Computation, Machines and Formal Languages - Solution of the Exam (spring 2002) Moed A

Lecturer: Prof. Michael Ben-Or.

Time: 2.5 hours.

Write your answers on the exam sheet.

If the answer is yes/no, **write down the answer that you chose**.

Answer all the questions.

Unless stated otherwise, you have to add a short explanation (within the designated area for each question) for your answers. Answers that will not be accompanied by explanations will not be credited. The explanation should include the main steps in the proof, do not get into technical details, notations etc. If your explanation is given by a contradicting example, you have to state the example together with a short explanation of the contradiction. If you have an example that contradicts an unproven but a reasonable assumption (such as $P \neq NP$), state the example, the reasonable assumption, and a short explanation of the contradiction.

Definitions and Notations: (identical to those we used in class)

For a language $\mathcal{L}$, we denote by $\overline{\mathcal{L}}$ the language $\Sigma^* \setminus \mathcal{L}$.

Complexity Classes:
REG is the class of regular languages.
CFL is the class of context-free languages.
R is the class of languages that are decidable by Turing Machines (TM).

RE is the class of languages that are recognizable by TM's.
P is the class of languages that are decidable by deterministic TM's in polynomial time.
NP is the class of languages that are decidable by non-deterministic TM's in polynomial time.
$PSPACE$ is the class of languages that are decidable by deterministic TM's in polynomial space.
L is the class of languages that are decidable by deterministic TM's in logarithmic space.
NL is the class of languages that are decidable by non-deterministic TM's in logarithmic space.
For a class of languages $\mathcal{C}$, we denote by $co\mathcal{C}$ the class of languages that their complements are in $\mathcal{C}$.

Computational problems:
$SUBSET-SUM = \{(S,t) :$ S is a (multi) set of numbers that contains a subset that sums to $t\}$
$HAM-PATH = \{(G,s,t) :$ G is a directed graph, that contains a path from s to t, that visits every node in the graph exactly once$\}$
We say that a Boolean formula is $3-cnf$ if it is of the form:

$$\bigwedge_i (a_1^i \vee a_2^i \vee a_3^i)$$

where a_j^i is a variable or its negation.
$3-SAT = \{\phi : \phi$ is a satisfiable $3-cnf\}$
$k-color = \{G :$ given k colors, it is possible to assign a color to each node in the graph G, such that no two adjacent nodes get the same color$\}$
$ALL_{CFG} = \{G : G$ is a context free grammar, and $L(G) = \Sigma^*\}$
$ALL_{DFA} = \{A : A$ is a deterministic finite automaton, and $L(A) = \Sigma^*\}$
$QBF = \{\phi : \phi$ is a true fully quantified Boolean formula$\}$

Reductions:
For two languages $\mathcal{L}_1, \mathcal{L}_2$, $\mathcal{L}_1 \leq_P \mathcal{L}_2$ denotes that there is a polynomial-time mapping reduction from $\mathcal{L}_1$ to $\mathcal{L}_2$.
For two languages $\mathcal{L}_1, \mathcal{L}_2$, $\mathcal{L}_1 \leq_L \mathcal{L}_2$ denotes that there is a logarithmic-space mapping reduction from $\mathcal{L}_1$ to $\mathcal{L}_2$.

1. (12 points)

   Version A Draw a **minimal deterministic** finite automaton for the language $\mathcal{L} = \{w : |w| = 2k, k \geq 0\}$ (over $\Sigma = \{0, 1\}$).
   Give a short proof for the correctness and the minimality.

   Solution: The automaton $\mathcal{A}$ is in Figure 1. Indeed, the run of $\mathcal{A}$ on a word of even length ends in the initial state, which is also accepting, and the run of $\mathcal{A}$ on a word of odd length ends in the non-accepting state. The automaton $\mathcal{A}$ is minimal, since $\mathcal{L}$ defines two equivalence classes: of words of even length and of words of odd length.

