1. Question 1.

(1) Show that PCP is decidable relative to $E_{TM}$.

Here is a recognizer for PCP:

$R = \text{"On input } \langle P \rangle \text{, where } P \text{ is a set of tiles}

\text{Let } P = \{T_1, \ldots, T_n\} \text{ be the } n \text{ tiles}

\text{Enumerate all strings in } \{1, \ldots, n\}^* \text{ in nondecreasing order of length}

\text{For each } w = i_1 \cdots i_\ell \text{ enumerated}

\text{Concatenate tiles } T_{i_1}, \ldots, T_{i_\ell} \text{ in that order.}

\text{If the top strings and bottom strings are identical, accept."

Here is a decider for PCP relative to $E_{TM}$:

$D^{E_{TM}} = \text{"On input } \langle P \rangle \text{, where } P \text{ is a set of tiles}

\text{Query the } E_{TM} \text{ oracle with the input } \langle R, P \rangle.$

\text{If the oracle says yes, reject; if it says no, accept."}

(2) Show that $A_{TM}$ is decidable relative to PCP.

This is essentially the mapping reduction from $A_{TM}$ to PCP given in Sipser
Theorem 5.15 and in class.

2. Question 2.

(1) $EVEN_{PDA} = \{ \langle M \rangle : \text{the language recognized by the PDA } M \text{ only contains}

\text{strings of even length}\}$.

Prove or disprove: $EVEN_{PDA}$ is decidable.

$EVEN_{PDA}$ is decidable. Form a DFA $D$ that accepts $\Sigma(\Sigma\Sigma)^*$, where $\Sigma$ is

the input alphabet of $M$. (This accepts exactly the strings of odd length.)

Form a PDA $M'$ that accepts $L(M) \cap L(D)$ (using the standard product construction).

Then $L(M') = \emptyset$ if and only if $M$ accepts only strings of even

length. Now observe that $E_{PDA}$ is decidable, so we are done.
2. **\( \text{EVEN}_{TM} = \{ \langle M \rangle : \text{the language recognized by the TM } M \text{ only contains strings of even length} \} \). Prove or disprove: \( \text{EVEN}_{TM} \) is decidable.

   \( \text{EVEN}_{TM} \) is undecidable. We make two observations. First, if \( M \) and \( M' \) are equivalent TMs, either both belong to \( \text{EVEN}_{TM} \) or neither do – so this is a property of the language. It is nontrivial because the machine whose language is \( \emptyset \) is in \( \text{EVEN}_{TM} \) but the machine whose language is \( \Sigma^* \) is not. So we have met the two requirements to apply Rice’s Theorem, which now states that \( \text{EVEN}_{TM} \) is not decidable.

3. **Question 3.**

   (1) Something about the recursion theorem confuses my friend. Here is his argument:
   
   Suppose (to the contrary) that the recursion theorem is true. Then we can construct a Turing machine \( M \) as follows:
   
   \[ M = \text{“On input } x \text{:\n}
   \text{Obtain own description } \langle M \rangle \text{\n   Simulate } M \text{ on } x \text{ and do the opposite.”} \]
   
   Then \( M \) accepts \( x \) if and only if \( M \) does not accept \( x \). But then \( M \) cannot exist, and the recursion theorem cannot be true!
   
   Please explain to my friend his error (if indeed he is wrong).

   He is wrong. There is no real contradiction, because \( M \) will run forever on every input, and so the “do the opposite” instruction can never be reached. (Think about what happens when \( M \) is run. There’s an infinite recursion.)

   (2) Something else about the recursion theorem confuses my friend. He says that if a Turing machine can obtain its own description, it cannot possibly be a decider. (His explanation: Of course it cannot be! Didn’t we learn that in kindergarten?) I do not respond well to proof by intimidation and do not believe him. Is my friend wrong? Explain carefully (it is not sufficient to say that we did not learn that in kindergarten).

   He is wrong again (perhaps I need new friends). Let’s ignore the foolishness about kindergarten. All we need to do is show him a counterexample. So modify SELF so that after it prints itself, it always accepts. (As given in class and the text, it always halts.) Then SELF is a decider that gets its own description.

4. **Question 4.**

   A Turing machine with an output tape can **enumerate** a language, and a language is **Turing-enumerable** if there is a Turing machine that enumerates it.
(1) Is it true that, whenever a language \( L \) is Turing-recognizable, there is a Turing machine that enumerates \( L \) so that each string in the language appears exactly once on the output tape? Explain.

This is true. Let \( R \) be a recognizer for \( L \). Let \( E \) be an enumerator for \( \Sigma^* \). Here is an enumerator \( Z \) for \( L \) that only lists each string in \( L \) once:

\[
Z = \text{"On any input:}
\text{Initialize a library tape to empty}
\text{for } \ell = 1, 2, \ldots
\text{Run } E \text{ to generate } \ell \text{ strings } w_1, w_2, \ldots, w_\ell
\text{for } i \text{ from } 1 \text{ to } \ell
\text{Run } R \text{ on input } w_i \text{ for } \ell \text{ steps.}
\text{If } R \text{ accepts, then check whether } w_i \text{ is on the library tape,}
\text{and if it is not, then write it on the output and library tapes.}
\]

This enumerates \( L \) but never prints the same string twice.

