Computational Complexity	October 3, 2018

Lecture 4 (Notes)

Lecturer: Ola Svensson *Scribes: Ola Svensson*

Disclaimer: These notes were written for the lecturer only and may contain inconsistent notation, typos, and they do not cite relevant works. They also contain extracts from the two main inspirations of this course:

1. The book *Computational Complexity: A Modern Approach* by Sanjeev Arora and Boaz Barak;
2. The course *http://theory.stanford.edu/ trevisan/cs254-14/index.html* by Luca Trevisan.

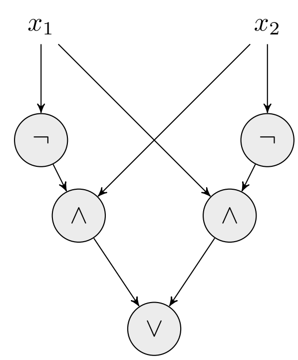
1 Introduction

- Today: circuits, non-uniform computation.
- Proof of the Cook-Levin Theorem.

2 Recall basic circuit definitions

A circuit C has n inputs and m outputs, and is constructed with AND, OR, and NOT gates. Each gate has fan-in 2 except the NOT gate which has fan-in 1. The out-degree can be any number. A circuit is *not* allowed to have any cycles.

Example 1 *A circuit C computing the XOR function, i.e., $C(x_1, x_2) = 1$ iff $x_1 \neq x_2$:*



Definition 1 (Size) *The size of a circuit C, denoted by $|C|$, is the number of its gates.*

- The size of the XOR circuit C above is 5.

Definition 2 (Circuit families and language recognition) *Let $T : \mathbb{N} \to \mathbb{N}$ be a function . A $T(n)$-*size circuit family *is a sequence of $\{C_n\}_{n\in\mathbb{N}}$ of Boolean circuits, where C_n has n inputs and a single output, and its size $|C_n| \leq T(n)$ for every n.*

We say that language L is in $\mathbf{SIZE}(T(n))$ *if there exists a $T(n)$-size circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that for every $x \in \{0,1\}^n$, $x \in L \Leftrightarrow C_n(x) = 1$.*

Example 2 *For any $B \subseteq \{0,1\}^*$, the unary language $U_B = \{1^n :$ exists a string of length n in $B\}$ has a linear-sized circuit family. If $1^n \in U_B$ the circuit is simply a tree of AND gates and otherwise if $1^n \notin U_B$ then the circuit C_n is the trivial circuit that always outputs 0.*

Example 3 *The language $\{\langle m, n, m+n\rangle : m, n \in \mathbb{Z}\}$ also has linear-sized circuits that implement the grade-school algorithm for addition.*

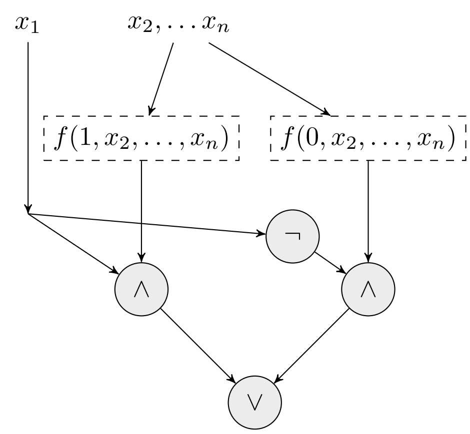
3 Basic Circuit Upper and Lower Bounds

- Notice that, unlike the complexity classes we defined with Turing machines, circuits is a non-uniform computational model: we can have different circuits for each size of the problem/language. For Turing machines we had the same machine for infinite (all) inputs of a problem (an uniform computational model).
- Indeed, unlike other complexity measures such as time and space, for which there are languages of arbitrarily high complexity, the size complexity of a problem is always at most exponential.

Theorem 3 *For every language L, $L \in \mathbf{SIZE}(2^n)$.*

Proof

- We need to show that for every Boolean function $f : \{0,1\}^n \to \{0,1\}$, f has a circuit of size $O(2^n)$.
- Use the identity $f(x_1, x_2, \ldots, x_n) = (x_1 \wedge f(1, x_2, \ldots, x_n)) \vee (\bar{x}_1 \wedge f(0, x_2, \ldots x_n))$ to recursively construct a circuit for f as follows:



- The recurrence relation for the size of the circuit is $s(n) = 4 + 2 \cdot s(n-1)$ with say base case $s(1) = 0$ which solves to $s(n) = 2^n - 4$.