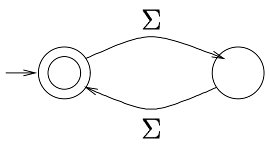

Figure 1: The minimal deterministic automaton for the language of words of even length

   Version B Draw a **minimal deterministic** finite automaton for the language $\mathcal{L} = \{w : |w| = 2k + 1, k \geq 0\}$ (over $\Sigma = \{0, 1\}$).
   Give a short proof for the correctness and the minimality.

   Solution: The automaton $\mathcal{A}$ is in Figure 2. Indeed, the run of $\mathcal{A}$ on a word of odd length ends in the accepting state, and the run of $\mathcal{A}$ on a word of even length ends in the non-accepting state. The automaton $\mathcal{A}$ is minimal, since $\mathcal{L}$ defines two equivalence classes: of words of even length and of words of odd length.

2. (13 points)

   Version A Let us define the following languages over $\Sigma = \{a, b\}$:

$$L = \{a^n b^m a^k : n \geq m \geq k\}$$

$$K = a^* b^*$$

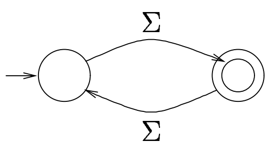

Figure 2: The minimal deterministic automaton for the language of words of odd length

For each one of the following languages, write the smallest class of languages that contains it out of the following list: REG, CFL, or P.

(a) $L \cap K$ - the answer is CFL

(b) $\overline{L}$ - the answer is CFL

(c) $\overline{L} \cap K$ - the answer is CFL

(d) L - the answer is P

Version B Let us define the following languages over $\Sigma = \{a, b\}$:

$$L = \{a^n b^m a^k : n \leq m \leq k\}$$

$$K = b^* a^*$$

For each one of the following languages, write the smallest class of languages that contains it out of the following list: REG, CFL, or P.

(a) $L \cap K$ - the answer is CFL

(b) $\overline{L}$ - the answer is CFL

(c) $\overline{L} \cap K$ - the answer is CFL

(d) L - the answer is P

3. (13 points)

   (a) For each one of the following languages, write the smallest class that contains it out of the following list: R, RE, $coRE$, or none of these classes.

i. ALL_{CFG}
Answer: $coRE$
In class we saw $\overline{A_{TM}} \leq ALL_{CFG}$: given a pair $\langle M, w\rangle$ we constructed a CFG G such that G derives all words that are not accepting computations of M on w. Then, M does not accept w iff $\mathcal{L}(G) = \Sigma^*$. Therefore, ALL_{CFG} is not in RE. The complement of ALL_{CFG} is recognizable, since given a CFG G, we can check for all words, trying them in alphabetic order, whether G derives them or not (we saw in class a polynomial algorithm that checks, given G and w, whether G derives w). For each word w the check is finite, and thus if there exists a word that G does not derive, the algorithm will stop and give a positive answer.

ii. ALL_{DFA}
Answer: R
We saw in class an algorithm that given a DFA $\mathcal{A}$ checks whether $\mathcal{L}(\mathcal{A}) = \Sigma^*$: the algorithm computes $\overline{\mathcal{A}}$, and then checks emptiness of $\mathcal{L}(\bar(\mathcal{A}))$.

iii. $\mathcal{L} = \{< M >: L(M)$ contains more than 100 words$\}$
Answer: RE
The language $\mathcal{L}$ is not in R by Rice theorem. It is in RE, since given a TM M, we can guess 101 words that it accepts and run M on all of them in parallel. If indeed M stops and accepts, then $\langle M\rangle \in \mathcal{L}$.

