The first three questions on this assignment should be completed as group work. One copy of the solution to each problem should be submitted by noon on Wednesday or Thursday 4/29 or 4/30 depending on your group's meeting time.

The final question on this assignment should be submitted independently by each student by Tuesday 4/28. You may discuss approaches to the question with members of your working group, but the final writeup you submit should represent your own work in the same sense that was expected for all homework submissions during the first half of the semester.

1. Below you will find a list of descriptions of languages languages that encode questions about automata or grammars. Some of these languages are decidable and some are not. Identify one of the languages in the list that is decidable and argue that you can construct a Turing machine that decides the language.

   Make sure that you make it clear which of the languages you are claiming is decidable.

   - $SUBSET_{CFL}^{DFA} =$
     $$\{\langle G, M\rangle \mid G \text{ is a CFG and } M \text{ is a DFA and } L(G) \subset L(M)\}$$
   - $FINITE_{TM} =$
     $$\{\langle M\rangle \mid M \text{ is a TM and } |L(M)| = n \text{ for some } n \geq 0\}$$
   - $CONTAINSPAL_{DFA} =$
     $$\{\langle M\rangle \mid M \text{ is a DFA and } ww^R \in L(M) \text{ for some } w \in \Sigma_M^*\}$$
   - $CONTAINSPAL_{TM} =$
     $$\{\langle M\rangle \mid M \text{ is a TM and } ww^R \in L(M) \text{ for some } w \in \Sigma_M^*\}$$

2. Using the same list of languages provided in problem 2, identify one language that is not decidable but is either recognizable or has a recognizable complement. First explain how one could construct a Turing machine to recognize the language (or its complement).

   Finally, argue that the language is not decidable by showing that you can reduce a language that we have already shown to be undecidable to the language you choose. That is, show that if a Turing machine that could decide that language you think is undecidable existed, you could use it to build a Turing machine that would decide a problem we already know to be undecidable. Likely candidate languages to use in this reduction would include A_{TM}, E_{TM} and their complements. Justify this claim by describing a Turing machine that can recognize the language or its complement.

3. (This is a big, long problem borrowed from K Schwarz of Stanford. Don't be scared. It really is not very hard, but the result is interesting.) There are two classes of languages associated with Turing machines — the recursively enumerable languages (RE), which can each be recognized by a Turing machine, and the recursive languages (R), which can each be decided by a Turing machine. Why didn't we talk about a model of computation that accepted just the R languages and nothing else? After all, having such a model of computation would be useful — if we could reason about automata that just accept recursive languages, it would be much easier to see what problems are and are not decidable.

   It turns out, interestingly, that there is no class of automata with this property, and in fact the only way to build automata that can decide all recursive languages is to have automata that also accept some languages that are RE but not R. This problem explores why.

   Suppose, for the sake of contradiction, that there is a type of automaton called a deciding machine (or DM for short) that has the computational power to decide precisely the R languages. That is, $L \in R$ iff there is a DM that decides L. We will make the following (reasonable) assumptions about deciding machines:

   - Any recursive language is accepted by some DM, and each DM accepts a recursive language.
   - Since DMs accept precisely the recursive languages, all DMs halt on all inputs. That is, all DMs are deciders.
   - Since deciding machines are a type of automaton, each DM is finite and can be encoded as a string. For any DM D, we will let the encoding of D be represented by $\langle D \rangle$.
   - DMs are an effective model of computation.

   Thus the Church-Turing thesis says that the Turing machine is at least as powerful as a DM. Thus there is some Turing machine UD that takes as input a description of a DM D and some string w, then accepts if D accepts w and rejects if D rejects w. Note that UD can never loop infinitely, because D is a deciding machine and always eventually accepts or rejects. More specifically, UD is the decider "On input $\langle D, w \rangle$, simulate the execution of D on w. If D accepts w, accept. If D rejects w, reject."

   Unfortunately, these four properties are impossible to satisfy simultaneously.

   (a) Consider the language $REJECT_{DM} = \{\langle D \rangle \mid D \text{ is a DM that rejects } \langle D \rangle \}$. Prove that $REJECT_{DM}$ is decidable.

   (b) Prove that there is no DM that decides $REJECT_{DM}$.

   Your result from (b) allows us to prove that there is no class of automaton like the DM that decides precisely the R languages. If one were to exist, then it should be able to decide all of the R languages, including $REJECT_{DM}$. However, there is no DM that accepts the decidable language $REJECT_{DM}$.

Note: A word or two about what the impossible model of computation this problem talks about might look like may help you appreciate what it says. If used in simple ways (some might say appropriate ways), the header of a for loop makes it clear how often the loop will execute. That is, if you see something like

for (int x = init; x < max; x = x + increment) }

you can compute (max - init)/increment to get the number of times you expect the loop to execute. In Java, C, or C++, your estimate may be wrong because code in a loop's body can modify x, max or increment. If, however, you imagine a language in which the compiler enforces a restriction that the body of a loop cannot do anything that might modify x, max, or increment, you could get a language where for loops could never lead to infinite loops. Then, all you would have to do is eliminate while loops and recursion to get a language in which it was impossible to write an infinite loop. This question shows that it would be impossible to write certain program that always terminate in such a language.

4. Below you will find a list of informal questions about automata and grammars of various sorts. Some of these questions are decidable and some are not. For all of the question, you should formulate the question as a language. Then, you should identify two of the questions that are decidable and argue that there is a Turing machine that decides these languages.

   Make sure that you make it clear which of the original questions you are claiming are decidable.

   For example, if one of the questions below was "Does the language of a given finite automaton contain a specific string w?", you would formulate this as the language

$$A_{DFA} = \{\langle D, w\rangle \mid D \text{ is a DFA and } w \in L(D)\}$$

   Then, if you believed that this language was decidable (it is!) and wanted to justify this belief, you would 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.

   Note that in the description of the question as a language, I did not try to provide a detailed description of the definition of $L(D)$. The point it to precisely describe the structure and components of the strings that will belong to the language that captures the intent of the informal question (D and w in this case).

   (a) Does the language of a given deterministic finite automaton contain all strings over its alphabet?

(b) Given a Turing machine whose encoding requires l symbols, does the machine run for at least l steps on all possible inputs?

(c) Given two context-free grammars, do they describe the same language?

(d) Given a Turing machine, does it ever write a specified symbol a to its tape?

(e) Given a Turing machine and an input, does M ever change the contents of a tape cell while processing that input?

(f) Given a Turing machine M, is its language finite?