Theory of Formal Languages and Automata

Lecture 24

Mahdi Dolati

Sharif University of Technology

Fall 2023

December 25, 2023

Time Complexity

- Stephen Cook and Leonid Levin, 1970.
- NP-complete: If a polynomial time algorithm exists for any of these problems, all problems in NP would be polynomial time solvable.
- **Theoretical**: A researcher attempting to prove that P equals NP only needs to find a polynomial time algorithm for an NP-complete problem to achieve this goal.
- **Practical**: NP-complete is strong evidence of a problem's nonpolynomiality.

Time Complexity

Polynomial Time Reducibility

- Definitions:

A function $f : \Sigma^* \longrightarrow \Sigma^*$ is a ***polynomial time computable function*** if some polynomial time Turing machine M exists that halts with just $f(w)$ on its tape, when started on any input w.

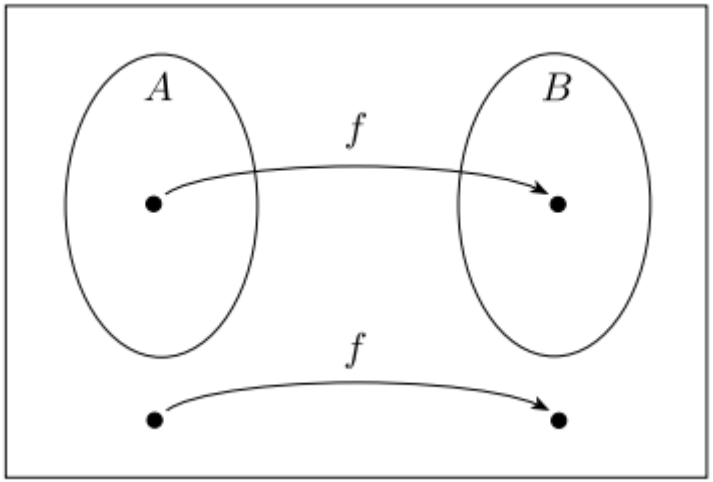
Language A is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

- Polynomial time reducibility is the **efficient** analog to mapping reducibility.

Time Complexity
Polynomial Time Reducibility



- A polynomial time reduction of A to B provides a way to convert membership testing in A to membership testing in B.

Time Complexity
Polynomial Time Reducibility

Theorem
If $A \leq_p B$ and $B \in P$, then A $\in P$.

- Proof:
  - M: Polynomial time algorithm deciding B,
  - f: Polynomial time reduction from A to B,
  - Describe a polynomial time algorithm N deciding A:

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

  - $w \in A \rightarrow f(w) \in B \rightarrow$ M accept f(w),
  - $w \notin A \rightarrow f(w) \notin B \rightarrow$ M rejects f(w),
  - N runs in polynomial time.

Time Complexity

NP-Completeness

- Definition:

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Theorem
If B is NP-complete and $B \in P$, then $\mathrm{P} = \mathrm{NP}$.

Time Complexity

NP-Completeness

Theorem
If B is NP-complete and $B \leq_p C$ for $C \in NP$, then C is NP-complete.

- Proof:
  - We must show that every $A \in NP$ is polynomial time reducible to C.
  - Since B is NPC, every language in NP is polynomial time reducible to B.
  - B, in turn, is polynomial time reducible to C.
  - Polynomial time reductions compose:
    - $A \leq_p B \leq_p C \rightarrow A \leq_p C$

Time Complexity

SAT

- The first NP-complete problem: Satisfiability problem.
- Boolean variables,
- Boolean operations AND, OR, and NOT,
- A Boolean formula is an expression involving Boolean variables and operations:

$$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$$

- A Boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.
- Satisfiability problem:

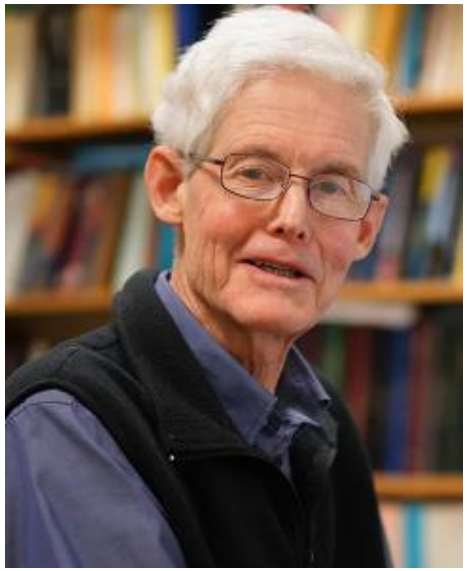
$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}.$$

Time Complexity

Cook–Levin Theorem

Theorem
SAT is NP-complete.

- Proof:
  - He and Stephen Cook independently discovered the existence of NP-complete problems.



Time Complexity

Cook–Levin Theorem

Theorem
SAT is NP-complete.

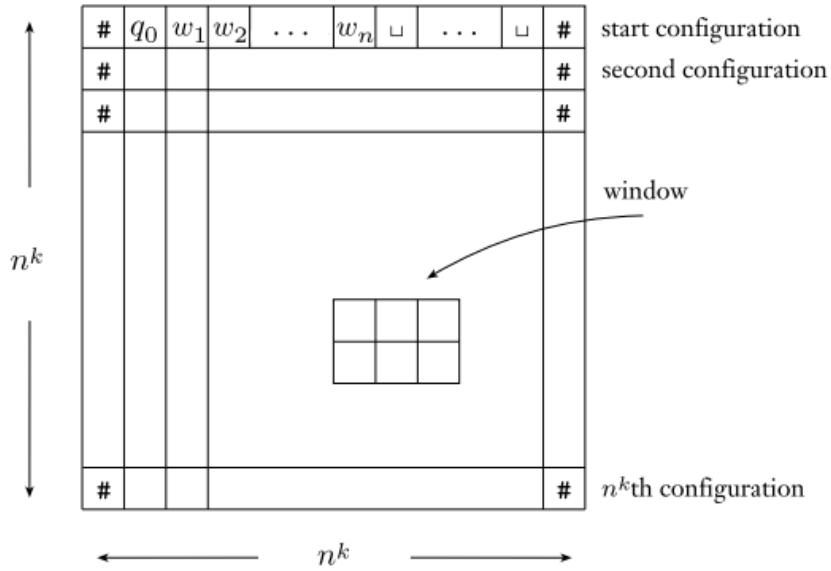
- Proof Idea:
  - SAT is in NP.
  - Any language in NP is polynomial time reducible to SAT.
  - Take a string w and produce a Boolean formula ϕ.
  - ϕ simulates the NP machine for A on input w.
  - If the machine accepts, ϕ is satisfiable.
  - If the machine rejects, ϕ is not satisfiable.

Time Complexity

Cook–Levin Theorem

Theorem
SAT is NP-complete.

- Proof:
  - N: A nondeterministic TM that decides A in n^k (A is in NP).
  - The formula corresponds to a computation history of N on input w.
  - We can represent a computation history in a $n^k \times n^k$ tableau.



Every accepting tableau for N on w corresponds to an accepting computation branch of N on w. Thus, the problem of determining whether N accepts w is equivalent to the problem of determining whether an accepting tableau for N on w exists.

Time Complexity

Cook–Levin Theorem

- Proof cont.:
  - Q: set of states of N.
  - Γ: tape alphabet of N.
  - $C = Q \cup \Gamma \cup \{\#\}$.
  - A variable for every **cell** in the tableau and symbol in C:

$$x_{i,j,s} \in \{True, False\}, (i,j) \in tableau \; and \; s \in C.$$
$$x_{i,j,s} = True \rightarrow cell[i,j] = s.$$

  - The formula:

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}.$$

Time Complexity

Cook–Levin Theorem

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right].$$

There should be at least one symbol in each cell.