(b) For each one of the languages for which you answered R, write the smallest class that contains it out of the following list: L, NL, P.
Answer: $ALL_{DFA} \in NL$
$ALL_{DFA} \in NL$, since the translation of a DFA $\mathcal{A}$ into a complement DFA $\overline{A}$ can be done in L by reversing the roles of accepting and non-accepting states, and then the check of emptiness of $\overline{\mathcal{A}}$ is done by running the algorithm $PATH$ for the initial state and each one of the accepting states. E_{DFA} is also NL-hard by the reduction from $\overline{PATH}$: given $\langle G, s, t\rangle$, there is a path from s to t in G iff the automaton $\mathcal{A}_G$, obtained from the graph G by setting states to be the nodes of G, and edges transitions (the determinism can be ensured by choosing the alphabet of size equal to the maximal branching degree of G), with the initial state S and the single accepting state t, has an empty language. Now, from NL-hardness of E_{DFA} follows NL-hardness of ALL_{DFA}, since the reduction from E_{DFA} to ALL_{DFA} is in L. Thus, it is likely that $ALL_{DFA} \notin L$.

4. (12 points)

Version A For each one of the following functions, answer whether it is $\Theta(1)$, $\Theta(\log n)$, or $\Theta(n)$.

(a) $f_1(n) = max\{|K(x) - K(xx^R)| : x \in \{0,1\}^n\}$
Answer: $\Theta(1)$
First we show that $K(xx^R) \leq K(x) + c$ (for some constant c). Let $(< M >, w)$ be the minimal description of the string x. Define the machine M', that on every input y, outputs yy^R. Now define the machine M'' that on every input, first runs M and then M'. Clearly, $(< M'' >, w)$ is a description of the string xx^R, and its length is it most $K(x) + | < M' > | = K(x) + c$.
Similarly we show that $K(x) \leq K(xx^R) + c$. Let $(< M >, w)$ be the minimal description of the string xx^R. Define the machine M', that on every even length input y, it outputs the $|y|/2$-long prefix of y. Now define the machine M'' that on every input, first runs M and then M'. Clearly, $(< M'' >, w)$ is a description of the string x, and its length is it most $K(xx^R) + | < M' > | = K(xx^R) + c$.

(b) $f_2(n) = max\{|K(x) - K(y)| : x, y \in \{0,1\}^{2n}$, and in each one of the strings x and y, exactly half the symbols are 1 and half are $0\}$
Answer: $\Theta(n)$
Clearly the difference in the description complexity of every two strings of length $2n$ cannot be more than $O(n)$ because for every x, $K(x) \leq |x| + c$.
We show that there is a string of length $2n$ with exactly n 1's, and Kolmogorov complexity $\Omega(n)$. The number of such strings is $\binom{2n}{n} = 2^{\Omega(n)}$. Therefore we need this number of different strings to describe all the strings of length $2n$ with exactly n 1's. It follows that at least one of these descriptions must be of length $\log(2^{\Omega(n)}) = \Omega(n)$. Let x be a string with this Kolmogorov complexity. Let y be the string 0^n1^n. Clearly, $k(y) = O(\log n)$. Therefore,

$$f_2(n) \geq K(x) - K(y) = \Omega(n) - O(\log n) = \Omega(n)$$

Version B For each one of the following functions, answer whether it is $\Theta(1)$, $\Theta(\log n)$, or $\Theta(n)$.

(a) $f_1(n) = max\{|K(x) - K(double(x))| : x \in \{0,1\}^n\}$
Where $double(x)$ is the string x where each bit appears twice.
Example: $double(100) = 110000$
Answer: $\Theta(1)$
First we show that $K(double(x)) \leq K(x) + c$ (for some constant c). Let $(< M >, w)$ be the minimal description of the string x. Define the machine M', that on every input y, outputs $double(y)$. Now define the machine M'' that on every input, first runs M and then M'. Clearly, $(< M'' >, w)$ is a description of the string $double(x)$, and its length is it most $K(x) + | < M' > | = K(x) + c$.
Similarly we show that $K(x) \leq K(double(x)) + c$. Let $(< M >, w)$ be the minimal description of the string $double(x)$. Define the machine M', that on every even length input y, it outputs every second (even) bit of y. Now define the machine M'' that on every input, first runs M and then M'. Clearly, $(< M'' >, w)$ is a description of the string x, and its length is it most $K(double(x)) + | < M' > | = K(double(x)) + c$.

