


Lecture 6

Introduction to SAT and SMT

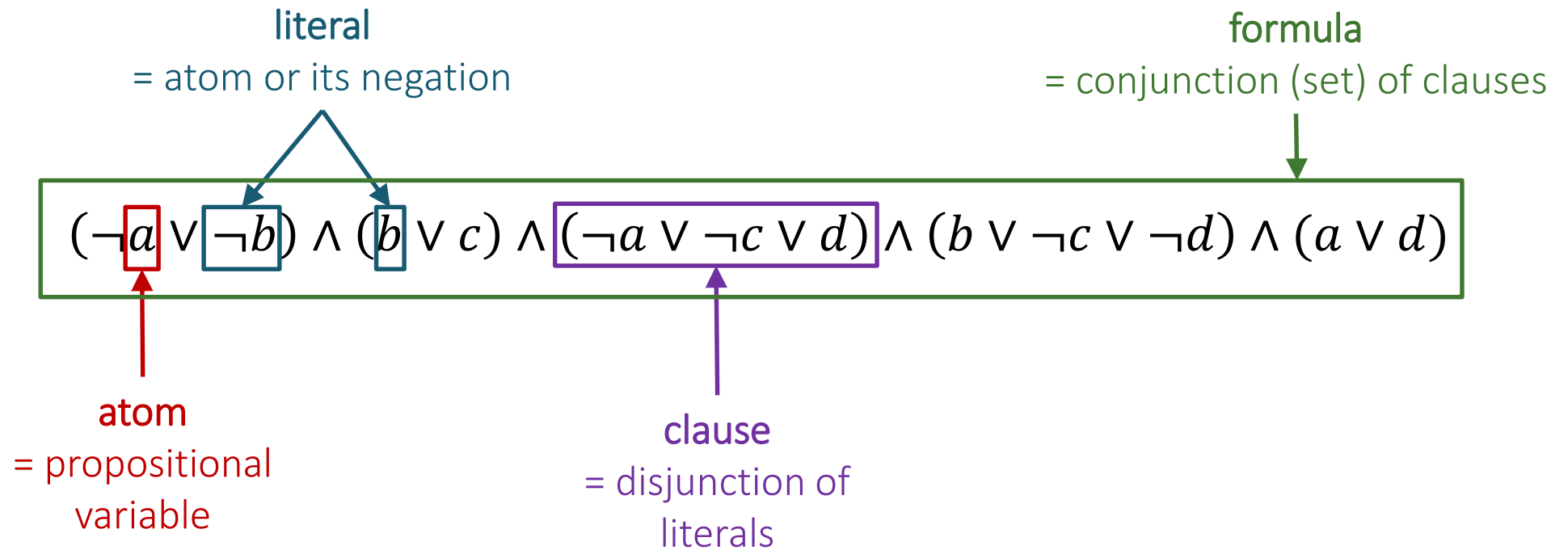
Nadia Polikarpova

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*

$$(\neg a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee d)$$

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:

$$\neg a \wedge b \wedge \neg c \wedge d$$

Naive solution

$$(\neg a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee d)$$

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

$2^{|P|}$

0000	0
0001	0
0010	0
0011	0
0100	0
0101	1
0110	0
0111	1
...	

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

DPLL algorithm

[Davis, Putnam '60]

[Davis, Logemann, Loveland '62]

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

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

Transitions:

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

DPLL: example

$$(\neg a \vee \neg b) \wedge (b \vee c) \wedge (\neg a \vee \neg c \vee d) \wedge (b \vee \neg c \vee \neg d) \wedge (a \vee d)$$

$M =$	\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
	$\neg a d \neg c^d b$	SAT!

DPLL + clause learning

$$(\neg a \vee b) \wedge (\neg c \vee d) \wedge (\neg e \vee \neg f) \wedge (f \vee \neg b) \wedge (\neg a \vee \neg e)$$

$M =$

\emptyset

a^d

$a^d b$

$a^d b c^d$

$a^d b c^d d$

$a^d b c^d d e^d$

$a^d b c^d d e^d \neg f$

$a^d b c^d d \neg e$

Bad decision!

decide

unit-propagate

decide

unit-propagate

decide

unit-propagate

backtrack

Wait, but why?

DPLL + clause learning

$$(\neg a \vee b) \wedge (\neg c \vee d) \wedge (\neg e \vee \neg f) \wedge (f \vee \neg b) \wedge (\neg a \vee \neg e)$$

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

Beyond propositional logic

What if our formula looks like this?

