Cook–Levin Theorem
CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Chinese University of Hong Kong

Fall 2017
NP-completeness

Theorem (Cook–Levin)

Every language in NP polynomial-time reduces to SAT
Cook–Levin theorem

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

Need to find a polynomial-time reduction $R$ such that

$L$ \hspace{10em} SAT

$z$ \hspace{5em} $R$

$z \in L$ \hspace{10em} $\phi$ is satisfiable

Boolean formula $\phi$
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 $V$

- $T$:
  - $q_0: 0 1 1 0 \# 1 0$
  - $q_1: 1 1 0 \# 1 0$
  - $1: q_{\text{acc}} 0 \cdots$
Tableau of computation history

\[ q_0 \theta 1 1 0 \# 1 0 \square \]

\[ 0 q_1 1 1 0 \# 1 0 \square \]

\[ T \]

\[ S \]

\[ u \]

\[ 1 q_{\text{acc}} \theta \cdots \]

\[ n = \text{length of } z \]

\[ \text{height of tableau is } O(n^c) \text{ for some constant } c \]

\[ \text{width of tableau is } O(n^c) \]

\[ k \text{ possible tableau symbols} \]

\[ x_{T,S,u} = \begin{cases} 
\text{True} & \text{if cell } (T, S) \text{ contains symbol } u \\
\text{False} & \text{otherwise} 
\end{cases} \]
Reduction to SAT

\[ L \rightarrow \text{SAT} \]

Boolean formula \( \varphi \)

\[ z \in L \leftrightarrow \varphi \text{ is satisfiable} \]

Will design a formula \( \varphi \) such that

variables of \( \varphi \)

assignment to \( x_T, S, u \) \( \approx \) assignment to tableau symbols

satisfying assignment \( \iff \) accepting computation history

\( \varphi \text{ is satisfiable} \) \( \iff \) \( V \text{ accepts } \langle z, s \rangle \) for some \( s \)
Reduction to SAT

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

\[
\varphi = \varphi_{\text{cell}} \land \varphi_{\text{init}} \land \varphi_{\text{move}} \land \varphi_{\text{acc}}
\]

- $\varphi_{\text{cell}}$: Exactly one symbol in each cell
- $\varphi_{\text{init}}$: First row is $q_0 z \# s$ for some $s$
- $\varphi_{\text{move}}$: Moves between adjacent rows follow the transitions of $V$
- $\varphi_{\text{acc}}$: Last row contains $q_{\text{acc}}$
\( \varphi_{\text{cell}} : \text{exactly one symbol per cell} \)

\[
\varphi_{\text{cell}} = \varphi_{\text{cell},1,1} \land \cdots \land \varphi_{\text{cell},\#\text{rows},\#\text{cols}} \quad \text{where}
\]

\[
\varphi_{\text{cell},T,S} = (x_{T,S,1} \lor \cdots \lor x_{T,S,k}) \\
\quad \land (x_{T,S,1} \land x_{T,S,2}) \\
\quad \land (x_{T,S,1} \land x_{T,S,3}) \\
\quad \vdots \\
\quad \land (x_{T,S,k-1} \land x_{T,S,k})
\]

at least one symbol

no two symbols in one cell
$\varphi_{\text{init}}$ and $\varphi_{\text{acc}}$

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

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

Last row contains $q_{\text{acc}}$ somewhere

$\varphi_{\text{acc}} = x_{\#\text{rows},1}, q_{\text{acc}} \land \cdots \land x_{\#\text{rows},\#\text{cols}}, q_{\text{acc}}$
Legal and illegal transitions windows

legal windows

... abx ...  
... abx ...  
... a$q_3$a ...  
... $q_6$ax ...  
... aba ...  
... ab$q_6$ ...  
... aa□ ...  
... xa□ ...  

illegal windows

... q$3$ab ...  
... ab$q_3$ ...  
... q$3$q$3$a$ ...  
... q$3$q$3x ...  
... a$q_3$a ...  
... q$6$ab ...  
... a$q_3$a ...  
... a$q_6$x ...
\( \varphi_{\text{move}} : \text{moves between rows follow transitions of } V \)

\[
\begin{array}{c|cccc|c|c|c}
q_0 & 0 & 1 & 1 & 0 & # & 1 & 0 & \square \\
\hline
0 & q_1 & 1 & 1 & 0 & # & 1 & 0 & \square \\
\hline
& a_1 & a_2 & a_3 & & b_1 & b_2 & b_3 & \\
\hline
1 & q_{\text{acc}} & 0 & \cdots & \\
\end{array}
\]

\( \varphi_{\text{move}} = \varphi_{\text{move},1,1} \land \cdots \land \varphi_{\text{move},\#\text{rows}-1,\#\text{cols}-2} \)

\( \varphi_{\text{move},T,S} = \bigvee_{\text{legal}} \begin{pmatrix}
\begin{pmatrix}
x_T,S,a_1 \land x_T,S+1,a_2 \land x_T,S+2,a_3 \land 
\end{pmatrix} \\
\begin{pmatrix}
x_{T+1,S},b_1 \land x_{T+1,S+1},b_2 \land x_{T+1,S+2},b_3 
\end{pmatrix}
\end{pmatrix} \)
NP-completeness of SAT

Let $V$ be a polynomial-time verifier for $L$

$$R = \text{On input } z,$$

1. Construct the formulas $\varphi_{\text{cell}}, \varphi_{\text{init}}, \varphi_{\text{move}}, \varphi_{\text{acc}}$
2. Output $\varphi = \varphi_{\text{cell}} \land \varphi_{\text{init}} \land \varphi_{\text{move}} \land \varphi_{\text{acc}}$

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

$V$ accepts $\langle z, s \rangle$ for some $s$ if and only if $\varphi$ 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
Step 1: TRICOVER is in NP

What is a solution for TRICOVER?
A subset of vertices like \{D, F\}

\[ V = \text{On input } \langle G, k, S \rangle, \text{ where } S \text{ is a set of } k \text{ vertices} \]

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

\[ VC = \{ \langle G, k \rangle \mid G \text{ has a vertex cover of size } k \} \]
Some vertex in every edge is covered

\[ TRICOVER = \{ \langle G, k \rangle \mid G \text{ has a } k\text{-cover for triangles} \} \]
Some vertex in every triangle is covered

Idea: replace edges by triangles
VC polynomial-time reduces to TRICOVER

\[ R = \text{On input } \langle G, k \rangle, \text{ where graph } G \text{ has } n \text{ vertices and } m \text{ edges,} \]

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)

\[ \langle G, k \rangle \in VC \implies \langle G', k \rangle \in \text{TRICOVER} \]

- \( G \) 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)

\[ \langle G, k \rangle \in \text{VC} \iff \langle G', k \rangle \in \text{TRICOVER} \]

\[ G \text{ has a } k\text{-vertex cover } S' \]

\[ G' \text{ has a } k\text{-triangle cover } S \]

\[ S' \text{ is obtained after moving some vertices of } S \]

\[ S' \text{ covers all triangles in } G', \text{ it covers all edges in } G \]

\[ \text{Some vertices in } S \text{ may not come from } G!\]

\[ \text{But we can move them and still cover the same triangle} \]