(b) $f_2(n) = max\{|K(x) - K(y)| : x, y \in \{0,1\}^n$, and in each one of the strings x and y, exactly 3 of the symbols are $1\}$
Answer: $\Theta(\log n)$
The Kolmogorov complexity of every string of length n and exactly three 1's, is at most $O(\log n)$, because we only have to tell the machine the number n and the positions of the three 1's, which takes another $O(\log n)$. Therefore, the difference in the description complexity of every two such strings cannot be more than $O(\log n)$.
We show that there is a string of length n with exactly three 1's, and Kolmogorov complexity greater than $2 \log n$. The number of such strings is $\binom{n}{3} = \Theta(n^3) > n^2$. Therefore we need at least n^2 different strings to describe all the strings of length n with exactly three 1's. It follows that at least one of these descriptions must be of length $\log(n^2) = 2 \log n$. Let x be a string with this Kolmogorov complexity. Let y be the string $1^3 0^{n-3}$. Clearly, $k(y) \leq \log n + c$. Therefore,

$$f_2(n) \geq K(x) - K(y) \geq 2 \log n - (\log n + c) = \Omega(\log n)$$

5. (13 points)

Version A Let $\mathcal{L}$ be the following language. $\mathcal{L} = \{\langle \varphi_1, \varphi_2 \rangle : \varphi_1$ and φ_2 are formulas in $3 - cnf$ and exactly one of φ_1, φ_2 is satisfiable $\}$. Answer the following questions.

(a) Is $\mathcal{L} \in NP$?

Unlikely, since $\mathcal{L}$ is co-NP-hard. We show a reduction from $\overline{3-SAT}$. Let x be a variable. We define a tautology $\varphi_x = x \vee \neg x$ (or in $3-cnf$: $(x \wedge x \wedge x) \vee (\neg x \wedge \neg x \wedge \neg x)$). Now, given a formula φ in $3-cnf$, it is clear that φ is unsatisfiable (belongs to $\overline{3-SAT}$) iff $\langle \varphi, \varphi_x \rangle \in \mathcal{L}$.

(b) Is $\mathcal{L} \in coNP$?

Unlikely, since $\mathcal{L}$ is NP-hard. We show a reduction from $3-SAT$. Let x be a variable. We define a contradiction $\varphi_x = x \wedge \neg x$ (or in $3-cnf$: $(x \wedge \neg x \wedge x) \vee (x \wedge \neg x \wedge x)$). Now, given a formula φ in $3-cnf$, it is clear that φ is satisfiable (belongs to $3-SAT$) iff $\langle \varphi, \varphi_x \rangle \in \mathcal{L}$.

(c) Is $\mathcal{L} \in PSPACE$?

Yes. For a given formula φ, we can check in $PSPACE$ whether φ is satisfiable (by trying all possible assignments one after another, using the same memory). Thus, we can perform the satisfiability check for both formulas and accept iff the results are opposite.

Version B The same question, where $\mathcal{L}$ is defined as $\mathcal{L} = \{\langle \varphi_1, \varphi_2 \rangle : \varphi_1$ and φ_2 are formulas in $3-cnf$ and either both φ_1, φ_2 are satisfiable or both are unsatisfiable. The answer is the same, and the reductions are similar to ones in Version A, where we pair an input formula with a contradiction for the reduction from $\overline{3-SAT}$, and with a tautology for the reduction from $3-SAT$.

6. (13 points)

Version A Under the assumption that $3-color \notin coNP$, are the following true?

(a) $3-SAT \in coNP$? **No**.

There is a polynomial reduction from $3-color$ to $3-SAT$, thus if we assume that $3-SAT \in coNP$, then $3-color$ should also be in $coNP$ by the reduction to $3-SAT$ and then solving $3-SAT$ in $coNP$, and this contradicts the assumption.

