1. [4 marks] Show that the following language L is decidable by giving a high-level description of a decider M with L(M) = L.

$$\mathbf{L = \{\langle D\rangle \mid D \text{ is a DFA over } \Sigma^* \text{ for some } \Sigma \text{ and } L(D) = \Sigma^*\}}$$

Solution

A language L is Turing Decidable (or just decidable in short) if there exists a halting TM M (decider) such that L = L(M). L is decidable if there exists a decider that decides the language. To show that the language *L = {⟨D⟩ | D is a DFA over Σ** and *L(D) = Σ** is decidable, we need to describe a high-level algorithm for a decider machine (M) that accepts descriptions of DFAs and decides whether those DFAs accept all strings over their alphabet Σ. The decider M confirms *L(D) = Σ* for DFA(A)* by checking if the complement DFA (B) has an empty language, which is a decidable problem.

If D is a DFA and L(D) = Σ^* then the complement of the language D can be obtained by converting all the accept states in D to reject states, and all the reject states to accept states.

The complement of a language is defined in terms of set difference from Σ^*

$$\begin{aligned} L &= \Sigma* - L \\ L(D) &= \Sigma* - L(D) \\ &= \Sigma^* - \Sigma^* \\ &= \emptyset \end{aligned}$$

Given a DFA D and L(D) = Σ^* we can construct the complement of the DFA D called DFA B with L(B) = ∅. Then we can use TM T on input ⟨B⟩ to check if L(B) = ∅. If T accepts, M accepts.
If T rejects, M rejects.

The following TM M decides L:
M = On input ⟨D⟩ where D is a DFA and L(D)=Σ^*
1. Let B be the DFA obtained by swapping accept and reject states of A.
2. Run TM T on input ⟨B⟩ see if L(B) = ∅.
3. If so accept, else reject.

Therefore, the decider (M) effectively decides the language (L), proving that (L) is decidable.

2. [4 marks] Using a high-level TM description, give a TM M that recognizes the complement of E_{TM}, E_{TM} = {⟨M⟩ | M is a TM and L(M) = ∅ }.

Solution

The problem asks for a Turing Machine M that recognizes the complement of E_{TM}, where ETM = {⟨M⟩ | M is a TM and L(M) = ∅ }. The complement of E_{TM}, denoted as $\overline{E_{TM}}$, consists of all Turing Machines M such that L(M) ≠ ∅, i.e., M accepts at least one string.

To show that the complement of E_{TM}. E_{TM} is Turing-recognizable, we can construct a Turing machine that recognizes E_{TM} by simulating the behavior of a given Turing machine M on a given input s, and accepting if and only if M does not accept s.

Here's a high-level description of the Turing machine M:
- On input ⟨M⟩, M simulates the execution of the Turing machine M on a blank input tape.
- If the simulation of M halts and accepts, then M accepts the input ⟨M⟩, as this means that L(M) ≠ ∅.
- If the simulation of M runs forever or halts and rejects, then M rejects the input ⟨M⟩, as this means that L(M) = ∅.

We want to determine if any of the strings (s1, s2, s3…) is accepted by M, If M accepts at least one string s, then L(M) $\neq \emptyset$. On input <M, s>, where M is a Turing machine and s is a string: construct a new Turing machine N such that:

- Simulate M on s for some fixed number of steps (say, i steps).
- If M has accepted s within those steps, then reject s. Otherwise, accept s.
- Run N on <M, s>. If N accepts, reject <M, s>. If N rejects, accept <M, s>.

It's important to note that M as described is a recognizer for $\overline{E_{TM}}$ rather than a decider. It accepts inputs in $\overline{E_{TM}}$ by finding at least one string that M accepts. However, it cannot always determine if a given TM M belongs to $\overline{E_{TM}}$ (i.e., if L(M) = ∅) due to the undecidability of the halting problem. M can only recognize, not decide, because it cannot always halt with a definitive answer for all possible inputs.

3. [3 marks] Prove that ETM = {⟨M⟩ | M is a TM and L(M) = ∅ } is not Turing-recognizable. You may use your answer from question 2 as well as any proof shown in class.

