

# CS244: THEORY OF COMPUTATION

Fu Song  
ShanghaiTech University

Fall 2020

# Outline

## Intractability

- Hierarchy Theorems

- Relativization

- Circuit Complexity

# Intractability

## Definition (Informal definition)

Computational problems that are solvable in principle, but the solutions require so much time or space that they can't be used in practice, are called **intractable**.

- ▶ Some problems thought to be intractable but none that have been proven to be intractable, e.g., SAT problem and all other NP-complete problems
- ▶ We give examples of problems that we can prove to be intractable
- ▶ Hierarchy theorems: relate the power of TM to the amount of time or space available for computation
- ▶ Relativization: Oracle Turing Machines
- ▶ Circuit complexity: complexity on a simplified model of the silicon chips

# Outline

## Intractability

Hierarchy Theorems

Relativization

Circuit Complexity

# Space constructible function

## Definition

A function  $f : \mathbb{N} \rightarrow \mathbb{N}$  where  $f(n) \geq O(\log n)$ , is **space constructible** if there is a  $O(f(n))$  space TM  $M$  that given an input (e.g.,  $1^n$ ) of length  $n$  writes  $f(n)$  in binary on the (work) tape.

- ▶  $f(n) = n^2$  is space constructible: given  $1^n$ , compute the binary representation of  $n$ , then multiplying  $n$  with itself. The total space used is  $O(n)$
- ▶  $f(n) = \log n$  (rounded down to the next lower integer for non-integers) is space constructible: given  $1^n$ , compute the binary representation of  $n$ , then count the number of bits of  $n$  in binary. The total space used is  $O(\log n)$
- ▶  $f(n) = \begin{cases} 2n & n \in A \\ 2n+1 & n \notin A \end{cases}$  for some  $A$  in  $\text{SPACE}(n^2) \setminus \text{SPACE}(n)$  is not **space constructible**: as determining whether  $n \in A$  or not needs  $O(n^2)$  space.

# Space hierarchy theorem

## Theorem

For any space constructible function  $f : \mathbb{N} \rightarrow \mathbb{N}$ , there is a language  $A$  decidable in  $O(f(n))$  space, but *not in  $o(f(n))$  space*.

- ▶  $g(n) = O(f(n))$  if positive integers  $c$  and  $n_0$  exist such that for every integer  $n \geq n_0$ ,  $g(n) \leq c \cdot f(n)$ .
- ▶  $g(n) = o(f(n))$  means that for any real number  $c > 0$ , a number  $n_0$  exists, where  $g(n) < c \cdot f(n)$  for all  $n \geq n_0$ .

# Proof (1)

## Theorem

For any space constructible function  $f : \mathbb{N} \rightarrow \mathbb{N}$ , there is a language  $A$  decidable in  $O(f(n))$  space, but *not in  $o(f(n))$  space*.

- ▶ Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  be a space constructible function such that  $f(n) \geq O(\log n)$ .
- ▶ We want to define a language  $A$  that can be solved in space  $O(f(n))$  but not in space  $o(f(n))$ .
- ▶ We define a  $O(f(n))$  space TM  $D$  that decides  $A$ .
- ▶ We use the diagonalization method: if  $M$  is a  $o(f(n))$  space TM, then  $D$  guarantees that  *$A$  differs from  $M$ 's language*

## Proof (2)

To design the  $O(f(n))$  space TM  $D$  that decides  $A$ , we use the diagonalization method

$D$  on input  $w$ :

1.  $n = |w|$   $[O(\log n) \text{ space}]$
  2. mark off  $f(n)$  space ( $f$  is space constructible)  $[O(f(n)) \text{ space}]$
  3. if  $w \neq \langle M \rangle 1^*$  for some TM  $M$ , **reject**  $[O(f(n)) \text{ space}]$
  4. simulate  $M$  on  $w$ :  $[d \ g(n) \text{ space}]$ 
    5. if the number of steps  $> 2^{f(n)}$ , **reject**  $[O(f(n)) \text{ space}]$
    6. if  $M$  tries to use more than  $f(n)$  space, **reject**  $[O(f(n)) \text{ space}]$
    7. if  $M$  accepts, then **reject**
    8. if  $M$  rejects, then **accept**
- Suppose  $M$  uses  $g(n) = o(f(n))$  space, then exists  $n_0$  such that  $d \ g(n) < f(n)$  for all  $n \geq n_0$
- If  $|\langle M \rangle| \geq n_0$ , then  $D(\langle M \rangle) \neq M(\langle M \rangle)$
- If  $|\langle M \rangle| < n_0$ , then  $D(\langle M \rangle 1^{n_0}) \neq M(\langle M \rangle 1^{n_0})$  ( $1^{n_0}$  gives enough space to  $D$  to simulate  $M$ )