(b) $\overline{3-SAT} \notin P$? **Yes**.

If $3-color \notin coNP$, then $NP \neq coNP$, and thus $P \neq NP$ (since P is closed under complementation). Thus, NP-complete problems are not in P, and in particular $3-SAT \notin P$. Now, if $\overline{3-SAT}$ if in P, then also $3-SAT \in P$ (because P is closed under complementation), and we got a contradiction.

(c) $QBF \in NP$? **No**.
If $3 - color \notin coNP$, then $NP \neq coNP$, and thus $NP \neq PSPACE$ (since $PSPACE$ is closed under complementation). Therefore, $PSPACE$-complete problems are not in NP, and since QBF is $PSPACE$-complete, it is not in NP.

Version B Under the assumption that $HAM - PATH \notin coNP$, are the following true?

(a) $\overline{SUBSET - SUM} \notin P$? **Yes**.
If $HAM - PATH \notin coNP$, then $NP \neq coNP$, and thus $P \neq NP$ (since P is closed under complementation). Thus, NP-complete problems are not in P, and in particular $SUBSET - SUM \notin P$. Now, if $\overline{SUBSET - SUM}$ if in P, then also $SUBSET - SUM \in P$ (because P is closed under complementation), and we got a contradiction.

(b) $QBF \in NP$? **No**.
If $3 - color \notin coNP$, then $NP \neq coNP$, and thus $NP \neq PSPACE$ (since $PSPACE$ is closed under complementation). Therefore, $PSPACE$-complete problems are not in NP, and since QBF is $PSPACE$-complete, it is not in NP.

(c) $3 - SAT \in coNP$? **No**.
There is a polynomial reduction from $3 - color$ to $3 - SAT$, thus if we assume that $3 - SAT \in coNP$, then $3 - color$ should also be in $coNP$ by the reduction to $3 - SAT$ and then solving $3 - SAT$ in $coNP$, and this contradicts the assumption.

7. (12 points) Let Σ be some fixed alphabet. The operation of *shuffle* is important in the study of concurrent systems. If $x, y \in \Sigma^*$, then $x \,||\, y$ is the set of all strings that can be made by shuffling x and y together like a deck of cards, that is, mixing the letters of x with the letters of y, while keeping the order of their appearance in the original words. For example,

$$ab \,||\, cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$$

The shuffle of two languages A and B, denoted $A \,||\, B$, is the set of strings obtained by shuffling a string from A with a string from B:

$$A \,||\, B = \bigcup_{\substack{x \in A \\ y \in B}} x \,||\, y$$

A class $\mathcal{C}$ of languages is **closed** under shuffle if for all $A, B \in \mathcal{C}$, we have $A \,||\, B \in \mathcal{C}$. Are the following classes closed under shuffle?

(a) REG
Answer: yes
Let $A, B \in REG$, and $\mathcal{A}_A, \mathcal{A}_B$ the corresponding DFA's. We define (informally) a NFA M for $A \,||\, B$. M is a cartezian product of the automata $\mathcal{A}_A$ and $\mathcal{A}_B$. That is, the states of M are pairs $\langle q, q' \rangle$, where q is a state of $\mathcal{A}_A$ and q' is a state of $\mathcal{A}_B$. The transition relation of M is defined as follows. $\langle q, q' \rangle \in \delta_M(\langle q_1, q_1' \rangle, \sigma)$ iff either $q = q_1$ and $q_1' = \delta_B(q_1, \sigma)$ or $q' = q_1'$ and $q_1 = \delta_A(q, \sigma)$. That is, on each step, M non-deterministically chooses which automaton to follow, and keeps the state of the other automaton unchanged. If at the end both automata are in accepting states, M accepts, otherwise it rejects (recall the cartezian product automaton we constructed to prove that REG is closed under intersection). Now if an input w is in $w_1 \,||\, w_2$ where $w_1 \in A$ and $w_2 \in B$, then there is a nondeterministic choice that picks the correct automaton in each step. And since $w_1 \in A$ and $w_2 \in B$, at the end of the computation, both automata will be in an accepting state.

