

# Cook-Levin Theorem

Qi'ao Chen  
21210160025@m.fudan.edu.cn

March 6, 2022

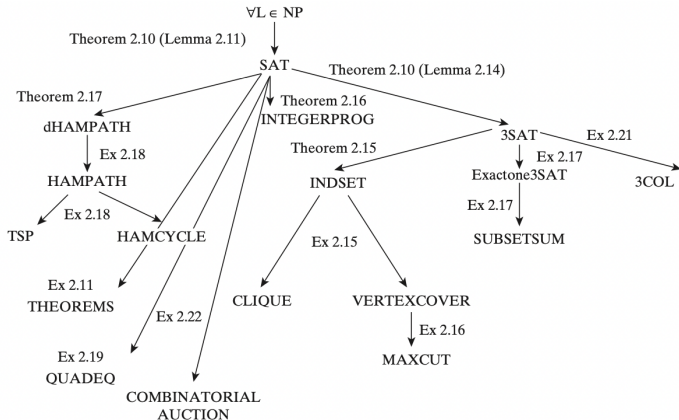
# Outline

- 1 Goal
- 2 Intro
- 3 Cook-Levin Theorem
- 4 **TODO** Total

## Theorem (Cook-Levin Theorem)

- 1 *SAT is **NP**-complete*
- 2 *3SAT is **NP**-complete*

# The web of reductions

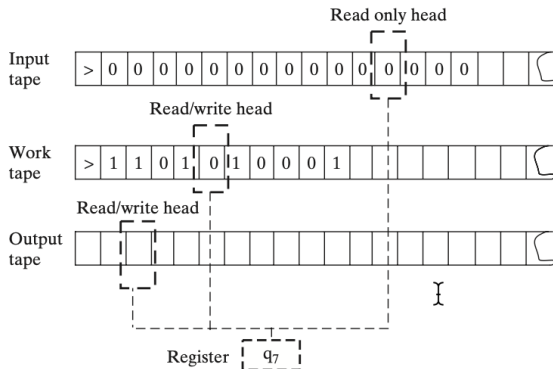


## Definition

A TM  $M$  is described by a tuple  $(\Gamma, Q, \delta)$  containing

- A finite set  $\Gamma$  of the symbols that  $M$ 's tapes can contain. We assume that  $\Gamma$  contains a designated “blank” symbol, denoted  $\square$ ; a designated “start” symbol, denoted  $\triangleright$ ; and the numbers 0 and 1. We call  $\Gamma$  the **alphabet** of  $M$
- A finite set  $Q$  of possible states  $M$  register can be in. We assume that  $Q$  contains a designated start state, denoted  $q_{\text{start}}$ , and a designated halting state, denoted  $q_{\text{halt}}$
- A function  $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ , where  $k \geq 2$ , describing the rules  $M$  use in performing each step. This function is called the **transition function** of  $M$

# Turing machine



## Definition (Computing a function and running time)

Let  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  and let  $T : \mathbb{N} \rightarrow \mathbb{N}$  be some functions, and let  $M$  be a Turing machine. We say that  $M$  **computes**  $f$  if for every  $x \in \{0, 1\}^*$  whenever  $M$  is initialized to the start configuration on input  $x$ , then it halts with  $f(x)$  written on its output tape. We say  $M$  **computes**  $f$  in  **$T(|x|)$ -time** if its computation on every input  $x$  requires at most  $T(|x|)$  steps

# The class **P**

A **complexity class** is a set of functions that can be computed within given resource bounds. We say that a machine **decides** a language  $L \subseteq \{0, 1\}^*$  if it computes the function  $f_L : \{0, 1\}^* \rightarrow \{0, 1\}$  where  $f_L(x) = 1 \Leftrightarrow x \in L$

## Definition

Let  $T : \mathbb{N} \rightarrow \mathbb{N}$  be some function. A language  $L$  is in **DTIME**( $T(n)$ ) iff there is a deterministic Turing machine that runs in time  $c \cdot T(n)$  for some constant  $c > 0$  and decides  $L$

## Definition

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c)$$



# The class **NP**

## Definition

A language  $L \subseteq \{0, 1\}^*$  is in **NP** if there exists a polynomial function  $p : \mathbb{N} \rightarrow \mathbb{N}$  and a polynomial-time TM  $M$  (called the **verifier** for  $L$ ) such that for every  $x \in \{0, 1\}^*$ ,

$$x \in L \Leftrightarrow \exists u \in \{0, 1\}^{p(|x|)} \text{ s.t. } M(x, u) = 1$$

