# Lecture 6 Introduction to SAT and SMT

Nadia Polikarpova

#### Logistics

#### Project proposals

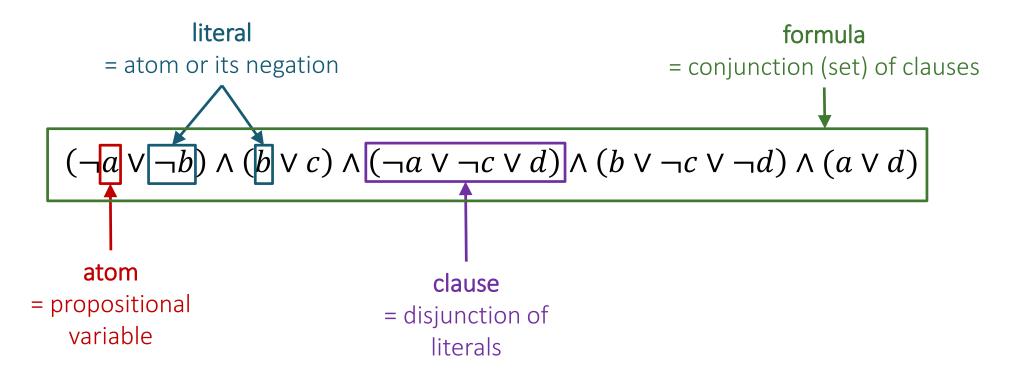
- Due Friday (Oct 20) by the end of the day
- Upload to the Proposals directory inside the shared Google folder
- Can be a Google Doc or a PDF
- File name must be "Team-N", where N is your team ID

## Why do we care?

- 1. Synthesis is combinatorial search, and so is SAT
- 2. SAT solvers are really good these days
- 3. ??? **←** this week
- 4. Profit!!!

## The SAT problem

Input: propositional formula in CNF



# The SAT problem

**Problem:** find a *satisfying assignment* (also called a *model*)

• or determine that the formula is *unsatisfiable* 

$$(\neg a \lor \neg b) \land (b \lor c) \land (\neg a \lor \neg c \lor d) \land (b \lor \neg c \lor \neg d) \land (a \lor d)$$

a satisfying assignment:

$$\{a \mapsto 0, b \mapsto 1, c \mapsto 0, d \mapsto 1\}$$

can be written as a set of literals:

$$\{\neg a, b, \neg c, d\}$$

or as a formula:

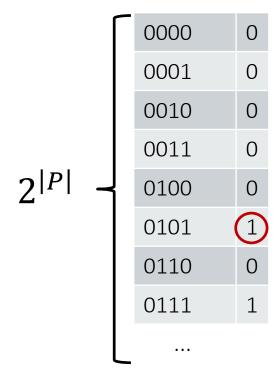
$$\neg a \land b \land \neg c \land d$$

#### Naive solution

$$(\neg a \lor \neg b) \land (b \lor c) \land (\neg a \lor \neg c \lor d) \land (b \lor \neg c \lor \neg d) \land (a \lor d)$$

#### Build a truth table!

- We can't do fundamentally better: it's an NP-complete problem
- But we can do way better in practice for common instances



#### Intuition: Sudoku

Easy vs hard: what's the difference?

| 7 | 9 |   |   |   |   | 3 |   |   |
|---|---|---|---|---|---|---|---|---|
|   |   |   |   |   | 6 | 9 |   |   |
| 8 |   |   |   | 3 |   |   | 7 | 6 |
|   |   |   | 9 | 6 | 5 |   |   | 2 |
|   |   | 5 | 4 | 1 | 8 | 7 |   |   |
| 4 |   |   | 7 | 2 | 3 |   |   |   |
| 6 | 1 |   |   | 9 |   |   |   | 8 |
|   |   | 2 | 3 |   |   |   |   |   |
|   |   | 9 |   |   |   |   | 5 | 4 |

Copyright 2005 M. Feenstra, Den Haag

|   |   | 9 | 7 | 4 | 8 |   |   |   |
|---|---|---|---|---|---|---|---|---|
| 7 |   |   |   |   |   |   |   |   |
|   | 2 |   | 1 |   | 9 |   |   |   |
|   |   | 7 |   |   |   | 2 | 4 |   |
|   | 6 | 4 |   | 1 |   | 5 | 9 |   |
|   | 9 | 8 |   |   |   | 3 |   |   |
|   |   |   | 8 |   | 3 |   | 2 |   |
|   |   |   |   |   |   |   |   | 6 |
|   |   |   | 2 | 7 | 5 | 9 |   |   |

