CS 341-008, Spring 2021
Solutions for Midterm 2, Hybrid

1. (a) True. If $A \subseteq B$, then $x \in A$ implies that $x \in B$, so $|A| \leq |B|$. Thus, if B is countable, we must have that A is also countable.

   (b) False. Theorem 4.11 shows that A_{TM} is undecidable, so no TM can decide A_{TM}. The universal TM recognizes A_{TM} but doesn't decide it.

   (c) False. The set $\mathcal{N} = \{1, 2, 3, \ldots\}$ is infinite and countable.

   (d) True, as is shown in the proof of Theorem 3.16.

   (e) False. Every regular language is context-free by Corollary 2.32. Every context-free language is decidable by Theorem 4.9, and every decidable language is Turing-recognizable because the definition of Turing-recognizable is less restrictive than the definition of decidable (also see slide 4.55). Thus, every regular language is Turing-recognizable.

   (f) False. For example, we always have that $\emptyset \subseteq A$ for any set A, countable or uncountable, and $|\emptyset| = 0$, which is finite so countable.

   (g) False. Suppose $A = \{a\}$ and $B = \{a, aa\}$ are languages defined over alphabet $\Sigma = \{a\}$. Then $\overline{A} \cap B = \{aa\}$ and $A \cap \overline{B} = \emptyset$, so the statement "$\overline{A} \cap B = \emptyset$ or $A \cap \overline{B} = \emptyset$" is true because at least one is empty, but $A \neq B$.

   (h) False, by slide 4-38.

   (i) False. Just because a language A is recognized by a TM T that loops on some $w \notin A$, that doesn't necessarily mean there isn't another TM M that also recognizes A but never loops so M decides A. For example, we could modify the TM M on slide 4-7 for A_{DFA} to create another TM T that is the same as M except we change stage 2 to instead do the following: "If B ends in state $q \in F$, then M *accepts*; otherwise, loop." Then T recognizes A_{DFA} but does not decide A_{DFA} because T loops on $\langle B, w \rangle \notin A_{\text{DFA}}$. But A_{DFA} is decided by TM M.

   (j) False. TM M can loop on w.

2. (a) No, because $f(1) = f(3) = b$.

   (b) Yes, because everything in R is hit by f.

   (c) No, because f is not one-to-one.

   (d) A language L_1 that is Turing-recognizable is recognized by a Turing machine M_1 that may loop forever on a string $w \notin L_1$. A language L_2 that is Turing-decidable is recognized by a Turing machine M_2 that always halts.

   (e) An algorithm is a Turing machine that always halts.

3. $q_1 baab\#aaba \quad xq_3aab\#aaba \quad xaq_3ab\#aaba \quad xaaq_3b\#aaba \quad xaabq_3\#aaba \quad xaab\#q_5aaba$
   $xaab\#aq_{\text{reject}}aba$

4. This is HW 9, problem 1. Let $\mathcal{B}$ be the set of infinite binary sequences. Each element in $\mathcal{B}$ is an infinite sequence $(b_1, b_2, b_3, \ldots)$, where each $b_i \in \{0, 1\}$. Suppose $\mathcal{B}$ is countable. Then we can define a correspondence f between $\mathcal{N} = \{1, 2, 3, \ldots\}$ and $\mathcal{B}$. Specifically, for $n \in \mathcal{N}$, let $f(n) = (b_{n1}, b_{n2}, b_{n3}, \ldots)$, where b_{ni} is the ith bit in the nth sequence, i.e.,

n	$f(n)$
1	$(b_{11}, b_{12}, b_{13}, b_{14}, b_{15}, \ldots)$
2	$(b_{21}, b_{22}, b_{23}, b_{24}, b_{25}, \ldots)$
3	$(b_{31}, b_{32}, b_{33}, b_{34}, b_{35}, \ldots)$
4	$(b_{41}, b_{42}, b_{43}, b_{44}, b_{45}, \ldots)$
$\vdots$	$\vdots$

Now define the infinite sequence $c = (c_1, c_2, c_3, c_4, c_5, \ldots) \in \mathcal{B}$ over $\{0, 1\}$, where $c_i = 1 - b_{ii}$. In other words, the ith bit in c is the opposite of the ith bit in the ith sequence. For example, if

n	$f(n)$
1	$(0, 1, 1, 0, 0, \ldots)$
2	$(1, 0, 1, 0, 1, \ldots)$
3	$(1, 1, 1, 1, 1, \ldots)$
4	$(1, 0, 0, 1, 0, \ldots)$
$\vdots$	$\vdots$

then we would define $c = (1, 1, 0, 0, \ldots)$. Thus, $c \in \mathcal{B}$ differs from each sequence by at least one bit, so c does not equal $f(n)$ for any n, which is a contradiction. Hence, $\mathcal{B}$ is uncountable.

