Lecture 3

Sat	NP-complete
Validity	coNP-complete
Model counting	#P-complete

We can describe Boolean formulas using the grammar:

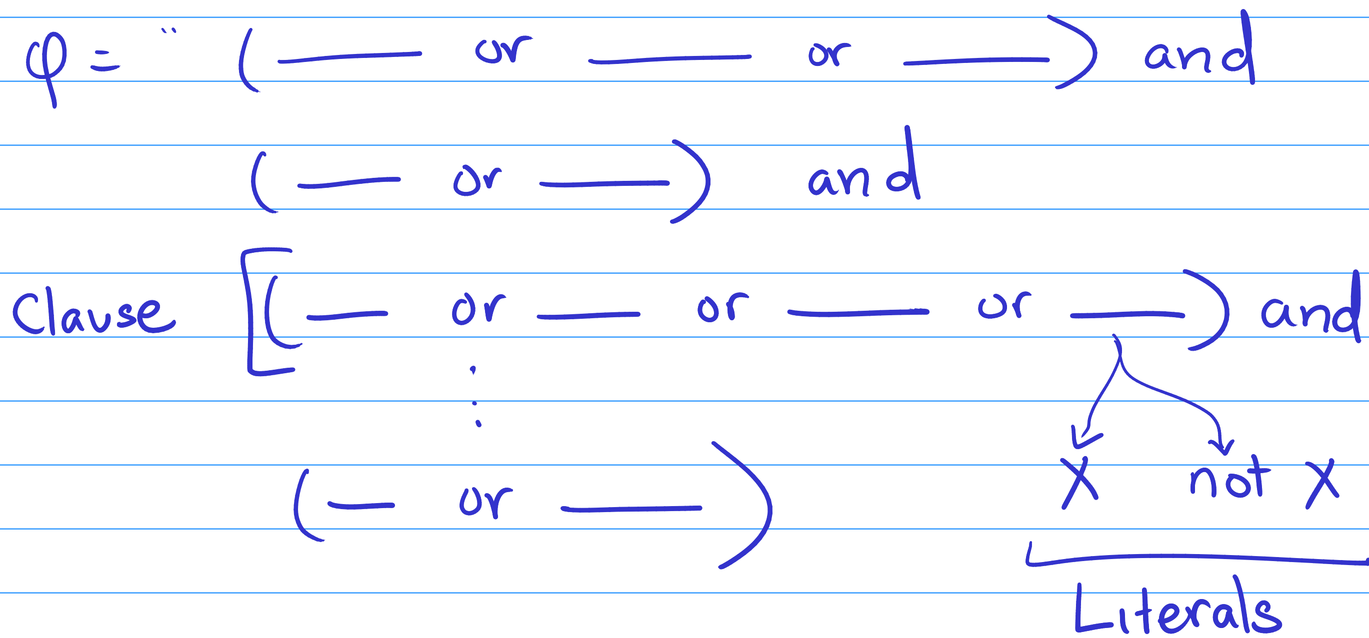
φ ::= p (Atomic proposition)
| φ_1 and φ_2 (Conjunctions)
| φ_1 or φ_2 (Disjunctions)
| not φ_1 (Negations)

x is a proposition

not (x and y) is also a proposition

(not (x or (y and z))) or (y and not x)

Conjunctive Normal Form



CNF $\varphi ::= \text{Clause}_1 \wedge \text{Clause}_2 \wedge \cdots \wedge \text{Clause}_n$ (∧ = and)

$\text{Clause} ::= \text{Lit}_1 \vee \text{Lit}_2 \vee \cdots \vee \text{Lit}_k$ (∨ = or)

$\text{Lit} ::= p \mid \text{not } p$

k-CNF : is a CNF formula where each clause is at most k literals long

k-SAT is the problem of checking if a k-CNF formula is satisfiable

---

3-SAT is NP complete

2-SAT can be solved in linear time

CNF-SAT is also NP-complete

- ## Disjunctive Normal Form

DNF $\varphi ::= D_1$ or D_2 or D_3 or ·· or D_n

Disjunct $D ::= Lit_1$ and Lit_2 and ··· and Lit_k

$Lit ::= p \mid$ not p

- **Question**: What is the complexity of DNF-sat?

