Cook–Levin Theorem

CSCI 3130 Formal Languages and Automata Theory

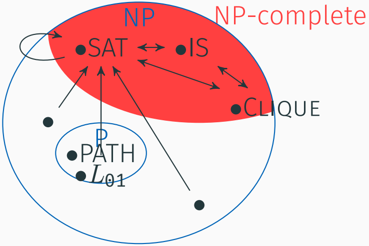
Siu On CHAN

Fall 2021

Chinese University of Hong Kong

Cook-Levin theorem (optional)

NP-completeness



Theorem (Cook–Levin)

Every language in NP polynomial-time reduces to SAT

Cook–Levin theorem

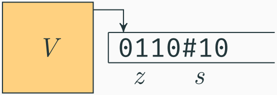
Every $L \in$ NP polynomial-time reduces to SAT

Need to find a polynomial-time reduction R such that

L		SAT
z	$\longrightarrow \boxed{R} \longrightarrow$	Boolean formula ϕ
$z \in L$	$\longleftrightarrow$	ϕ is satisfiable

NP-completeness of SAT

All we know: L has a polynomial-time verifier V



$z \in L$ if and only if
V accepts $\langle z, s \rangle$ for some s

Tableau of computation history of V

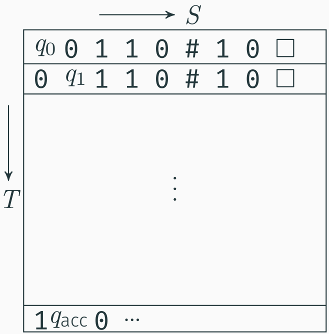
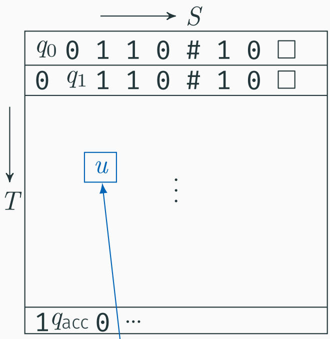


Tableau of computation history



$n =$ length of z

height of tableau is $O(n^c)$ for some constant c

width of tableau is $O(n^c)$

k possible tableau symbols

$$x_{T,S,u} = \begin{cases} \text{True} & \text{if cell } (T, S) \text{ contains symbol } u \\ \text{False} & \text{otherwise} \end{cases}$$

Reduction to SAT

L		SAT
z	$\longrightarrow \boxed{R} \longrightarrow$	Boolean formula ϕ
$z \in L$	$\longleftrightarrow$	ϕ is satisfiable

Will design a formula ϕ such that

variables of ϕ		$x_{T,S,u}$
assignment to $x_{T,S,u}$	$\approx$	assignment to tableau symbols
satisfying assignment	$\leftrightarrow$	accepting computation history
ϕ is satisfiable	$\leftrightarrow$	V accepts $\langle z, s \rangle$ for some s

Reduction to SAT

Will construct in $O(n^{2c})$ time a formula ϕ such that
$\phi(x)$ is True precisely when the assignment to $\{x_{T,S,u}\}$ represents
legal and accepting computation history

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{init}} \wedge \phi_{\text{move}} \wedge \phi_{\text{acc}}$$

ϕ_{cell} : Exactly one symbol in each cell

ϕ_{init} : First row is $q_0 z\#s$ for some s

ϕ_{move} : Moves between adjacent rows follow the transitions of V

ϕ_{acc} : Last row contains q_{acc}

q_0	0	1	1	0	#	1	0	$\square$
0	q_1	1	1	0	#	1	0	$\square$
				$\vdots$				
1	q_{acc}	0	$\cdots$					

ϕ_{cell} : exactly one symbol per cell

$$\phi_{\text{cell}} = \phi_{\text{cell},1,1} \wedge \cdots \wedge \phi_{\text{cell},\#\text{rows},\#\text{cols}} \quad \text{where}$$

$$\begin{aligned}
\phi_{\text{cell},T,S} = {} & (x_{T,S,1} \vee \cdots \vee x_{T,S,k}) && \text{at least one symbol} \\
& \left.\begin{aligned}
&\wedge \overline{(x_{T,S,1} \wedge x_{T,S,2})} \\
&\wedge \overline{(x_{T,S,1} \wedge x_{T,S,3})} \\
&\qquad \vdots \\
&\wedge \overline{(x_{T,S,k-1} \wedge x_{T,S,k})}
\end{aligned}\right\} && \text{no two symbols in one cell}
\end{aligned}$$

ϕ_{init} and ϕ_{acc}

First row is $q_0 z\#s$ for some s

$$\phi_{\text{init}} = x_{1,1,q_0} \wedge x_{1,2,z_1} \wedge \cdots \wedge x_{1,n+1,z_n} \wedge x_{1,n+2,\#}$$

Last row contains q_{acc} somewhere

$$\phi_{\text{acc}} = x_{\#\text{rows},1,q_{\text{acc}}} \vee \cdots \vee x_{\#\text{rows},\#\text{cols},q_{\text{acc}}}$$