## Corollary

For any two functions  $f_1, f_2 : \mathbb{N} \rightarrow \mathbb{N}$ , where  $f_1(n)$  is  $o(f_2(n))$  and  $f_2$  is space constructible,

$$\text{SPACE}(f_1(n)) \subset \text{SPACE}(f_2(n))$$

Separate various space complexity classes:

- ▶  $\text{SPACE}(n^{c_1}) \subset \text{SPACE}(n^{c_2})$  for all natural numbers  $c_1 < c_2$
- ▶  $\text{SPACE}(n^{c_1}) \subset \text{SPACE}(n^{c_2})$  for all positive rational numbers  $c_1 < c_2$
- ▶  $\text{SPACE}(n^{c_1}) \subset \text{SPACE}(n^{c_2})$  for all positive real numbers  $c_1 < c_2$

## Corollary

$$\text{coNL} = \text{NL} \subset \text{PSPACE}$$

$$\text{NL} = \text{NSPACE}(\log n) \subseteq \text{SPACE}(\log^2 n) (\text{Savitch's theorem}) \subset \text{SPACE}(n) \subset \text{PSPACE}$$

# Space Intractable Problems

## Definition

$$\mathbf{EXPSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(2^{n^k})$$

$$\text{SPACE}(n^k) \subset \text{SPACE}(n^{\log n}) \subset \text{SPACE}(2^n)$$

## Theorem

$$\text{PSPACE} \subset \mathbf{EXPSPACE}$$

# Time constructible function

## Definition

A function  $t : \mathbb{N} \rightarrow \mathbb{N}$  where  $t(n) \geq O(n \log n)$ , is **time constructible** if there is a  $O(t(n))$  time TM  $M$  that given an input (e.g.,  $1^n$ ) of length  $n$  writes  $t(n)$  in binary on the (work) tape.

- ▶  $t(n) = n^2$  is time constructible: given  $1^n$ , compute the binary representation of  $n$ , then multiplying  $n$  with itself. **The total time used is  $O(n)$**
- ▶  $t(n) = n \log n$  (rounded down to the next lower integer for non-integers) is time constructible: given  $1^n$ , compute the binary representation of  $n$ , then count the number of bits of  $n$  in binary. **The total time used is  $O(n)$**
- ▶  $t(n) = \begin{cases} n^2 & n \in A \\ n^2 + 1 & n \notin A \end{cases}$  for some  $A$  in  $\text{TIME}(n^3) \setminus \text{TIME}(n^2)$  is not **time constructible**: as determining whether  $n \in A$  or not needs  **$O(n^3)$  time**.

# Time hierarchy theorem

## Theorem

For any time constructible function  $t : \mathbb{N} \rightarrow \mathbb{N}$ , there is a language  $A$  decidable in  $O(t(n))$  time, but *not in  $o(\frac{t(n)}{\log t(n)})$  time*.

NOTE: We do not know how to prove that  $A$  is *not in  $o(t(n))$  time*

- ▶ We construct a  $O(t(n))$  time TM  $D$  that decides a language  $A$
- ▶ show that  $A$  cannot be decided in  $o(\frac{t(n)}{\log t(n)})$  time

# Proof

To design the  $O(t(n))$  time TM  $D$  that decides a language  $A$ , we use the diagonalization method

$D$  on input  $w$ :

1.  $n = |w|$  [ $O(t(n))$  time]
2. Compute and store  $\frac{t(n)}{\log t(n)}$  ( $t$  is time constructible) [ $O(t(n))$  time]
3. if  $w \neq \langle M \rangle 1^*$  for some TM  $M$ , **reject** [ $O(t(n))$  time]
4. simulate  $M$  on  $w$ : [ $d \cdot g(n)$  time]
5. if  $M$  uses more than  $\frac{t(n)}{\log t(n)}$  steps, **reject** [ $O(\log t(n))$  time]
6. if  $M$  accepts, then **reject**
7. if  $M$  rejects, then **accept**

► Suppose  $M$  uses  $g(n) = o(\frac{t(n)}{\log t(n)})$  time, then exists  $n_0$  such that

$$d \cdot g(n) < \frac{t(n)}{\log t(n)} \text{ for all } n \geq n_0$$

► If  $|\langle M \rangle| \geq n_0$ , then  $D(\langle M \rangle) \neq M(\langle M \rangle)$

► If  $|\langle M \rangle| < n_0$ , then  $D(\langle M \rangle 1^{n_0}) \neq M(\langle M \rangle 1^{n_0})$  ( $1^{n_0}$  gives enough time to  $D$  to simulate  $M$ )

## Corollary

For any two functions  $t_1, t_2 : \mathbb{N} \rightarrow \mathbb{N}$ , where  $t_1(n)$  is  $o(\frac{t_2(n)}{\log t_2(n)})$  and  $t_2$  is time constructible,

$$\text{TIME}(t_1(n)) \subset \text{TIME}(t_2(n))$$

Separate various space complexity classes:

