1. Consider the languages

\[ \text{SUBSET}_{DFG} = \{ \langle G, M \rangle \mid G \text{ is a CFG, } M \text{ is a DFA and } L(G) \subseteq L(M) \} \]

and

\[ \text{SUBSET}_{\text{CFG}} = \{ \langle G, M \rangle \mid G \text{ is a CFG, } M \text{ is a DFA and } L(M) \subseteq L(G) \} \]

For each of these languages, prove that it is or is not decidable. Hint: One is decidable and one is not, but this hint cannot be used as the basis of your proof.

2. Show that \( A \) is Turing-recognizable iff \( A \leq_m A_{TM} \).

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 ⟨D, w⟩, 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 \( \text{REJECT}_{DM} = \{ ⟨D⟩ \mid D \text{ is a DM that rejects } ⟨D⟩ \} \). Prove that \( \text{REJECT}_{DM} \) is decidable.

(b) Prove that there is no DM that decides \( \text{REJECT}_{DM} \).

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 \( \text{REJECT}_{DM} \). However, there is no DM that accepts the decidable language \( \text{REJECT}_{DM} \).

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

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

you can compute \((\text{max} - \text{init})/\text{increment}\) 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. Use mapping reductions to show that neither the language

\[
\text{TOTAL}_{TM} = \{ ⟨M⟩ \mid M \text{ is a TM that halts on all inputs} \}
\]

nor its complement is recognizable. (Hint: Feel free to think about how to solve this problem without using a mapping reduction. Then, later you can figure out how to convert your “free-style” proof into one using mapping reductions.)