Solution:

To prove that ETM = {⟨M⟩ | M is a TM and L(M) = ∅ } is not Turing-recognizable, we will use a proof by contradiction, specifically leveraging the concept of reduction from a known non-Turing-recognizable problem. The problem we'll use for the reduction is the Halting Problem, which is known to be undecidable. The Halting Problem is defined as the set of all pairs ⟨M, s⟩ where M is a TM that halts on input. Halting problem is not Turing-recognizable, meaning there's no TM that can recognize all instances of *HP*.

We will assume, for the sake of contradiction, that ETM is Turing-recognizable. This means there exists a Turing Machine, let's call it **R**, that recognizes ETM. We will then show how we could use ***R*** to construct another Turing Machine, ***S***, that recognizes *HP*, contradicting the known fact that *HP* is not Turing-recognizable.

Assumption: We assume ETM is Turing-recognizable, and there exists a Turing Machine R that recognizes ETM.

Construction of S: We construct a Turing Machine S that takes as input a pair ⟨M, w⟩, where M is a TM and w is an input string, and does the following:
- S constructs a new Turing Machine M' that on any input x:
  - Simulates M on w.
  - If M halts on w, M' enters an infinite loop.
  - Otherwise, M' halts.
- S then runs R on ⟨ M ⟩
- If R accepts ⟨ M' ⟩, meaning L(M) = ∅, S accepts ⟨M, w⟩.
- If R rejects, S rejects.

Contradiction: The construction of S shows that if ETM were Turing-recognizable, then HP would also be Turing-recognizable because S effectively decides HP by recognizing when M halts on w. This contradicts the known fact that HP is not Turing-recognizable.

Conclusion: Therefore, our initial assumption that ETM is Turing-recognizable must be false. Hence, ETM is not Turing-recognizable.

This proof uses the concept of reduction, showing that if we could recognize ETM, we could also recognize HP, which is impossible. Thus, ETM cannot be Turing-recognizable.

4. Consider the following language L1:

$$\mathbf{L_1 = \{\langle M\rangle \mid M \text{ is a TM and } M \text{ accepts at least one string of form } 0{*}1{*}0{*}\}}$$

(a) Prove that L_1 is undecidable by showing a reduction from ATM to L_1.
(b) Prove the correctness of your reduction by explaining how the decider S for A_{TM} that you create in the reduction works, illustrating that S is indeed a decider for A_{TM}, and explaining why S always halts.

Solution (a)

To prove that the language *L_1 = {⟨M⟩ | M is a TM and M accepts at least one string of form 0∗1∗0∗}* is undecidable, we will show a reduction from *ATM* to *L_1*. *ATM* is the Acceptance Problem for Turing Machines, defined as ATM = { ⟨M, w⟩ | M is a TM and M accepts w }, which is known to be undecidable.

The strategy is to take an arbitrary instance of *ATM*, which is a pair `, and construct a Turing Machine *M'* such that *M'* accepts at least one string of the form 0*1*0* if and only if M accepts w. This construction will show that if we could decide L_1, we could also decide *ATM*, which is a contradiction since *ATM* is undecidable.

Construction of M'

Given an instance ⟨M, w⟩ of *ATM*, construct a Turing Machine M' that operates as follows on any input x:

1. Check the input format: M' first checks if x is of the form 0*1*0*. If not, M' rejects.
2. Simulate M on w: If x is of the correct form, M' then simulates M on input w.
   - If M accepts w, then M' accepts x.
   - If M rejects w or runs indefinitely, M' rejects x.

Proof of Reduction

- If M accepts w, then M' will accept any string of the form 0*1*0*, because it only needs to find one such string to accept, and the construction ensures it will accept if M accepts w. Thus, ⟨M, w⟩ in L_1.
- If M does not accept w, then M' will not accept any string of the form 0*1*0*, because it rejects or runs indefinitely on all inputs, following the behavior of M on w. Thus, ⟨ M' ⟩ $\notin$ L_1.

This reduction shows that if we had a decider for L_1, we could use it to decide ATM by constructing M' for any given ⟨M, w⟩ and checking if ⟨ M' ⟩ $\notin$ L_1. Since ATM is undecidable, this implies that L_1 must also be undecidable. The undecidability of L_1 follows directly from the undecidability of ATM and our reduction.

