Problem 1

A universal Turing machine is one that simulates another Turning machine given its definition. This is not unusual, we use compilers every day, and we have all probably compiled a C program using a compiler written in C. Assume we have a Turing machine $M$ which is given as input a triple $(T, x, n)$. $T$ is a Turing machine, $x$ is an valid input to $T$, and $n$ is an integer. Write a high level description of $M$ which decides the following language:

$L = \{\langle T, x, n \rangle \mid T \text{ is a turing machine and } T \text{ accepts } x \text{ within } n \text{ steps} \}$

Idea

$L$ places an upper bound $n$ on the number of steps a machine may run before it accepts. So, we can simply run $T$ on $x$ for $n$ steps and accept if and only if $T$ accepts in that time: if $T$ accepts on $x$ within $n$ steps, $\langle T, x, n \rangle \in L$, otherwise, it is not.

High level description

$M$ : "on input $\langle T, x, n \rangle$

1. Run $T$ on $x$ for $n$ steps.

   a. If, after $n$ steps $T$ accepts, accept;

   b. Otherwise, reject."
Problem 2

Part A

Consider the problem

\[ A = \{ (M, w) | M \text{ moves left at some point during its computation on } w \} \]

Show that \( A \) is decidable.

Proof Idea

Suppose that a machine \( M \) has \( n \) states. If \( M \) is started on an input \( w \), then in a finite number of steps (equal the length \( n \) of the input \( w \)) we can determine if \( M \) moves left while reading \( w \). As \( M \) moves to the right of \( w \), reading only blanks, it can move right only at most \( m \) times without looping. This is because each state has a single instruction for what to do when reading a \( B \). So, if \( M \) reads a \( B \) \( m + 1 \) times then, by the pigeonhole principle, it must enter one of its states at least twice.

This means that, in \( m + n + 1 \) moves, we can determine if \( M \) moves left at least once.

Proof. In order to show that \( A \) is decidable, we will construct a machine \( N \) and prove that \( N \) decides \( A \). \( N \) will take as input a machine-input pair \( < M, w > \), such that the number of \( M \)'s states is \( m \) and the length of \( w \) is \( n \).

Define \( N \): "on input \( < M, w > \)

1. Run \( M \) on \( w \) for \( n \) steps.
2. If, during any one of those steps, \( M \) moves left, accept.
3. Continue running \( M \) on \( w \) for \( m + 1 \) steps.
4. If, during any one of those steps, \( M \) moves left, accept.
5. Else reject.

\( N \) decides \( A \):

Since \( N \) always halts after a finite number of steps in either an accepting or rejecting state, there are only two cases we need to consider. First, suppose \( N \) accepts on input \( < M, w > \). Then \( M \) moves left in either \( n \) steps or \( m + n + 1 \) steps. Therefore, \( M \) moves left on \( w \) and \( < M, w > \in A \).

Suppose \( N \) rejects on \( < M, w > \). Then \( M \) does not move left on \( w \) in \( m + n + 1 \) steps. If \( M \) does not move left in \( n \) steps, then \( C_n = s_1 s_2 \ldots s_n q_i \), for some \( i \) (i.e. at step \( n \), \( M \) is reading the first blank after the end of the input string).

For any \( q_j \in Q \), there is exactly one instruction of the form:

\[ < q_j, B > \to < q_k, a, d > \] (otherwise \( M \) would have conflicting instructions).

Further, if \( M \) does not move left in \( p = m + n + 1 \) steps, then \( C_p = s_1 s_2 \ldots s_n \ldots s_p q_r \), for some \( r \), where \( p - n = m + 1 \). Since \( M \) has only \( m \) states, each state has exactly one instruction for reading \( B \), and \( M \) reads only \( B \)'s from \( s_n \) to \( s_p \), by the pigeonhole principle \( M \) must repeat some state \( q_e \) in some \( C_x \), for \( n \leq x \leq p \). Suppose without loss of generality, that \( r = e \), that is, that the state \( M \) enters in \( C_p \) is the repeated state. Because \( M \) is reading a \( B \) in \( C_p \), and we know that \( M \) moves right when it reads a \( B \) in state \( q_e \), \( C_y = s_1 s_2 \ldots s_y q_e \), for any \( x \geq p \). Less formally, we know that \( M \) has entered an infinite loop and will move right forever.

