

Final Exam
CAS705
December 6, 2006
Examiner: Michael Soltys

Duration: 3hrs

5 questions, each worth 20%, plus a bonus question

1. Suppose that C is a boolean circuit. Let $C^{(xy)}$ be C with input variables x and y exchanged (every occurrence of x replaced by y and every occurrence of y replaced by x). Consider the language $L = \{\langle C, (xy) \mid C \equiv C^{(xy)} \rangle\}$. Is this language likely to be in \mathbf{P} ? Justify your answer.
2. We know that most boolean functions require large circuits, but the best lower bounds we can currently give are linear. Let B_n be the set of boolean functions on n variables (semantic objects), and consider circuits with gates $\{\wedge, \vee, \neg, 0, 1\}$ of fan-in exactly two except for fan-in one negations at input level only (these circuits are syntactic objects). Let the size of a circuit be the number of $\{\wedge, \vee\}$ gates (so the negation gates at the input level do not count for size).
 - (a) What is $|B_n|$? (I.e., how many boolean functions on n variables are there?)
 - (b) Show that the number of circuits with n variables and size s is bounded above by $(2 \cdot (s + 2n + 2)^2)^s$. (**Hint.** Note that this number is a gross overestimate—just count “graphs” with s nodes labeled by a gate with two inputs, not being concerned about the fact that many such “graphs” are not really circuits because they may contain cycles.)
 - (c) Show that for $s = 2^n / (10n)$ the value of the expression in (2b) is bounded above by $2^{2^n/5}$, and conclude that “almost all” boolean functions in B_n require circuits of size $\Omega(2^n/n)$.
 - (d) Show that $O(n2^n)$ many gates are sufficient to compute any boolean function in B_n .
 - (e) Now give a slightly better construction, from which it follows that every boolean function can be computed with $O(2^n)$ gates. (**Hint.** Do this by induction on the number of inputs.)

- (f) In fact we can get an even tighter upper bound showing that every boolean function f in B_n can be computed with circuits of size $O(2^n/(n - \log n))$, showing that the lower bound in (2c) is very exact. (**Hint.** First show that for any k there is a circuit with multiple outputs and size $O(2^{2^k})$ such that for every f in B_k there is an output computing f ; use the idea in (2e) to do this. Suppose now that $f \in B_n$, and for any $k \leq n$, note that $f(x_1, \dots, x_n)$ equals

$$\bigvee_{a_1, \dots, a_k \in \{0,1\}} (x_1^{a_1} \wedge \dots \wedge x_k^{a_k}) \wedge f(a_1, \dots, a_k, x_{k+1}, \dots, x_n)$$

where x^a is x if $a = 1$, and $\neg x$ if $x = 0$.)

- (g) Let $\text{Th}_{k,n}$ be the threshold function which outputs 1 iff at least k of its n variables are 1. Show that $\text{Th}_{2,n}$ requires circuit size at least $2n - 4$.
(Hint. Do the proof by induction on n . For $n = 2$ and $n = 3$ it is easy. Otherwise, let C be an optimal circuit for $\text{Th}_{2,n}$, and suppose that the bottom most gate acts on variables x_i, x_j , where $i \neq j$. Consider the four possible settings to x_i, x_j .)
- (h) Using (2c) show that there exists a boolean function $f = \langle f_n \rangle$ computable in exponential space but requiring exponential-size circuits.

3. Show that if $\mathbf{NP} \subseteq \mathbf{BPP}$ then $\mathbf{RP} = \mathbf{NP}$.

4. In this question we are concerned with the equality of \mathbf{P} and \mathbf{NP} relative to some oracle, i.e., with $\mathbf{P}^A \stackrel{?}{=} \mathbf{NP}^A$ relative to some A .

- (a) Show that there exists an oracle A such that $\mathbf{P}^A = \mathbf{NP}^A$.

Now suppose that we generate an oracle A according to the following random procedure: for every string $x \in \{0,1\}^*$ we flip a coin to determine if $x \in A$. In other words, $\forall x, \Pr[x \in A] = \frac{1}{2}$. In what follows you are going to show that for such an A ,

$$\Pr[\mathbf{P}^A \neq \mathbf{NP}^A] = 1$$

which means that there must exist an oracle separating \mathbf{P} and \mathbf{NP} , and in fact “most” oracles separate \mathbf{P} and \mathbf{NP} ! (For this question, it is useful to know that $\lim_{m \rightarrow \infty} (1 - \frac{1}{m})^m = \frac{1}{e} = 0.367 \dots$)

There is a list of all polynomial time oracle Turing machines, $M_1^\square, M_2^\square, M_3^\square, \dots$ (Note that the \square indicates a “slot” into which we put an oracle A of our choosing.) For any oracle A , $\mathbf{P}^A = \{L(M_i^A) \mid i \geq 1\}$.

