\mathcal{T}(\mathcal{F}, \mathcal{V})$  smallest set such that
  - ▶  $\mathcal{V} \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$
  - ▶ if  $f \in \mathcal{F}$  has arity  $n$  and  $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{V})$  then  $f(t_1, \dots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$
- ▶ subterms

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

## Example

- ▶ for  $\mathcal{F} = \{f/1, g/2, a/0\}$  and  $x, y \in \mathcal{V}$  have terms  $a, f(x), f(a), g(x, f(y)), \dots$
- ▶ for  $t = g(g(x, x), f(f(a)))$  have  $\text{Sub}(t) = \{t, g(x, x), x, f(f(a)), f(a), a\}$

## Congruence Closure

Input: set of equations  $E$  and equation  $s \approx t$ , both without variables

Output: *valid* ( $E \models_{\tau} s \approx t$ ) or *invalid* ( $E \not\models_{\tau} s \approx t$ )

- 1 build congruence classes
  - (a) put different subterms of  $E \cup \{s \approx t\}$  in separate sets
  - (b) merge sets  $\{\dots, t_1, \dots\}$  and  $\{\dots, t_2, \dots\}$  for all  $t_1 \approx t_2$  in  $E$
  - (c) 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  $1 \leq i \leq n$
  - (d) repeat (c) until no change
- 1 if  $s$  and  $t$  belong to same set then return *valid* else return *invalid*

## Example (1)

- ▶ given set of equations  $E$

$$f(f(f(a))) \approx g(f(g(f(b)))) \quad f(g(f(b))) \approx f(a) \quad g(g(b)) \approx g(f(a)) \quad g(a) \approx b$$

and test equation  $f(a) \approx g(a)$

- ▶ sets

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

- ▶ conclusion:  $E \models f(a) \not\approx g(a)$

12

13

## Example (2)

- ▶ given set of equations  $E$

$$f(f(f(a))) \approx a \quad f(f(f(f(f(a)))) \approx a$$

and test equation  $f(a) \approx a$

- ▶  $\{a, f(a), f(f(a)), f(f(f(a))), f(f(f(f(a))))\}$
- ▶ conclusion:  $E \models f(a) \approx a$

## Ok, But How About a Solver for EUF?

### Definition (Skolemization)

given formula  $\varphi$  with free variables  $x_1, \dots, x_n$ ,

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

### Deciding Satisfiability of EUF Conjunctions

given EUF conjunction  $\varphi$  with free variables  $x_1, \dots, x_n$ :

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

$\varphi = (\bigwedge P) \wedge (\bigwedge \neg N)$	unsatisfiable	
$\iff \exists x_1 \dots x_n. (\bigwedge P) \wedge (\bigwedge \neg N)$	unsatisfiable	
$\iff (\bigwedge \hat{P}) \wedge (\bigwedge \neg \hat{N})$	unsatisfiable	skolemization
$\iff \neg ((\bigwedge \hat{P}) \wedge (\bigwedge \neg \hat{N}))$	valid	$\varphi$ unsat iff $\neg \varphi$ valid
$\iff \bigwedge \hat{P} \rightarrow \bigvee \hat{N}$	valid	de Morgan
$\iff \exists s \approx t$ in $\hat{N}$ such that $\bigwedge \hat{P} \rightarrow s \approx t$	valid	semantics of $\bigvee$
$\iff \exists s \approx t$ in $\hat{N}$ such that $\bigwedge \hat{P} \models_{\tau} s \approx t$		semantics of $\models_{\tau}$

14

semantics of  $\models_{\tau}$

## Obtained Satisfiability Check

$$(\bigwedge P) \wedge (\bigwedge \neg M) \text{ unsatisfiable} \iff \exists s \approx t \text{ in } \hat{N} \text{ such that } \bigwedge \hat{P} \models_{\mathcal{T}} s \approx t$$

## Example

- 1  $g(a) \approx c \wedge f(g(a)) \not\approx f(c) \wedge c \not\approx d$ 
  - ▶ split into  $P = \{g(a) \approx c\}$  and  $N = \{f(g(a)) \approx f(c), c \approx d\}$
  - ▶ have  $g(a) \approx c \models_{\mathcal{T}} f(g(a)) \approx f(c)$ , so **unsatisfiable**
- 2  $g(a) \approx c \wedge f(g(a)) \approx f(c) \wedge g(a) \approx d \wedge c \not\approx d$ 
  - ▶ split into  $P = \{g(a) \approx c, f(g(a)) \approx f(c), g(a) \approx d\}$  and  $N = \{c \approx d\}$
  - ▶ have  $g(a) \approx c, f(g(a)) \approx f(c), g(a) \approx d \models_{\mathcal{T}} c \approx d$ , so **unsatisfiable**
- 3  $g(a) \approx c \wedge c \approx d \wedge f(x) \approx x \wedge d \not\approx g(x) \wedge f(x) \not\approx d$ 
  - ▶  $P = \{g(a) \approx c, c \approx d, f(x) \approx x\}$  and  $N = \{d \approx g(x), f(x) \approx d\}$
  - ▶ **skolemize**  $P = \{g(a) \approx c, c \approx d, f(e) \approx e\}$ ,  $N = \{d \approx g(e), f(e) \approx d\}$ 
    - ▶  $g(a) \approx c, c \approx d, f(e) \approx e \not\models_{\mathcal{T}} d \approx g(e)$
    - ▶  $g(a) \approx c, c \approx d, f(e) \approx e \not\models_{\mathcal{T}} f(e) \approx d$

so **satisfiable**

16

```

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

```

17