CSE491/596, Fall 2020 Problem Set 5 Due Mon. Nov. 9, 11:59pm
Plus topics for presentations on Nov. 5–6

Reading: We are now in section 4.4 of ALR chapter 28, and will be there Monday and Wednesday; the problems there rather than those in Debray's section 16 will be lectured on. Starting Friday 11/6, however, we will give full treatment to space complexity and its relation to time complexity before handling the completeness results in section 4.5 of that chapter. There is a "hole" in both ALR chapter 27 and in Debray's notes, where ALR states Theorem 2.3 (in the short "Basic Relationships" section 2.4) without giving proofs, while Debray sketches the proof tersely at the top of page 56 (near the end of section 17) but does not make a separate theorem statement. The hole is filled by page 1 of this handout linked from the course webpage as item 6 of required reading:

`https://cse.buffalo.edu/~regan/cse491596/CSE596inclusions.pdf`

The remaining pages 2–4 are heavy going but they tie together material that is heavy going in Debray's notes (split between Theorems 13.7–13.8 and Theorem 18.1) and ALR chapter 27, Theorem 2.5 in section 2.6 (where, however, the proof of the hierarchy theorem for deterministic time is not given, and you should skim/skip the NTIME proof). Thus the reading for the second week of November becomes:

- The rest of Debray, sections 13–18, except we are still postponing the discussion of oracles in the first two pages of section 17, and the TQBF proof will be covered the following week when we rejoin ALR chapter 28 in section 5.
- ALR chapter 27, sections 2.4–2.7.

The sequence can be followed from the lecture notes from 2018 and 2019 beginning in week 10 of both (we are one lecture ahead, plus we've already covered the subtopic of the easy reductions involving CLIQUE, IND. SET, and VERTEX COVER).

Assignment 5, written part due Mon. Nov. 9. 11:59pm

(1) For each of the following decision problems, say whether its language L is known to belong to NP or to co-NP. In each case, say what the "witness" is, either for membership in L or in $\tilde{L}$. Give a polynomial bound on the size of the witness in terms of the input length, and briefly explain why a given witness can be checked in polynomial time. (The I/O alphabet Σ always includes at least two characters. In many cases, "polynomial" can be "linear." The problems are named after films from the year 2000.)

(a) "Unbreakable"

Instance: An undirected graph G with an even number n of vertices, and a number k such that $0 < k < (n^2 - n)/2$.

Question: Is it possible to break G into two disjoint pieces of $n/2$ nodes each by removing at most k edges?

(b) "Pay It Forward"

INSTANCE: A nondeterministic Turing machine N with three-way branching at each step, an input $x \in \Sigma^*$ to N, and a number $d > |x|$ given in unary notation as the string 0^d.

QUESTION: Do all possible computation paths by $N(x)$ lead to halting and accepting within d steps?

(c) "TRAFFIC"

INSTANCE: An airport with k runways and n flights that have to land within an h-hour period. The input specifies for each flight a 30-minute time period when it can land, and which runways it can and cannot land on. No two flights may land on the same runway within fewer than 5 minutes of each other.

QUESTION: Is there an air-traffic control schedule that enables all the planes to land?

(d) "HIDDEN DRAGON"

INSTANCE: A deterministic Turing machine M whose code includes a "clock" that shuts off computations on inputs of length n after n^2 steps, and a string $y \in \Sigma^*$.

QUESTION: Is there a string x of the same length as y such that $M(x)$ outputs y and halts?

(e) "GLADIATOR"

INSTANCE: A 'bot program P written in C that plays "Player 1" of two-player "Quake" against other 'bots that play "Player 2." Let n be the size in bytes of the source code for P.

QUESTION: Does P defeat all Player 2 'bots of the same source-code size, in "Quake" games that last at most n milliseconds and use no randomness?

