Exercise III, Computational Complexity 2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. Solve as many problems as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

Circuit complexity

1 In the CNF-Equivalence problem, the input consists of a pair of CNF formulas (φ, ψ) both defined over the same set of n variables $x = (x_1, \ldots, x_n)$. The goal is to decide if they are *equivalent*, that is, do they compute the same boolean function: $\varphi(x) = \psi(x)$ for all $x \in \{0,1\}^n$? Classify this problem as best as you can—is it in P, NP, or coNP? is it complete for any class?

Solution: Let us abbreviate CNF-Equivalence by simply C and show that the language C is coNP-complete. Note that C $\in$ coNP because $\overline{\text{C}}$ is the language of all pairs of functions (φ, ψ) that are not equivalent and non-equivalency can be certified by exhibiting an assignment $x \in \{0,1\}^n$ with $\varphi(x) \neq \psi(x)$.

We further show that the coNP-complete problem $\overline{\text{SAT}}$ reduces to C. Indeed, let $f : \{0,1\}^* \to \{0,1\}^*$ be the poly-time computable function that takes a formula φ and returns the C instance $(\varphi, x_1 \wedge \neg x_1)$. Then, for any formula φ:

$$\varphi \in \overline{\text{SAT}} \iff \varphi \equiv 0 \iff \varphi \equiv x_1 \wedge \neg x_1 \iff f(\varphi) \in \text{C}$$

This shows that most likely C $\notin$ NP.

2 Complete the proof of Circuit-SAT $\leq_p$ SAT from the lecture by finding, for each of the following logical predicates, an equivalent CNF formula.

(a) $y \leftrightarrow (x \vee z)$
(b) $y \leftrightarrow (x \wedge z)$
(c) $y \leftrightarrow \neg x$

Solution: One can find an equivalent CNF formula by applying standard boolean algebra identities. We do the details for the first item only.

$$\begin{aligned} y \leftrightarrow (x \vee z) &= (y \to (x \vee z)) \wedge ((x \vee z) \to y) \\ &= (\neg y \vee x \vee z) \wedge (\neg(x \vee z) \vee y) \\ &= (\neg y \vee x \vee z) \wedge (\neg x \wedge \neg z) \vee y) \\ &= (\neg y \vee x \vee z) \wedge (\neg x \vee y) \wedge (\neg z \vee y) \end{aligned}$$

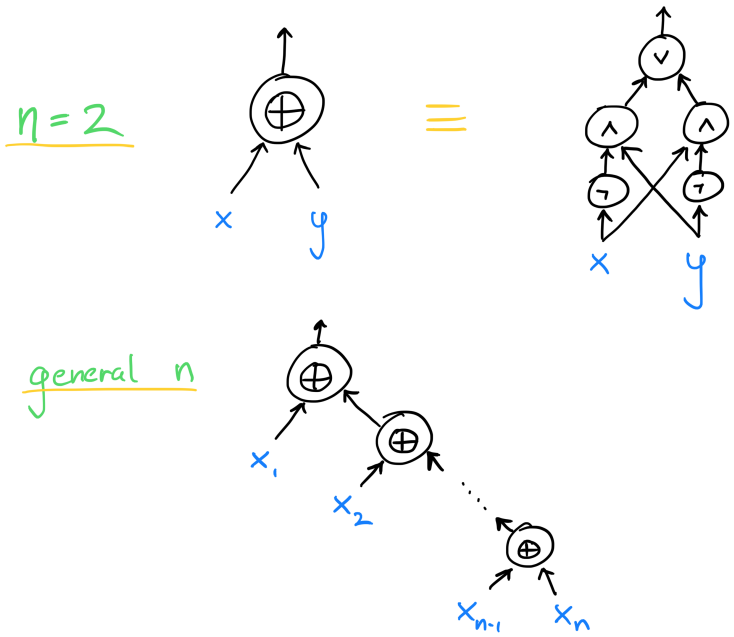
$$y \leftrightarrow (x \wedge z) = (\neg y \vee x) \wedge (\neg z \vee z) \wedge (\neg x \vee \neg z \vee y)$$

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

3 The n-bit function $\textsc{Xor}_n\colon \{0,1\}^n \to \{0,1\}$ outputs 1 iff the number of 1-bits in the input is odd. Show that $\textsc{Xor}_n$ can be computed with a boolean circuit (gates $\vee$, $\wedge$, $\neg$) of size $O(n)$. Can you also make the circuit have *depth* (i.e., length of longest directed path) at most $O(\log n)$?