- ▶  $\text{TIME}(n^{c_1}) \subset \text{TIME}(n^{c_2})$  for all natural numbers  $c_1 < c_2$
- ▶  $\text{TIME}(n^{c_1}) \subset \text{TIME}(n^{c_2})$  for all positive rational numbers  $c_1 < c_2$
- ▶  $\text{TIME}(n^{c_1}) \subset \text{TIME}(n^{c_2})$  for all positive real numbers  $c_1 < c_2$

## Definition

$$\text{EXPTIME} = \bigcup_{k \in \mathbb{N}} \text{TIME}(2^{n^k})$$

## Corollary

$$\mathbf{P} \subset \text{EXPTIME}.$$

# EXPSPACE-complete

## Definition

A language  $B$  is **EXPSPACE-complete** if

1.  $B$  is in **EXPSPACE**, and
2. every  $A \in \mathbf{EXPSPACE}$  is polynomial time reducible to  $B$ .

If  $B$  merely satisfies condition 2, then it is **EXPSPACE-hard**.

# Regular expression with exponentiation

Syntax of **regular expressions with exponentiation** is defined by the following rules,

$$r := a \mid \varepsilon \mid \emptyset \mid r_1 \cup r_2 \mid r_1 \circ r_2 \mid r^* \mid r^k.$$

**Semantics,**

- ▶  $L(a) = \{a\}$ ,  $L(\varepsilon) = \{\varepsilon\}$ ,  $L(\emptyset) = \emptyset$ ,
- ▶  $L(r_1 \cup r_2) = L(r_1) \cup L(r_2)$ ,
- ▶  $L(r_1 \circ r_2) = L(r_1) \circ L(r_2)$ ,
- ▶  $L(r^*) = (L(r))^*$ ,
- ▶  $L(r^k) = (L(r))^k = \underbrace{L(r) \circ L(r) \cdots \circ L(r)}_k$

where

- ▶  $L_1 \circ L_2 = \{uv \mid u \in L_1, v \in L_2\}$  ( $L \circ \emptyset = \emptyset \circ L = \emptyset$ ),
- ▶  $L^* = \bigcup_{i=0}^{\infty} L^i$ ,
- ▶  $L^0 = \{\varepsilon\}$ ,  $L^i = L^{i-1} \circ L$  for any  $i > 0$ .



# Regular expression with exponentiation

Syntax of **regular expressions with exponentiation** is defined by the following rules,

$$r := a \mid \varepsilon \mid \emptyset \mid r_1 \cup r_2 \mid r_1 \circ r_2 \mid r^* \mid r^k.$$

$$EQ_{\text{REX}\uparrow} = \{ \langle r_1, r_2 \rangle \mid r_1 \text{ and } r_2 \text{ are two equivalent regular expressions with exponentiation} \}$$

## Theorem

$EQ_{\text{REX}\uparrow}$  is **EXSPACE**-complete.

## $EQ_{RE\uparrow}$ is in **EXPSpace**

Naive idea:

1. transform REs with exponentiation  $r_1$  and  $r_2$  to standard REs  $r'_1$  and  $r'_2$  whose size is  $n2^n$
2. construct DFA  $A$  from  $r'_1$  and DFA  $B$  from  $r'_2$ , DFA has size at most  $2^{n2^n}$
3. construct a NFAC such that  
$$L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right),$$
 cost  $2^{n2^n}$  space
4. test emptiness of  $C$

# $EQ_{REX\uparrow}$ is in $EXPSpace=NEXPSpace$

An alternative idea:

1. transform REs with exponentiation  $r_1$  and  $r_2$  to standard REs  $r'_1$  and  $r'_2$  whose size is  $n2^n$
2. construct NFA  $A$  from  $r'_1$  and NFA  $B$  from  $r'_2$ , NFA has size at most  $O(n2^n)$
3. test equivalent on  $A$  and  $B$

$N$  on input  $\langle A, B \rangle$ :

1. place a marker each of the start states of  $A$  and  $B$
2. Repeat  $2^{|A|+|B|}$  times: [only  $2^{|A|+|B|}$  subsets of states]
3. Nondeterministically select an input symbol and change the positions of the markers on the states of  $A$  and  $B$  to simulate reading that symbol [nondeterministic linear space]
4. If at any point a marker was placed on an accepting state of one of the finite automaton and not on any accept state of the other finite automaton, **accept**. Otherwise, **reject**

$$L(A) = L(B) \iff N \text{ rejects}$$

## $EQ_{REX\uparrow}$ is **EXSPACE**-hard

- ▶ Consider a  $2^{n^k}$  space TM  $M$  for some constant  $k$  which decides a language  $A$
- ▶ For an input  $w$
- ▶  $r_1$  is the language  $\Delta^*$ , where  $\Delta = Q \cup \Gamma \cup \{\#\}$
- ▶  $r_2$  is the language over  $\Delta$  that are all non-rejecting computation histories of  $M$  on  $w$
- ▶  $r_1$  and  $r_2$  are equivalent iff  $M$  accepts  $w$

## $EQ_{REX\uparrow}$ is **EXSPACE**-hard

When a string over  $\Delta$  fails to be a non-rejecting computation history?