You *can* answer the last question without knowing any details of the game "Quake" or "game-bot" programs. Depending on those details and whether 'bots can be optimized, the language of (e) *may* even be empty, but the point is that the above knowledge does allow you to classify it as belonging to one of NP or co-NP. The "clock" in (d) could be implemented on an independent set of tapes, after a routine that copies the input x to one of those tapes before starting "M itself." Regarding (a), note that the problem of whether a (sub-)graph is connected belongs to P, as was covered on Assignment 4 in 2019. ($5 \times 6 = 30$ pts)

(2) Let $C = (\ell_1 \vee \ell_2 \vee \ell_3)$ be a clause with three literals, where ℓ_1 is x or $\bar{x}$, ℓ_2 is y or $\bar{y}$, and ℓ_3 is z or $\bar{z}$. (It is often convenient to use this notation so that you can write, e.g., $\overline{\ell_1}$ without having to specify whether ℓ_1 is x or $\bar{x}$.) Show how to map C to a sub-formula ψ_C, where ψ_C is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ with extra variables $w, v, u, \dots$, such that:

> an assignment a to x, y, z satisfies C if and only if a can be extended to an assignment a' of all the variables that not only satisfies ψ_C, but also makes **exactly one** literal in each of its k clauses true.

Note that if a is the one assignment (out of eight) that fails to satisfy C, then you may still be able to satisfy ψ_C, but the point is that you won't be able to make *exactly* one literal in each of its clauses true. Use this fact to reduce 3SAT to each of the following two problems:

EXACTLY ONE 3SAT

Instance: A Boolean formula $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ in 3CNF.

Question: Is there an assignment to the variables that makes *exactly one literal* in each clause true?

EXACTLY TWO 3SAT

Instance: A Boolean formula $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ in 3CNF.

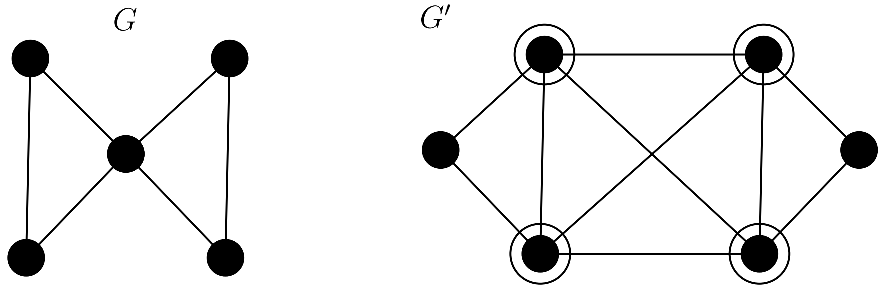
Question: Is there an assignment to the variables that makes *exactly two literals* in each clause true?

(30 pts. total)

(3) Given an undirected graph $G = (V, E)$, define its *line graph* $G' = (V', E')$ by $V' = E$ and

$$E' = \{((u, v), (v, w)) : (u, v) \in E \wedge (v, w) \in E\},$$

recalling that in an undirected graph, $(u, v) \in E \iff (v, u) \in E$. That is, G' has a vertex for every edge of G, and two edges that "touch" in G become an edge in G'. For example, the graph G' at right is the line graph of the 5-node "bowtie graph" at left, and the circles show how the four edges touching the center vertex of the bowtie become a clique in G':



Now consider the following decision problem:

EDGE COVER

Instance: An undirected graph G and an integer $k \geq 1$.

Question: Does there exist a set H of at most k edges such that every other edge in G touches an edge in H?

(a) Does $f(G, k) = (G', k)$ give a polynomial-time mapping reduction (i) from VERTEX COVER to EDGE COVER?, (ii) from EDGE COVER to VERTEX COVER? (iii) both? (iv) neither? Justify your answer. (12 pts.)

(b) Show that EDGE COVER is NP-complete *by mapping reduction from* 3SAT. You are allowed to use extra properties of the reduction in the proof of the Cook-Levin theorem or the reduction in problem (2), but you may not need to do so. (30 pts., making 42 on the problem and 102 on the set)

Presentation Options For 11/5–6, numbered in Roman this time

Again, the first two can be teamed, as can (IV) and (V)

(I) Define a *two-head deterministic finite automaton* (2HDFA) $M = (Q, \Sigma, \delta, s, F)$ to be like a DFA except that its instructions have the form $(p, (c_1, c_2)/(D_1, D_2), q)$ where D_1 and D_2 can be R for move right or S for stay. Without loss of generality we may suppose that each instruction moves at least one of the two heads right.