If  $x \in L$  and  $u \in \{0, 1\}^{p(|x|)}$  satisfy  $M(x, u) = 1$ , then we call  $u$  a **certificate** for  $x$  w.r.t.  $L$  and  $M$

# Non-deterministic Turing machine

## Definition

**Non-deterministic Turing machine** has **two** transition function  $\delta_0$  and  $\delta_1$ , and a special state denoted by  $q_{\text{accept}}$ . When an NDTM  $M$  computes a function, we envision that at each computational step  $M$  makes an arbitrary choice as to which of its two transition functions to apply. For every input  $x$ , we say that  $M(x) = 1$  if there **exists** some sequence of these choices that would make  $M$  reach  $q_{\text{accept}}$  on input  $x$ . We say that  $M$  runs in  $T(n)$  time if for every input  $x \in \{0, 1\}^*$  and every sequence of nondeterministic choices,  $M$  reaches the halting state or  $q_{\text{accept}}$  within  $T(|x|)$  steps.

# The class **NP**

## Definition

For every function  $T : \mathbb{N} \rightarrow \mathbb{N}$  and  $L \subseteq \{0, 1\}^*$ , we say that  $L \in \mathbf{NTIME}(T(n))$  if there is a constant  $c > 0$  and a  $c \cdot T(n)$ -time NDTM  $M$  s.t. for every  $x \in \{0, 1\}^*$ ,  $x \in L \Leftrightarrow M(x) = 1$

## Theorem

$$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$$

## Proof.

The main idea is that the sequence of nondeterministic choices made by an accepting computation of an NDTM can be viewed as a certificate that the input is in the language, and vice versa □

# The class **NP**

## Theorem

$$\mathbf{NP} = \bigcup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$$

## Proof.

Suppose  $p : \mathbb{N} \rightarrow \mathbb{N}$  is a polynomial and  $L$  is decided by a NDTM  $N$  that runs in time  $p(n)$ . For every  $x \in L$ , there is a sequence of nondeterministic choices that makes  $N$  reach  $q_{\text{accept}}$  on input  $x$ . We can use this sequence as a **certificate** for  $x$ .

Conversely, if  $L \in \mathbf{NP}$ , then we describe a polynomial time NDTM  $N$  that decides  $L$ . On input  $x$ , it uses the ability to make nondeterministic choices to write down a string  $u$  of length  $p(|x|)$ . (Having transition  $\delta_0$  correspond to writing a 0 and  $\delta_1$  to 1). Then it runs the deterministic verifier  $\square$

## Definition

A language  $L \subseteq \{0, 1\}^*$  is **polynomial-time Karp reducible to a language**  $L' \subseteq \{0, 1\}^*$  (sometimes shortened to just “polynomial-time reducible”), denoted by  $L \leq_p L'$  if there is a polynomial-time computable function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  s.t. for every  $x \in \{0, 1\}^*$ ,  $x \in L$  iff  $f(x) \in L'$ . We say that  $L'$  is **NP-hard** if  $L \leq_p L'$  for every  $L \in \mathbf{NP}$ . We say that  $L'$  is **NP-complete** if  $L'$  is **NP-hard** and  $L' \in \mathbf{NP}$ .

We denote by SAT the language of all satisfiable CNF formulae and by 3SAT the language of all satisfiable 3CNF formulae

## Theorem (Cook-Levin Theorem)

- 1 SAT is **NP**-complete
- 2 3SAT is **NP**-complete

# Oblivious Turing machine

## Definition

Define a TM  $M$  to be **oblivious** if its head movements do not depend on the input but only on the input length. That is,  $M$  is oblivious if for every input  $x \in \{0, 1\}^*$  and  $i \in \mathbb{N}$ , the location of each of  $M$ 's heads at the  $i$ th step of execution on input  $x$  is only a function of  $|x|$  and  $i$ .

## Theorem

*For any Turing machine  $M$  that decides a language in time  $T(n)$ , there exists an oblivious Turing machine that decides the same language in  $T(n)^2$*

# A lemma

## Lemma

*For every Boolean function  $f : \{0, 1\}^l \rightarrow \{0, 1\}$ , there is an  $l$ -variable CNF formula  $\varphi$  of size  $l2^l$  s.t.  $\varphi(u) = f(u)$  for every  $u \in \{0, 1\}^l$ , where the size of a CNF formula is defined to be the number of  $\wedge/\vee$  symbols it contains*

## Proof.

For every  $v \in \{0, 1\}^l$ , there exists a clause  $C_v(z_1, \dots, z_l)$  s.t.  $C_v(v) = 0$  and  $C_v(u) = 1$  for every  $u \neq v$ .

