Sketch of Cook-Levin's theorem (Circuit version)

Saturday, June 19, 2021 12:27 AM

3. Cook-Levin's theorem.

Given $X=\{x_1, \dots, x_n\}$ boolean variable ($x_i \in \{0,1\}$)

Clause: Variable (and negation) conjugated by OR

$x_2 \vee x_3 \vee (\neg x_4)$

Formula: Clause conjugated by AND

$F = (x_2 \vee x_3 \vee (\neg x_4)) \wedge (x_1 \vee x_3 \vee x_5) \dots$

$F: \{0,1\}^n \to \{0,1\}$

Fact: Every formula can be written in this form

SAT problem: Given $X=\{x_1, \dots, x_n\}$ and formula F, decide whether it $\exists A \in \{0,1\}^n$ s.t. $F(A)=1$

3-SAT problem: SAT problem, but every Clause have at most 3 variable

Thm (Cook-Levin): 3-SAT is NP-complete

or 1. 3-SAT $\in$ NP

2. Every NP problem can be reduce to solving an instance of 3-SAT problem

"reduce": Transform solving any problem in NP $\to$ Solving the 3-SAT problem

(Aside) 3-SAT $\in$ P $\Rightarrow$ P = NP

proof (sketch): (Tseytin Transformation)

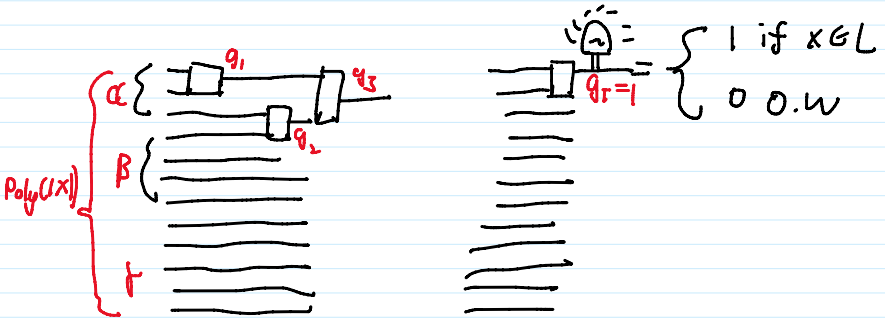
For any $L \subseteq \{0,1\}^n$. How can I "convince" you that $L \in$ NP

Fix n, consider the circuit C_n, $T=poly(n)$ gates.

If you pick some $x \in L$, $|x|=n$,

I should be able to provide some certificate y with $|y|=m=poly(n)$

How can we check the circuit actually works.



We want to encode the computational step in a 3-SAT. Formula ϕ_x

s.t: Computation step is valid $\Rightarrow \exists$ assign ϕ_x s.t $\phi_x=1$

# of variable

s.t : Computation step is valid $\Rightarrow \exists$ assign ϕ_x s.t $\phi_x = 1$

# of variable

α, β, τ Input variable $g_1 \cdots g_T$ gate variable

$\underbrace{\alpha, \beta, \tau}_{poly(n)}$

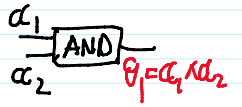
$O(poly(n))$ size

1. Start Ok: $\alpha = x, \quad \gamma = 0$

$(\alpha_i \vee x_i) \wedge (\neg\alpha_i \vee x_i) \quad \forall i \in [n]$ $poly(n)$ clause

↳ encode $\alpha_i = x_i$

2. Evolve Ok: Every step of the computation is consistant

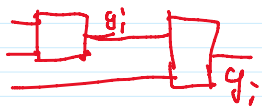
For 

x	y	x∧y
0	0	0
0	1	0
1	0	0
1	1	1

This can be encode as :

$(\neg\alpha_1 \vee \neg\alpha_2 \vee g_1) \wedge (\alpha_1 \vee \neg g_1) \wedge (\alpha_2 \vee \neg g_1)$

1, 2 bit gates $\Rightarrow O(1)$ Clause for each gate



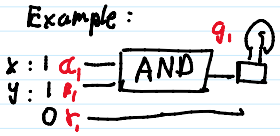
$O(T) = O(poly(n))$ clause

3. End Ok. $g_T = 1$, easy!

Let $\phi_x = (\text{Input clause}) \wedge (\text{Evolve clause}) \wedge (\text{End Ok})$

$O(poly(n))$ clause!

Example :



Start Ok:

$S = (\alpha_1 \vee 0) \wedge (\neg\alpha_1 \vee 1) \wedge (\tau_1 \vee 1) \wedge (\neg\tau_1 \vee 0)$

Evolve Ok:

$E = (\neg\alpha_1 \vee \neg\beta_1 \vee g_1) \wedge (\alpha_1 \vee \neg g_1) \wedge (\beta_1 \vee \neg g_1)$

End Ok:

$F = (g_1 \vee 0) \wedge (\neg g_1 \vee 1)$

$\phi = S \wedge E \wedge F$

Correctness :

$x \in L \Rightarrow C_{1x1}(x, y) = 1 \quad \Rightarrow$ the Boolean formula can be satisfied

$x \notin L \Rightarrow$ No matter what y I put into C_{1x1}, always reject

$\Rightarrow \phi_x$ cannot be satisfy

$\Rightarrow \forall x$ cannot be satisfy

□

One step of computation is Local! Hence only a constant # of Clause is needed!

Reference:

[SIP06]: Introduction to the Theory of Computation

[Yue 20]: The Complexity of Entanglement. Lecture 2.