Problem 1

In class we proved that the language

\[ \text{HALT}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ halts on input } w \} \]

is not decidable by reduction from the acceptance problem \( A_{TM} \).

In this problem you are asked to give a direct proof of the same result. Prove (using a diagonalization argument) that the language \( \text{HALT}_{TM} \) is undecidable.

**Solution:** Assume for contradiction that \( \text{HALT}_{TM} \) is decidable and let \( H \) be a Turing machine that decides \( \text{HALT}_{TM} \). Define a new Turing machine \( D \) defined as follows:

1. \( D \) takes as input the description \( \langle M \rangle \) of a Turing machine \( M \).

2. On input \( \langle M \rangle \), TM \( D \) run \( H \) on input \( \langle M, \langle M \rangle \rangle \) to decide whether or not \( M \) terminates when run on its own description \( \langle M \rangle \).

3. If \( H \) rejects, then \( D \) accepts.

4. If \( H \) accepts, then \( D \) enters an infinite loop.

We now consider what happens if we run \( D \) on its own description \( \langle D \rangle \). The question is: does \( D \) terminate when run on input \( \langle D \rangle \)? We examine both possibilities and show that in both cases we get a contradiction.

1. First assume that \( D \) terminates on input \( \langle D \rangle \). Then \( \langle D, \langle D \rangle \rangle \in \text{HALT}_{TM} \) and at step (2) \( H \) accepts \( \langle D, \langle D \rangle \rangle \). Therefore \( D \) enter an infinite loop at step (4) and does not terminate: a contradiction!

2. Now assume that \( D \) does not terminate on input \( \langle D \rangle \). Then \( \langle D, \langle D \rangle \rangle \notin \text{HALT}_{TM} \) and at step (2) \( H \) rejects \( \langle D, \langle D \rangle \rangle \). Therefore \( D \) will terminate accepting the input at step (3): a contradiction!
Problem 2

In the second midterm you were asked to prove that the language

\[ L = \{ \langle G, E \rangle \mid G \text{ is a CFG and } E \text{ is a regular expression such that } \mathcal{L}(G) \subseteq \mathcal{L}(E) \} \]

is decidable. In this problem you are asked to study what happens if you exchange \( E \) and \( G \) in the definition of \( L \). That is, consider the language

\[ L' = \{ \langle G, E \rangle \mid G \text{ is a CFG and } E \text{ is a regular expression such that } \mathcal{L}(E) \subseteq \mathcal{L}(G) \} \].

Is \( L' \) decidable? (Prove or disprove)

**Solution:** \( L' \) can be easily proved undecidable by reduction from the language \( \text{ALL}_{CFG} \) which was proved undecidable in class. Recall that \( \text{ALL}_{CFG} \) is the set of all strings \( \langle G \rangle \) such that \( G \) is a context free grammar that generates the entire language \( \Sigma^* \).

We now give a simple reduction from \( \text{ALL}_{CFG} \) to \( L' \). Let \( M \) a TM deciding \( L' \). We describe a TM \( R \) that using \( M \) as a subroutine decides \( \text{ALL}_{CFG} \).

On input \( \langle G \rangle \), \( R \) does the following:

1. Build a regular expression \( E \) that generates the language \( \Sigma^* \), where \( \Sigma \) is the alphabet of \( G \).
2. Run \( M \) on input \( \langle G, E \rangle \) to check whether \( \mathcal{L}(E) \subseteq \mathcal{L}(G) \).
3. If \( M \) accepts, then \( R \) also accepts
4. If \( M \) rejects, then \( R \) also rejects

It is easy to see that this is in fact a map reduction from \( \text{ALL}_{CFG} \) to \( L' \) with reduction function

\[ f : \langle G \rangle \mapsto \langle G, \Sigma^* \rangle \]

Problem 3

Do problem 5.20(b) from the book. I.e., show that the emptiness problem \( E_{2DFA} \) for two headed finite automata is undecidable. (See the textbook for precise definition of 2DFA and \( E_{2DFA} \).)

Also, answer the following questions: is \( E_{2DFA} \) enumerable? is \( E_{2DFA} \) co-enumerable? (Briefly justify your answer)

**Solution:** We prove that \( E_{2DFA} \) is undecidable by reduction from \( E_{TM} \) (a similar reduction is also possible from \( E_{LBA} \) or \( A_{TM} \)). I.e., we show how to transform a TM \( M \) into a 2DFA \( A \) such that \( \mathcal{L}(M) \) is empty if and only if \( \mathcal{L}(A) \) is empty. The idea is to define a 2DFA \( A \) that accepts all strings that represent accepting computations for \( M \). If \( M \) does not accept any string then the set of accepting computation for \( M \)
is also empty. On the other hand, if $M$ accept some string, then it has at least one accepting computation and therefore $L(A)$ is not empty.

The 2DFA $A$ can be defined as follows. Let $M = (Q; \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$. The automaton $A$ has input alphabet $\Gamma \cup Q \cup \{\#\}$ where $\#$ is some special symbol not in $\Gamma$.

Let $w = c_0\#c_1\#\ldots\#c_n$ be the input to $A$ where each $c_i$ is a string not containing the special symbol $\#$. Automaton $A$ will check that each $c_i$ represent a configuration, $c_0$ is a start configuration (i.e. a string of the form $q_{\text{start}} \Sigma^*$), each $c_i$ yields $c_{i+1}$ according to the transition function $\delta$ of TM $M$, and $c_n$ is an accepting configuration, i.e. $c_n$ contains the state $q_{\text{accept}}$. Details follow.

