

General Instructions: Same as in Homework 1.

Honor Principle: For the first two problems, you should work entirely on your own and not discuss them with anyone else. For the third problem, the honor principle is the same as in Homework 1, i.e., you may discuss, but write up solutions on your own.

11. An *unbounded fan-in circuit* is just like the circuits we defined in class — i.e., DAGs whose vertices (gates) are inputs (x_1, \dots, x_n) , negated inputs $(\neg x_1, \dots, \neg x_n)$, and logic gates (AND/OR), with one or more gates of fan-out 0 designated as output(s) — except that the restriction that gates have fan-in 2 is removed. Size and depth are defined as before: number of edges (wires) and maximum path length, respectively.

Prove that there exists a suitable size function $s : \mathbb{N} \rightarrow \mathbb{N}$ such that every Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ has unbounded fan-in circuits with $O(1)$ depth and $O(s(n))$ size. Find the smallest possible $s(n)$ you can.

[2 points]

12. For integers $i \geq 0$, the complexity classes AC^i are defined as follows:

$$AC^i = \{L \subseteq \{0, 1\}^* : \exists c \in \mathbb{N} (L \text{ has an unbounded fan-in circuit family } \langle C_n \rangle_{n=1}^{\infty} \text{ with } \text{size}(C_n) = O(n^c) \text{ and } \text{depth}(C_n) = O(\log^i n))\}.$$

In other words, AC^i is the unbounded fan-in analogue of NC^i . Note that AC^0 does not suffer from the severe shortcoming that NC^0 does, where the output can only depend on a constant number of inputs.

Prove that $\bigcup_{i=0}^{\infty} AC^i = \bigcup_{i=0}^{\infty} NC^i$.

[2 points]

13. For this problem, we return to the usual notion of circuits with fan-in 2. Let $k > 0$ be an integer. Construct a language in PH that is not in $SIZE(n^k)$. [2 points]

This is a hard problem. To get started, recall from the lectures that we proved that there *exist* languages in $SIZE(n^{2k})$ that are not in $SIZE(n^k)$. Try writing out this fact formally, using quantifiers: you should have a small, fixed number of quantifier alternations. This suggests that you might be able to place the required language in either Σ_i^p or Π_i^p , for some *fixed* i , independent of k .

A further hint is given on the next page. I *strongly* recommend that you turn the problem over in your mind for a day at least, before looking at that hint.

Hint for Problem 13: For a string $w \in \{0, 1\}^n$, let B_w denote the Boolean circuit described by w ; if w is not a well-formed encoding of a circuit due to syntax errors, define B_w to be a trivial circuit that always outputs 0 (say). Let $C(x)$ denote the output of circuit C on input x . Argue that, for $s \in \mathbb{N}$ and $w, x \in \{0, 1\}^*$, the predicates “ $\text{size}(B_w) \leq s$ ” and “ $B_w(x) = 1$ ” are decidable in polynomial time. Therefore, if we use a fixed number of quantifier alternations and then perform an inner computation that involves evaluating these types of predicates (maybe a few times), we’ll have either a Σ_i^P or a Π_i^P computation, depending upon whether we start with a “ \exists ” or a “ \forall ” quantifier.

Now consider the following statement $\phi_n(x)$, for an $x \in \{0, 1\}^n$, and figure out what it’s saying:

$$\phi_n(x) = \exists w \in \{0, 1\}^* (\text{size}(B_w) \leq n^{2k} \wedge B_w(x) \wedge \forall v \in \{0, 1\}^* (\text{size}(B_v) \leq n^k \Rightarrow \exists y \in \{0, 1\}^n (B_v(y) \neq B_w(y)))).$$

Once you have digested it, you’ll find that this is close to what we need to create a suitable language in PH. But unfortunately $\{x \in \{0, 1\}^* : \phi_{|x|}(x)\}$ ends up containing *all* sufficiently long Boolean strings, so this is not the language we seek! Figure out why this happens, and then think of what you can do to fix it. Perhaps you need more quantifiers.