

CS-E4530

# Computational Complexity Theory

**Janne H. Korhonen**

Aalto University

Spring 2018

Lecture 14:

# **Circuit Complexity**

# Boolean Circuits

# Boolean Circuits

- **Boolean circuits** are a model of computing modelling a computation of a **Boolean function**

$$f: \{0, 1\}^n \rightarrow \{0, 1\}$$

in terms of elementary **Boolean operations**

- **Motivation:** modelling **physical circuits**
- **Motivation:** understanding **non-uniform computation**
  - Circuits are purely combinatorial objects
  - Possibly easier to analyse?

# Boolean Circuits

## Definition (Boolean circuits)

A **Boolean circuit**  $C$  with  $n$  inputs and 1 output is a directed acyclic graph with  $n$  sources (vertices with no incoming edges) and 1 sink (vertex with no outgoing edges) such that

- all non-source vertices are labelled with either  $\vee$ ,  $\wedge$ , or  $\neg$ ,
  - vertices labelled with  $\vee$  or  $\wedge$  have in-degree 2,
  - vertices labelled with  $\neg$  have in-degree 1, and
  - the  $n$  sources are labelled with integers  $1, 2, \dots, n$ .
- 
- Vertices are called **gates**
  - Source vertices are called **inputs**
  - The sink vertex is called **output**
  - **Fan-in** and **fan-out** refer to the in-degree and out-degree

# Boolean Circuits

## Definition (Value of a Boolean circuit)

Given an input  $x \in \{0, 1\}^n$ , the value  $v_g(x)$  of a gate  $g$  is defined as follows:

- if  $g$  is an input labelled with  $i$ , then  $v_g(x) = x_i$ , and
- if  $g$  is a non-input gate, then value of  $g$  is defined naturally in terms of the label of  $g$ .

The **value** of the circuit  $C(x)$  is defined as the value of the output gate.

- A circuit  $C$  **computes** a function  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  if for all  $x \in \{0, 1\}^n$ , we have that  $C(x) = f(x)$

# Boolean Circuits

- **Definitions are robust in terms of small modifications**
  - We can allow  $\wedge$  and  $\vee$  gates to have **unbounded fan-in**, as a gate with fan-in  $k \geq 3$  can be simulated with a binary tree
  - We can define circuits with **multiple output gates**
  - We can allow **constant gates** 0 and 1

# Circuits and Boolean Formulas

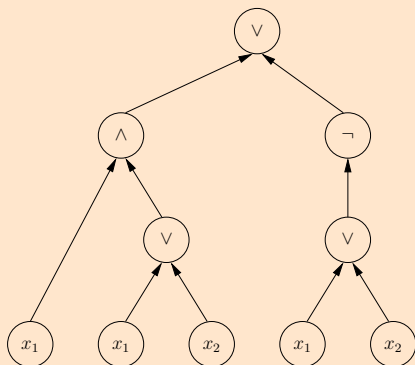
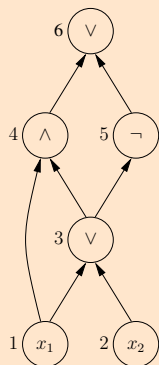
- **Boolean formulas** can be seen as special type of circuits
  - Only input gates can have fan-out large than one
  - Alternatively: duplicated input gates
- **All Boolean functions  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  can be described by a CNF of size at most  $n2^n$** 
  - All functions have circuits of size  $n2^n$
  - This is pretty close to optimal:  $O(2^n/n)$  suffices



# Circuits and Boolean Formulas

## Example

$$\phi = (x_1 \wedge (x_1 \vee x_2)) \vee \neg(x_1 \vee x_2)$$



# Languages and Circuits

- **Size**  $|C|$  of a circuit  $C$  is the number of vertices in  $C$

## Definition (Family of circuits)

Let  $T: \mathbb{N} \rightarrow \mathbb{N}$ . A  **$T(n)$ -size circuit family** is a sequence  $\{C_n\}_{n \in \mathbb{N}}$  of circuits such that  $C_n$  has  $n$  inputs and one output, and  $|C_n| \leq T(n)$  for all  $n$ .