1. not start with the start configuration:  $r_{\text{bad-start}}$  [not a computation history]
2. not end with the rejecting configuration:  $r_{\text{bad-reject}}$  [non rejecting computation history]
3. configuration does not follow from the preceding one:  $r_{\text{bad-window}}$  [not a computation history]

$$r_2 = r_{\text{bad-start}} \cup r_{\text{bad-reject}} \cup r_{\text{bad-window}}$$

## $EQ_{REX\uparrow}$ is **EXSPACE**-hard

When a string over  $\Delta$  fails to be a non-rejecting computation history?

1. not start with the start configuration: [not a computation history]

$$r_{\text{bad-start}} = S_0 \cup S_1 \cup \dots \cup S_n \cup S_b \cup S_{\#}$$

2.  $S_0 = (\Delta \setminus \{q_0\})\Delta^*$  [not start with the starting state]
3.  $S_i = \Delta^i(\Delta \setminus \{w_i\})\Delta^*$  for  $1 \leq i \leq n$  [the  $(i+1)$ -th cell is not  $w_i$ ]
4.  $S_b = \Delta^{n+1}(\Delta \cup \{\varepsilon\})^{2^{n^k} - n - 2}(\Delta \setminus \sqcup)\Delta^*$  [some cell between  $n+2$  and  $2^{n^k}$  is not  $\sqcup$ ]
5.  $S_{\#} = \Delta^{2^{n^k}}(\Delta \setminus \{\#\})\Delta^*$  [ $\#$  is not in  $(2^{n^k} + 1)$ -th cell]

# $EQ_{REX\uparrow}$ is **EXSPACE**-hard

When a string over  $\Delta$  fails to be a non-rejecting computation history?

1. not start with the start configuration:

$$r_{\text{bad-start}} = S_0 \cup S_1 \cup \dots \cup S_n \cup S_b \cup S_{\#} \quad [\text{not a computation history}]$$

2. not end with the rejecting configuration:  $r_{\text{bad-reject}} = (\Delta \setminus \{q_{\text{accept}}\})^*$   
[non rejecting computation history]

# $EQ_{REX\uparrow}$ is **EXSPACE**-hard

When a string over  $\Delta$  fails to be a non-rejecting computation history?

1. not start with the start configuration:

$$r_{\text{bad-start}} = S_0 \cup S_1 \cup \dots \cup S_n \cup S_b \cup S_{\#} \quad [\text{not a computation history}]$$

2. not end with the rejecting configuration:  $r_{\text{bad-reject}} = (\Delta \setminus \{q_{\text{accept}}\})^*$   
[non rejecting computation history]

3. configuration does not follow from the preceding one: [not a computation history]

$$r_{\text{bad-window}} = \bigcup_{\text{bad}(avc, def)} \Delta^* abc \Delta^{2^{n^k} - 2} def \Delta^*$$

where  $abc$  does not yield  $def$  according to the transition function  
(similar to Cook-Levin theorem)

$$r_2 = r_{\text{bad-start}} \cup r_{\text{bad-reject}} \cup r_{\text{bad-window}}$$

Size of  $r_2$  is polynomial in  $n$



# EXPTIME-complete

## Definition

A language  $B$  is EXPTIME-complete if

1.  $B$  is in EXPTIME, and
2. every  $A \in \text{EXPTIME}$  is polynomial time reducible to  $B$ .

If  $B$  merely satisfies condition 2, then it is EXPTIME-hard.

# Outline

## Intractability

Hierarchy Theorems

Relativization

Circuit Complexity

# Oracle Turing Machines

The meaning of the word **Oracle** according to Webster: a person (as a priestess of ancient Greece) through whom a deity is believed to speak; a shrine in which a deity reveals hidden knowledge or the divine purpose through such a person

## Definition

**Oracle machines** are TMs  $O^A$  that are given access to a **black box** or **oracle**  $O_A$  that can solve the decision problem for some language  $A$ .

## Definition

An **oracle Turing machine** (OTM)  $M^O$  is an ordinary TM with an extra tape called the **oracle** tape with the capability of querying whether the string on the oracle tape is in  $O$  or not.

## New Classes: $P^O$ and $NP^O$

### Definition

For every oracle  $O$ ,  $P^O$  is the set of all languages that can be decided by a poly-time deterministic TM with oracle access to  $O$ .

$NP^O$  is the set of all languages that can be decided by a poly-time non-deterministic TM with oracle access to  $O$ .

