CSE396, Spring 2021 Problem Set 9 Due Tue. 4/27, 11:59pm

Reading:

For next week, read Chapter 5, sections 5.1 and 5.3, skim/skipping section 5.2. The last week will go into Chapter 7, but for the Cook-Levin Theorem of the NP-completeness of SAT, will use the "alternative proof" in Chapter 9. Chapter 6 is skipped. The "computation histories" part of 5.1 will be covered after section 5.3 and may even be held over into the Tuesday of the last week.

I have already jumped over the subject of countability and uncountability in chapter 4, but I recommend *skimming* it now as a way to reinforce your understanding of *functions* and for reviewing "diagonalization" the way it was covered in the Thu. 4/22 lecture using functions. The point of Cantor's Theorem is that this logic works the same way even when A is infinite—and when $A = \mathbb{N}$ it leads to the inference that the power set of $\mathbb{N}$ (which under the correspondence between numbers and strings we can think of as the class of *all* languages, decidable and undecidable) is uncountable. But uncountability isn't driving the bus—diagonalization is—so I soft-pedal it.

The thing to realize about Chapter 5 is that it is titled "Reducibility" but Sipser defers the key concept until section 5.3. I put it up-front, doing the definitions in section 5.3 before tackling the examples in section 5.1. The chain of reductions will continue by reducing A_{TM} to NE_{TM} (defined as $\{\langle M \rangle : L(M) \neq \emptyset\}$, which is essentially the complement of E_{TM} as defined in the text), and then reducing A_{TM} to the historical Halting Problem.

The *TopHat* portion is lengthier but important for final-exam (p)review. It is still coded as questions with one right answer and a single attempt. However, here are some notes and reminders about facts related to Chapters 2 and 4 that would be given if it had been coded as having 2 attempts:

- The complement of a CFL need not be a CFL, *so* there is no general way to transform a CFG G into a CFG G' such that $L(G') = \sim L(G)$.
- The problem E_{CFG} of whether $L(G) = \emptyset$ is decidable, but the problem ALL_{CFG} of whether $L(G) = \{0,1\}^*$ is undecidable.
- A PDA P can be converted into an equivalent CFG and vice-versa. This plus the decidability of the A_{CFG} problem via Chomsky normal form makes the A_{PDA} problem decidable too. (In fact, though the text doesn't say this, the PDA-to-CFG process is also computationally efficient, so that when we hit Chapter 7 in the last week we will "officially" say that A_{CFG} and A_{PDA} belong to polynomial time, which is stronger than saying all individual CFLs belong to polynomial time.)
- A *deterministic* PDA M can be converted into a DPDA M' such that $L(M') = \sim L(M)$. The proof must first make M total before interchanging q_{acc} and q_{rej}.
- If the answer to "Is $L(M) = \emptyset$" is *no*, this doesn't mean $L(M) = \Sigma^*$. (Although, we will bring this about for certain very special Turing machines M that we construct when doing reductions.)

- We cannot do Cartesian product of two PDAs. Intuitively this is because they might "fight" over control of the single stack a PDA is allowed. But we can do Cartesian product of a PDA and a DFA, so long as the Boolean operation involved is not something like symmetric difference which would involve complementing the PDA (which can't be done in general unless the PDA is deterministic). In particular, this is AOK for $\cap$ as well as $\cup$, so—as was mentioned midway through the Week 10, Tue. 4/6 lecture regarding closure properties after the CFL Pumping Lemma—the intersection of a CFL and a regular language is still a CFL.

Homework—part online (TopHat), part written, and all *individual work*:

(1) Using *TopHat*, the "Worksheet" titled *S21 HW9 Online Part* (10 Qs, 20 pts.)

The other two problems are to be submitted as PDFs using the *CSE Autograder* system.

(2) Consider the following decision problem: Given a CFG $G = (V, \Sigma, R, S)$ with $\Sigma = \{a, b\}$, is $L(G) \cap a^+ \neq \emptyset$? That is, does S generate one or more strings that consist only of one or more a's? Sketch a decision procedure in prose similar to what the text does in section 4.1 with E_{CFG} and what my lectures did when comparing the latter to the algorithm for whether $\epsilon \in L(G)$.

(You may if you wish assume the conversion from G to a G' without ϵ-rules as the first step of your procedure with no other comment needed on how to do it, but further steps should show all details including sketching any while-loops that may be needed. 24 pts.)

(3) Consider the following decision problem—and its two variants in parts (b) and (c) where the machine M is allowed to be a deterministic pushdown automaton or a deterministic Turing machine, respectively.

INSTANCE: A DFA $M = (Q, \Sigma, \delta, s, F)$ and two strings $u, v \in \Sigma^*$.

QUESTION: Does $L(M)$ contain all strings that can be formed by concatenating u and v as often as desired in any order?

(a) Give in pseudocode a decision procedure for this problem. (First ask yourself, what kind of language is $(u \cup v)^*$? 18 pts.)

(b) With reference to the above bulleted notes, tweak your procedure so that it solves the somewhat more general problem where M can be given as a DPDA not just a DFA. (6 pts.)

(c) Show that the problem becomes *undecidable* when M is allowed to be any DTM. You need not use a mapping reduction *per se*, but you should consider modified machines M' that run a given machine M on its own code, and only if and when M accepts its own code, does M' then think about accepting any strings. (12 pts., for 36 on the problem and 80 total on the set)