## Definition (Size classes)

We define **SIZE( $T(n)$ )** as the set of languages  $L \subseteq \{0, 1\}^*$  for which there exists a  $T(n)$ -size circuit family  $\{C_n\}_{n \in \mathbb{N}}$  such that for all  $x \in \{0, 1\}^n$ , we have  $x \in L$  if and only if  $C_n(x) = 1$ .

# Polynomial Circuits

# Polynomial Circuits

## Definition

The class  $\mathbf{P}_{/\text{poly}}$  is the set of languages with polynomial-sized circuits, that is,

$$\mathbf{P}_{/\text{poly}} = \bigcup_{d=1}^{\infty} \text{SIZE}(n^d).$$

# Circuits from Turing Machines

## Theorem

$$P \subseteq P_{/poly}$$

- **Proof:** apply the reduction from Cook-Levin theorem
  - The proof can be modified to show that for any  $n$ , there is a CNF of size  $O(T(n)^2)$  that ‘computes’ the output of  $T(n)$ -time Turing machine  $M$  on an input of length  $n$
  - Alternatively, one can transform  $M$  into an **oblivious** Turing machine with running time  $O(T(n) \log T(n))$
  - A more careful construction shows that an oblivious Turing machine can be simulated by a linear-size circuit
- **General observation:** polynomial-time TM can be simulated by a circuit family of polynomial size

# Undecidable Languages

## Theorem

$P_{/poly}$  contains undecidable languages.

- **Proof:**

- $P_{/poly}$  contains all unary languages  $L \subseteq \{1^n : n \in \mathbb{N}\}$ 
  - If  $1^n \in L$ , then the circuit  $C_n$  is an AND of all variables
  - If  $1^n \notin L$ , then the circuit  $C_n$  is circuit that outputs always 0
- The unary language  $\{1^n : n \text{ in binary encodes a TM that halts on empty input}\}$  is undecidable and in  $P_{/poly}$

# Uniform Circuits

# Uniform Circuits

- **Nonuniform circuits are very powerful, as  $P_{/poly}$  contains undecidable languages**
  - What happens if we want a degree of uniformity for our circuit families?

## Definition

A circuit family  $\{C_n\}_{n \in \mathbb{N}}$  is **P-uniform** if there is a polynomial-time Turing machine that on input  $1^n$  outputs a description of circuit  $C_n$ .



# Uniform Circuits

## Theorem

*A language  $L \subseteq \{0, 1\}^*$  is in  $P$  if and only if it is decided by a  $P$ -uniform circuit family.*

- **Proof:**
  - If  $L$  has a  $P$ -uniform circuit family, we can compute the circuit corresponding to input length and simulate it in polynomial time
  - If  $L \in P$ , we can modify the proof of the Cook-Levin theorem to obtain an algorithm that outputs the circuit in polynomial time
- **If we add uniformity requirement,  $P_{/poly}$  collapses to  $P$**

# Uniform Circuits

- Same thing happens if we impose the even stricter constraint of **logspace-uniformity**

## Definition

A circuit family  $\{C_n\}_{n \in \mathbb{N}}$  is **logspace-uniform** if the mapping from  $1^n$  to a description of circuit  $C_n$  is implicitly logspace-computable.

## Theorem

*A language  $L \subseteq \{0, 1\}^*$  is in P if and only if it is decided by a logspace-uniform circuit family.*

- **Proof:** Cook-Levin reduction can be done in implicit logspace

# **Turing Machines with Advice**

# Turing Machines with Advice

## Definition

Let  $T, a: \mathbb{N} \rightarrow \mathbb{N}$ . The class of languages **decidable in time  $T(n)$  with  $a(n)$  bits of advice**, denoted by  **$\text{DTIME}(T(n))/a(n)$** , is the class of languages  $L \subseteq \{0, 1\}^*$  such that there exists

- a sequence of strings  $\{\alpha_n\}_{n \in \mathbb{N}}$  with  $\alpha_n \in \{0, 1\}^{a(n)}$ , and
- a Turing machine  $M$ ,

such that for all  $x \in \{0, 1\}^n$ , we have  $x \in L$  if and only if  $M(x, \alpha_n) = 1$  and  $M$  runs in time  $O(T(|x|))$  on input  $(x, \alpha_n)$ .