Solution (b)

Given an instance ⟨M, w⟩ of ATM, the reduction constructs a Turing Machine M' as described. The role of the hypothetical decider S is to decide whether ⟨M, w⟩ is in ATM by utilizing a decider for L1, which we assumed exists for the sake of contradiction.

For the given ⟨M, w⟩, S constructs M' as per the reduction. M' is designed to accept any string of the form 0*1*0 if and only if M accepts w. S then uses the assumed decider for L1 to determine whether ⟨ M' ⟩ is in L1. This is equivalent to checking if M' accepts at least one string of the form 0*1*0. If the decider for L1 determines that ⟨ M' ⟩ is in L1, S concludes that M accepts w and thus decides that ⟨M, w⟩ is in ATM. If the decider for L1 determines that ⟨ M' ⟩ is not in L1, S concludes that M does not accept w and thus decides that ⟨M, w⟩ is not in ATM.

The correctness of S hinges on the construction of M' and the behavior of the decider for L_1. By construction, M' is designed to accept strings of the form 0*1*0* if and only if M accepts w, ensuring that S's decision about ⟨M, w⟩ being in ATM is accurate.

S always halts because the construction of M' from ⟨M, w⟩ is a finite process, the use of the decider for L1 is crucial. By assumption, this decider always halts, providing a definitive answer about whether ⟨ M' ⟩ is in L1. Based on the decider for L1's answer, S makes a finite decision about ⟨M, w⟩ being in ATM. The hypothetical decider S for ATM, constructed through the reduction to L1, demonstrates that if L1 were decidable, then ATM would also be decidable. However, since ATM is known to be undecidable, this leads to a contradiction, reinforcing the correctness of the reduction and confirming that L1 is undecidable.

The key to my proof is the assumption that L1 is decidable, which allows for the construction of S, a decider for ATM, thereby leading to the contradiction since ATM is undecidable. This contradiction proves that our initial assumption (that L_1 is decidable) is false, thus establishing the undecidability of L1.

5. Consider the following language L2:

L2 = {⟨M⟩ | M is a TM and M accepts exactly 2 strings}

(a) Prove that L2 is undecidable by showing a reduction from ATM to L2.
(b) Prove the correctness of your reduction by explaining how the decider S for AT M that you create in the reduction works, illustrating that S is indeed a decider for AT M , and explaining why S always halts.

Solution:

To prove that the language *L2 = {⟨M⟩ | M is a TM and M accepts exactly 2 strings}* is undecidable, we will show a reduction from ATM to L2. Recall that ATM = {⟨M, w⟩ | M is a TM and M accepts w} is known to be undecidable.

The strategy is to take an arbitrary instance of ATM, which is a pair ⟨M, w⟩, and construct a Turing Machine M_2 such that M_2 accepts exactly 2 strings if and only if M accepts w. This construction will show that if we could decide L_2, we could also decide ATM, which is a contradiction since ATM is undecidable.

Given an instance ⟨M, w⟩ of ATM, construct a Turing Machine M2 that operates as follows:

1. On any input x: M_2 first checks if x is one of two special strings, say s_1 or s_2, where s_1 and s_2 are distinct and known in advance. If x is neither s_1 nor s_2, M_2 rejects.
2. If x = s1: M2 simulates M on input w.
   - If M accepts w, then M_2 accepts x.
   - If M rejects w or runs indefinitely, M_2 rejects x.
3. If x = s_2: M_2 accepts x unconditionally.

Proof of Reduction

- If M accepts w: Then M_2 will accept both s_1 and s_2, because it is designed to accept s_2 unconditionally and to accept s_1 if M accepts w. Thus, ⟨ M2 ⟩ ∈ L2.
- If M does not accept w: Then M2 will only accept s2, because it will reject s1 following the behavior of M on w. Thus, ⟨ M2 ⟩ ∉ L2, as it does not accept exactly 2 strings.

My reduction shows that if we had a decider for L2, we could use it to decide ATM by constructing M2 for any given ⟨M, w⟩ and checking if ⟨ M2 ⟩ ∈ L2. Since ATM is undecidable, this implies that L2 must also be undecidable. The undecidability of L2 follows directly from the undecidability of ATM and our reduction, demonstrating that no algorithm can decide L2.