Advanced Complexity

Partial Exam

Wednesday, October 24th 2018

All questions below are subjectively ranked from (*) to (***) stars depending on whether they are more or less difficult, hence require more careful and detailed answers.

When giving a reduction, define it rigorously and *argue its correctness.*

The marking will be strict w.r.t. mathematical rigour and completeness of the reasoning. Rigour does not mean length and (*) questions can be answered in at most 3 sentences.

SPARSE languages. We fix an alphabet Σ with at least two letters. Σ^n is the set of words of length n, and $\#E$ denotes the number of elements in the finite set E. For simplicity, we shall assume that $\Sigma = \{0,1\}$, although this is not strictly necessary. A language $L \subseteq \Sigma^*$ is *sparse* if and only if $\#(L \cap \Sigma^n) \leq p(n)$ for some polynomial p (that depends on L). One also say that L "has polynomial density". Let SPARSE be the class of all sparse languages.

(*) 1. For $k \in \mathbb{N}$, we define $L_k = \{u \in \{0,1\}^* \ : \ |u|_1 = k\}$, where $|u|$ is the length of u and $|u|_1$ is the number of times the letter 1 occurs in u. Show that $L_k \in \mathsf{SPARSE} \cap \mathsf{L}$ for any $k \in \mathbb{N}$.

Solution:
L_k is regular hence in L. It is in SPARSE because for any n, $\#L_k \cap \Sigma^n = \binom{n}{k} \leq n^k$.

(*) 2. Show that SPARSE contains some undecidable languages (NB : Only look for very simple examples).

Solution:
Take the language L of strings 1^n such that n is the Gödel number of a halting Turing machine (or of any string from some undecidable language). L is sparse because $\#(L \cap \Sigma^n) \leq 1$.

(*) 3. Given $L, L_1, L_2 \in \mathsf{NL} \cap \mathsf{SPARSE}$, do we have :
(1) $L_1 \cdot L_2 \in \mathsf{NL}$? (2) $L_1 \cdot L_2 \in \mathsf{coNL}$? (3) $L_1 \cdot L_2 \in \mathsf{SPARSE}$?
(4) $L^* \in \mathsf{NL}$? (5) $L^* \in \mathsf{coNL}$? (6) $L^* \in \mathsf{SPARSE}$?
Recall that $L_1 \cdot L_2 = \{uv \ : \ u \in L_1, v \in L_2\}$ and that $L^* = \{\epsilon\} \cup L \cup L \cdot L \cup L \cdot L \cdot L \cup \cdots$

Solution:
3. Yes : if L_1 and L_2 have density bounded by p then $\#L_1 \cdot L_2 \cap \Sigma^n \leq \sum_{i=0}^{n} p(i)p(n-i)$ which is in $O(n.p(n))$. 6 : No, e.g., Σ is sparse but Σ^* is not. 1,2,4 & 5 : Yes.

UNARY languages are languages included in $\{1\}^*$.

(*) 4. Show that **UNARY** $\subset$ SPARSE.

Solution:
At most one instance for each length n.

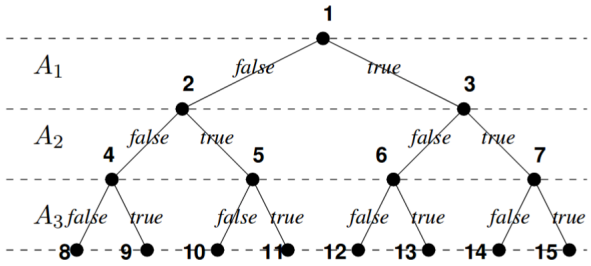
(*) 5. Show that if EXPSPACE = EXPTIME, then PSPACE $\cap$ **UNARY** $\subset$ P and NPSPACE $\cap$ **UNARY** $\subset$ P .

Solution:
There was an error in the subject, we only show the result if $\mathsf{SPACE}(2^{O(n)}) = \mathsf{TIME}(2^{O(n)})$.
Take a language L in $\mathsf{PSPACE} \cap \mathbf{UNARY}$. With L we associate $V = \{n \ : \ 1^n \in L\}$. Note that $V \subseteq \{0,1\}^*$ contains the **binary encodings** of the lengths of the words in L. Now $V \in \mathsf{SPACE}(2^{O(n)})$ since, given some binary n, we can compute 1^n and call the PSPACE algorithm for L on the exponentially long input 1^n. By hypothesis, we thus have $V \in \mathsf{TIME}(2^{O(n)})$. But then we can decide L by the following method : on input x, check that x has the shape $x = 1^n$, compute n (of size $\log |x|$) and use the $\mathsf{TIME}(2^{O(n)})$ algorithm for V on n. This takes a time polynomial in $|x|$. Hence $L \in \mathsf{P}$.
We conclude that $\mathsf{PSPACE} \cap \mathbf{UNARY} \subset \mathsf{P}$, and of course $\mathsf{NPSPACE} \cap \mathbf{UNARY} \subset \mathsf{P}$ as $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

Mahaney's Theorem. We wish to examine in which cases SAT is polynomial-time reducible to a non-empty sparse language. We consider many-one reductions similar to the ones we used in the course *except that they can be polynomial-time computable instead of the weaker logspace computable*. We write $L \leq_P L'$ when L is polynomial-time reducible to L'.