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

- 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, ...}, {=}, {

$$\forall x. x = x$$

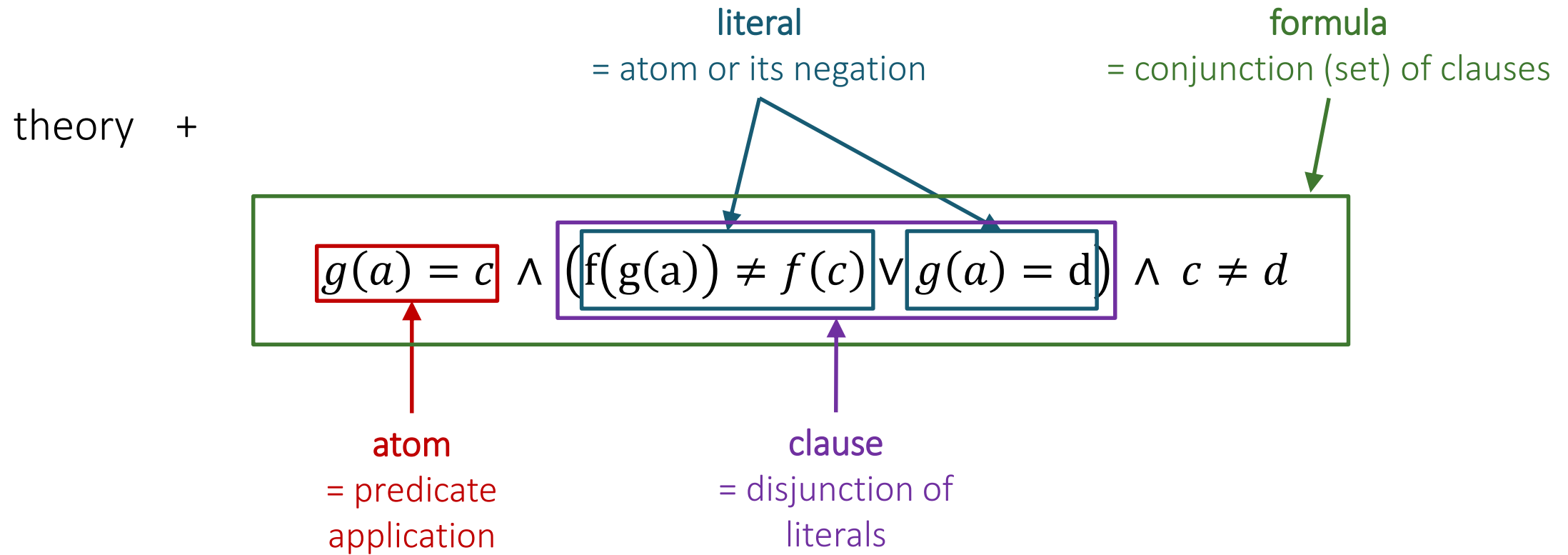
$$\forall x y. x = y \Rightarrow y = x$$

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

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

}>

The SMT problem



Theories for our purpose

theory = <function symbols, predicate symbols, ~~axioms~~>

solver

can decide consistency of
conjunctions of literals

$$f(a) = c$$

$$f(b) \neq d$$

$$c = d$$

$$a = b$$

EUf solver



Inconsistent!

Basic DPLL(T)

$$g(a) = c \wedge (f(g(a)) \neq f(c) \vee g(a) = d) \wedge c \neq d$$

abstract atoms to
propositional variables

$$p \wedge (\neg q \vee r) \wedge \neg s$$

$$p \wedge (\neg q \vee r) \wedge \neg s \xrightarrow{\text{SAT solver}} p \neg q \neg s$$

$$\xleftarrow{\text{EUF solver}} \boxed{g(a) = c} \quad \boxed{f(g(a)) \neq f(c)} \quad c \neq d$$

Inconsistent!

$$p \wedge (\neg q \vee r) \wedge \neg s \wedge (\neg p \vee q) \xrightarrow{\text{SAT solver}} p q r \neg s$$

$$\xleftarrow{\text{EUF solver}} \boxed{g(a) = c} \quad f(g(a)) = f(c) \quad \boxed{g(a) = d} \quad \boxed{c \neq d}$$

Inconsistent!

$$p \wedge (\neg q \vee r) \wedge \neg s \wedge (\neg p \vee q) \wedge (\neg p \wedge \neg r \wedge s) \xrightarrow{\text{SAT solver}} \text{Unsat}$$

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 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

$\text{EUF} = \langle \{\mathbf{f}, \mathbf{g}, \mathbf{h}, \dots\}, \{=\}, \text{axioms of equality \& congruence} \rangle$

Linear Integer Arithmetic

$\text{LIA} = \langle \{\mathbf{0}, \mathbf{1}, \dots, +, -\}, \{=, \leq\}, \text{axioms of arithmetic} \rangle$

Arrays

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

Theories can be combined!

Nelson-Oppen combination

Popular SMT solvers

Z3 (Microsoft): <https://github.com/Z3Prover/z3/wiki>

CVC4 (Stanford): <http://cvc4.cs.stanford.edu/web/>

Yices (SRI): <http://yices.csl.sri.com/>

Boolector (JKU Austria): <https://boolector.github.io/>

SMT-LIB

Uniform format for SMT problems understood by all solvers

```
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (> x 0))
(assert (> y 0))
(assert (> z 0))
(assert (> (* 2 x) (+ y z)))
(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?
 - Feng, Martins, Bastani, Dillig: [Program synthesis using conflict-driven learning](#). PLDI'18
- Coarse-grained reasoning and gradual refinement like in DPLL(T)?
 - Wang, Dillig, Singh: [Program synthesis using abstraction refinement](#). POPL'18

Next week

Topics:

- Constraint-based search

Paper: Jha, Gulwani, Seshia, Tiwari: [Oracle-guided component-based program synthesis](#)

- Questions coming soon

Project: proposals due next Friday