## Theorem

$\overline{\text{SAT}}$  is in  $\mathbf{P}^{\text{SAT}}$ , where  $\overline{\text{SAT}}$  denotes the language of unsatisfiable formulae.

1. Given oracle access to SAT,
2. A deterministic poly-time  $M^O$  can query its oracle whether  $\phi \in \text{SAT}$
3. Then give the opposite answer.

## Theorem

$\mathbf{P} = \mathbf{P}^O$  for all  $O \in \mathbf{P}$

1. It trivially follows  $\mathbf{P} \subseteq \mathbf{P}^O$ .
2. If  $O \in \mathbf{P}$ , then replace each oracle call with a deterministic poly-time computation of  $O$ .
3. Number of oracle calls can be polynomial and product of two polynomials is another polynomial. Hence,  $\mathbf{P}^O \subseteq \mathbf{P}$ .

### Theorem

$\text{EXPTIME} = \mathbf{P}^O = \mathbf{NP}^O$  for all  $O \in \text{EXPTIME}$

1. It trivially follows  $\text{EXPTIME} \subseteq \mathbf{P}^O \subseteq \mathbf{NP}^O$ .
2. If  $O \in \text{EXPTIME}$ , then replace each oracle call with a deterministic exp-time computation of  $O$ .
3. Number of oracle calls can be exponential and product of two exponentials is another exponential. Hence,  $\mathbf{NP}^O \subseteq \text{EXPTIME}$ .

### Theorem

$\mathbf{NP} \subseteq \mathbf{P}^{\text{SAT}}$  and  $\mathbf{coNP} \subseteq \mathbf{P}^{\text{SAT}}$ .

1. It trivially follows  $\mathbf{NP} \subseteq \mathbf{P}^{\text{SAT}}$ .
2.  $\mathbf{coNP} \subseteq \mathbf{co(P}^{\text{SAT}}) = \mathbf{P}^{\text{SAT}}$



# Quiz

Two Boolean formulas  $\phi$  and  $\psi$  are **equivalent** if the formulas have the same value on any assignment to the boolean variables. A formula is **minimal** if no smaller formula is equivalent to it.

$$\text{NONMIN} = \{\langle \phi \rangle \mid \phi \text{ is not a minimal Boolean formula}\}$$

## Theorem

$\text{NONMIN} \in \mathbf{NP}^{\text{SAT}}$ .

# Relativization

We go back to [diagonalization](#) and its limits. For that, we use the concept of [relativization](#).

## Definition (Relativization)

- ▶ The OTM  $M^O$  introduced earlier allows us to [magically](#) solve the SAT problem in one step if our  $O$  was SAT.
- ▶ Irrespective of the relation between **P** and **NP**, this OTM can solve any **NP** problem in deterministic polynomial time because every problem in **NP** is poly-time reducible to SAT.
- ▶ Such a OTM is said to be computing [relative](#) to SAT. This is the significance of the term [relativization](#).

# Limits of the Diagonalization Method

Our goal is to show the **limits of diagonalization** in proving the **P** vs **NP** question. To that end, we will prove the following theorem:

## Theorem

*There exist oracles  $A$  and  $B$  such that  $\mathbf{P}^A \neq \mathbf{NP}^A$  and  $\mathbf{P}^B = \mathbf{NP}^B$ .*

- ▶ But, how does the above theorem help?
- ▶ The diagonalization method is a simulation of one TM  $M$  by another TM  $U$ .  $U$  can determine the behaviour of  $M$  and then behave differently.
- ▶ Suppose,  $M$  and  $U$  were given identical oracles  $O$
- ▶ Now, whenever,  $M$  queries the oracle, so can  $U$  and the simulation can proceed as before
- ▶ So, any theorem proved about TMs using only diagonalization would still hold if both machines were given the same oracle.

# Limits of the Diagonalization Method

## Theorem

*There exist oracles  $A$  and  $B$  such that  $\mathbf{P}^A \neq \mathbf{NP}^A$  and  $\mathbf{P}^B = \mathbf{NP}^B$ .*

- ▶ If we could show  $\mathbf{P} \neq \mathbf{NP}$  by the diagonalization method, we conclude that they are different relative to any oracle as well, namely  $\mathbf{P}^B \neq \mathbf{NP}^B$
- ▶ But  $\mathbf{P}^B = \mathbf{NP}^B$ , so the conclusion is false
- ▶ On the other side, no proof that relies on simulation (as in diagonalization) can show that  $\mathbf{P} = \mathbf{NP}$  because  $\mathbf{P}^A \neq \mathbf{NP}^A$ .