Therefore, \( < M, w > \notin A \). \( \square \)
Part B
Consider the problem

\[ B = \{ \langle M, w \rangle | M \text{ moves left at least 4 times during its computation on } w \} \]

Is B decidable? Why or why not?

Answer

B is decidable.

Proof Idea

From Part B, we know that if \( M \) moves left at all, it will move left in \( m + n + 1 \) steps. We will place an upper bound on the size of the string to the right of the tapehead after \( p \) left moves. This will put an upper bound on the number of steps that must be checked to determine if \( M \) moves left \( p + 1 \) times. (As in Part A, \( q \) is the number of \( M \)'s states, and \( n \) the length of its input.)

Proof. Suppose \( M \) moves left \( p \) number of times, for arbitrary \( p \). \( M \) is, then, in some configuration \( C_h = s_1s_2\ldots q_t s_j \ldots s_k \), where \( s_k \) is the last non-blank symbol before the infinite, unbroken string of \( B \)'s to the right. Since for any \( k \) and \( j \), \( k - j \) is finite, for the \( p^{th} \) left move there will be a finite string \( S_p \) of symbols to right of the tapehead after the move.

By the reasoning in Part A, in order to determine whether or not \( M \) makes a \( p + 1^{th} \) left move, we must to run \( M \) at most \( |S_p| + m + 1 \) steps. So, in general, in order to determine if \( M \) makes \( p \) left moves, we must run \( M \) at most

\[
\sum_{i=0}^{p} |S_i| + (m + 1) \text{ steps.}
\]

Where \( |S_i| \) = the length of \( S_i \), and \( |S_0| \) = the length of the input. Therefore, in order to determine if \( M \) makes four left moves, we must run \( M \) at most

\[
(|S_0| + m + 1) + (|S_1| + m + 1) + (|S_2| + m + 1) + (|S_3| + m + 1) = \text{ steps.}
\]

Since each of the \( S_i \)'s is finite in length, there is a finite upper bound on the number of steps it takes to determine if \( M \) makes four left moves. Therefore, \( B \) is decidable. \( \square \)
Problem 3

A language, $L_1$, can be concatenated with another language, $L_2$ (denoted $L_1 \circ L_2$) as follows:

$$s \in L_1 \circ L_2 \iff s = s_1 \circ s_2 \text{ for some } s_1 \in L_1 \text{ and some } s_2 \in L_2$$

We can repeat this construction to create $L_1 \circ L_2 \circ L_3$

Prove that if $L_1$ and $L_2$ are decidable then $L_1 \circ L_2 \circ \overline{L_1}$ is decidable. ($\overline{L_1}$ is the compliment of $L_1$.)

Proof Idea

In order to determine if a string $s = w_1 \circ w_2 \circ w_3 \in L_1 \circ L_2 \circ \overline{L_1}$, we must determine if there is some way of dividing $s$ into $w_1$, $w_2$, and $w_3$ such that $w_1 \in L_1$, $w_2 \in L_2$, and $w_3 \in \overline{L_1}$. In order to make this determination, we must test every possible way of dividing $s$ into three consecutive strings. Since there are only a finite number of ways of making this division, there are only a finite number of checks to do. Therefore, $L_1 \circ L_2 \circ \overline{L_1}$ is decidable.

Proof. Because $L_1$ and $L_2$ are decidable, there exist machines $M_1$ and $M_2$ that decide $L_1$ and $L_2$ respectively.

Define $M_{10201}$: on input $\langle w \rangle$:

Let $n = |w|$, or, the length of $w$ and let $w[h-k]$ be the substring of $w$ consisting of the $h$th symbol through the $k$th symbol.

1. For $i = 0$,
   a. For $j = 0$,
      i. Partition $w$ into $x = w[0-i]$, $y = w[i+1-k]$, and $z = w[k+1-n]$.
      ii. Simulate $M_1$ on $x$.
      If $M_1$ accepts, then, or
      iii. Simulate $M_2$ on $y$.
      If $M_2$ accepts, then
      iv. Simulate $M_1$ on $z$.
      If $M_1$ rejects, accept.