■

On the other hand, most languages do require exponential size circuits:

Theorem 4 *There are languages L such that $L \notin \mathbf{SIZE}(o(2^n/n))$. In particular, for every $n \geq 11$, there exists $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $2^n/4n$.*

Proof This is a counting argument:

- There are 2^{2^n} functions $f : \{0,1\}^n \to \{0,1\}$.
- We claim that the number of circuits of size s is at most $2^{O(s \log s)}$, assuming $s \geq n$.
- To bound the number of circuits of size s, we create a compact binary encoding of such circuits.
- Identify gates with numbers $1, 2, \ldots, s$. For each gate, specify where the two/one inputs are coming from, and the type of the gate. The total number of bits required to represent the circuits is

$$s \cdot (2 \log(n + s) + 2) \leq s \cdot (2 \log 2s + 3) = s \cdot (2 \log s + 5).$$

- So the number of circuits of size s is at most $2^{2s\log s+5s}$ and this is not sufficient to compute all possible functions if
$$2^{2s\log s+5s} < 2^{2^n}.$$
- This is satisfied if $s \leq 2^n/(4n)$ and $n \geq 11$.

■

3.1 Some comments

Although almost all functions $f : \{0,1\}^n \to \{0,1\}$ require large circuits, we are unable to show that "natural ones" require large circuits. The best lower bound on an NP language is something like $5n$. We do not even know if every language in NEXP does have a polysize circuit family.

4 Simulation of Efficient Computation by Small Circuits

Definition 5 *An* Oblivious *Turing machine (OTM) is a machine for which, at every time t, the j:th head is at cell $s_j(t)$ for some function s_j that only depends on the length of the input.*

We show that any $T(n)$-time OTM can be simulated by a circuit of size at most $O(T(n))$. As any TM can be simulated by an OTM by incurring a logarithmic multiplicative loss in the running time (see book and exercise session) it follows that
$$\mathbf{P} \subseteq \mathbf{P_{/poly}} := \cup_c \mathbf{SIZE}(n^c).$$

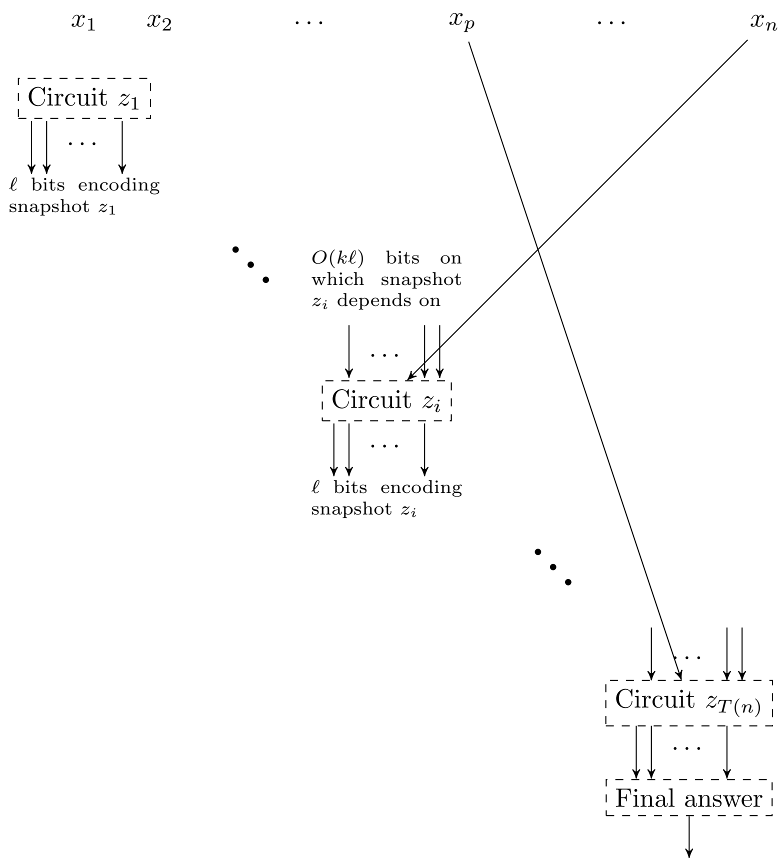
Theorem 6 *Let M be a $T(n)$-time OTM. There exists an $O(T(n))$-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that*
$$C_n(x) = M(x) \qquad \textit{for every } x \in \{0,1\}^n.$$

