Complexity Theory

Homework Sheet 3

Hand in before the lecture of Tuesday 28 Feb.

Preferably by email to bannink@cwi.nl

21 February 2017

Exercise 1. Is there an oracle such that, relative to this oracle, ...? If so, then exhibit such an oracle and prove it works. If not, prove why not.

(a) $\mathbf{P} = \mathbf{EXP}$

(b) $\mathbf{coNP} \subseteq \mathbf{P}$ *and* $\mathbf{NP} \not\subseteq \mathbf{P}$

(c) $\mathbf{NP} = \mathbf{coNP} \neq \mathbf{EXP}$

For example, in **(a)** you have to either show: there exists an oracle A such that $\mathbf{P}^A = \mathbf{EXP}^A$, or: such an oracle does not exist. In **(b)** you show either: there exists an oracle A such that $\mathbf{coNP}^A \subseteq \mathbf{P}^A$ and $\mathbf{NP}^A \not\subseteq \mathbf{P}^A$, or: such an oracle does not exist.

Solution.

(a) By the Time Hierarchy Theorem we know that $\mathbf{P} \subsetneq \mathbf{EXP}$ and this is a proof that relativizes, so this holds relative to any oracle. Hence there is no oracle such that relative to this oracle $\mathbf{P} = \mathbf{EXP}$.

Common mistake regarding $\mathbf{P} \subsetneq \mathbf{EXP}$
The time hierarchy theorem is stated in terms of the classes $\mathbf{DTIME}(f(n))$ and $\mathbf{DTIME}(g(n))$. From this theorem it directly follows that $\mathbf{DTIME}(n^c) \subsetneq \mathbf{DTIME}(2^{n^{c'}})$ (for all c, c'). One might think that this means $\mathbf{P} \subsetneq \mathbf{EXP}$. This by itself, however, is **not** sufficient to show that $\mathbf{P} \subsetneq \mathbf{EXP}$ where $\mathbf{P} = \cup_{c\geq 1}\mathbf{DTIME}(n^c)$ and $\mathbf{EXP} = \cup_{c\geq 1}\mathbf{DTIME}(2^{n^c})$.
To see why this is not enough, consider the closed interval $A_j = [1/j, 1]$ and the union $A = \cup_{j\geq 1} A_j$. Then for each j we have $A_j \subsetneq (0,1]$, (for every j there is an $x \in (0,1]$ with $x \notin A_j$). However, once we take the union we see that $A = (0,1]$.
Similarly, with the argument above we have shown that seperately for each part of the union there is a language in $\mathbf{EXP}$ that is not in that part of $\mathbf{DTIME}(n^c)$. But instead we have to show that there is such a language that does the job for all c at the same time. To do this, we can instead separate two classes *within* $\mathbf{EXP}$: note that $\mathbf{DTIME}(2^n) \subsetneq \mathbf{DTIME}(2^{n^2})$ (by the time hierarchy theorem). Since $\mathbf{DTIME}(n^c) \subset \mathbf{DTIME}(2^n)$ for all c, we have that $\mathbf{P} \subset \mathbf{DTIME}(2^n)$ and hence $\mathbf{P} \subsetneq \mathbf{DTIME}(2^{n^2})$ so $\mathbf{P} \subsetneq \mathbf{EXP}$.

(b) Let A be any language such that assume $\mathbf{coNP}^A \subseteq \mathbf{P}^A$. Now let $L \in \mathbf{NP}^A$, then $\bar{L} \in \mathbf{coNP}^A$ and by assumption $\bar{L} \in \mathbf{P}^A$. We then have $L \in \mathbf{P}^A$ because one can flip the output of the polynomial time machine that decides $\bar{L}$. This shows $\mathbf{NP}^A \subseteq \mathbf{P}^A$. We conclude that there is no oracle such that the required inclusions hold.

(c) Use the language EXPCOM (called K_{EXP} in the lecture). It was shown in the lecture (also in the book) that with this oracle one has $\mathbf{NP}^{\text{EXPCOM}} = \mathbf{coNP}^{\text{EXPCOM}} = \mathbf{P}^{\text{EXPCOM}} = \mathbf{EXP}$. Furthermore from the argument in (a) it follows that $\mathbf{P}^{\text{EXPCOM}} \neq \mathbf{EXP}^{\text{EXPCOM}}$ and hence the required (in)equalities are satisfied relative to the oracle EXPCOM.

