Cook–Levin Theorem

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN
Fall 2019

Chinese University of Hong Kong
NP-completeness

Theorem (Cook–Levin)

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

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

$z \in L \iff \phi$ is satisfiable

$\phi$ 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 $V$

$$
\begin{array}{|c|c|c|c|c|c|c|}
\hline
q_0 & \emptyset & 1 & 1 & 0 & \# & 1 & 0 \\
\hline
q_1 & 0 & 1 & 1 & 0 & \# & 1 & 0 \\
\hline
\end{array}
$$
Tableau of computation history

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

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

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

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

\( k \) possible tableau symbols
Reduction to SAT

\[ L \quad \xrightarrow{\quad R \quad} \quad \text{SAT} \]

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

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

- \( z \in L \) if and only if \( \varphi \) is satisfiable.
- The assignment to \( x_T, S, u \) satisfying assignment \( \varphi \) is satisfiable is equivalent to the assignment to tableau symbols satisfying an accepting computation history. 
- \( V \) accepts \( \langle z, s \rangle \) for some \( s \) if \( \varphi \) is satisfiable.
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 $q_{\text{acc}}$
\( \varphi_{\text{cell}} \): exactly one symbol per cell

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

where

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

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

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

**Legal windows**

- ... abx ...
- ... abx ...
- ... a_q3\ a ...
- ... q_6\ ax ...
- ... aba ...
- ... ab_{q_6} ...
- ... aa□ ...
- ... x_{a□} ...

**Illegal windows**

- ... q_3\ ab ...
- ... ab_{q_3} ...
- ... q_3\ q_3\ a ...
- ... q_3\ q_3\ x ...
- ... a_{q_3}\ a ...
- ... q_6\ ab ...
- ... a_{q_6}\ a ...
- ... a_{q_6}\ x ...
$\varphi_{\text{move}}$ : moves between rows follow transitions of $V$

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

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

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

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

\[
R = \text{On input } z, \\
1. \text{ Construct the formulas } \varphi_{\text{cell}}, \varphi_{\text{init}}, \varphi_{\text{move}}, \varphi_{\text{acc}} \\
2. \text{ Output } \varphi = \varphi_{\text{cell}} \land \varphi_{\text{init}} \land \varphi_{\text{move}} \land \varphi_{\text{acc}}
\]

$R$ 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
$k$-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

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

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 \( G \)  \[ R \]  cover for triangles in \( G' \)
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)

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

$G$ has a $k$-vertex cover $S'$

$S'$ is obtained after moving some vertices of $S$

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

$G'$ has a $k$-triangle cover $S$

Some vertices in $S$ may not come from $G$!

But we can move them and still cover the same triangle