1. Let $C \subseteq \Sigma_C^*$ be a language. Prove that $C$ is Turing-recognizable if and only if a Turing-decidable language $D \subseteq \Sigma_D^*$ exists such that

$$C = \{x \mid \text{there exists } y \text{ such that } x\#y \in D\ \}.$$

where $\#$ is a delimiter symbol such that $\# \notin \Sigma_C$ but $\# \in \Sigma_D$.

You should think of $y$ as acting as a *witness* to $x$'s membership in $C$.

Hint: The next problem is provided as a hint for this problem and vice versa.

To prove the forward direction, suppose we have a turing machine $M_D$ that recognizes the language $D$ such that

$$C = \{x \mid \text{there exists } y \text{ such that } x\#y \in D\ \}.$$

We can build a turing machine $M_C$ that recognizes $C$ in the following way. For an input $x$ on $M_C$, we make the turing machine generate all strings $y \in \Sigma^*$. Now, for every $y$ generated, we run $x\#y$ on $M_D$. If $M_D$ accepts a $x\#y$, then $M_C$ accepts $x$. On the other hand, if $M_D$ does not accept a given $x\#y$, then we try another $y$ until one is accepted. We know $M_D$ must halt because $D$ is Turing-decidable, and if no possible $x\#y$ is accepted by $M_D$ then $M_C$ does not halt on input $x$. Thus, $M_C$ would not recognize that $x$.

To prove the backwards direction, suppose we have a turing machine $T_C$ that recognizes the language $C$, that is $L(T_C) = C$. We can build a turing machine that decides a language $D$ such that

$$D = \{x\#b(n) \mid x \in C \text{ and } n \geq 0\}$$

$b(n)$ is a string representation of how many steps it takes $T_C$ to recognize $x$. To build such turing machine $T_D$, we take the input $x\#b(n)$ and run $x$ on $T_C$. If $T_C$ halts within $n$ steps and accepts $x$, then $T_D$ accepts $x\#b(n)$. However, if $T_C$ would take more than $n$ steps on input $x$, then we know the machine could possibly not recognize $x$ and not halt. Since $T_D$ only does up to $n$ steps, it decides $D$.

2. An unrestricted grammar is a quadruple $G = (V, \Sigma, R, S)$ where

   - $V$ is a finite set of non-terminal symbols;
   - $\Sigma$ is a finite set of terminal symbols disjoint from $V$;
   - $S \in V$ is the unique start symbol.
   - $R \subset (V \cup \Sigma)^*(V)(V \cup \Sigma)^* \times (V \cup \Sigma)^*$ is a set of rules .

   The only difference between an unrestricted grammar and a context-free grammar is in the rules. Rules in context-free grammars have a single non-terminal on the left hand side, whereas rules in an unrestricted grammar may have any string of terminals and non-terminals on the left side, but must include at least one non-terminal. As with context-free grammars, although rules are formally tuples, they are conventionally written with an arrow separating the right and left sides.

   Derivations are similar to context-free grammars except the definition of "yields" is slightly different: we may substitute the right hand side of any rule into a derivation if the derivation has a substring matching the left hand side of the rule. More formally:

   **Yields** Given an unrestricted grammar, G = $(V, \Sigma, S, R)$, and two strings $x$ and $y$ in $(V \cup \Sigma)^*$ such that $x = \alpha\sigma\beta$ and $y = \alpha\gamma\beta$ where $\alpha, \sigma, \gamma, \beta \in (V \cup \Sigma)^*$ and $(\sigma, \gamma) \in R$ we say that $x$ *yields (or directly derives)* $y$. In this case we write

   $$x \Longrightarrow y$$

   For example, here's a grammar that generates the language $\{a^n b^n c^n \mid n \geq 1\}$:

   | | | | | |
   |---|---|---|---|---|
   | S | $\rightarrow$ | ABCS | \| | $T_c$ |
   | CA | $\rightarrow$ | AC | | |
   | BA | $\rightarrow$ | AB | | |
   | CB | $\rightarrow$ | BC | | |
   | C$T_c$ | $\rightarrow$ | $T_c$c | \| | $T_b$c |
   | B$T_b$ | $\rightarrow$ | $T_b$b | \| | $T_a$b |
   | A$T_a$ | $\rightarrow$ | $T_a$a | | |
   | $T_a$ | $\rightarrow$ | $\varepsilon$ | | |

   Let $L \subseteq \Sigma^*$ be a language generated by an unrestricted grammar $G$. Show that $L$ is Turing recognizable using the result of problem 1 (even if you were unable to complete problem 1).