# P/poly as Advice Class

## Theorem

$$\mathbf{P}/\text{poly} = \bigcup_{d,c>0} \mathbf{DTIME}(n^c)/n^d$$

- **Proof:**

- $\mathbf{P}/\text{poly} \subseteq \bigcup_{d,c>0} \mathbf{DTIME}(n^c)/n^d$ : give a description of circuit  $C_n$  as advice
- $\bigcup_{d,c>0} \mathbf{DTIME}(n^c)/n^d \subseteq \mathbf{P}/\text{poly}$ : construct a circuit simulating execution of the Turing machine  $M$  on input  $x$  (inputs of the circuit) and  $\alpha_n$  (hardwired into the circuit)

# NP and P/poly

- **Even if  $P \neq NP$ , it is in principle possible that all problems in NP have polynomial circuits**
  - For example, maybe CNF-SAT has circuits of size  $n^2$ ?
  - This just requires that the circuits cannot be constructed in polynomial time
- **However, there is evidence suggesting this is not the case**

# NP and P/poly

## Theorem

If  $\text{NP} \subseteq \text{P}/\text{poly}$ , then  $\text{PH} = \Sigma_2^p$ .

- **Proof idea:**

- If CNF-SAT has polynomial-size circuits, then there is also a polynomial-size circuit that *outputs* a satisfying assignment of an input CNF
- This can be used to show that a  $\Pi_2^p$ -complete problem is in  $\Sigma_2^p$ : we can use the existential quantifier to guess the above circuit, and use it to replace the second quantifier in  $\Pi_2^p$

# EXP and P/poly

- **Similar result shows that EXP is unlikely to have polynomial circuits**

## Theorem

*If  $\text{EXP} \subseteq \text{P}/\text{poly}$ , then  $\text{EXP} = \Sigma_2^p$ .*

- **Note that this implies that if  $\text{P} = \text{NP}$ , then  $\text{EXP} \not\subseteq \text{P}/\text{poly}$ :**
  - If  $\text{P} = \text{NP}$ , then  $\text{P} = \Sigma_2^p$
  - If also  $\text{EXP} \subseteq \text{P}/\text{poly}$ , then  $\text{P} = \text{EXP}$ , which is impossible by the time hierarchy theorem
- **Upper bounds can imply circuit lower bounds!**
  - Used in a fairly recent breakthrough result by Williams



# Circuit Lower Bounds

# Circuit Lower Bounds?

- **Proving  $\text{NP} \not\subseteq \text{P}_{/\text{poly}}$  would imply  $\text{P} \neq \text{NP}$** 
  - The hope is that since circuits are much more explicit than Turing machines, they might be mathematically easier to handle
  - So far, this has not proven very successful
- **However, it is very easy to show that **some** functions are difficult to compute with circuits**

# Counting Arguments

## Theorem

*For every  $n > 1$ , there exists a function  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  that cannot be computed by a circuit of size  $2^n/10n$ .*

### • Proof:

- The number of functions  $f: \{0, 1\}^n \rightarrow \{0, 1\}$  is  $2^{2^n}$
- Circuit of size at most  $S$  can be represented with, say,  $9S \log S$  bits
- Thus, there are at most  $2^{9S \log S}$  circuits of size  $S$
- Setting  $S = 2^n/10n$ , the number of circuits of size  $S$  is at most

$$2^{9S \log S} \leq 2^{2^n 9n/10n} < 2^{2^n}$$

- Thus, there are more functions than circuits of size  $S$

# Nonuniform Time Hierarchy

- Using a similar counting argument, one can prove a **hierarchy theorem** for circuit size classes