We let  $\varphi$  be the AND of all the clauses  $C_v$  for  $v$  s.t.  $f(v) = 0$

$$\varphi = \bigwedge_{v: f(v)=0} C_v(z_1, \dots, z_l)$$

Note that  $\varphi$  has size at most  $l2^l$ . □



# Main lemma

## Lemma

*SAT is **NP**-hard*

## Proof.

Let  $L$  be an **NP** language. By definition, there is a polynomial time TM  $M$  s.t. for every  $x \in \{0, 1\}^*$ ,  $x \in L \Leftrightarrow M(x, u) = 1$  for some  $u \in \{0, 1\}^{p(|x|)}$ , where  $p : \mathbb{N} \rightarrow \mathbb{N}$  is some polynomial. We show  $L$  is polynomial-time Karp reducible to SAT by describing a polynomial-time transformation  $x \rightarrow \varphi_x$  from strings to CNF formulae s.t.  $x \in L$  iff  $\varphi_x$  is satisfiable. Equivalently

$$\varphi_x \in \text{SAT} \quad \text{iff} \quad \exists u \in \{0, 1\}^{p(|x|)} \text{ s.t. } M(x \circ u) = 1$$

where  $\circ$  denotes concatenation

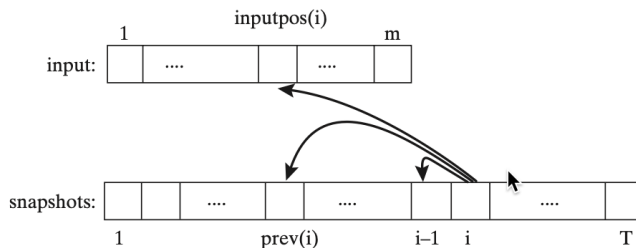


# Assumption

## Assume

- ①  $M$  only has two tapes - an input tape and a work/output tape
- ②  $M$  is an oblivious TM in the sense that its head movement does not depend on the contents of its tapes. That is,  $M$ 's computation takes the same time for all inputs of size  $n$ , and for every  $i$  the location of  $M$ 's head at the  $i$ th step depends only on  $i$  and the length of the input

# Proof



The previous discussion shows this latter condition occurs iff there exists a string  $y \in \{0, 1\}^{n+p(n)}$  and a sequence of strings  $z_1, \dots, z_{T(n)} \in \{0, 1\}^c$  (where  $T(n)$  is the number of steps  $M$  takes on inputs of length  $n + p(n)$ ) satisfying the following four conditions

- 1 The first  $n$  bits of  $y$  are equal to  $x$
- 2 The string  $z_1$  encodes the initial snapshot of  $M$ . That is,  $z_1$  encodes the triple  $\langle \triangleright, \square, q_{\text{start}} \rangle$ .
- 3 For every  $i \in \{2, \dots, T(n)\}$ ,  $z_i = F(z_{i-1}, z_{\text{prev}(i)}, y_{\text{inputpos}(i)})$ .
- 4 The last string  $z_{T(n)}$  encodes a snapshot where the machine halts and outputs 1

## Lemma

$$SAT \leq_p 3SAT$$

## Proof.

Suppose  $\varphi$  is a 4CNF. Let  $C$  be a clause of  $\varphi$ , say  $C = u_1 \vee \bar{u}_2 \vee \bar{u}_3 \vee u_4$ . We add a new variable  $z$  to the  $\varphi$  and replace  $C$  with the pair  $C_1 = u_1 \vee \bar{u}_2 \vee z$  and  $C_2 = \bar{u}_3 \vee u_4 \vee \bar{z}$ . If  $C$  is true, then there is an assignment to  $z$  that satisfies both  $C_1$  and  $C_2$ . If  $C$  is false, then no matter what value we assign to  $z$  either  $C_1$  or  $C_2$  will be false. For every clause  $C$  of size  $k > 3$ , we change it into an equivalent pair of clauses  $C_1$  of size  $k - 1$  and  $C_2$  of size 3. □

nondeterministic turing machines

Maybe some examples