Possible mistake when showing $\mathbf{EXP} \neq \mathbf{EXP}^{\text{EXPCOM}}$.
Intuitively, the reason that $\mathbf{EXP}^{\text{EXPCOM}}$ is strictly larger is that on an input of size n, an exponential-time Turing Machine can write queries to EXPCOM that are of size 2^{n^c}. The EXPCOM oracle then does a computation that is exponential in 2^{n^c}, so it is effectively doing a *double* exponential computation in terms of the original input size n. This intuition is valid, and by the time hierarchy theorem we have that $\mathbf{DTIME}(2^{n^{c'}}) \subsetneq \mathbf{DTIME}(2^{2^{n^c}})$. However, since $\mathbf{EXP}$ is defined with an infinite union, one has to be careful about directly concluding $\mathbf{EXP} \neq \mathbf{EXP}^{\text{EXPCOM}}$, for the same reason as explained at 'common mistakes' in (a).

Exercise 2. Let $\mathbf{C}$ be the class of sets decidable by Turing machines which use polynomial space, but are not allowed to reuse space. I.e., the Turing machine can write over blank tape cells, but it can neither erase nor overwrite previously used cells – it *is* allowed to go back over the tape and read them.

(a) Prove that $\mathbf{C} \subseteq \mathbf{P}$.

(b) Prove that $\mathbf{P} \subseteq \mathbf{C}$.

Definition 1. We say that a set A is *1-query length-decreasing self-reducible* if there is a polytime oracle Turing machine M, such that

$$x \in A \iff M^A(x) = 1.$$

Moreover, the computation of $M^A(x)$ makes at most one query to the oracle, and that query has to be a string of length strictly less than $|x|$.

Exercise 3. Show that every 1-query length-decreasing self-reducible set is in $\mathbf{P}$.

Exercise 4. Prove that in the certificate definition of $\mathbf{NL}$ (Book §4.3.1), if we allow the verifier machine to move its head back and forth on the certificate, the class being defined changes to $\mathbf{NP}$. Hint: Consider 3-SAT.

Solution. Let $\mathbf{NL}'$ be the alternate definition where the verifier can read the certificate tape cells more than once. The goal is to show that $\mathbf{NL}' = \mathbf{NP}$. By counting the number of configurations of the configuration graph one can see that any $\mathbf{NL}'$ verifier machine takes at most polynomial time and hence $\mathbf{NL}' \subseteq \mathbf{NP}$. It remains to show that $\mathbf{NP} \subseteq \mathbf{NL}'$. To show this, we will first show that SAT $\in \mathbf{NL}'$ and then show that any language in $\mathbf{NP}$ reduces to SAT *using only logarithmic space.*

We first show that 3-SAT $\in \mathbf{NL}'$. The certificate represents a truth assignment to the boolean variables, and has length exactly n when there are n variables. A verifier machine M gets an input φ (a formula in 3-CNF form)[1] and a certificate u. Now M goes over all clauses of the formula, and for each clause does the following. Set a bit z to zero. Now look up the (three) relevant bits in u. This requires a pointer (counter) of size $\log n$, since the formula makes a reference to a variable, say x_i, and M then has to look up the i-th bit of u. After looking up the bit, check if this makes the clause true and set z to 1 if so. If not, continue with the next variable in that clause. If after these three lookups z is still zero then output false and halt. If z was one then, clear all space and continue with the next clause. After the final clause has been validated, output true and halt. This machine requires only a pointer of size $\log n$ (and some constant size memory) and correctly verifies any formula. We conclude that 3-SAT $\in \mathbf{NL}'$.
(Note that this algorithm also works for SAT because even if the size of a single clause becomes large, M only verifies the clause one literal at a time, and the variable z keeps track of the 'or' of the literals so far. The pointer that is used to loop up the bit in u can be re-used for every literal in the clause. Hence we also have SAT $\in \mathbf{NL}'$.)

By the Cook-Levin theorem we know that any language L in **NP** reduces to SAT (or 3-SAT) in polynomial time. (i.e. a polytime Karp reduction). This is not enough for this exercise. However the result of the theorem is stronger because these reductions can also be done using only logarithmic space. To see this, look at the proof of the Cook-Levin theorem and note that this proof deals with 'snapshots' of constant size and with 'head positions' and 'times' which are logarithmic in size. To construct a SAT instance, the reduction machine creates *constant size clauses* that verify these snapshots and positions. Once such a clause has been written to the output tape, the machine can reuse all work-space to create the next clause. The output tape does not count as space usage, and hence these reductions can completely be done in logarithmic space.

Now we have that any language $L \in \mathbf{NP}$ is in $\mathbf{NL}'$ because $L \leq_{\log}$ SAT and SAT $\in \mathbf{NL}'$, so we conclude $\mathbf{NL}' = \mathbf{NP}$.

Something to think about: Why does the following *not* show that SAT is in **NL**: Consider the same machine as before, but as a certificate simply use many copies of the truth-assignment, so simply repeat the bitstring u many times and use that as a new certificate. Then everytime the verification machine wants to loop up a bit in u, it can go to the next copy and never has to move back. Now it is a *read-once* certificate. Why does this *not* mean that SAT is in **NL**?

[1]The machine has to check if the input is of this form; this can be done in logspace