Legal and illegal transitions windows

legal windows	illegal windows
… abx … … abx …	… $q_3$ab … … abq_3 …
… aq_3a … … $q_6$ax …	… $q_3 q_3$a … … $q_3 q_3$x …
… aba … … abq_6 …	… aq_3a … … $q_6$ab …
… aa□ … … xa□ …	… aq_3a … … aq_6x …

$$q_3 \xrightarrow{\text{a/xL}} q_6$$

ϕ_{move} : moves between rows follow transitions of V

q_0	0	1	1	0	#	1	0	□
0	q_1	1	1	0	#	1	0	□
		a_1	a_2	a_3				
		b_1	b_2	b_3				
1	q_{acc}	0	···					

$$\phi_{\text{move}} = \phi_{\text{move},1,1} \wedge \cdots \wedge \phi_{\text{move},\#\text{rows}-1,\#\text{cols}-2}$$

$$\phi_{\text{move},T,S} = \bigvee_{\text{legal } \boxed{\begin{smallmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{smallmatrix}}} \begin{pmatrix} x_{T,S,a_1} \wedge x_{T,S+1,a_2} \wedge x_{T,S+2,a_3} \wedge \\ x_{T+1,S,b_1} \wedge x_{T+1,S+1,b_2} \wedge x_{T+1,S+2,b_3} \end{pmatrix}$$

NP-completeness of SAT

$$z \longrightarrow \boxed{R} \longrightarrow \text{Boolean formula } \phi$$

$$z \in L \quad \longleftrightarrow \quad \phi \text{ is satisfiable}$$

Let V be a polynomial-time verifier for L

$R =$ **On input** z
1. Construct the formulas $\phi_{\text{cell}}, \phi_{\text{init}}, \phi_{\text{move}}, \phi_{\text{acc}}$
2. Output $\phi = \phi_{\text{cell}} \wedge \phi_{\text{init}} \wedge \phi_{\text{move}} \wedge \phi_{\text{acc}}$

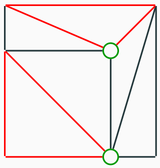
R takes time $O(n^{2c})$

V accepts $\langle z, s\rangle$ for some s if and only if ϕ is satisfiable

NP-completeness: More examples

Cover for triangles

k-cover for triangles: k vertices that touch all triangles



Has 2-cover for triangles?
Yes

Has 1-cover for triangles?
No, it has two vertex-disjoint triangles

$$\text{TRICOVER} = \{\langle G, k\rangle \mid G \text{ has a } k\text{-cover for triangles}\}$$

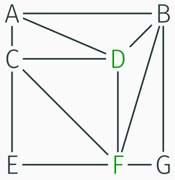
TRICOVER is NP-complete

What is a solution for TRICOVER?

A subset of vertices like {D, F}

$V =$ On input $\langle G, k, S\rangle$, where S is a set of k vertices

1. For every triple (u, v, w) of vertices:
   If $(u, v), (v, w), (w, u)$ are all edges in G:
   If none of u, v, w are in S, reject
2. Otherwise, accept



Running time = $O(n^3)$

Step 2: Some NP-hard problem reduces to TRICOVER

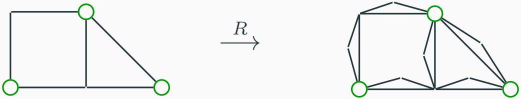
$\text{VC} = \{\langle G, k\rangle \mid G \text{ has a vertex cover of size } k\}$

Some vertex in every edge is covered

$\text{TRICOVER} = \{\langle G, k\rangle \mid G \text{ has a } k\text{-cover for triangles}\}$

Some vertex in every triangle is covered

Idea: replace edges by triangles



vertex cover in G cover for triangles in G'

VC polynomial-time reduces to TRICOVER

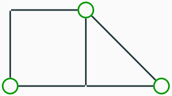
$R =$ **On input $\langle G, k\rangle$, where graph G has n vertices and m edges**

1. Construct the following graph G':
   G' has $n + m$ vertices:
   $v_1, \ldots, v_n$ are vertices from G
   introduce a new vertex u_{ij} for every edge (v_i, v_j) of G
   For every edge (v_i, v_j) of G:
   include edges $(v_i, v_j), (v_i, u_{ij}), (u_{ij}, v_j)$ in G'
2. Output $\langle G', k\rangle$

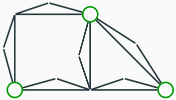
Running time is $O(n + m)$

Step 3: Argue correctness (forward)

$$\langle G, k\rangle \in \text{VC} \quad \Rightarrow \quad \langle G', k\rangle \in \text{TRICOVER}$$



$\Rightarrow$



G has a k-vertex cover S

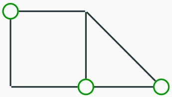
G' has a k-triangle cover S

old triangles from G are covered

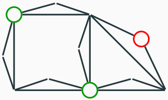
new triangles in G' also covered

Step 3: Argue correctness (backward)

$\langle G, k \rangle \in$ VC $\quad \Leftarrow \quad \langle G', k \rangle \in$ TRICOVER



$\Leftarrow$



G has a k-vertex cover S'

G' has a k-triangle cover S

S' is obtained after moving some vertices of S

Since S' covers all triangles in G', it covers all edges in G

Some vertices in S may not come from G!

But we can move them and still cover the same triangle