(*) 6. Show that, if $\mathsf{P} = \mathsf{NP}$, then SAT is polynomial-time reducible to some non-empty sparse language (NB : Only look for very simple answers.)

Solution:
In that case, SAT is in P. The reduction first **decides the SAT instance** x in polynomial time. If x is satisfiable, then we return, say, 1, otherwise something else, say 0. This is a polynomial-time reduction showing SAT $\leq_P \{1\}$.



Given a finite list of propositional variables $A_1, \ldots, A_m$, we define a binary tree T_m that organises the 2^m valuations ρ from $\{A_1, \ldots, A_m\}$ to $\{true, false\}$. Instead of defining it formally, let us give an example. We show T_3 above.

The leaves of T_m are in bijection with the (complete branches) from the root, and define valuations. For example, leaf 11 define the valuation $[A_1 \mapsto false; A_2 \mapsto true; A_3 \mapsto true]$.

The nodes of T_m (not just the leaves) are, accordingly, called *partial valuations* σ. For example, node 5 maps A_1 to *false*, A_2 to *true*, and does not map A_3 to anything.

We say that the leaf ρ is *to the left of* the node σ if and only if (1) ρ is a descendant of node σ, or (2) is graphically to the left of σ. For example, the leaves that are to the left of node 5 are 8, 9 (left) and 10, 11 (descendants).

Let LEFT-SAT be the following language.
INPUT : a pair of a set E of propositional clauses, a partial valuation σ.
QUESTION : is there a valuation ρ to the left of σ that satisfies E ?

In this definition we are not 100% explicit on how the input (E, σ) is encoded. We shall assume that E is encoded "as usual", that $|\sigma|$ is $\#dom(\sigma)$, the number of variables that receive some value, and that $|(E, \sigma)|$ is in $|E| + |\sigma| + O(1)$.

(*) 7. Show that LEFT-SAT is NP-complete.

Solution:

LEFT-SAT is in NP : **guess** ρ, then check that it validates E, and that it is to the left of σ. Concretely, we can also find the largest i such that $\sigma(A_1) = \sigma(A_2) = \cdots = \sigma(A_i) = \mathit{false}$, fix $\rho(A_1) = \cdots = \rho(A_i) = \mathit{false}$ and then guess the remaining $\rho(A_{i+1}), \ldots, \rho(A_m)$. This way ρ will be a descendant of σ, or will have $\rho(A_{i+1}) = \mathit{false} \neq \sigma(A_{i+1})$ hence will be to the left of σ. That concrete solution avoids having to explain why "being to the left of" is polynomial-time decidable.

LEFT-SAT is NP-hard. We build a reduction from SAT to LEFT-SAT : on input E (to SAT), we produce (E, ϵ) where ϵ is the empty partial valuation (the root of the tree). The reduction is correct since there is a satisfying valuation for E if and only if there is one **to the left of** ϵ.

(*) 8. Let $f : (E, \sigma) \mapsto f(E, \sigma)$ be some polynomial-time reduction from LEFT-SAT to some sparse language S. Let E be a set of propositional clauses on variables $A_1, \ldots, A_m$. We define the *level* of a partial valuation as its distance from the root of T_m. So node 5 in our example is at level 2, and node 11 is at level 3.

Let σ, σ' be two nodes of T_m at the same level ℓ. Assume that σ is strictly to the left of σ' (graphically), and that $f(E, \sigma) = f(E, \sigma')$. Assume also that E is satisfiable and take the leftmost satisfying valuation τ. Can τ be a descendant of σ' ?

Solution:

Answer : No. Proof : assume, by way of contradiction, that τ is a descendant of σ'. Then (E, σ') is a positive instance of LEFT-SAT. Then $f(E, \sigma') \in S$ since the reduction is correct. Now since $f(E, \sigma) = f(E, \sigma')$, $f(E, \sigma)$ is also in S, so (E, σ) too is a positive instance. So there is a satisfying valuation to the left of σ', contradicting the fact that τ is leftmost.

(**) 9. Let $E, \sigma \mapsto f(E, \sigma)$ be some polynomial-time reduction from LEFT-SAT to some sparse language S. Let p and q be two polynomials such that $\#S \cap \Sigma^n \leq p(n)$ on one hand, and f works in time $\leq q(n)$ for inputs of size n on the other hand. Give a polynomial bound for the number of distinct elements of S that can be obtained as $f(E, \sigma)$ when σ ranges over all the partial valuations for the variables in E. (NB : Give a big O upper bound, not an exact formula.)