Copyright 2005 M. Feenstra, Den Haag

Most real-world SAT instances allow a lot of inference

[Davis, Logemann, Loveland '62]

**State:** current model M (a sequence of annotated literals)

$$M = a^{d} \neg b \ c$$
 decision literal

#### **Transitions:**

- decide  $M \longrightarrow M l^d$  if / undefined in M
- unit-propagate  $M \longrightarrow M \ l$  if there is a clause where are literals are false except  $\mathit{l}$ , which is undefined
- backtrack  $Ml^dM' \longrightarrow M \neg l$  if there is a conflicting clause and M' has no decision literals
- fail  $M \longrightarrow Unsat$  if there is a conflicting clause and no decision literals

#### **DPLL**: example

$$(\neg a \lor \neg b) \land (b \lor c) \land (\neg a \lor \neg c \lor d) \land (b \lor \neg c \lor \neg d) \land (a \lor d)$$

$$M = \emptyset$$
 decide  $a^d$  unit-propagate  $a^d \neg b$  unit-propagate  $a^d \neg b \ c$  unit-propagate  $a^d \neg b \ c \ d$  backtrack  $\neg a$  unit-propagate  $\neg a \ d$  decide  $\neg a \ d \neg c^d$  unit-propagate  $\neg a \ d \neg c^d$  SAT!

#### DPLL + clause learning

$$(\neg a \lor b) \land (\neg c \lor d) \land (\neg e \lor \neg f) \land (f \lor \neg e \lor \neg b)$$

$$M = \emptyset$$

$$a^{d}$$

$$a^{d}b$$

$$a^{d}bc^{d}$$

$$a^{d}bc^{d}d$$

$$a^{d}bc^{d}de^{d}$$

$$a^{d}bc^{d}de^{d} - f$$

$$a^{d}bc^{d}d - e$$

decide
unit-propagate
decide
unit-propagate
decide
unit-propagate
decide
unit-propagate
backtrack

Wait, but why?

#### DPLL + clause learning

$$(\neg a \lor b) \land (\neg c \lor d) \land (\neg e \lor \neg f) \land (f \lor \neg e \lor \neg b) \land (\neg a \lor \neg e)$$

$$M = \emptyset$$
 decide unit-propagate  $a^d b$  Bad decision! decide  $a^d b c^d$  unit-propagate  $a^d b c^d d$  decide  $a^d b c^d d e^d$  unit-propagate  $a^d b c^d d e^d$  unit-propagate  $a^d b c^d d e^d$  backtrack Wait, but why?  $a^d b c^d d e^d$ 

#### DPLL + clause learning

$$(\neg a \lor b) \land (\neg c \lor d) \land (\neg e \lor \neg f) \land (f \lor \neg e \lor \neg b) \land (\neg a \lor \neg e)$$

$$M = \emptyset$$
 decide  $a^d$  unit-propagate  $a^d$   $b$  decide  $a^d$   $b$   $c^d$  unit-propagate  $a^d$   $b$   $c^d$  decide  $a^d$   $b$   $c^d$   $d$  decide  $a^d$   $b$   $c^d$   $d$   $e^d$  unit-propagate  $a^d$   $b$   $c^d$   $d$   $e^d$  unit-propagate  $a^d$   $b$   $c^d$   $d$   $e^d$  backjump  $a^d$   $b$   $\neg e$ 

## Beyond propositional logic

What if our formula looks like this?

$$(p \land \neg q \lor a = f(b-c)) \land (g(g(b) \neq c \lor a - c \leq 7))$$

• talks about integers, functions, sets, lists...

