Undergraduate Complexity Theory **CMU 15-455, Spring 2017**

HOMEWORK 3
Due: 5:00pm, Thursday February 9

---

1. **(Not even EXP.)** [10 points.] Prove there exists a decidable language L that is not in EXP.

2. **(Coloring better than brute force.)** [10 points.] Recall the 3-coloring problem: Given as input an undirected graph $G = (V, E)$, determine whether or not it has a "valid 3-coloring". A valid 3-coloring is an assignment of "Red", "Blue", or "Yellow" to each vertex such that no edge has both its endpoints the same color. Prove that the 3-coloring problem can be solved in time $2^n \cdot \text{poly}(n)$, where $n = |V|$.

   You should write your answer in pseudocode, and justify the correctness and time analysis of your algorithm.

3. **(CSPs.)** [10 points.] In this problem we define *constraint satisfaction problems* (CSPs). Informally, an instance of CSP consists of some n *variables* $x_1, \ldots, x_n$, together with a list of m *constraints* on them. The variables are to be assigned values from some finite *domain* D, and each constraint specifies a small subset of the variables and a restriction on how they are allowed to be assigned. The task is to determine if there is an assignment to the variables that satisfies all the constraints.

   For example, in the 3SAT problem (which I hope you remember from 15-251), the variables are supposed to take values from the domain $D = \{\mathsf{T}, \mathsf{F}\}$, and an input 3CNF formula can be thought of as a list of m constraints, each of which is the logical OR of up to three literals (variables/negated-variables). For example, a 3SAT instance might looks like

   $$(x_1 \vee x_2 \vee \overline{x}_4),\ (x_3 \vee \overline{x}_5 \vee x_6),\ (x_4 \vee \overline{x}_6 \vee \overline{x}_7),\ \ldots$$

   (It's common to see the OR-constraints separated by the AND symbol $\wedge$, but I've just separated them by commas here.) The first constraint above can be thought of as the predicate $(\bullet \vee \bullet \vee \overline{\bullet})$ applied to the tuple of variables (x_1, x_2, x_4). In other words, it's saying that that tuple of variables should be assigned one of the following 7 triples from D^3:

   $$(\mathsf{F},\mathsf{F},\mathsf{F}), (\mathsf{F},\mathsf{T},\mathsf{F}), (\mathsf{F},\mathsf{T},\mathsf{T}), (\mathsf{T},\mathsf{F},\mathsf{F}), (\mathsf{T},\mathsf{F},\mathsf{T}), (\mathsf{T},\mathsf{T},\mathsf{F}), (\mathsf{T},\mathsf{T},\mathsf{T}).$$

   Usually in a CSP not all possible kinds of constraints are allowed. For example, in 3SAT there are exactly 14 different kinds of constraints allowed, the ones that look like this:

   $$x_i,\ \ \overline{x}_i,\ \ x_i \vee x_j,\ \ x_i \vee \overline{x}_j,\ \ \overline{x}_i \vee x_j,\ \ \overline{x}_i \vee \overline{x}_j,\ \ x_i \vee x_j \vee x_k,\ \ x_i \vee x_j \vee \overline{x}_k,\ \ \ldots,\ \ \overline{x}_i \vee \overline{x}_j \vee \overline{x}_k.$$

   Another example of a CSP is 3-coloring, as in Problem 2. This is the CSP in which the domain is $D = \{R, B, Y\}$, and there is exactly one kind of constraint allowed: it can be applied to any pair of variables and constrains their values to be one of the following 6 triples from D^2:

   $$(R,B),\ \ (R,Y),\ \ (B,R),\ \ (B,Y),\ \ (Y,R),\ \ (Y,B).$$

   Alternatively, you can describe it as the "$\neq$" predicate. This is not quite how we described 3-coloring in Problem 2, but you should convince yourself it's really the same problem!

On to the present problem. Consider the CSP in which the domain is $D = \{0, 1\}$ and there are three kinds of constraints allowed. The first two are unary constraints: you can specify either $x_i = 0$ or $x_i = 1$. The third constraint is binary: you can specify $x_i \leq x_j$. In other words, a typical input to this CSP might look like this:

$$x_1 = 1, \quad x_3 \leq x_2, \quad x_5 = 0, \quad x_{25} \leq x_{83}, \quad x_3 \leq x_{15}, \quad x_{55} = 1, \quad \ldots \quad x_{42} \leq x_n.$$

Prove that the associated decision problem — given an instance, is there an assignment to the variables that satisfies all the constraints — can be solved in polynomial time.

4. **(Poly-time can be syntactically checked.)** [10 points.] Suppose you have a Turing Machine M which you somehow know for a fact is a decider TM with polynomial running time. You give it to your friend Alice and say, "Hey Alice, feel free to use this polynomial-time Turing Machine M." Now you've had a shaky relationship with Alice in the past, so she is always skeptical of your claims. And in fact, the problem of "Given a Turing Machine $\langle M \rangle$, decide if it has polynomial running time" happens to be undecidable, so in general Alice can't do anything on her own to become convinced of your claim.

   However, all is not lost. Describe a procedure you can do to transform your Turing Machine M into a new one M' (which may be a multi-tape TM, if you want). The transformation should have the following properties:

   (a) M' should have the same accept/reject behavior as M for all strings. (In showing this, you may assume that M *is indeed* a decider TM with polynomial running time. Further, you may assume you know a number k such that M's running time is at most kn^k for all n.)

   (b) The running time of M' should also be polynomial. (In fact, it should be at most polynomially larger than the running time of M itself.)

   (c) The transformation from $\langle M \rangle$ to $\langle M' \rangle$ should be doable in polynomial time.

   (d) When Alice gets $\langle M' \rangle$, she should be able to run a polynomial-time algorithm on $\langle M' \rangle$ that convinces her that M' indeed is a decider TM with polynomial running time.

   For this problem, you should describe your transformation in English prose, although that prose will of course be talking about the details of Turing Machines. You should be able to explain in at most a few sentences why properties (a), (b) above hold. You *don't* really need to go into many details to justify property (c), as long as it's pretty clear from the prose description of your transformation that it's doing relatively straightforward string manipulations. Finally, you *should* spend a little bit of time justifying property (d); specifically, why Alice also only needs to do some relatively straightforward string-parsing to become convinced.

   (Hint: a previous homework problem.)