Problem 1

Consider the language:

\[ 2P = \{ \langle M \rangle \mid M \text{ accepts at least 2 palindromes, } \Sigma = \{0,1\} \} \]

Recall, that a palindrome is an input which is the same forward as it is backward, \( w = w^R \), where \( w^R \) is \( w \) input in reverse.

Prove that \( 2P \) is undecidable.

Proof idea

We reduce the problem of deciding \( A_{TM} \) to the problem of deciding \( 2P \). That is, we show that if \( 2P \) were decidable, \( A_{TM} \) would also be decidable. To perform the reduction, we suppose that there is a machine \( M_{2P} \) which decides \( 2P \). We then demonstrate how, given a machine \( M \) and input \( w \), to construct a machine \( N \) which accepts two arbitrary palindromes if and only if \( M \) accepts on input \( w \). We then construct \( M_A \) which takes as input a machine-input pair \( \langle M, w \rangle \), constructs a machine \( N \), then runs \( M_{2P} \) on \( N \) and outputs whatever \( M_{2P} \) outputs. By stipulation, \( M_{2P} \) accepts if \( N \) accepts two palindromes (indicating that \( M \) accepts on \( w \)) and rejects if \( N \) does not accept two palindromes (indicating that \( M \) does not accept on \( w \)). Therefore, \( M_A \) decides \( A_{TM} \).

Proof. Suppose there is some machine \( M_{2P} \) that decides \( 2P \). Given an arbitrary machine \( M \) and input \( w \), can construct the following machine \( N \):

Define \( N \): "on input \( x \),
1. If \( x = 010 \), accept.
2. If \( x = 101 \), run \( M \) on \( w \).
   a. If \( M \) accepts on \( w \), ACCEPT.
   b. If \( M \) rejects on \( w \), REJECT.
3. Else, REJECT."

The only inputs that \( N \) can possibly accept are ‘010’ and ‘101’, but it only accepts ‘101’ if \( M \) accepts on input \( w \). So, \( N \) accepts two palindromes if and only if \( M \) accepts on \( w \).
Construct $M_A$ as follows:

Define $M_A$: "on input $<M,w>$,
1. Construct $N$ (as above).
2. Run $M_{2P}$ on $N$.
   a. If $M_{2P}$ accepts, ACCEPT.
   b. If $M_{2P}$ rejects, REJECT."

$M_A$ decides $A_{TM}$.

There are two cases to consider: $\langle M, w \rangle \in A_{TM}$ and $\langle M, w \rangle \notin A_{TM}$.

Case 1. Suppose $\langle M, w \rangle \in A_{TM}$. Then $M$ accepts on $w$ and $N$ accepts on ‘101’. Which means that $N$ accepts two palindromes, so $M_{2P}$, when run on $N$, accepts. So, $M_A$ accepts $\langle M, w \rangle$.

Case 2. Suppose $\langle M, w \rangle \notin A_{TM}$. Then $M$ either loops on $w$ or rejects. Suppose it rejects. Then $N$ rejects on ‘101’ and, so, accepts only one palindrome. So $M_{2P}$, when run on $N$, rejects and $M_A$ rejects $\langle M, w \rangle$. Suppose $M$ loops on $w$. Then $N$ loops on ‘101’ and, so, accepts only one palindrome. So $M_{2P}$, when run on $N$, rejects and $M_A$ rejects $\langle M, w \rangle$.

Therefore, $M_A$ decides $A_{TM}$. Since $A_{TM}$ is undecidable, $2P$ is undecidable. □
Problem 2
Consider the language from Problem 1:

\[ 2P = \{ \langle M \rangle | M \text{ accepts at least 2 palindromes, } \Sigma = \{0, 1\} \} \]

Prove that \(2P\) is recognizable.

Proof idea
We construct a machine \(M_{2P}\) which takes as input a machine \(\langle M \rangle\). \(M_{2P}\) runs \(M\) on an increasing number of inputs, for an increasing number of steps, at each iteration. If \(M\) accepts two palindromes, \(M_{2P}\) eventually halts and accepts.

Proof. Construct the machine \(M_{2P}\) on input \(\langle M \rangle\). Let \(M =< Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta >\) and let \(s_1, s_2, s_3, s_4, s_5, \ldots\) be an enumeration of \(\Sigma^*\).

Define \(M_{2P}\): "on input \(\langle M \rangle\),

1. \(i := 1\) and \(j := 1\) and \(k := 0\).
2. Repeat the following steps.
   a. If \(s_j\) is a palindrome
      aa. Run \(M\) on \(s_j\) for \(i\) steps.
     bb. If \(M\) accepts \(s_j\), increment \(k\).
   b. If \(k = 2\), ACCEPT.
   c. If \(j = i\), increment \(i\), \(j := 1\), and \(k := 0\).
   d. Else, increment \(j\).