# Proof

## Theorem

*There exist oracles  $A$  and  $B$  such that  $\mathbf{P}^A \neq \mathbf{NP}^A$  and  $\mathbf{P}^B = \mathbf{NP}^B$ .*

1. How to fix  $B$
2. We have shown that  $\text{EXPTIME} = \mathbf{P}^O = \mathbf{NP}^O$  for all  $O \in \text{EXPTIME}$
3.  $O$  can be any PSPACE-complete language

# Proof

## Theorem

There exist oracles  $A$  and  $B$  such that  $\mathbf{P}^A \neq \mathbf{NP}^A$  and  $\mathbf{P}^B = \mathbf{NP}^B$ .

1. How to fix  $A$ ?
2. Consider  $L_A = \{1^n \mid \exists w \in A. |w| = n\}$ ,  $L_A$  is in  $\mathbf{NP}^A$  for any  $A$
3. It remains to show that  $L_A \notin \mathbf{P}^A$
4. Let  $M_1, M_2, \dots$  (countable) be all the possible polynomial time oracle (A) TMs with time complexity  $f_1, f_2, \dots$
5. Let  $A = \bigcup_{i \geq 1} A_i$ , where  $A_0 = \emptyset$  and  $n_0 = 1$ , for every  $i \geq 1$ ,
  - ▶ run  $M_i$  on  $1^{n_i}$  for the minimum  $n_i$  such that  $2^{n_i} > f_i(n_i)$  and  $n_i > n_{i-1}$  ( $M_i$  cannot query its oracle on all strings of length  $n_i$ )
  - ▶ let  $S$  be the set of strings with length  $n_i$  that are queried by  $M_i$  to the oracle, let  $x \in \{0, 1\}^{n_i} \setminus S$  (such  $x$  must exist)
  - ▶ If  $M_i$  accepts  $1^{n_i}$ ,  $A_i = A_{i-1}$   $[\exists w \in A_i \text{ with } |w| = n_i]$
  - ▶ If  $M_i$  rejects  $1^{n_i}$ ,  $A_i = A_{i-1} \cup \{x\}$   $[x \in A_i \wedge |x| = n_i]$
6. Thus, no  $i$  exists such that  $M_i$  recognize  $L_A$ .

# Outline

## Intractability

Hierarchy Theorems

Relativization

Circuit Complexity

# Boolean circuits

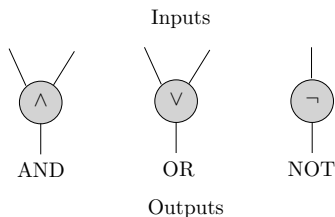
- ▶ Computers are built from electronic devices wired together in a design called a **digital circuit**.
- ▶ We can also simulate theoretical models, such as Turing machines, with the theoretical counterpart to digital circuits, called **Boolean circuits**.
- ▶ Two purposes are served by establishing the connection between TMs and Boolean circuits.
  - ▶ First, researchers believe that circuits provide a convenient computational model for attacking the **P** versus **NP** and related questions.
  - ▶ Second, circuits provide an alternative proof of the Cook-Levin theorem that SAT is **NP**-complete.



# Boolean circuits

## Definition

A **Boolean circuit** is a collection of **gates** and **inputs** connected by wires. Cycles aren't permitted. Gates take three forms: **AND** gates, **OR** gates, and **NOT** gates



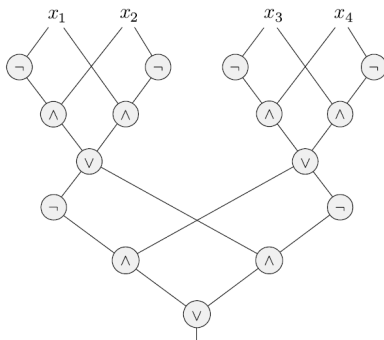
- ▶ A Boolean circuit  $C$  with  $n$  inputs and 1 outputs can be modeled as a function  $f_C : \{0, 1\}^n \rightarrow \{0, 1\}$
- ▶ A Boolean circuit  $C$  with  $n$  inputs and  $k$  outputs can be modeled as a function  $f_C : \{0, 1\}^n \rightarrow \{0, 1\}^k$

## Example

The  $n$ -input parity function  $\text{parity}_n : \{0,1\}^n \rightarrow \{0,1\}$  outputs 1 if an odd number of 1's appear in the input variables.

$$\text{parity}_n(x_1, \dots, x_n) = \bigoplus_{i=1}^n x_i$$

$$x \oplus y = (\neg x \wedge y) \vee (x \wedge \neg y)$$



# Boolean family

## Definition

A **circuit family**  $C$  is an infinite list of circuits,  $(C_0, C_1, C_2, \dots)$ , where  $C_n$  has  $n$  input variables. We say that  $C$  decides a language  $A$  over  $\{0, 1\}$  if for every string  $w$ ,

$$w \in A \iff C_n(w) = 1$$

where  $n$  is the length of  $w$ .

