Assignment 5 **CAS705 Fall 2005** **Due on November 21th**

Solutions by **Grzegorz Herman**.

The work you submit must be your own. *You may discuss problems with each other; however, you should prepare written solutions alone. In particular, you should not leave with any written notes from such discussions. The style and clarity of your answers will be an important factor in the grade.*

The following questions are problems from chapter 11 in Papadimitriou.

1. The classes $\mathbf{RP}, \mathbf{BPP}, \mathbf{PP}$ are defined in terms of quantity of accepting/correct computations of precise, polytime machines. As logspace reductions are deterministic, polytime and preserve the notion of acceptance, they can be "incorporated" into appriopriate machines preserving all above requirements. Therefore the discussed classes are closed under logspace reductions.

2. Both **BPP** and **RP** allow "magnifying", therefore we can obtain a required bound for probability of error/false negatives. It is also easy to see that in case of **RP** we can avoid any false positives.
   To prove that **PP** is closed under complement, we need to show that it is possible to modify a nondeterministic Turing machine in such a way, that the numbers of accepting and rejecting computation paths will always differ (of course preserving the inequality between them). To achieve this we first make sure that the left-most computation path is always an accepting one (this can be done preserving the accept/reject relation by splitting the computation tree into 4: an always-accepting, an always-rejecting and two identical to the original one). Then we add an "additional bit" to the state of the machine to keep track whether the computation follows the left-most path. If this is the case, we nondeterministically accept or reject. Otherwise we do as before (of course we need to create two identical branches to make the machine precise). Now if the original machine would accept on exactly half of the branches, the new one will accept on one less than a half. Therefore we can get a machine for the complement of the original language by reversing the accept and reject answers.
   Knowing that **PP** is closed under complement it is easy to show that it is also closed under symmetric difference. That is because for any numbers p, q such that $\frac{1}{2} < p, q \leq 1$ either $p(1-q)+q(1-p)$ or $pq+(1-p)(1-q)$ has to be greater than $\frac{1}{2}$. Therefore for any two languages $L_1, L_2 \in \mathbf{PP}$ we can use one of the following to get their symmetric difference:

$$L_1 \div L_2 = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$$
$$L_1 \div L_2 = \overline{(L_1 \cap L_2) \cup (\overline{L_1} \cap \overline{L_2})}$$

3. Consider any problem $\pi \in \mathbf{PP}$. Having the machine solving it we can create a boolean formula encoding possible computations (as in the Cook's Theorem). Let us consider two parts of this formula: ϕ, meaning "the computation is correct (well-formed)", and

ψ, meaning "the computation is accepting". Let us introduce a new variable x and consider the following formula:

$$\theta := (\phi \supset \psi) \wedge (\neg\phi \supset x)$$

It is easy to see, that if a truth assignment does not satisfy ϕ, then to satifsy θ it must satisfy x. It means that exactly half of these assignments satisfy θ. On the other hand if a truth assignment satisfies ϕ then, to satisfy θ, it must satisfy ψ. It means that more than half of all assignments satisfy θ if and only if more than a half correct computations are accepting. And, as the size of θ is of course polynomial in the size of the instance of π, the reduction works correctly.
Thus any problem in **PP** can be reduced do $MAJSAT$ which means that $MAJSAT$ is **PP**-complete.

4. We know that $\mathbf{RP} \subseteq \mathbf{NP}$. Assume that $\mathbf{NP} \subseteq \mathbf{BPP}$. Then $SAT \in \mathbf{BPP}$.
Consider the following algorithm for SAT: given a formula ϕ we first use the **BPP** algorithm to see if it is satisfiable. If the answer comes "no", we reject. If the answer comes "yes", we set the first variable x_1 to 0 and 1 and ask the **BPP** machine again. If both answers are "no", we reject. Otherwise we proceed with x_1 set to the value for which the answer was "yes". We repeat this procedure until we get a total truth assignment. Then we verify (deterministically!) that it really satisfies ϕ and accept if and only if it does.
From the last step it is clear, that our algorithm will not return "false positive" answers. The probability of a "false negative" can be bounded from above by the sum of probabilities of errors on each steps (and there are n of them). And as the probability of error of an **BPP** algorithm can be made arbitrarily (up to exponentially) small by "magnifying", we can easily limit the resulting probability of false negatives by $\frac{1}{2}$.
Thus $SAT \in \mathbf{RP}$, but as SAT is **NP**-complete, $\mathbf{NP} \subseteq \mathbf{RP}$ and finally, $\mathbf{NP} = \mathbf{RP}$.