Given a random oracle A , define the language $L(A)$ as follows: arrange all strings in $\{0,1\}^*$ in lexicographic order. Given an x , $|x| = n$, to determine if $x \in L(A)$, look at

the segment of $n2^n$ strings that follow x (in this lexicographic order). Imagine this segment consisting of 2^n blocks, of n strings each. Then, $x \in L(A)$ iff there is at least one such block of n strings all of which are in A .

$$0, 1, 00, 01, \dots, x = x_i, \underbrace{\dots, \dots}_{n}, \underbrace{\dots, \dots}_{n}, \dots, \underbrace{\dots, \dots}_{n}, x_{i+(n2^n+1)}, \dots$$

- (b) Given a random oracle A , what is the probability that a string x with $|x| = n$ is in $L(A)$? What does this probability converge to as n grows large?
- (c) Show that for *any* oracle A , $L(A) \in \mathbf{NP}^A$.

We conclude that $\Pr[\mathbf{P}^A = \mathbf{NP}^A] \leq \Pr[L(A) \in \mathbf{P}^A]$.

- (d) Show that

$$\begin{aligned} & \Pr[L(A) \in \mathbf{P}^A] && (1) \\ & \leq \sum_i \Pr[\forall j (x_j \in L(M_i^A) \square L(A))] && (2) \\ & \leq \sum_i \prod_j \underbrace{\Pr[x_j \in L(M_i^A) \square L(A) \mid x_k \in L(M_i^A) \square L(A), \forall k < j]}_{(4)} && (3) \end{aligned}$$

where $X \square Y = \{w \mid w \in X \iff w \in Y\}$, the x_j 's are any subsequence of the lexicographic ordering of $\{0, 1\}^*$, and $\Pr[A|B] = \Pr[A \cap B] / \Pr[B]$ is the conditional probability. (You can use the given equation numbers in your answer.)

- (e) For the rest of this question we are concerned with finding an appropriate subsequence $\{x_j\}$ so that we can bound (4) by 0.9. Once we are able to do that, we are done, since $\sum_{i=1}^{\infty} \prod_{j=1}^{\infty} 0.9 = \sum_{i=1}^{\infty} 0 = 0$. (**Hint.** Choose the x_j 's very far apart from each other, so that the events " $x_j \in L(A)$ " are independent. It is not possible to make the events " $x_j \in L(M_i^A)$ " completely independent, since nothing stops M_i^A from making small queries; but we can make them sufficiently independent by noting that M_i^A cannot make *huge* queries and cannot make *too many* queries. Simplify your reasoning with the diagram:

	$x_j \in L(M_i^A)$	$x_j \notin L(M_i^A)$
$x_j \in L(A)$	E_1	E_2
$x_j \notin L(A)$	E_3	E_4

and examine the events E_1, E_2, E_3, E_4 .)

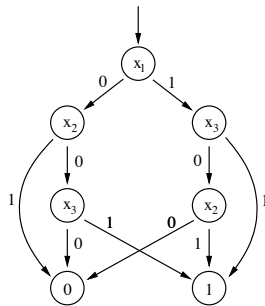
5. We say that a (possibly non-total) function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is *honest* if there exists a polynomial q such that for all $y \in \text{range}(f)$, there exists an x such that $|x| \leq q(|y|)$ and $f(x) = y$. We say that it is *polytime invertible* if there is a polytime (possibly non-total) function g such that for all $y \in \text{range}(f)$, $y \in \text{domain}(g)$ and $g(y) \in \text{domain}(f)$ and $f(g(y)) = y$.

We say that a (possibly non-total) function f is *one-way* if (i) f is polytime, (ii) f is not polytime invertible, and (iii) f is honest.

Show that a one-way function exists $\iff \mathbf{P} \neq \mathbf{NP}$.

(**Hint for** [\Leftarrow]). Assume you have some pairing function $\langle \cdot, \cdot \rangle : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$, which is a bijection and it is polytime computable, and polytime invertible. Suppose $\mathbf{P} \neq \mathbf{NP}$, and say $A \in \mathbf{NP} - \mathbf{P}$, so it has a nondeterministic polytime machine N deciding it. Let $f(\langle x, w \rangle)$ be $0x$ if w is an accepting path of $N(x)$, and $1x$ otherwise.)

6. [BONUS QUESTION] A *branching program* is a directed acyclic graph where all nodes are labeled by variables, except for two output nodes labeled 0 and 1. The nodes that are labeled by variables are called *query nodes*. Every query node has two outgoing edges, one labeled 0 and the other labeled 1. Both output nodes have no outgoing edges. One of the nodes in a branching program is designated the start node.



Here is an example of a branching program. Note that this branching program is a *read-once branching program*: each variable is queried at most once on each directed path from the start node to an output node.

Let $\text{EQ}_{\text{ROBP}} = \{\langle B_1, B_2 \rangle \mid B_1, B_2 \text{ are read-once BP such that } L(B_1) = L(B_2)\}$. Note that $x \in L(B)$ if on string x the branching program outputs 1 (in other words, BPs compute boolean functions in the natural way).

Show that $\text{EQ}_{\text{ROBP}} \in \mathbf{BPP}$. (**Hint.** Associate a polynomial with each edge and node. Assign the constant 1 to the start node. If a node labeled x has been assigned a polynomial p , assign the polynomial xp to the outgoing 1-edge and $(1 - x)p$ to the outgoing 0-edge. If the edges incoming to some node have all been assigned polynomials, assign the sum of those polynomials to this node.) **End of Exam**