2. Reject.

Each input can be split into 3 pieces a finite number of ways, and thus $M_{10201}$ has a finite number possible checks for $s$. Because each partition only requires simulating 3 machines, each of which is a decider, the computation for each has a finite time length. If $M_1$ accepts $x$, $M_2$ accepts $y$, and $M_1$ rejects $z$, this indicates that a partition $w_1w_2w_3$ has been found such that $w_1 \in L_1$, $w_2 \in L_2$, and $w_3 \in \overline{L_1}$. If one of these is not the case, such a partition has not been found. If any partition is found, the machine accepts, otherwise it rejects. Thus, this machine is a decider for $L_1 \circ L_2 \circ \overline{L_1}$. 

\[\square\]
Problem 4
Part A
Prove that the collection of decidable languages is closed under set difference.

Proof Idea
Given two decidable sets, $A$ and $B$, the set difference $A - B$ is the set of things in $A$ and not in $B$. Since $B$ is decidable, $\overline{B}$ is also decidable. Since $A$ and $\overline{B}$ are decidable, $A \cap \overline{B}$ is decidable, and $A \cap \overline{B} = A - B$. Therefore, $A - B$ is decidable.

Proof

Lemma 1: If $B$ is decidable, $\overline{B}$ is decidable.

If $B$ is decidable, then there is some machine $N$ which decides it- that is, accepts on input $w$ if $w \in B$, and rejects on $w$ if $w \notin B$. Construct a machine $N'$ such that $N'$ accepts if $N$ rejects, and $N'$ rejects if $N$ accepts. $N'$ decides $\overline{B}$.

Lemma 2: If $A$ and $B$ are decidable, then $A \cap B$ is decidable.

If $A$ is decidable and $B$ is decidable, then there are machines $N_A$ and $N_B$ which decide them. Construct a machine $N$ such that for input $w$, $N$ runs $N_A$ and $N_B$ on input $w$. If both $N_A$ and $N_B$ accept, then $N$ accepts. Otherwise, $N$ rejects. $N$ decides $A \cap B$.

Suppose that sets $A$ and $B$ are decidable. By Lemma 1, $\overline{B}$ is decidable. So, by Lemma 2, $A \cap \overline{B}$ is decidable. Since $A \cap \overline{B} = A - B$, the set difference $A - B$ is decidable.
Part B
Is the same true for computably enumerable (Turing-recognizable) languages?

Solution
The set of Turing-recognizable languages is not closed under set difference.

Proof idea
Take the difference of $T$ (the set of all Turing machines) and $A_{TM}$. If the set of Turing-recognizable languages were closed under set difference, then this set difference would be recognizable. However, then the set $A_{TM}$ would be decidable as we could always establish whether something was a Turing machine and in $A_{TM}$ or a Turing machine and not in $A_{TM}$. Since $A_{TM}$ is not decidable, the set of Turing-recognizable language is not closed under set difference.

Proof
[Proof by contradiction] Suppose the set of Turing-recognizable languages were closed under set difference. Let $T$ be the set of Turing machines, and $A_{TM}$ be the set of Turing machines $M$ that accept on input $< M >$. $T$ is decidable and, therefore, recognizable. $A_{TM}$ is recognizable. So, by supposition, $T - A_{TM}$ is recognizable. Let $N_A$ be some machine that recognizes $A_{TM}$, and $N_D$ some machine that recognizes $T - A_{TM}$. Define a machine $N$ as follows:

Define $N$: "on input $w$
1. Set $i$ equal to 0
2. Repeat the following 5 steps:
   2a. Run $N_A$ on $w$ for $i$ steps.
   2b. Run $N_D$ on $w$ for $i$ steps.
   2c. If $N_A$ accepts, accept.
   2d. If $N_D$ accepts, reject.
   2e. Increment $i$, return to step 2a."

$N$ decides $A_{TM}$. Since $A_{TM}$ is not decidable, the set of Turing-recognizable languages is not closed under set difference.
Problem 5

Consider the language, $L$, defined below:

$$L = \{ \langle M, x \rangle \mid M \text{ halts on input } x \text{ and during the computation it enters at least 10 different non-halting states at least once.} \}$$

Prove that $L$ is undecidable.