(a) Sketch how a 2HDFA can recognize a few of the languages we've proved to be nonregular. (Giving the full machine is optional; it's not much more than a sketch anyway.)

(b) Can a 2HDFA recognize the language of palindromes? What do you think? (Proof not needed.)

(c) Explain how, for any fixed one-tape DTM M, a 2HDFA H can decide the Kleene T-predicate for M under a natural encoding scheme where $\vec{c}$ is a sequence of IDs. That is, show how H, given $x, I_0, I_1, \dots, I_t$ as a string, can decide whether this is a valid accepting computation of M on input x. (To get you started, note that I_0 is $sx_1x_2\cdots x_n$, or if you prefer using members of $Q \times \Gamma$ as single chars, $I_0 = \binom{s}{x_1}x_2\cdots x_n$. You may suppose the comma is not a char in the work alphabet Γ of M.)

(II) Deduce from (I) that the emptiness problem for 2HDFAs, whose language can be called E_{2HDFA}, is undecidable—indeed, mapping equivalent to E_{TM}. Conclude along lines of the presentation option (A) on HW4 (whose key is being given) that for any program verification logic F, there exist 2HDFAs H such that $L(H) = \emptyset$ but F cannot prove it. (IMHO, 2HDFAs are about the simplest kind of code whose behavior cannot be generally verified.)

(III) In his original 1936 paper, Alan Turing thought in terms of "computable numbers" z_L rather than decidable languages L. To define what he meant by example, the language $P = \{2, 3, 5, 7, 11, 13 \dots\}$ of prime numbers becomes the binary irrational number

$$z_P = 0.0110101000101\dots$$

That is, we identify binary strings and positive integers under the correspondence $1 \leftrightarrow \epsilon$, $2 \leftrightarrow 0$, $3 \leftrightarrow 1$, $4 \leftrightarrow 00$, and so on. Then the i-th bit of z_L is a 1 if i is in L, and is 0 otherwise. (If we want to allow zero as a number we can use the bit before the "binary decimal point" for it.)

Prove that a language L is decidable if and only if there is a total TM T that on any input n outputs the first n bits of z_L, which is what Turing meant by an irrational number being "computable." What happens for rational numbers whose denominator is a power of 2?

(IV) Show that the following decision problem is not only undecidable, its language is not c.e.:

Instance: A deterministic Turing machine M.
Question: Does M run in polynomial time?

Deduce, again along lines of the presentation option (A) on HW4, that for any reasonable system of logic F, there are algorithms that run in polynomial time, but F cannot prove that they do so.

(V) Gödel's Second Incompleteness Theorem in full says that an adequate and effective system of logic F cannot prove the statement $(\forall\pi)\neg R_F(0 = 1, \pi)$, which expresses that F is consistent, unless F actually is inconsistent (and hence proves all statements it can formulate). Use this idea to create a Turing machine M_F that runs in linear time if F is consistent, but not otherwise. You can do this using the "delay switch" framework, but with "Search For Pie" in place of the "Simulate M on its own code" as the action to do for n steps, and with a time-wasting action in the "else:panic" branch.

(VI) Show that a language L is c.e. if and only if L is the range of a computable function f. Then show that L is in NP if and only if L is the range of a polynomial-time computable function g, where also there is a polynomial q such that for all x and y, if $g(x) = y$ then $q(|y|) \geq |x|$. This latter condition, called "polynomial honesty," prevents g from making strings exponentially shorter. For a topical example, consider the following function that takes a 3CNF formula ϕ and an assignment to it:

$$g(\langle\phi, a\rangle) = \begin{cases} \phi & \text{if } a \text{ satisfies } \phi \\ \phi_n & \text{otherwise,} \end{cases}$$

where ϕ_n is a fixed 3CNF formula of the same size as ϕ that is trivially satisfiable. Because a cannot be longer than the encoding of ϕ, g gives linear honesty, and its range is 3SAT.