Automaton $A$ starts with both heads to the left of the input string $w$ and does the following:

1. Check that $w$ starts with a string of the form $q_{\text{start}} \Sigma^*$ moving the first tape head to the right until a $\#$ is found. If not, reject.

2. At this point the two tape heads are positioned at the beginning of two successive configuration in the computation history $w$ and $A$ must check that the first configuration $c_i$ yields the second one $c_{i+1}$. This is done as follows:
   
   (a) Move both tape heads in parallel to the right checking that the symbols read are equal

   (b) As soon as a difference is found, read the next three symbols with both heads to check that they correspond to a valid transition according to $\delta$. More precisely, if the three symbols from the first configuration are $u_1 = aqb$ and $\delta(q, b) = (r, c, L)$ then the three symbols from the second configuration should be $u_2 = rcb$; while if $u_1 = qab$ and $\delta(q, a) = (r, c, R)$ then it should be $u_2 = crb$. Special rules are needed when the TM tape head is either at the beginning or the end of the tape. So, for example if $u_1 = aq\#$ and $\delta(q, \ldots) = (r, b, L)$ the it should be $u_2 = qab$.

   (c) finally checks that the remaining symbols of the two configurations (up to the next $\#$) are the same.

3. After $A$ has checked the $c_i$ yields $c_{i+1}$, the tape heads are at the beginning of configurations $c_{i+1}$ and $c_{i+2}$ and $A$ can go on to check that $c_{i+1}$ yields $c_{i+2}$. If $A$ ever encounter a configuration containing state $q_{\text{accept}}$, then an accepting configuration has been found and $A$ accept the computation.

The transformation we just described from $M$ to $A$ is in fact a map-reduction from $E_{TM}$ to $E_{2DFA}$, and therefore it proves that $E_{2DFA}$ is undecidable.

In fact, $E_{2DFA}$ is co-enumerable but not enumerable. The co-enumerability of $E_{2DFA}$ follows from the observation that the acceptance problem for 2DFA ($A_{2DFA}$) is decidable. The proof is similar to the one for $A_{LBA}$ from section 5.1 of the book.
Alternatively you can easily reduce $A_{2DFA}$ to $A_{LBA}$ which is proved decidable in the book.

Then a TM recognizing $\overline{E_{2DFA}}$ can be defined as follows: on input 2DFA $A$ consider all the strings $w \in \Sigma^*$ in lexicographic order, and for each one of them check if $A$ accepts $w$ using the decoder for $A_{2DFA}$. If some string $w$ is ever accepted by $A$, then accept $A$. Otherwise keep trying with some other string $w$.

This proves that $E_{2DFA}$ is $\omega$-enumerable. Finally, $E_{2DFA}$ cannot be enumerable, because if it were both enumerable and $\omega$-enumerable it would be decidable which we know is false.

Problem 4

Let $L$ be a language.

(a) Show that if $L \leq_m \overline{L}$ (i.e., $L$ map reduces to the complement of $L$), then $L$ is either (1) decidable or (2) neither enumerable nor $\omega$-enumerable.

**Solution:** The language $L$ is either enumerable or not.

First assume $L$ is enumerable. Notice that $L \leq_m \overline{L}$ implies $\overline{L} \leq_m L$. Since $L$ is enumerable, then also $\overline{L}$ is enumerable and by theorem 4.16, $L$ is decidable.

Now assume $L$ is not enumerable and assume for contradiction that $L$ is $\omega$-enumerable. By definition, this means that $\overline{L}$ is enumerable. Since $L \leq_m \overline{L}$, this implies that also $L$ is enumerable, contradicting our assumption.

(b) The following statement is false:

"If $L$ is decidable then $L \leq_m \overline{L}$"

Find a counterexample! I.e., find a decidable language $L_1$ such that $L_1$ does not map reduce to its complement. Now, find another counterexample! I.e., find a decidable language $L_2$ different from $L_1$ such that $L_2$ does not map reduce to $\overline{L_2}$. Finally, prove that $L_1$ and $L_2$ are the only two counterexamples! I.e., show that for any decidable language $L$ different from $L_1$ or $L_2$, it holds $L \leq_m \overline{L}$.

**Solution:** Let $L_1$ be the set of all strings over the alphabet $\Sigma$. Then $L_1$ cannot be map reduced to $\overline{L_1}$ because a map reduction from $L_1$ to $\overline{L_1}$ should map strings in $L_1$ to some string in $\overline{L_1}$, but $\overline{L_1}$ is the empty set and no such string exists.

Similarly the empty set $L_2 = \emptyset$ cannot be map-reduced to its complement $\overline{L_2} = L_1$, giving another counter example to the statement in the problem.

Now consider any decidable language $L$ which is neither $\Sigma^*$ nor the empty set $\emptyset$. We want to prove that $L$ is map-reducible to $\overline{L}$. Since $L$ is not empty, there exists a string $w_1$ such that $w_1 \in L$. Similarly, since $L$ is not $\Sigma^*$, there exists a string $w_2$ such that $w_2 \notin L$. Finally since $L$ is decidable, there exists a Turing
machine $M$ that decides $L$. Consider the function defined by the following Turing machine:

(a) On input $w$, simulates $M$ on input $w$.
(b) If $M$ accepts, then output $w_2$
(c) If $M$ rejects, then output $w_1$.

This is obviously a Turing computable function, and it maps all strings in $L$ to $w_2$ which belongs to $\overline{L}$, and all strings not in $L$ to $w_1$ which is not an element of $\overline{L}$. Therefore, the function is a reduction from $L$ to $\overline{L}$. 