Proof idea

[Proof by contradiction] We suppose that $L$ is decidable, and infer that there is a machine $M_L$ that decides it. If $M_L$ accepts on $\langle M, x \rangle$, that means that $M$ halts on $x$ after entering at least 10 non-halting states. If $M_L$ rejects on $\langle M, x \rangle$, that means that either $M$ did not halt on $x$, or that $M$ did not enter 10 non-halting states.

We construct a machine $M_{10}$ that runs through 10 non-halting states without changing the input, and use $M_{10}$ and $M_L$ to decide $\text{HALT}$.

To decide $\text{HALT}$, we must determine for arbitrary $N$ and $w$ whether $M$ halts on $x$. By 'hooking' $M_{10}$ to $N$, we construct the machine $M_{10} \rightarrow N$. We then run $M_L$ on input $\langle M_{10} \rightarrow N, w \rangle$. Since we know that $M_{10} \rightarrow N$ enters 10 non-halting states, if $M_L$ accepts, it means that $N$ halts on $w$. Likewise, if $M_L$ rejects, it means that $N$ does not halt on $w$. Therefore, we have a general procedure for deciding $\text{HALT}$.

Proof. Suppose $L$ is decidable. Then there is some machine $M_L$ which decides it.

Define $M_{10}$: "on input $w$

1. Start in state $q_{1'}$.
2. Transition into $q_{2'}$, do not change the tape, and move right.
3. Transition into $q_{3'}$, do not change the tape, and move left.
4. Transition into $q_{4'}$, do not change the tape, and move right.
5. Transition into $q_{5'}$, do not change the tape, and move left.
6. Transition into $q_{6'}$, do not change the tape, and move right.
7. Transition into $q_{7'}$, do not change the tape, and move left.
8. Transition into $q_{8'}$, do not change the tape, and move right.
9. Transition into $q_{9'}$, do not change the tape, and move left.
10. Transition into $q_{10'}$, do not change the tape, and move right.
11. Transition into $q_{11'}$, do not change the tape, and move left.

For arbitrary $N$, construct a machine $M_{10} \rightarrow N$ by setting $M_{10}$’s accepting state to $N$’s start state. So, $M_{10} \rightarrow N$’s start state is $M_{10}$’s start state, and $M_{10} \rightarrow N$’s accepting/rejecting states are the same as $N$’s. Also, since $M_{10}$ halts with its tapehead at the beginning of the tape without changing the input, $M_{10} \rightarrow N$ is equivalent to $N$.

Run $M_L$ on input $\langle M_{10} \rightarrow N, w \rangle$ for arbitrary $w$. Since $M_L$ is a decider, it either accepts or rejects on $\langle M_{10} \rightarrow N, w \rangle$.

Suppose $M_L$ accepts on $\langle M_{10} \rightarrow N, w \rangle$. Then $M_{10} \rightarrow N$ enters 10 non-halting states and halts on $w$. Since $M_{10} \rightarrow N$ is equivalent to $N$, $N$ halts on $w$. So, $\langle N, w \rangle \in \text{HALT}$.
Suppose $M_L$ rejects on $< M_{10} \rightarrow N, w >$. Then $M_{10} \rightarrow N$ either does not enter 10 non-halting states, or does not halt on $w$. By construction, $M_{10} \rightarrow N$ enters 10 non-halting states. Therefore, $M_{10} \rightarrow N$ does not halt on $w$. Since $M_{10} \rightarrow N$ is equivalent to $N$, $N$ does not halt on $w$. So, $< N, w > \notin HALT$.

Define $M_{HALT}$: "on input $< N, w >$

1. Construct $M_{10} \rightarrow N$.
2. Run $M_L$ on $< M_{10} \rightarrow N, w >$.
3. If $M_L$ accepts, accept.
4. If $M_L$ rejects, reject.

$M_{HALT}$ decides $HALT$. Since $HALT$ is not decidable and $M_{HALT}$ is constructed from $M_L$, $L$ is not decidable.