5. (This is HW 8, problem 2.) The language of the decision problem is

$$A\varepsilon_{\text{CFG}} = \{\, \langle G \rangle \mid G \text{ is a CFG that generates } \varepsilon \,\}.$$

If a CFG $G = (V, \Sigma, R, S)$ includes the rule $S \to \varepsilon$, then clearly G can generate ε. But G could still generate ε even if it doesn't include the rule $S \to \varepsilon$. For example, if G has rules $S \to XY$, $X \to aY \mid \varepsilon$, and $Y \to baX \mid \varepsilon$, then the derivation $S \Rightarrow XY \Rightarrow \varepsilon Y \Rightarrow \varepsilon\varepsilon = \varepsilon$ shows that $\varepsilon \in L(G)$, even though G doesn't include the rule $S \to \varepsilon$. So it's not sufficient to simply check if G includes the rule $S \to \varepsilon$ to determine if $\varepsilon \in L(G)$.

But if we have a CFG $G' = (V', \Sigma, R', S')$ that is in Chomsky normal form, then G' generates ε if and only if $S' \to \varepsilon$ is a rule in G'. Thus, we first convert the CFG G into an equivalent CFG $G' = (V', \Sigma, R', S')$ in Chomsky normal form. If $S' \to \varepsilon$ is a rule in G', then clearly G' generates ε, so G also generates ε since $L(G) = L(G')$. Since G' is in Chomsky normal form, the only possible ε-rule in G' is $S' \to \varepsilon$, so the only way we can have $\varepsilon \in L(G')$ is if G' includes the rule $S' \to \varepsilon$ in R. Hence, if G' does not include the rule $S' \to \varepsilon$, then $\varepsilon \notin L(G')$. Thus, a Turing machine that decides $A\varepsilon_{\text{CFG}}$ is as follows:

M = "On input $\langle G \rangle$, where G is a CFG:
  1. Convert G into an equivalent CFG $G' = (V', \Sigma, R', S')$ in Chomsky normal form.
  2. If G' includes the rule $S' \to \varepsilon$, *accept*. Otherwise, *reject*."

An alternative correct solution is as follows. Let T be a TM that decides $A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \}$. Then the following TM M' decides $A\varepsilon_{\text{CFG}}$:

M' = "On input $\langle G \rangle$, where G is a CFG:
  1. Run T on input $\langle G, \varepsilon \rangle$, where TM T decides A_{CFG}.
  2. If T accepts, then *accept*. Otherwise, *reject*."

6. (This is HW 8, problem 4.) We need to show there is a Turing machine that recognizes $\overline{E_{\text{TM}}}$, the complement of E_{TM}. Let $s_1, s_2, s_3, \ldots$ be a list of all strings in Σ^*. For a given Turing machine M, we want to determine if any of the strings $s_1, s_2, s_3, \ldots$ is accepted by M. If M accepts at least one string s_i, then $L(M) \neq \emptyset$, so $\langle M \rangle \in \overline{E_{\text{TM}}}$; if M accepts none of the strings, then $L(M) = \emptyset$, so $\langle M \rangle \notin \overline{E_{\text{TM}}}$. However, we cannot just run M sequentially on the strings $s_1, s_2, s_3, \ldots$. For example, suppose M accepts s_2 but loops on s_1. Since M accepts s_2, we have that $\langle M \rangle \in \overline{E_{\text{TM}}}$. But if we run M sequentially on $s_1, s_2, s_3, \ldots$, we never get past the first string. The following Turing machine avoids this problem and recognizes $\overline{E_{\text{TM}}}$:

R = "On input $\langle M \rangle$, where M is a Turing machine:
  1. Repeat the following for $i = 1, 2, 3, \ldots$.
  2. Run M for i steps on each input $s_1, s_2, \ldots, s_i$.
  3. If any computation accepts, *accept*.