(Hint: Construct a circuit for $n = 2$ and then use many copies of that circuit for general n.)

Solution:



4 Let φ be any DNF formula over n variables that computes $\textsc{Xor}_n$. Recall that $\varphi = T_1 \vee \cdots \vee T_m$ where each T_j is a *term*, that is, a conjunction of literals.

(a) Show that any term T_j either contains n distinct variables or is *contradictory*, meaning that it contains x_i and $\overline{x}_i$ for some variable x_i.

(Hint: Show that if T_j is not contradictory and omits both x_i and $\bar{x}_i$ for some i, then φ fails to compute $\textsc{Xor}_n$ correctly. Use the fact that the value of $\textsc{Xor}_n$ is flipped if we flip the value of x_i.)

(b) Show that φ must contain $m \geq 2^{n-1}$ terms.

Conclude that circuits can be exponentially more expressive than DNF/CNF formulas.

Solution: (a) Suppose T_j is not contradictory. Then it accepts some input $x \in \{0,1\}^n$ so that $T_j(x) = 1$ and hence $\varphi(x) = \textsc{Xor}_n(x) = 1$. Suppose for the sake of contradiction that T_i does not contain all n variables, say, variable x_i. Consider $x' \in \{0,1\}^n$ which is the same as x but with the value of variable x_i flipped. We still have that $T_j(x') = 1$ and hence $\varphi(x') = \textsc{Xor}_n(x') = 1$. But this is a contradiction since x and x' have different parities! Hence we conclude that T_j contains all n variables.

(b) Item (a) implies that each term of φ can accept at most one input. But there are 2^{n-1} many inputs x with $\textsc{Xor}_n(x) = 1$ and hence φ must contain one term for each such x.

5 (*) Consider the $2n$-variate CNF formula defined by $\varphi = (x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge \ldots \wedge (x_n \vee y_n)$. Show that any DNF formula equivalent to φ requires at least 2^n terms.

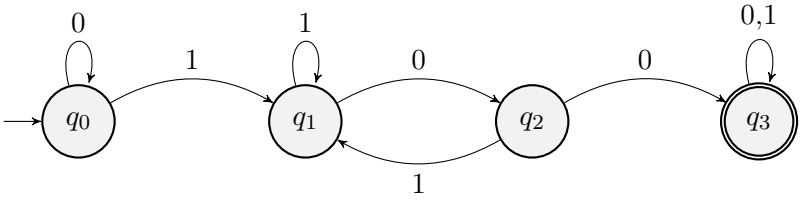
Solution: Let $\psi = T_1 \vee T_2 \vee \cdots \vee T_k$ be an equivalent DNF formula. Consider the family of inputs $\mathcal{F} \subseteq \{0,1\}^{2n}$:

$$\mathcal{F} = \big\{(x_i, y_i)_{i\in[n]} : (x_i, y_i) \in \{01, 10\}\big\}$$

Observe that each term T of ψ can accept at most one element of $\mathcal{F}$. Indeed, suppose toward contradiction that $T(f) = T(f') = 1$ for two distinct elements $f, f' \in \mathcal{F}$. Without loss of generality, we may assume that the mismatch happens at index i with $f_i = 01$ and $f'_i = 10$. Since T is a conjunction, it must be that T is not looking at variables x_i and y_i at all so that T also accepts the input $\widetilde{f}$ which is a copy of f where f_i is modified to be 00. Ultimately, this implies that $\psi(\widetilde{f}) = 1$: a contradiction with the fact that $\varphi(\widetilde{f}) = 0$.

Note that all strings of $\mathcal{F}$ are accepted by φ. Correspondingly, for each $f \in F$, there must be a term T of ψ accepting f. Using our above observation, we directly get that $k \geq |\mathcal{F}| = 2^n$.