There should not be more than one symbol in each cell.

Should be like this for every cell in the tableau.

- Where,

$$\bigvee_{s \in C} x_{i,j,s} = x_{i,j,s_1} \vee x_{i,j,s_2} \vee \cdots \vee x_{i,j,s_l}$$

Time Complexity

Cook–Levin Theorem

- Start configuration:

$$\phi_{\text{start}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}.$$

- Accept configuration:

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}.$$

Time Complexity

Cook–Levin Theorem

- Each row of the tableau corresponds to a configuration that **legally** follows the preceding row's configuration according to N's rules.
- We can check this by looking at windows of size 2×3.
- **Example (legal windows)**:

$$\delta(q_1, \mathtt{a}) = \{(q_1, \mathtt{b}, \mathrm{R})\}$$

$$\delta(q_1, \mathtt{b}) = \{(q_2, \mathtt{c}, \mathrm{L}), (q_2, \mathtt{a}, \mathrm{R})\}$$

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

(c)

a	a	q_1
a	a	b

(d)

#	b	a
#	b	a

(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b

Time Complexity

Cook–Levin Theorem

- Each row of the tableau corresponds to a configuration that **legally** follows the preceding row's configuration according to N's rules.
- We can check this by looking at windows of size 2×3.
- **Example (illegal windows)**:

$$\delta(q_1, \mathtt{a}) = \{(q_1, \mathtt{b}, \mathrm{R})\}$$

$$\delta(q_1, \mathtt{b}) = \{(q_2, \mathtt{c}, \mathrm{L}), (q_2, \mathtt{a}, \mathrm{R})\}$$

(a)

a	b	a
a	a	a

(b)

a	q_1	b
q_2	a	a

(c)

b	q_1	b
q_2	b	q_2

Time Complexity

Cook–Levin Theorem

- Legal movement:

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \big(\text{the } (i,j)\text{-window is legal}\big).$$

- The (i, j)-window has cell[i, j] as the upper central position.
- Express the legality of each window by:

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

Time Complexity

Cook–Levin Theorem

- Complexity of the reduction: Size of ϕ.
- Number of variables:
  - The tableau has $n^k \times n^k$ cells.
  - Each cell may contain one of $l = |C|$ symbols.
  - l is a constant that does not depend on the input.
  - Thus, the number of variables is $O(n^{2k})$.
- Size of the formula:
  - ϕ_{cell}: Constant size for each cell: $O(n^{2k})$.
  - ϕ_{start}: Constant size for each cell in the first row: $O(n^k)$.
  - ϕ_{accept}: Constant size for each cell: $O(n^{2k})$.
  - ϕ_{move}: Constant size for each cell: $O(n^{2k})$.
  - $O(\log n)$: Extra coefficient for indices.

Time Complexity
3SAT

- 3SAT:
  - A literal is a Boolean variable or a negated Boolean variable,
  - A clause is several literals connected with OR,
  - A Boolean formula is in conjunctive normal form (cnf-formulat) is it comprises several clauses connected with AND:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6}).$$

- 3cnf-formula: All clauses has three literals:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6).$$

- 3SAT:

$$3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}.$$

Time Complexity

3SAT

Theorem
3SAT is NP-complete.

- Proof:
  - We can modify every formula to be in conjunctive normal form.
  - We can replicate a variable to have three literals per clause:

$$(a_1 \vee a_2) \rightarrow (a_1 \vee a_2 \vee a_2)$$

  - We can use auxiliary variables to split large clauses into smaller ones that have at most three literals:

$$(a_1 \vee a_2 \vee \cdots \vee a_l)$$

  - Replace with:

$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_3 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee a_l)$$

"Begin at the beginning,
and go on till you come to the end:
then stop."

— Lewis Carroll, Alice in Wonderland