# Complexity of Boolean family

- ▶ The **size** of a circuit  $C$  is the number of gates that it contains
- ▶ A circuit is size **minimal** if no smaller circuit is equivalent to it
  - ▶ The problem of minimizing circuits has obvious engineering applications but is very difficult to solve in general
  - ▶ Even the problem of testing whether a particular circuit is minimal does not appear to be solvable in **P** or in **NP**.
- ▶ A circuit family is **minimal** if every  $C_i$  on the list is a **minimal**
- ▶ The **size complexity** of a circuit family  $(C_0, C_1, C_2, \dots)$  is the function  $f : \mathbb{N} \rightarrow \mathbb{N}$ , where  $f(n)$  is the size of  $C_n$
- ▶ The **depth** of a circuit is the **length** (number of wires) of the **longest** path from an input variable to the output gate. **Depth minimal circuits** and **circuit families**, and the **depth complexity** of circuit families are similar

## Definition

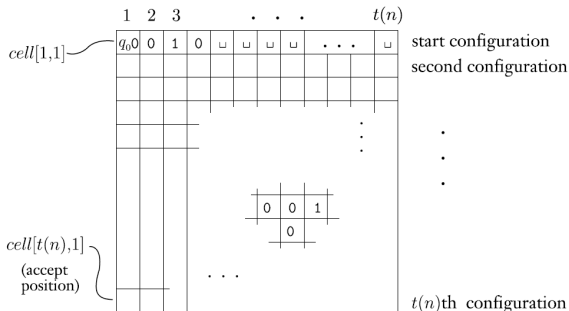
The **circuit complexity** of a language is the **size complexity** of a **minimal** circuit family for that language.

The **circuit depth complexity** of a language is defined similarly, using **depth** instead of size.

## Theorem

*Let  $t : \mathbb{N} \rightarrow \mathbb{N}$  be a function, where  $t(n) \geq n$ . If  $A \in \text{TIME}(t(n))$ , then  $A$  has circuit complexity  $O(t^2(n))$ .*

# Proof (1)



1.  $M$  be a TM that decides  $A$  in time  $t(n)$  and assume that the tape alphabet of  $M$  is  $\{0, 1\}$
2.  $M$  moves head onto the leftmost and writes  $\sqcup$  on that cell prior to entering the accept state,  $q_{accept}\sqcup$ , and stays here forever, so that we can designate a gate in the final row of the circuit to be the output gate
3. The state and the tape symbol under the tape head by a single composite character, e.g.,  $q_00$

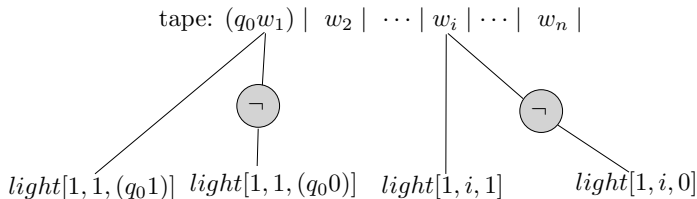
## Proof (2)

1. Construct a circuit  $C_n$  that simulates  $M$  on inputs of length  $n$ 
  - ▶ Start configuration  $cell[1, 1], cell[1, 2], \dots, cell[1, n]$
  - ▶ Accepting configuration look at  $cell[t[n], 1]$
  - ▶ Configuration moving using **windows** of tableau,  
 $cell[i, j - 1], cell[i, j], cell[i, j + 1]$  decides  $cell[i + 1, j]$  from transition of  $M$
2. The gates of  $C_n$  are organized in rows, **one row for one configuration**
3. Each row is wired into the previous row so that it can calculate its configuration from the previous row's configuration
4.  $|Q \times \Gamma| + |\Gamma|$  lights for each cell
5.  **$light[i, j, s]$  is on iff  $cell[i, j]$  contains the symbol  $s$**
6. then  $O(t^2(n))$  lights