Algorithm to check satisfiability of DNF φ

1. Drop all disjuncts which contain both a variable & its negation
   "x and not x"
2. If any disjuncts remain, then φ is sat
   otherwise φ is unsat.

DNF-sat can be checked in linear time

(Also, not necessarily easy to change representation)

- Representation matters!

	Formula	CNF	DNF	Circuits	ROBDD (Binary decision diagrams)
sat	NPC	NPC	Linear	NPC	Trivial
valid	coNPC	Linear	coNPC	coNPC	Trivial
How many models?	#PC	#PC	#PC	#PC	Simple
				Variable ordering	NPC.

Distributivity: φ and $(\psi$ or $\omega)$ CNF → DNF

$\equiv (\varphi$ and $\psi)$ or $(\varphi$ and $\omega)$

φ or $(\psi$ and $\omega)$ DNF → CNF

$\equiv (\varphi$ or $\psi)$ and $(\varphi$ or $\omega)$

De Morgan not $(\varphi$ and $\psi) \equiv$ (not φ) or (not ψ)

not $(\varphi$ or $\psi) \equiv$ (not φ) and (not ψ)

- **Problem**: Applying the algebraic laws to convert arbitrary formulas to CNF will require exponential time.

  If we use a CNF-SAT solver to check satisfiability of arbitrary formulas, we will spend most of our time doing the conversion.

Problem: We are given an unrestr. formula φ

We want to check satisfiability

Defn: A problem is in NP if there is a poly time translation from instances of problem to instances of 3-SAT.

Conflict!

Resolution: The conversion from arbitrary formulas to CNF formulas need not preserve equivalence. You can add vars/add clauses/..

(equivalence ← Same truth table)

It merely preserves satisfiability

3-CNF is NP complete

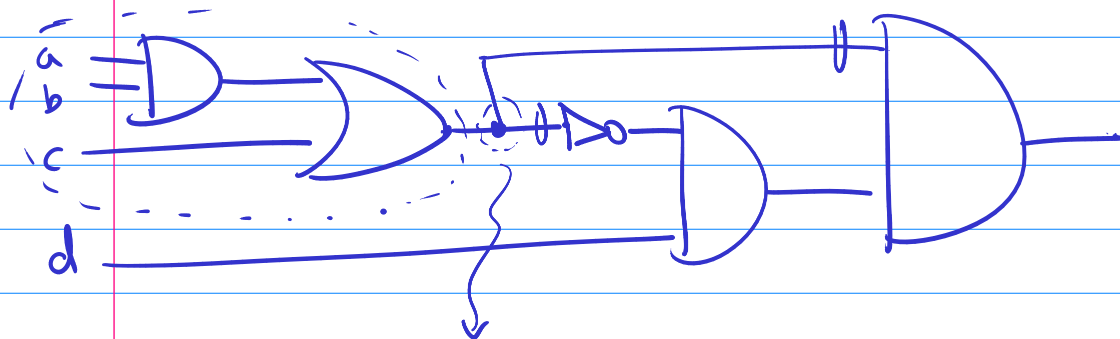
Decision Problem → [f.] (Polytime) → 3-CNF