One idea: bit-blast everything and use SAT

- can only find solutions within bounds
- very inefficient, so bounds are small

Better idea: combine SAT with special solvers for theories

• they "natively understand" integers, functions, etc

#### First-order theories

theory = <function symbols, predicate symbols, axioms>

ground first-order formulas over functions and predicates

**Example:** theory of Equality and Uninterpreted Functions

EUF = 
$$\{f, g, h, ...\}, \{=\}, \{$$

$$\forall x. x = x$$

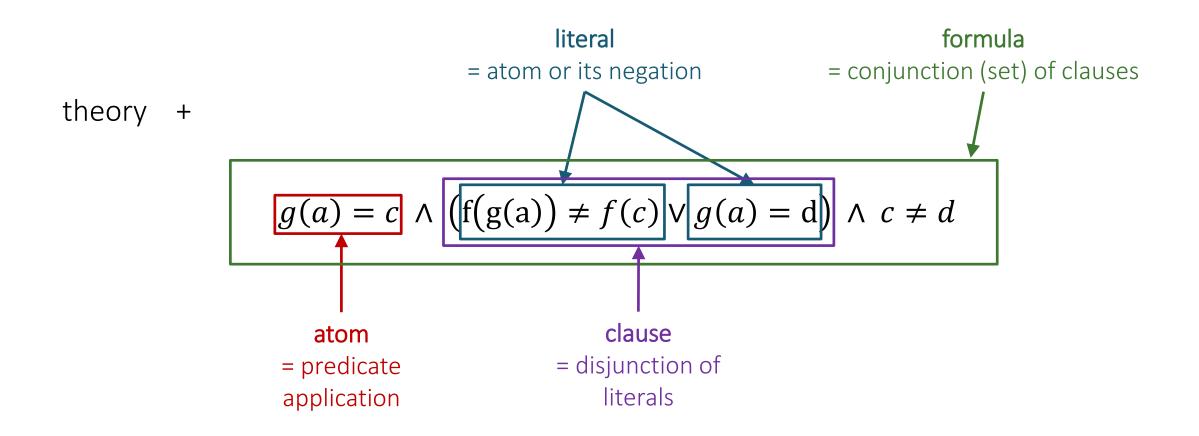
$$\forall x y. x = y \Rightarrow y = x$$

$$\forall x y z. x = y \land y = z \Rightarrow x = z$$

$$\forall x y. x = y \Rightarrow f(x) = f(y)$$

$$\} >$$

## The SMT problem



# Theories for our purpose

a = b

theory = <function symbols, predicate symbols, axioms>



can decide consistency of conjunctions of literals

$$f(a) = c$$
  
 $f(b) \neq d$   
 $c = d$ 
EUF solver
Inconsistent!

# Basic DPLL(T)

$$g(a) = c \land (f(g(a)) \neq f(c) \lor g(a) = d) \land c \neq d$$

$$\downarrow \qquad \qquad \downarrow \qquad$$

## DPLL(T) optimizations

#### Basic

Check consistency of full propositional models

Upon inconsistency, add clause and restart

Check consistency after adding a literal

#### Advanced

Check consistency of partial assignment while being built

Upon inconsistency, do conflict analysis and backjump

Add a theory-propagate rule to DPLL

• like unit-propagate, but infers all literals that follow from the theory

## Popular theories

**Equality and Uninterpreted Functions** 

 $EUF = \langle \{f, g, h, ...\}, \{=\}\}$ , axioms of equality & congruence>

Linear Integer Arithmetic

LIA =  $\{0, 1, ..., +, -\}, \{=, \leq\}$ , axioms of arithmetic>

Arrays

Arrays = 
$$\langle \text{sel, store} \rangle$$
,  $\{=\}$ ,  $\forall a \ i \ v. \text{sel(store}(a, i, v), i) = v$   
 $\forall a \ i \ j \ v. \ i \neq j \Rightarrow \text{sel(store}(a, i, v), j) = \text{sel}(a, j) >$ 

Theories can be combined!

Nelson-Oppen combination

## Popular SMT solvers

**Z3** (Microsoft): <a href="https://github.com/Z3Prover/z3/wiki">https://github.com/Z3Prover/z3/wiki</a>

CVC4 (Stanford): <a href="http://cvc4.cs.stanford.edu/web/">http://cvc4.cs.stanford.edu/web/</a>

Yices (SRI): <a href="http://yices.csl.sri.com/">http://yices.csl.sri.com/</a>

#### **SMT-LIB**

Uniform format for SMT problems understood by all solvers

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (>= (* 2 x) (+ y z)))
(declare-fun f (Int) Int)
(declare-fun g (Int Int) Int)
(assert (< (f x) (g x x)))
(assert (> (f y) (g x x)))
(check-sat)
(get-model)
```

#### Z3 demo

https://rise4fun.com/Z3/tutorial

# Why do we care?

If we can encode a synthesis problem as SAT/SMT, we can use solvers to do the search for us

Get some inspiration from how solvers search

- Unit propagation similar to top-down propagation (pruning through inference of consequences of a guess)
- Backjumping / clause learning?
- Coarse-grained reasoning and gradual refinement like in DPLL(T)?