(b) P
Answer: no (under the assumption $P \neq NP$)
Let us look at the languages Σ^* (over $\Sigma = \{0, 1\}$), and $L = \{(\phi, a) : \phi$ is a satisfiable $3 - cnf$, and a is a satisfying assignment$\}$. Clearly, both languages are in P. Suppose that P is closed under shuffle. Then there is a deterministic polynomial-time algorithm A, for $\Sigma^* \,||\, L$. We describe a deterministic polynomial-time algorithm for $3 - SAT$. On input ϕ, a $3 - cnf$ on n variables, the algorithm runs A on $(\phi, (01)^n)$. The algorithm accepts if and only if A accepts. Now if ϕ is satisfiable, then its satisfying assignment (actually, all the satisfying assignments) appear in the string $(01)^n$, shuffled with some string from Σ^*, and A will accept. On the other hand, if ϕ is unsatisfiable, then A will never accept because ϕ will never be part of a word in L.
There is one subtle point, we must define L such that the symbols in the "ϕ" part will not be in $\{0, 1\}$. Because otherwise, the shuffle can take a non-satisfiable formula and add there symbols from Σ^* to create a satisfiable formula.
We conclude that if P is closed under shuffle then $P = NP$.

(c) NP
Answer: yes

This is similar (and simpler) to the case of REG.
Let $A, B \in NP$, and M_A, M_B the corresponding nondeterministic TM's. We define (informally) a nondeterministic TM M for $A \,||\, B$. On input w, M chooses a separation of w into two shuffled words. It then runs M_A on the first and M_B on the second. If both accept then M accepts.

8. (12 points) A language C **separates** two disjoint languages A, B if $A \subseteq C$ and $B \subset \bar{C}$ (see Figure 3). Prove that for all pairs of disjoint languages A, B in $coRE$ there is a language in R that separates them.

   Hint: All languages in RE have enumerators that print them.

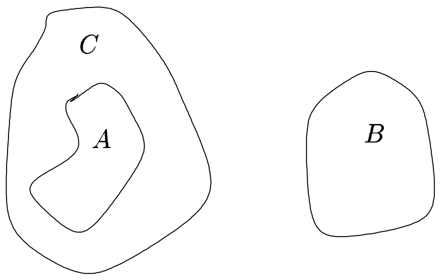

Figure 3: A separating language

Solution:

Let $E_{\overline{A}}$ and $E_{\overline{B}}$, be the enumerators of $\overline{A}$ and $\overline{B}$ respectively. These enumerators exist because of the definition of A and B, together with the hint.

Define the (deciding) machine M_C as follows: on input x, M_C runs $E_{\overline{A}}$ and $E_{\overline{B}}$ on x step after step (i.e. one step for the first then one for the second and so on), until one of them prints x. If $E_{\overline{A}}$ is the first to print x, then M_C rejects. If $E_{\overline{B}}$ is the first, then M_C accepts. Define the language $C = L(M_C)$.

Since A and B are disjoint, it must hold that $\overline{A} \cup \overline{B} = \Sigma^*$. Therefore, for each string x, either $E_{\overline{A}}$ or $E_{\overline{B}}$ (or both) print it at some stage. Therefore M_C is a deciding machine and $C \in R$. For every $x \in A$, $x \in \overline{B} \setminus \overline{A}$, therefore it will only be printed by $E_{\overline{B}}$ and M_C will accept it. On the other hand, For every $x \in B$, $x \in \overline{A} \setminus \overline{B}$, therefore it will only be printed by $E_{\overline{A}}$ and M_C will reject it.

In other words, $A \subseteq C$, and $B \subseteq \overline{C}$, hence C separates A and B.