"Yes" iff sat ← This is asking for satisfiability preservation

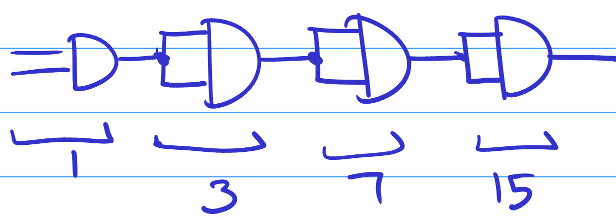
Simple algebraic conversion

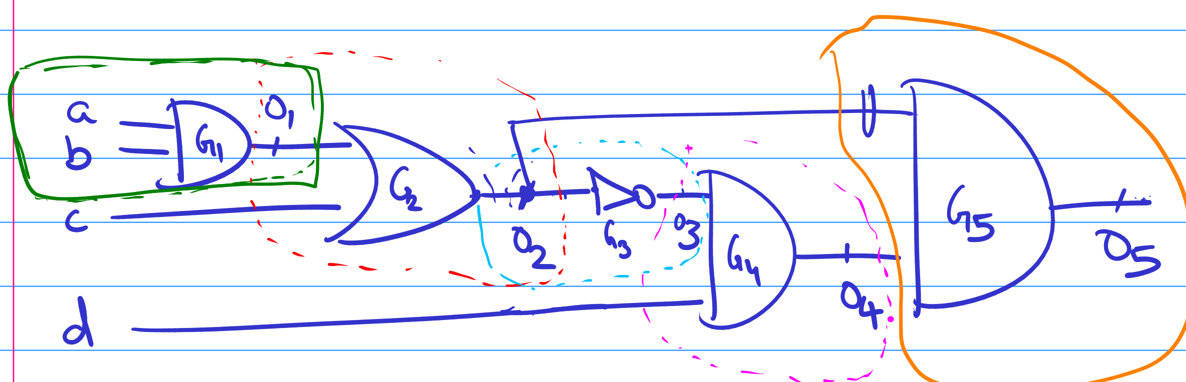
← Not asking to preserve equivalence

- Tseitin's transform



Fanout / sharing





$o_1 = a$ and b | $\bar{a} \Rightarrow \bar{o}_1$ $\quad \bar{b} \Rightarrow \bar{o}_1$ $\quad o_1 \Rightarrow a \wedge b$

Clause $[\bar{\bar{a}} \vee \bar{o}_1 \quad \bar{\bar{b}} \vee \bar{o}_1]$ $\quad \bar{o}_1 \vee (a \wedge b)$

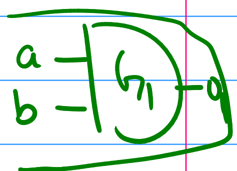
Clause

$(a \vee \bar{o}_1)$ and $(b \vee \bar{o}_1)$ and $(\bar{o}_1 \vee a)$ and $(\bar{o}_1 \vee b)$

$\overline{o_1} \Rightarrow \bar{a} \vee \bar{b}$

$\overline{\overline{o_1}} \vee \bar{a} \vee \bar{b} \quad (o_1 \vee \bar{a} \vee \bar{b})$

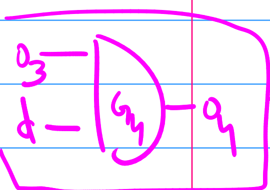
- And gate G_1 is characterized by 3 clauses:



$(a \vee \overline{o_1})$ and $(b \vee \overline{o_1})$ and $(\bar{a} \vee \bar{b} \vee o_1)$

$o_4 = o_3$ and d

$o_4 \Rightarrow o_3$ and d | $\overline{o_4} \Rightarrow \overline{o_3}$ or $\bar{d}$

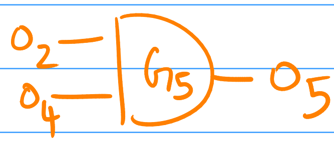


$\overline{o_4}$ or $(o_3$ and $d)$ | $\overline{o_4}$ or $\overline{o_3}$ or $\bar{d}$

- Gate G_4 is described by 3 clauses

$(o_3$ or $\overline{o_4})$ and $(d$ or $\overline{o_4})$ and $(\overline{o_3}$ or $\bar{d}$ or $o_4)$

