Computational Complexity

Take-home exam

Hand in via Canvas before Wednesday May 28, 2025, at 23:59

`https://canvas.uva.nl/courses/49698/assignments/588931`

Definition 1. 'Universal variable forgetting' in propositional logic is defined as follows. Let φ be a propositional logic formula over the propositional variables X, and let $W \subseteq X$ be a subset of variables. The result of *universally forgetting* W *in* φ is a propositional logic formula ψ over the variables $X \setminus W$ such that for all truth assignments $\alpha : X \setminus W \to \{0,1\}$ it holds that α makes ψ true if and only if for all truth assignment $\beta : X \to \{0,1\}$ that extend α it holds that β makes φ true. Note that such a formula ψ is not unique—there are other formulas ψ' that are logically equivalent, and thus also express the result of universally forgetting W in φ.

For example, consider the propositional formula $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ over the variables $X = \{x_1, x_2, x_3\}$, and let $W = \{x_3\}$. The formula $\psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ expresses the result of universally forgetting W in φ.

We say that a function f implements universal forgetting for propositional logic if for each propositional formula φ and each $W \subseteq \text{Vars}(\varphi)$ it holds that $f(\varphi, W)$ is a propositional logic formula that expresses the result of universally forgetting W in φ.

Question 1 (*3½pt; a: ¾pt; b: ¾pt, c: 2pt*).

(a) Prove that there is a polynomial-time function f that implements universal forgetting for propositional logic formulas in CNF.

(b) Prove that if there is a function f that implements universal forgetting for (arbitrary formulas of) propositional logic and that can be computed in polynomial time, then $\mathsf{P} = \mathsf{NP}$.

(c) Prove that if there is a function f that implements universal forgetting for (arbitrary formulas of) propositional logic and that is of polynomial-size, then the Polynomial Hierarchy collapses. A function f is of *polynomial-size* if there exists some polynomial p such that $|f(x)| \leq p(|x|)$—i.e., the size of the result is upper bounded by a polynomial of the size of the input, but there are no restrictions on the time needed to compute the function (or whether it is computable at all).

- *Hint:* use the fact that $\mathsf{coNP} \subseteq \mathsf{P/poly}$ implies that $\mathsf{PH} = \Sigma_2^{\mathrm{p}}$.
- *Hint:* for different values of $\ell \in \mathbb{N}$, consider the formula:

$$\varphi_\ell = \bigvee_{1 \leq i \leq (2\ell)^3} (y_i \wedge \delta_i),$$

where $\delta_1, \ldots, \delta_{(2\ell)^3}$ is an enumeration of all terms[1] of size 3 over the variables $x_1, \ldots, x_\ell$.

Question 2 (*1pt*). Prove that $\mathsf{ZPP}^{\mathsf{ZPP}} = \mathsf{ZPP}$.

[1] Conjunctions of literals are called *terms*—i.e., terms are negations of clauses.

Definition 2. We say that a language L is *p-selective* if there exists a polynomial-time computable function $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ such that for bit strings x and y, the value $f(x,y)$ is either x or y, and if $x \in L$ or $y \in L$ then $f(x,y) \in L$. We call f a *selector* for L. In other words, if only one of its inputs is in L, then the polynomial-time computable function f is guaranteed to select that input; otherwise, f is allowed to output either of its inputs.

Question 3 (*1½pt*)**.** Recall that CLIQUE is the problem of deciding—given an undirected graph G and a positive integer k—whether G has a clique of size k. Prove that CLIQUE is p-selective if and only if $\mathsf{P} = \mathsf{NP}$. (See Definition 2 above.)

Definition 3. Consider the following problem FEW LONG CLAUSES SAT:

Input: A propositional logic formula φ over n variables that is in CNF where all but k clauses are of size 2.

Question: Is φ satisfiable?

Question 4 (*3pts; a: ½pt, b: 1½pts, c: 1pt*)**.**

(a) Prove that there exists an algorithm that solves FEW LONG CLAUSES SAT in time $n^{O(k)}$.

- *Hint:* 2SAT is polynomial-time solvable. You may use this fact without proving it.

(b) Prove that there does not exist an algorithm that solves FEW LONG CLAUSES SAT in time $2^k \cdot n^{o(k)}$, assuming the ETH.

- *Hint:* Consider the following reduction from 3COL to FEW LONG CLAUSES SAT. Let $G = (V, E)$ be an instance of 3COL with $|V| = n$. The reduction partitions the nodes (arbitrarily) into $\log n$ groups $V_1, \dots, V_{\log n}$ consisting each of at most $n/\log n$ nodes.

  It then constructs an instance φ of FEW LONG CLAUSES SAT as follows. We set $k = \log n$. The set X of variables contains a variable x_μ for each 3-coloring μ such that:

  (A) μ is defined on exactly one of $V_1, \dots, V_k$—i.e., $\mu : V_i \to \{1,2,3\}$ for some $1 \le i \le k$; and

  (B) μ does not assign any two nodes connected by an edge in E to the same color—i.e., for no edge $\{v_1, v_2\} \in E$ it holds that $v_1, v_2 \in V_i = \text{dom}(\mu)$ and $\mu(v_1) = \mu(v_2)$.

  The formula φ then consists of the conjunction of the following clauses.

  * For each $1 \le i \le k$ and each $\mu_1, \mu_2 : V_i \to \{1,2,3\}$ such that $\mu_1 \neq \mu_2$, the formula φ contains the clause $(\neg x_{\mu_1} \lor \neg x_{\mu_2})$.
  * For each $1 \le i < j \le k$, each $\mu_i : V_i \to \{1,2,3\}$ and each $\mu_j : V_j \to \{1,2,3\}$ that assign two nodes that are connected by an edge in E to the same color—i.e., such that there is some $\{v_i, v_j\} \in E$ with $v_i \in V_i$ and $v_j \in V_j$ for which $\mu_i(v_i) = \mu_j(v_j)$—the formula φ contains the clause $(\neg x_{\mu_i} \lor \neg x_{\mu_j})$.
  * For each $1 \le i \le k$, the formula φ contains the clause:

$$\bigvee_{\substack{\mu_i : V_i \to \{1,2,3\} \\ \mu_i \text{ satisfies (A) and (B)}}} x_{\mu_i}$$

- *Note:* you still have to prove that this reduction is correct.

(c) Prove that there does not exist an algorithm that solves FEW LONG CLAUSES SAT in time $2^{2^k} \cdot n^{o(k)}$, assuming the ETH.

- *Note:* for **(c)**, it suffices to indicate where (and how exactly) your solution for **(b)** needs to be adapted.

- *Fun fact:* this result can be extended to the statement that there exists no algorithm for FEW LONG CLAUSES SAT running in time $f(k) \cdot n^{o(k)}$ for any computable function f, assuming the ETH. (But you don't have to prove this!)