Solution:

Let us write n for $|E|$ so that $|(E, \sigma)| \leq 2n+1$. In time $q(2n+1)$, $f(E, \sigma)$ can only produce a word of length at most $q(2n+1)$. The number of elements of S of length $\leq q(2n+1)$ is $\leq p(0) + p(1) + \cdots + p(q(2n+1))$.

We now use the fact that $0^k + 1^k + \cdots + j^k \leq \frac{1}{k+1}(j+1)^{k+1}$ (upper bound the sum by an integral, say), so that the required bound is $O(q(2n+1)p(q(2n+1)))$.

We shall later use the fact that this is polynomial in n.

(***) 10. Let $E, \sigma \mapsto f(E, \sigma)$ be some polynomial-time reduction from LEFT-SAT to some sparse language S. For a given E of size n, let us call $a(n)$ the polynomial found in Question 9. Let $A_1, \ldots, A_m$ be the propositional variables in E.

Imagine that we have a list L_ℓ of partial valuations σ at level $\ell < m$, ordered from left to right, with the guarantee that : (a) all the words $f(E, \sigma)$, for $\sigma \in L_\ell$, are distinct ; (b) L_ℓ contains at most $a(n)$ elements ; and (c) if E has a satisfying valuation, then the leftmost one is a descendant of some $\sigma \in L_\ell$.

Show that you can build, in polynomial time, a new list $L_{\ell+1}$ satisfying (a), (b), and (c) but at level $\ell + 1$. NB : Condition (b) is the hardest to satisfy. Question 8 should lead you to remove certain duplicates, i.e., certain nodes σ' at level $\ell + 1$ that have the same $f(E, \sigma')$ value as some others. If this is not enough, you should remove enough nodes from one side of the list you have obtained.

Solution:

We first form the list L' of partial valuations $\sigma + [A_{\ell+1} \mapsto \mathit{false}]$ and $\sigma + [A_{\ell+1} \mapsto \mathit{true}]$, for $\sigma \in L$, organised from left to right.

By Question 8, for every pair of partial valuations σ' and σ'' in L', with σ' strictly to the left of σ'', if $f(E, \sigma') = f(E, \sigma'')$, then we can **safely remove** σ'' from the list. Removing these duplicates can be done in polynomial time since L' contains at most $2a(n)$ elements, by (b).

After this duplication removal process, we obtain a new list L'', which satisfies (a) and (c). If (b) is satisfied as well, then we have found the desired $L_{\ell+1}$.

Otherwise, not all words $f(E, \sigma')$, $\sigma' \in L''$ can be in S, by Question 9. Since the list L'' is ordered from left to right, all the nodes of L'' whose f value is not in S will be to the left of all those whose f value is in S. This is because of the semantics of LEFT-SAT : any node σ'' strictly to the right of a node σ' at the same level with an f value in S must also have an f value in S.

Therefore it is enough to **let $L_{\ell+1}$ be the list of the $a(n)$ rightmost elements of L''**. $L_{\ell+1}$ will then satisfy (b), and clearly (a). It will also satisfy (c), because the nodes σ such that the leftmost satisfying valuation is a descendant of σ are amongst the last $a(n)$ distinct nodes of L''.

Note that the latter requires one to count up to $a(n)$ in polynomial time. But polynomials are proper, hence computable in polynomial time.

(***) 11. Using the previous questions, show that if LEFT-SAT reduces in polynomial time to some sparse language S, then $\mathsf{P} = \mathsf{NP}$. (NB : Using the assumption "LEFT-SAT $\leq_P S$", you need to explicitly give a polynomial-time algorithm solving some NP-hard problem).

Solution:

Under the given assumption, we solve SAT in polynomial time as follows. On input E, with propositional variables $A_1, \ldots, A_m$, we compute in variable L the successive lists L_0, L_1, $\ldots$, L_m given in Question 10, **level by level**. Initially, L is L_0, which is just the list containing the root node. For ℓ ranging from 0 to $m-1$, knowing that L contains L_ℓ, we compute $L_{\ell+1}$ as in Question 10, and store it in L. At the end of the loop, L contains L_m, and we accept if and only if some $\tau \in L$ satisfies E. This is correct because of property (c), and the fact that the descendants of nodes at level m are the nodes themselves.

(*) 12. Prove *Mahaney's Theorem* : there is a sparse S such that SAT $\leq_P S$ if and only if $\mathsf{P} = \mathsf{NP}$.

Solution:

The "if" direction is Question 6. Conversely, if SAT $\leq_P S$ for some sparse language S, then also LEFT-SAT $\leq_P S$ (this is because LEFT-SAT is in NP, entailing LEFT-SAT $\leq_P$ SAT, and not because it is NP-hard). By Question 11, $\mathsf{P} = \mathsf{NP}$.