\(M_{2P}\) recognizes \(2P\).

Suppose \(\langle M \rangle \in 2P\). Then \(M\), after some finite number of steps, accepts distinct inputs \(w, x\) which are palindromes. Since \(w, x \in \Sigma^*\), for some \(y\) and \(z\), \(w = s_y\) and \(x = s_z\). Suppose without loss of generality that \(y > z\) and that the number of steps \(n\) it takes \(M\) to accept \(w\) is greater than the number of steps it takes to accept \(x\). Then, on the iteration of step 2 of \(M_{2P}\) when \(i = n + z\), \(M\) will accept \(s_y\) and \(s_z\), \(k := 2\) and \(M_{2P}\) accepts (if it did not accept on an earlier iteration).

So, if \(\langle M \rangle \in 2P\), then \(n + z\) serves as an upper bound on the number times \(i\) is incremented in \(M_{2P}\), and so the computation will halt and accept after some finite amount of time.

If \(\langle M \rangle \notin 2P\), then, but the same reasoning, \(M_{2P}\) loops. Therefore, \(M_{2P}\) recognizes \(2P\). \(\square\)
Problem 3
Consider the language:

\[ \text{Palindrome} = \{ \langle M \rangle | M \text{ accepts only palindromes, } \Sigma = \{0, 1\} \} \]

Prove that \( \text{PALINDROME} \) is unrecognizable.

Proof idea

The solution presented here assumes that a machine \( M \) is a member of \( \text{PALINDROME} \) iff it accepts only palindromes. If you assumed that \( M \in \text{P} \) iff it accepts all and only palindromes, that is also fine, but your proof will look slightly different.

Let

\[ P = \text{PALINDROME} \]
\[ \overline{P} = \text{PALINDROME} = \{ \langle M \rangle | M \text{ accepts some non-palindrome, } \Sigma = \{0, 1\} \} \]

In order to show that \( P \) is unrecognizable, we will show that \( \overline{P} \) is both computably enumerable and not computable. We show that \( \overline{P} \) is computably enumerable using the same method as in Problem 2. We show that \( \overline{P} \) is not computable by proving that \( A_{TM} \leq_m \overline{P} \).

This reduction is proved using the same technique as Problem 1. Given any \( M \) and \( w \), we construct a machine \( N \) that accepts an arbitrary non-palindrome if and only if \( M \) accepts \( w \). The supposed decider for \( \overline{P} \) is run with \( N \) as input, and the resulting output is used to compute \( A_{TM} \).

Lemma 1: \( \overline{P} \) is computably enumerable.

Define \( R_{P} \): "on input \( \langle M \rangle \),

1. (As in Problem 2) run \( M \) on an increasing number of inputs, for an increasing number of steps.

2. If \( M \) accepts some \( s_m \) and \( s_m \) is not a palindrome, ACCEPT."

Suppose \( M \) accepts some non-palindrome string \( s_m \), and \( M \)'s computation on \( s_m \) takes \( n \) steps. Then (as in Problem 1) \( m + n \) places an upper bound on the number of ‘\( i \)’ iterations the machine must perform, and so \( R_{\overline{P}} \) will halt in an accepting state after some finite amount of time.

Lemma 2: \( \overline{P} \) is not computable.

Suppose there is a machine \( D_{\overline{P}} \) which computes \( \overline{P} \).
Define $M_A$: "on input $\langle M, w \rangle$,
1. Construct machine $N$, defined as follows:
   Define $N$: ‘on input $x$,
   1’. If ‘$x$’ = ‘10’, run $M$ on $w$.
      a’. If $M$ accepts, ACCEPT.
      b’. If $M$ rejects, REJECT.
   2’. Else, REJECT.
2. Run $D_P$ on input $\langle N \rangle$.
3. If $D_P$ accepts, ACCEPT.
4. If $D_P$ rejects, REJECT.

$M_A$ computes $A_{TM}$.

Case 1. Suppose $\langle M, w \rangle \in A_{TM}$. Then $N$ accepts '10' and so accepts some non-palindrome. So, $M_P$ on $\langle N \rangle$ accepts, and $M_A$ accepts.

Case 2. Suppose $\langle M, w \rangle \notin A_{TM}$. Then $N$ accepts no input strings. Since $N$ accepts no inputs, it accepts no non-palindrome inputs. So $M_P$ rejects on $N$ and $M_A$ rejects on $\langle M, w \rangle$.

Proof. [By contradiction.] Suppose $P$ is computably enumerable. By Lemma 1, $\overline{P}$ is also computably enumerable. So, we have a recognition procedure for both $P$ and $\overline{P}$, making $\overline{P}$ computable. By Lemma 2, $\overline{P}$ is not computable. Therefore, $P$ is not computably enumerable (i.e. is unrecognizable).  $\square$
Problem 4

Let \( C \) be a language. Prove that \( C \) is Turing-recognizable (i.e. computably enumerable) if and only if a decidable language \( D \) exists such that \( C = \{ x \mid \text{There exists a } y \text{ such that } \langle x, y \rangle \text{ is in } D \} \)

Proof idea

If \( C \) is finite, both directions of the biconditional are trivial, so we assume that \( C \) is infinite.