5. If a clause has at least $k\log(n)$ literals, then the number of truth assignments which falsify it is less than $2^{n-k\log(n)} = \frac{2^n}{n^k}$. Then the number of truth assignments which falsify all clauses is less than $n^k \frac{2^n}{n^k} = 2^n$ and therefore there must be at least one which satisfies the whole formula.
Consider the original formula ϕ and set the first variable to either 0 or 1 getting formulae ϕ_0 and ϕ_1. If we denote by $f(\psi)$ the number of truth assignments falsifying ψ, we can easily see that

$$f(\phi) = f(\phi_0) + f(\phi_1)$$

But we know that $f(\phi) < 2^n$ and it means that for i being either 0 or 1

$$f(\phi_i) < 2^{n-1}$$

It turns out, that if we select the i which satisfies the most clauses (and choose i arbitrarily if there is a tie) we can get a satisfying assignment (see Papadimitriou, theorem 13.2). Repeating the above step n times we will get a total truth assignment satisfying the original formula.

6. (a) If $L \in \mathbf{P}/n^k$ then it can be decided by a deterministic machine with advice. But then for each length n the advice is fixed, so we can encode the whole computation of this machine using a polynomial size circuit.
If, on the other hand, L has polynomial circuits, we can construct the following machine with advice to decide it:

- treat the advice as a description of a circuit
- simulate this circuit on your input
- accept iff the result was "1"

It is clear that this is a polytime machine which, when given descriptions of circuits for L as the advice, decides L. Therefore $L \in \mathbf{P}/n^k$.

(b) Assume that $SAT \in \mathbf{P}/\log(n)$. Then it has a machine with advice M deciding it. We can construct a new machine (without advice) to check whether a given formula ϕ is satisfiable. First let us denote by A the set of "valid" advice strings (defined later) and let $A_i := \{s \in A : |s| = \log(i)\}$. Then define recursive function $check(\phi, t, s)$ as follows:

- given a formula ϕ, a partial truth asssignment t (with i variables unset) and a string $s \in A_i$:
- if $i = 0$ then check if t satisfies ϕ and exit with appriopriate answer
- run M with advice s on $\phi[t]$
- if "no", answer "no"
- if "yes" then try setting the next variable in ϕ to 0 or 1 and running $check()$ recursively with all "advice candidates" from A_{i-1}
- if any of the answers is "yes", answer "yes"
- otherwise remove s from A and answer "no"

Now the algorithm to decide whether ϕ is satisfiable looks as follows:

- $A :=$ set of all possible advice strings of lengths up to $\log(n)$
- if $check(\phi, \emptyset, s)$ for any $s \in A_n$, answer "yes"
- answer "no"

It is easy to see that this algorithm correctly decides whether ϕ is satisfiable (it checks all possible advice strings, it never removes a "correct" advice string and validates every potential truth assignment to exclude false positives). The question is - does it run in polynomial time? The "worst case scenario" is when ϕ is not satisfiable and the algorithm has to "eliminate" all possible truth assignments. But then for each s at most one invocation of $check()$ will reach the phase of performing recursive calls (because then s will be removed from A). Thus the total number of calls to $check()$ is bounded by $|A|^2 = O(n^4)$.
Thus $SAT \in \mathbf{P}$ and therefore $\mathbf{P} = \mathbf{NP}$.

(c) As I was unable to solve (one direction of) the problem in its original statement, I will modify it slightly, allowing the considered circuit C_i to have polynomially many inputs (and still exactly i outputs).

Now if $L \in \mathbf{NP}/n^k$ then we have a nondeterministic machine M with advice deciding it. Thus for each input length i we can treat it as a "plain" nondeterministic machine. Thus we can construct a circuit C_i (of polynomial size) which given $i+i^k$ inputs will simulate computation of M on the word created from first i inputs, resolving all nondeterministic choices using the remaining inputs. If M accepts, the circuit outputs the string checked. Otherwise the output is 0^i. It is clear that this circuit has exactly the property defined in the question.
If, on the other hand, we know that L has a family of circuits with this property, we can create the following machine deciding L:

- on input x with $|x| = i$
- if $x = 0^i$ then return the first bit of advice
- nondeterministically choose $i + i^k$ bits
- treat the remaining bits of advice as the description of a circuit with $i + i^k$ inputs, and calculate the i-bit output: y
- accept iff $x = y$

As we know that the appriopriate family of circuits exists, we can define the advice strings to contain their descriptions. The additional (first) bit of advice enables us to distinguish if $0^i \in L$ (the circuits cannot give us this information). It is clear that the machine described decides L in polynomial time. Therefore $L \in \mathbf{NP}/n^k$. (This direction of course works well for circuit C_i having exactly i inputs).

(d) The Halting Problem, when encoded in unary, is of course undecidable. But the machine which simply returns the first bit of its advice as the result solves it (the advice contains a complete answer, because we need only one bit for every input length). This means that the Unary Halting Problem is in $\mathbf{P}/1 \subseteq \mathbf{P}/\log(n)$.