6 (*) Recall from your undergrad days that a language $L \subseteq \{0,1\}^*$ is *regular* if it is accepted by a *deterministic finite automaton* $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$. Here Q is a finite set of states, $\Sigma = \{0,1\}$ is the input alphabet, $\delta\colon Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final accepting states. For example, the following automaton accepts all binary strings that contain "100" as a substring:



Prove that every regular language can be computed by a linear-size circuit. That is, let $L \subseteq \{0,1\}^*$ be a regular language. For any input length $n \in \mathbb{N}$, show how to construct a boolean circuit C_n with n input variables and $O(n)$ gates such that

$$\forall x \in \{0,1\}^n : \quad C_n(x) = 1 \iff x \in L.$$

Solution: Let $\mathcal{D} = (Q, \{0,1\}, \delta, q_0, F)$ be a DFA that recognizes L. For simplicity, let us assume that the k states are labelled with $\{e_1, e_2, \ldots, e_k\}$ where $e_i \in \{0,1\}^k$ is the indicator bit-string which is 1 only at position i.

Note that the transition function $\delta : \{0,1\}^{k+1} \to \{0,1\}^k$ can be computed by a circuit Δ of constant size. Indeed, since the input size is $k+1$ and the output size is k, the function is described by a truth table of size $(2^k)^{2^{k+1}} \in O(1)$.

To construct the circuit that recognizes L on inputs of length n, we mimic $\mathcal{D}$ by processing the input iteratively, updating the state each time with Δ. When the last bit of the input has been processed, we check the final state and output 1 if and only if it belongs to F (this can be done with a constant number of gates), see Figure 1 for a depiction of the circuit. Since the circuit emulates $\mathcal{D}$, it correctly computes L while using $n \cdot O(1) + O(1) = O(n)$ gates.

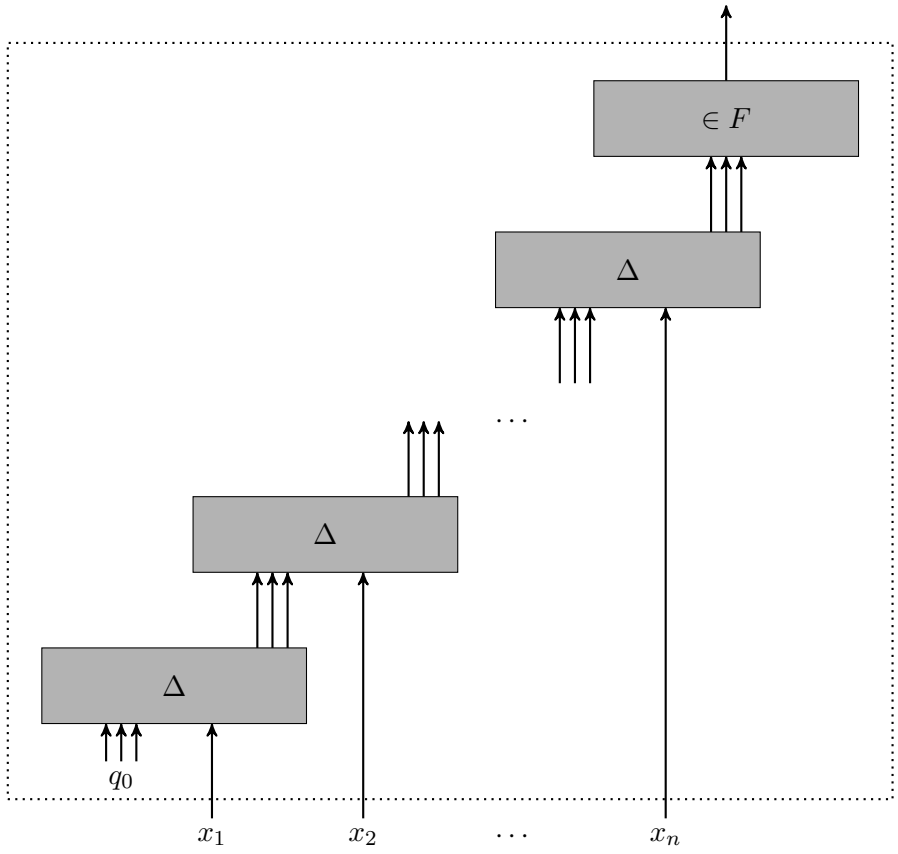


Figure 1. Architecture of the circuit that accepts n-bit words of L.