Problem 1

Consider the following two languages. For each language, determine whether you can use Rice’s Theorem to prove it is undecidable. If so, use Rice’s Theorem to prove it is undecidable. If not, explain why you cannot use Rice’s Theorem, and prove it is undecidable without using Rice’s Theorem.

a. $L_{yuca} = \{\langle M \rangle \mid |L(M)| \geq 1\}$

b. $L_{eek} = \{\langle M \rangle \mid \langle M \rangle \in L(M)\}$

Solution:

1. In this case, you can use Rice’s Theorem to prove that $L_{yuca}$ is undecidable.

Rice’s theorem says that if a language $L$ consisting of descriptions of Turing Machines satisfies the following three conditions it is undecidable:

(a) $\langle M \rangle \in L$ and $L(M) = L(N) \Rightarrow \langle N \rangle \in L$.
(b) $\exists \langle M \rangle \in L$.
(c) $\exists \langle M \rangle$ not in $L$. 
LYUCA satisfies (1): Suppose \( \langle M \rangle \in L \) and \( L(M) = L(N) \). \(| L(M) | \geq 1 \) so \(| L(N) | \geq 1 \) so \( \langle N \rangle \in L \).

LYUCA satisfies (2): Let \( M \) be the TM that accepts immediately on all inputs. Thus \(| L(M) | \geq 1 \), and \( \langle M \rangle \) is in \( LYUCA \).

LYUCA satisfies (3): Let \( M \) be the TM that rejects immediately on all inputs. Thus, \(| L(M) | = 0 \), and \( \langle M \rangle \) is not in \( LYUCA \).

So \( LYUCA \) must be undecidable, by Rice’s Theorem.

2. In this case, you cannot use Rice’s Theorem. Two machines could have exactly the same language, but because of differences in implementation, one machine might accept its own source code, while the other rejected its own source code. Hence, Rice’s Theorem does not apply.

Proof that \( L_{E_{EF}} \) is not decidable:

(a) Assume, to the contrary, that \( L \) is decidable. Then there exists a decider \( D \) for \( L \) that takes \( M \) as input and returns true if and only if \( M \) accepts itself. Using \( D \), we construct a TM \( A \) that takes \( \langle M, w \rangle \) as input, and does the following:

- Construct a TM \( M' \) that ignores its input and runs \( M \) on \( w \) instead.
- Runs \( D \) on \( M' \) and returns whatever \( D \) outputs.

\( A \) is a decider because it consists of steps that take finite length. (The modification \( M' \) just requires a few states to be added at the beginning of \( M \) to remove the current contents of the tape and replace them with \( w \), and \( D \) is a decider.) This is what \( A \) does:

- If \( M \) accepts \( w \), then \( M' \) will always accept, because \( M' \) simply runs \( M \) on \( w \) and returns the result. This is true even when \( M' \) is passed \( \langle M' \rangle \) as input. Hence, \( D \) will accept \( M' \), and so \( A \) will accept as well.
- If \( M \) does not accept \( w \), then \( M' \) will never accept. Therefore, \( M' \) does not accept itself, and so \( D \) will reject \( M' \). Therefore, \( A \) will reject as well.

In other words, \( A \) accepts if and only if \( M \) accepts \( w \). Hence, \( A \) is a decider for \( A_{TM} \), which we know to be undecidable, and so we have a contradiction. Hence, \( L \) is not decidable.
Problem 2

Explain why each of the following languages is decidable:

a. \( L = \{ \langle M, w \rangle \mid \text{when } M \text{ is run on } w, \text{ it only moves left} \} \)

b. \( L = \{ \langle M, w \rangle \mid \text{when } M \text{ is run on } w, \text{ the head reverses direction at least once} \} \)

Note: When a Turing machine moves left at the start of the tape, it just stays at the start of the tape.

Hint: Consider how you would detect an infinite loop, given that you haven’t changed direction.

Solution:

1. To decide \( L \), construct a table with each entry corresponding to a pair consisting of a state and a tape symbol. Then begin to simulate \( M \) on \( w \). As long as you keep moving left, you will stay on the first character of the tape and keep writing it. Every time you see a character in a particular state, mark that pair as visited. Because there are a finite number of symbols and a finite number of states, there are a finite number of \((\text{state}, \text{character})\) combinations, and so within a finite amount of time you must either revisit a \((\text{state}, \text{character})\) combination, move to the right, or halt. If the former occurs, then you know that you’re in an infinite loop, and can halt and accept. If the machine halts, then you can accept, since it has only moved left. And if you ever move right, reject.

2. (Proof that is the same as part (a) but only moving right) To decide \( L \), simulate \( M \) on \( w \). As soon as you follow a transition to the left, halt and reject. Since you have to keep moving to the right, you must eventually either halt (in which case you accept) or reach blank tape, at which point you will never read any non-blank characters anymore. At that point, start marking the states of \( M \) as visited. Because there are a finite number of TM states, you must eventually either halt (in which case you accept) or visit a state for a second time, at which point you know that the TM is in an infinite loop — because you are only reading blanks — and so you can halt and accept.

3. (Proof of part (b)) To see that \( L \) is decidable, note that we can easily combine the results from parts (a) and (b) of this problem. That is,
if we have a TM A which decides part a, and a TM B which decides part b, if we know that M doesn’t always go right and M also doesn’t always go left, then it must have switched directions. So construct a decider D which takes input M and runs A on M and B on M. If both A and B reject, then accept; else reject.

Problem 3

Consider the following language:

\[ L = \{ \langle M, k \rangle \mid M \text{ is a TM that halts on at least } k \text{ inputs} \} \]

Is this language Turing-recognizable? If so, is it decidable? Prove your answers. (Hint: you may want to use nondeterminism.)

Solution:

Yes, the language is Turing-recognizable. Consider an NTM that nondeterministically selects \( k \) inputs, then runs \( M \) on all of those inputs and accepts if they all halt. If there are at least \( k \) inputs that halt, then there is at least one accepting branch of the computation, and so we have constructed an NTM that will recognize the language.

However, the language is not decidable. Assume, to the contrary, that it is. Then there exists a decider \( D \) for this language. Construct a machine \( H \) using \( D \) that takes input \( \langle M, w \rangle \) and does the following:

Modify \( M \) to create \( M' \), which first deletes the current input and overwrites it with \( w \), and then runs \( M \). (In other words, no matter the input, \( M' \) runs \( M \) on \( w \).) Then run \( D \) on \( \langle M', 1 \rangle \), and return the result.

We claim that \( H \) solves the halting problem. Note that if \( M \) halts on \( w \), then \( M' \) halts on all inputs, and so \( D \) would accept \( M' \) no matter what \( k \) is selected. On the other hand, if \( M \) does not halt on \( w \), then \( M' \) loops on all inputs and halts on none of them, and so \( D \) would reject \( M' \). Therefore, \( H \) accepts \( \langle M, w \rangle \) if and only if \( M \) halts on \( w \). In other words, \( H \) decides the halting problem. However, the halting problem is undecidable, and so we have a contradiction. Thus, our assumption that \( L \) is decidable must be false, making \( L \) undecidable.