## Theorem

For any functions  $T_1, T_2: \mathbb{N} \rightarrow \mathbb{N}$  with  $2^n/n > T_2(n) > T_1(n) > n$ , we have

$$\text{SIZE}(T_1(n)) \subsetneq \text{SIZE}(T_2(n)).$$

# **Circuits and Parallel Computation**

# Circuits and Parallel Computation

- **Circuits can be viewed as a massively parallel computer**
  - Each node has its own processor computing the function at the gate
  - Messages are passed along the edges when computation at a gate completes
- **Relevant complexity measure: depth**
  - The **depth** of a circuit is the length of the longest path from an input gate to the output gate
  - Total parallel computing time corresponds to the depth of the circuit
- **We next look at two circuit complexity classes meant to model this type of parallelism**

# The Class NC

## Definition

Let  $d \geq 1$  be fixed. The class  $\mathbf{NC}^d$  is the class of languages  $L \subseteq \{0, 1\}^*$  that can be decided by a circuit family  $\{C_n\}_{n \in \mathbb{N}}$  such that each  $C_n$  has size polynomial in  $n$  and depth  $O((\log n)^d)$ . The class  $\mathbf{NC}$  is defined as

$$\mathbf{NC} = \bigcup_{d=1}^{\infty} \mathbf{NC}^d.$$

- **Uniform NC** is defined by requiring the circuits to be **logspace-uniform**

# The Class AC

## Definition

Let  $d \geq 1$  be fixed. The class  $\mathbf{AC}^d$  is defined similarly to  $\mathbf{NC}^d$ , but the AND and OR gates are allowed to have unbounded fan-in. The class  $\mathbf{AC}$  is defined as

$$\mathbf{AC} = \bigcup_{d=1}^{\infty} \mathbf{AC}^d .$$

- **Uniform AC** is again defined by requiring the circuits to be logspace-uniform
- Note that

$$\mathbf{NC}^d \subseteq \mathbf{AC}^d \subseteq \mathbf{NC}^{d+1} ,$$

since simulating unbounded fan-in adds at most  $\log n$  factor to depth



# Problems in NC

## Example

- **Some example problems in NC:**
  - **Parity** (input has an odd number of 1s)
  - **Integer operations** addition, multiplication and division
  - **Matrix multiplication** and related problems
  - **Maximal matching**

# NC and Parallel Computation

- **NC captures parallel computation as follows:**
  - Consider NC circuit family  $\{C_n\}_{n \in \mathbb{N}}$  with width  $N = O(n^d)$  and depth  $D = O((\log n)^d)$
  - Consider a parallel computer with  $N$  interconnected processors
  - Assigning one gate from each **layer** of circuit to one machine
  - At each step of parallel computation:
    - Each machine computes the output of their gate
    - Each machine sends their output to the machines that need it on the next step
- **More formally: NC is equivalent to logtime PRAMs**

# P-completeness

- **Do all problems in P have an efficient parallel algorithm?**
  - Can be formalised as the question whether  $P = NC$
  - Believed: **no**
  - Motivates the study of **P-completeness**

## Definition

A language  $L \subseteq \{0, 1\}^*$  is **P-complete** if  $L \in P$  and for any language  $L' \in P$ , there is a logspace reduction from  $L'$  to  $L$ .

# P-completeness

## Theorem

*If  $L \subseteq \{0, 1\}^*$  is P-complete, then*

- $L \in \text{NC}$  if and only if  $\text{P} = \text{NC}$ , and*
- $L \in \text{L}$  if and only if  $\text{P} = \text{L}$ , where  $\text{L}$  is logarithmic space.*

- P-complete problems don't have efficient parallel algorithm if  $\text{P} \neq \text{NC}$**

# P-completeness

## Circuit value

- **Instance:** A circuit  $C$  with  $n$  inputs and  $x \in \{0, 1\}^*$
  - **Question:** Does it hold that  $C(x) = 1$ ?
- 
- **Circuit value is P-complete:**
    - Circuit value is clearly in P
    - Hardness follows from the proof that all problems in P have logspace-uniform circuits

# Lecture 14: Summary

- Boolean Circuits
- Class  $P_{/poly}$
- $P_{/poly}$  and uniform complexity classes
- Counting arguments for circuit lower bounds
- NC, AC and P-completeness