(2) Is it true that, whenever a language \( L \) is Turing-recognizable, there is an integer constant \( c \) for which some Turing machine enumerates \( L \) so that if string \( w' \) is placed on the output after string \( w \), the length of \( w' \) plus \( c \) is at least the length of \( w \) (i.e., \( |w'| + c \geq |w| \))? Explain.

I interpreted ‘after’ in the statement as ‘immediately following’. Kaushik pointed out to me that it could be read as ‘following any time later on the list’, which would make the problem much simpler (but not change the conclusion). I have used the ‘immediately following’ interpretation here.

This is false, but the argument is maybe not the obvious one.

(Indeed if \( L \) is recognizable, and there is some constant \( c \) so that for every two strings if \( |w'| + c < |w| \) then there is string \( w'' \in L \) with \( |w'| < |w''| < |w| \), then the answer would be true for \( L \). So the argument must rely on finding a specific \( L \).)

To get a correct argument, let \( H \) be a language that is recognizable but not decidable. Now form a language \( L \) by replacing each string \( w \in H \) by \( w|w| \). This is recognizable but not decidable (the enumerator for \( H \) can be used to enumerate \( L \), and a decider for \( L \) could be used to decide \( H \)). The key observation is that no matter how you choose \( c \), once \( |w| > c \), there is no string in \( L \) of length between \( |w| + 1 \) and \( |w| + c \). This ensures that any enumerator for \( L \) satisfying the requirements must list all strings of length at most \( \ell^2 \) in \( L \) before any of length \( (\ell + 1)^2 \) can be listed, whenever \( \ell \geq c \). But knowing this, if we want to decide whether a string of length \( \ell^2 \) is in \( L \), an enumerator meeting the requirements can be run until it outputs the string (so we accept) or outputs a string of length \( \max(\ell, c) + 1)^2 \) (so we reject). But then \( L \) is decidable, which is a contradiction.
5. Question 5.

An unrestricted grammar is a pair $G = (\Gamma, R)$ where $\Gamma$ is a finite alphabet, and $R$ is a finite set of rules, each of which maps a string $w \in \Gamma^*$ to a string $y \in \Gamma^*$ (i.e. the rule is $w \rightarrow y$). A length-increasing grammar is an unrestricted grammar in which the right-hand side of each rule is at least as long as the left hand side. One symbol in $\Gamma$ is the start variable $S$, and a subset $\Sigma \subseteq \Gamma \setminus \{S\}$ consists of terminals. The language of $G$ is $L(G) = \{w : w \in \Sigma^* \text{ and } S \Rightarrow^* w\}$.

1) Prove or disprove: The question “Is $w \in L(G)$?” is decidable when $G$ is a length-increasing grammar.

“Is $w \in L(G)$?” is decidable, as follows. Suppose that $|\Gamma| = \gamma$. Given a length $n$, there are fewer than $\gamma^{n+1}$ strings that $S$ could generate. We make an enormous (but finite) table, to mark which of these strings can be generated. To do this, we start by marking $S$ and having all other strings unmarked. Then whenever $xuy$ is marked and there is a rule $u \rightarrow v$, mark $xvy$ if $|xvy| \leq |w|$. Keep applying rules until no new strings are marked. Then check whether $w$ is marked. If it is, accept; otherwise reject.

2) Prove or disprove: The question “Is $L(G) = \emptyset$?” is decidable when $G$ is a length-increasing grammar.

“Is $L(G) = \emptyset$?” is undecidable, as follows. We use linear bounded automata (LBA). From class we know that given a TM $M$ and input $w$, there is an LBA $N$ whose language is precisely the accepting computation histories of $M$ on $w$; hence $E_{LBA}$ is undecidable.

We produce from $N$ an equivalent length-increasing grammar $G$ as follows. When $N$ has states $Q$ and tape alphabet $\Gamma'$, we set $\Gamma = Q \cup \Gamma' \cup \{S, T, \#\}$. We set $\Sigma = \{q_{\text{accept}}\}$. The rules of $G$ are

(a) $S \rightarrow \# q_0 T$,
(b) $T \rightarrow a T$ for $a \in \Gamma'$, and $T \rightarrow \#$,
(c) $qa \rightarrow bq'$ when $\delta(q, a) = (q', b, R)$ for $q, q' \in Q \setminus \{q_{\text{reject}}\}$ and $a, b \in \Gamma'$,
(d) $cqa \rightarrow q' cb$ when $\delta(q, a) = (q', b, L)$ for $q, q' \in Q \setminus \{q_{\text{reject}}\}$ and $a, b, c \in \Gamma'$, and
(e) $q_{\text{accept}} a \rightarrow q_{\text{accept}} q_{\text{accept}}$ and $aq_{\text{accept}} \rightarrow q_{\text{accept}} q_{\text{accept}}$ for $a \in \Gamma' \cup \{\#\}$.

Then $G$ generates a string consisting entirely of $q_{\text{accept}}$ if and only if $N$ accepts some string.