   Given $C = L = L(G)$ consider the language:

   $$D = \{w\#w_0\#w_1\#...\#w_n \mid w \in L, w = S, \forall\, i < n\ w_i \Longrightarrow w_{i+1}\}$$

$D$ represents the language of strings $w$ in $L$ followed by derivations that lead to the corresponding strings $w$ using the rules of $G$. Each derivation is separated by a #. $D$ is of the form

$$C = \{x \mid \text{there exists } y \text{ such that } x\#y \in D \}.$$

as every $x \in C$ is the first substring of $D$. In order to show $C$ is Turing-recognizable, we must show $D$ is Turing -decidable (this is using our result from problem 1). We can build a turing machine $T_D$ that decides $D$ by first validating our string. That is, $w$ and $w_n$ must equal to each other and contain only terminals, every $w_i$ must contain symbols in either $\Sigma$ or $V$, and $w_0$ must be equal to the start state of the grammar. $T_D$ will then go through the start state of $G$ and verify that it can get to $w_1$, so on and so forth. If there is ever the case that $w_i \not\Longrightarrow w_{i+1}$, then $T_D$ rejects the input. Once $w_n$ is reached, then $T_D$ accepts the input. This machine will always halt as a string of terminals produced by $G$ will always be derived in a finite number of steps, thus $w$ and the derivations $w_0$ to $w_n$ are finite. The cases where $T_C$ would not halt are when the input is not produced by the grammar and $T_D$ will try an infinite number of configurations as input.

3. There are five remaining problem statements for this assignment. Each of the remaining problems describes a question informally. Three of these questions are decidable and two are not. For each of the questions that is decidable, you should formulate the question as a language and then argue that there is a Turing machine that decides this language. You don't have to turn in anything for the other two questions. To ensure that it is clear which questions you are answering, make sure that the number of each problem you decide to answer is clearly visible on the page(s) containing your answer.

   For example, if one of the questions below was "Does the language of a given finite automaton contain a specific string $w$?" and you believed that this language was decidable (it is!), you would formulate this as the language

   $$A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a DFA that accepts } w\}$$

   and then explain how a Turing machine could use the description of $D$ provided on its input to simulate $D$ on $w$ and determine whether it reached a final state.

4. Given a Turing machine $M$ whose encoding requires $l$ symbols, does $M$ run for at least $l$ steps on all possible inputs?

   $A_{STEP} = \{\langle M\rangle \mid M$ is a Turing Machine and $M$ runs for at least $|\langle M\rangle|$ steps for a given input$\}$.
   We can build a turing machine that simulates $M$ on all inputs of length up to $|\langle M\rangle|$. If $M$ accepts or rejects any strings before $|\langle M\rangle|$ steps then we reject the machine. However, if $M$ does not halt at $|\langle M\rangle| - 1$ steps for any input up to length $|\langle M\rangle|$, then we accept the machine. This is because all subsequent strings have one of the tested strings as a prefix, so they must take at least $|\langle M\rangle|$ steps.

5. Given two context-free grammars, do they describe the same language?

6. Given a Turing machine $M$ and an input $w$, does $M$ ever change the contents of a tape cell while processing $w$?

$A_{cell\Delta} = \{\langle M, w\rangle \mid M\}$ is a Turing Machine that changes the contents of its tape on input $w\}$
This is decideable. We can simulate $M$ on input $w$ and keep track of every configuration reached. If $M$ ever changes the content of its tape then we accept. Also, if the same configuration was reached more than once, then we know we have entered a loop so we terminate and reject. Additionally, if the machine moves to the right the same amount of times as the number of states after the last character of the input then we terminate. This is because there would not be a way to go back to the part of the tape containing the input.

7. Given a Turing machine $M$, is $L(M)$ finite?

8. Given a DFA $D$, does it always accept $w^R$ if it accepts $w$?

$A_{DFA^R} = \{\langle D \rangle \mid D \text{ is a DFA where } w \in L(D) \implies w^R \in L(D)\}$
This language is decideable. Given a DFA $D$, we have proven in class that we can construct a DFA $D'$ where $L(D) = L, L(D') = L^R = \{w \mid w^R \in L\}$. Since having a string $w \in L$ implies that $w^R \in L$, we can say $L^R \subseteq L$, and $L^R \cap \bar{L} = \emptyset$. We know regular languages are closed under intersection and complement, so we can build a DFA $D^*$ that recognizes $L^R \cap \bar{L}$. If the language of $D^*$ is empty, then we know that $D$ accepts $w^R$ if it accepts $w$. We showed in class that building a turing machine that determines if a DFA is empty is decidable. Thus, we can build a turing machine that given $\langle D \rangle$, it builds $D^*$, and accepts $D$ if it determines that the language of $D^*$ is empty.