## Proof (3)

Start configuration  $cell[1, 1], cell[1, 2], \dots, cell[1, n]$

$$w = w_1 w_2 \cdots w_n \text{ over } \{0, 1\}$$

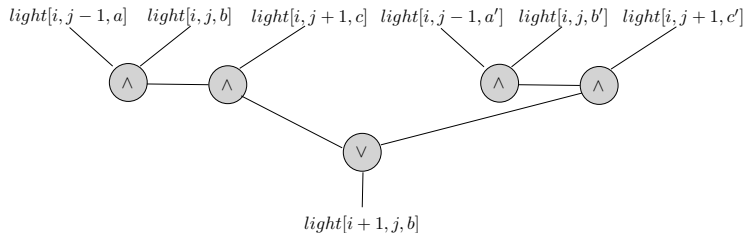




## Proof (4)

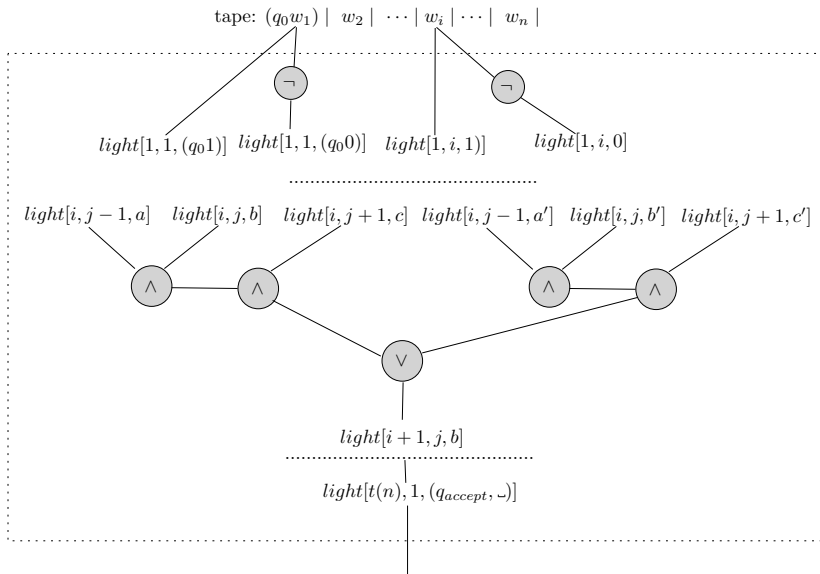
Configuration moving:  $cell[i, j - 1], cell[i, j], cell[i, j + 1]$  decides  $cell[i + 1, j]$

If  $cell[i, j - 1], cell[i, j], cell[i, j + 1]$  contain  $a, b, c$  or  $a', b', c'$ , then,  $cell[i + 1, j]$  contains  $b$  according to  $\delta$



# Proof (5)

Circuit  $C_n$



# Circuit-satisfiability

A Boolean circuit is **satisfiable** if some setting of the inputs causes the circuit to output 1.

## Definition

The **circuit-satisfiability problem** tests whether a circuit is satisfiable. Let

$$\text{CIRCUIT-SAT} = \{\langle C \rangle \mid C \text{ is a satisfiable Boolean circuit}\}$$

## Theorem

*CIRCUIT-SAT is **NP**-complete.*

# Circuit-satisfiability

## Theorem

*CIRCUIT-SAT is **NP**-complete.*

1. It is easy to show that CIRCUIT-SAT is in **NP**, whether  $C$  is true or not under an assignment of inputs can be done in PTIME
2. **NP**-hardness. Consider a language  $A$  is **NP**-complete. Construct a polynomial-time reduction  $f : A \rightarrow \text{CIRCUIT-SAT}$  such that

$$w \in A \iff f(w) \in \text{CIRCUIT-SAT}$$

- 2.1  $A$  is in **NP**, then it has a polynomial time verifier  $V$  whose input has the form  $\langle w, c \rangle$ , where  $c$  is certificate
- 2.2 Construct a circuit  $C$  from  $V$  as previously in polynomial time
- 2.3 Fill in the inputs to the circuit  $C$  that correspond to  $w$
- 2.4 Only remaining inputs to the circuit correspond to the certificate  $c$   
 $C$  is satisfiable (the assignment is a certificate  $c$ ) iff  $w \in A$

# An alternative proof of the Cook-Levin theorem

## Theorem

3SAT is **NP**-complete.

- ▶ 3SAT is obviously in **NP**
- ▶ We show that all languages in **NP** are polynomial-time reducible to 3SAT by reducing CIRCUIT-SAT to 3SAT

# Proof

## Theorem

3SAT is **NP**-complete.

- ▶ Given a circuit  $C$ . Let  $x_1, \dots, x_n$  are inputs of  $C$  and  $g_1, \dots, g_m$  are gates of  $C$
- ▶ Construct a 3SAT formula over variables  $y_1, \dots, y_n, y_{n+1}, \dots, y_{n+m}$
- ▶ For each NOT-gate  $y_j = \neg y_i$ : we have

$$(y_i \rightarrow \neg y_j) \wedge (\neg y_i \rightarrow y_j) \equiv (y_i \vee y_j) \wedge (\neg y_i \vee \neg y_j)$$

- ▶ For each AND-gate  $y_k = y_i \wedge y_j$ : we have

$$(y_i \wedge y_j \leftrightarrow y_k) \equiv (\neg y_i \vee \neg y_j \vee y_k) \wedge (\neg y_k \vee y_i) \wedge (\neg y_k \vee y_j)$$

- ▶ For each OR-gate  $y_k = y_i \vee y_j$ : we have

$$(y_i \vee y_j \leftrightarrow y_k) \equiv (y_i \vee y_j \vee \neg y_k) \wedge (\neg y_i \vee y_k) \wedge (\neg y_j \vee y_k)$$