Proof

- Let $x \in \{0,1\}^*$ be some input for M and define the *transcript* of M's execution on x to be the sequence $z_1, \ldots, z_{T(n)}$ of *snapshots* (the machine's state and symbols read by all heads) of the execution at each step in time.
- Each snapshot z_i can be encoded by a constant-sized binary string (say by ℓ bits).
- Moreover, we can compute the ℓ bits encoding z_i based on the following information:
  1. What is the state of the machine at time i?
  2. What is written on the heads of the tapes at time i?

  The answer to the first question depends on the snapshot z_{i-1}. The answer to the second question depends on (potentially) an input bit and the snapshots $z_{i_1}, \ldots, z_{i_k}$ where z_{i_j} denotes the last step the M's j:th head was in the same position as it is in the i:th step. (Notice that $i_1, \ldots, i_k$ depend only on i and not on the actual input x as M is oblivious).
- Because there are only a constant number of strings of constant length, we can compute the ℓ bits encoding z_i from these previous snapshots using a constant-sized circuit.

- The composition of all these constant sized circuits gives rise to a circuit that on input x computes the encoding of the snapshot $z_{T(n)}$. An overview of the circuit is as follows:



- If the snapshot $z_{T(n)}$ is accepting, the circuit outputs 1 and otherwise it outputs 0.
- Thus, there is a $O(T(n))$-sized circuit C_n such that $C_n(x) = M(x)$ for every $x \in \{0,1\}^n$.

■

Remark The proof of the above theorem actually gives a stronger result than in the statement: the circuit is not only of size $O(T(n))$ but it is also computable in time $O(T(n))$.

Remark The proof of the above theorem relied crucially on that computation is *local*.

5 Circuit Satisfiability and a proof of the Cook-Levin Theorem

Boolean circuits give an alternative proof of the central Cook-Levin Theorem that shows that 3-SAT is NP-complete.

Definition 7 (Circuit satisfiability or CKT-SAT) *The language CKT-SAT consists of all (strings representing) circuits that produce a single bit of output and that have a satisfying assignment.*

CKT-SAT is clearly in **NP** because the satisfying assignment can serve as the certificate. The Cook-Levin Theorem follows immediately from the next two lemmas.

Lemma 8 *CKT-SAT is* **NP***-hard.*

Proof

- If $L \in \mathbf{NP}$ then there is a polynomial-time TM M and a polynomial p such that $x \in L$ iff $M(x, u) = 1$ for some $u \in \{0,1\}^{p(|x|)}$.
- The proof of Theorem 6 yields a polynomial-time transformation from M, x to a circuit C such that $M(x, u) = C(u)$ for every $u \in \{0,1\}^{p(|x|)}$. Thus $x \in L$ iff $C \in$ CKT-SAT.

■

Lemma 9 *CKT-SAT* $\leq_p$ 3-*SAT.*

Proof Map a circuit C into a 3-SAT formula φ as follows:

- For every node/gate v_i of C, we will have a corresponding variable z_i in φ.
- If the node v_i is an AND of the nodes v_j and v_k then we add to φ the clauses that are equivalent to the condition $z_i = (z_j \wedge z_k)$.
- Similarly, if v_i is an OR of v_j and v_k we add the clauses that are equivalent to $z_i = (z_j \vee z_k)$.
- And, if v_i is the NOT of v_j then we add the clauses that are equivalent to $z_i = \neg z_j$.
- Finally, if v_i is the output node of C then we add the clause (z_i) to φ.
- It is not hard to see that the formula φ is satisfiable iff the circuit C is. Moreover, the reduction runs in polynomial time.

■