- Describing gate G_5

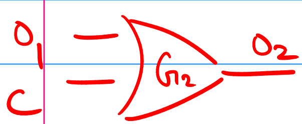


$o_5 = o_2$ and o_4

Exercise: Confirm that G_5 is described by 3 clauses:

$(o_2$ or $\overline{o_5})$ and $(o_4$ or $\overline{o_5})$ and $(\overline{o_2}$ or $\overline{o_4}$ or $o_5)$

- Describing gate G_2



$o_2 = o_1$ or c

$o_2 \Rightarrow o_1$ or c

$(\overline{o_2} \vee o_1 \vee c)$

$\overline{o_2} \Rightarrow \overline{o_1}$ and $\overline{c}$

$\overline{\overline{o_2}} \vee (\overline{o_1} \wedge \overline{c})$

$(o_2 \vee \overline{o_1}) \wedge (o_2 \vee \overline{c})$

Gate G_2 is described by 3 clauses

$(o_1 \vee c \vee \overline{o_2})$ and $(\overline{o_1} \vee o_2)$ and $(\overline{c} \vee o_2)$

- Describing gate G_3

o_2 —▷o— o_3 $\quad$ $o_3 = \text{not } o_2$

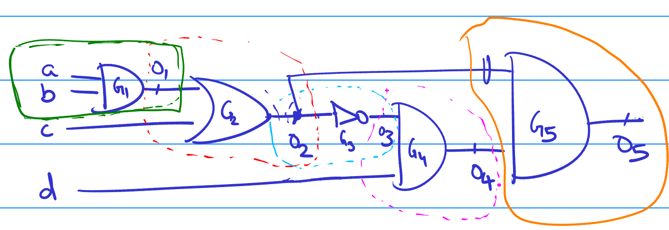
$o_3 \Rightarrow \overline{o_2}$ and $\overline{o_3} \Rightarrow o_2$

Clauses describing gate G_3:

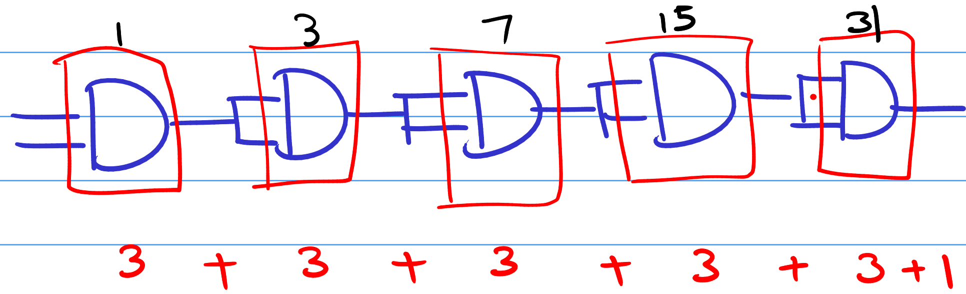
$(\overline{o_2} \vee \overline{o_3})$ and $(o_2 \vee o_3)$

Combining everything, the circuit is satisfiable iff the following formula is satisfiable:

$(a \vee \overline{o_1})$ and $(b \vee \overline{o_1})$ and $(\overline{a} \vee \overline{b} \vee o_1)$ and
$(o_1 \vee c \vee \overline{o_2})$ and $(\overline{o_1} \vee o_2)$ and $(\overline{c} \vee o_2)$ and
$(\overline{o_2} \vee \overline{o_3})$ and $(o_2 \vee o_3)$ and
$(o_3 \text{ or } \overline{o_4})$ and $(d \text{ or } \overline{o_4})$ and $(\overline{o_3} \text{ or } \overline{d} \text{ or } o_4)$ and
$(o_2 \text{ or } \overline{o_5})$ and $(o_4 \text{ or } \overline{o_5})$ and $(\overline{o_2} \text{ or } \overline{o_4} \text{ or } o_5)$ and o_5



Example:



Tseitin's transform produces a formula with 17 literals

Copy pasting produces a formula of size 31.