In order to prove the left to right direction of the biconditional, we suppose that \( C \) is computably enumerable. In the enumeration of \( C \) by some machine \( M \), each member of \( C \) will be associated with some place in the enumeration (i.e. there will be a 1st element of the enumeration, a 2nd, a 3rd, and so on). By pairing each element with the integer representing its place, we create a decidable set.

The right to left direction is trivial, and will be proved after.

Lemma 1: If \( C \) is computably enumerable, then there is a computable \( D \) such that for every \( x \in C \), there is a \( y \) such that \( \langle x, y \rangle \in D \).

Suppose \( C \) is computably enumerable. Then some machine \( M_C \) can enumerate \( C \) in an order \( E = c_1, c_2, c_3, c_4, \ldots \) Let \( D \) be defined

\[
D = \{ \langle c_x, x \rangle \mid c_x \text{ is some element of } E \}
\]

Define \( M_D \): "on input \( \langle x, y \rangle \)
   1. Use \( M_C \) to enumerate the first \( y \) elements of \( E \).
   2. If the \( y \)th element is \( x \), ACCEPT.
   3. If the \( y \)th element is not \( x \), REJECT.

\( M_D \) computes \( D \).

If \( \langle x, y \rangle \in D \), then the \( y \)th element of \( E \) is \( x \) and \( M_D \) will accept. If \( \langle x, y \rangle \notin D \), then the \( y \)th element of \( E \) is not \( x \) and \( M_D \) will reject. Therefore, \( D \) is computable.

Lemma 2: If \( C \) is some set and there is a computable \( D \) such that, for every \( x \in C \) there is a \( y \) such that \( \langle x, y \rangle \in D \), then \( C \) is computably enumerable.

Suppose the antecedent condition holds, and suppose that \( M_D \) computes \( D \). Construct \( M_C \) which takes input \( w \) and runs \( M_D \) on each \( \langle w, z \rangle \). Since \( D \) is computable, \( M_D \) will halt on all inputs, and so, if \( w \in C \), \( M_C \) will eventually halt.

Proof. The conclusion follows directly from lemmas 1 and 2. \( \square \)
Problem 5

Prove that every infinite Turing-recognizable (i.e. computably enumerable) language \( C \) contains an infinite subset \( D \) that is decidable.

Proof idea

We, again, assume that \( C \) is infinite.

We generate a decidable set from \( C \)'s enumeration by selecting only those strings that appear in some order in the enumeration. Specifically, we select only those elements that appear in lexicographic order. By eliminating those elements that appear out of order, we impose an order on the enumeration and are able to decide, for any given element in a finite amount of time, whether or not it appears in its place.

Proof. Suppose \( C \) is computably enumerable. Then some machine \( M_C \) enumerates \( C \) in an order \( E = e_1, e_2, e_3, e_4, \ldots \). Let \( D \) be defined

\[
D = \{ e_x | e_x \text{ appears in } E, \text{ and } \\
-\exists e_y \text{ in } E \text{ s.t. } y < x \text{ and } c_y > c_x \text{ (} c_y \text{ comes after } c_x \text{ in lexicographic order)} \}
\]

\( D \subseteq E \).

Since, by definition, every member of \( D \) is also a member of \( E \), \( D \subseteq E \).

\( D \) is infinite.

Suppose \( D \) is finite. Then there is some maximal element of \( E \) (i.e. some element for which no other element of \( E \) comes after it in lexicographic order). If there is a maximal element, then \( E \) is finite. By stipulation, \( E \) is not finite. Therefore, \( D \) is infinite.

\( D \) is computable.

Define \( M_D \): "on input \( w \)

1. Run \( M_C \).
2. If \( M_C \) outputs \( w \), ACCEPT.
3. If \( M_C \) outputs some \( x > w \), REJECT.

Suppose \( w \in D \). Then \( w \) appears in \( E \) and every element of \( E \) prior to \( w \) is lexicographically before, or identical with, \( w \). So, before \( M_C \) outputs any \( x > w \), it will output \( w \) and \( M_D \) will accept.

Suppose \( w \notin D \). Then \( w \) either does not appear in \( E \), or it appears after some \( x > w \). If \( w \) does not appear in \( E \), then eventually \( M_C \) will output some \( x > w \). If \( w \) appears in \( E \) after some \( x > w \), then eventually \( M_C \) will output \( x \). In either case, as soon as the \( x \) is output, \( M_D \) will reject. \( \square \)