Computability and Complexity (Summer 2025) – Homework 1

Shir Altman, Yedidya Even-chen

August 2025

Problem 1

Build (formally, with a program) a Turing Machine (you can use the type of standard TM you prefer: 1-tape, k-tape, deterministic or non-deterministic):

Part 1a

Computing the arithmetic addition $f(x, y) = x + y$, where $x, y \in \{0, 1\}$.

Solution: We use a 3-tape DTM and assume that the input is given in from left to right (LSB on the left), and that the input is given with x on tape 1 and y on tape 2.
We progress over tape 1 and tape 2, performing binary addition with tape 3 as the output. We have states for 0-carry and 1-carry. On tapes 1 and 2, we only read; on tape 3, we only write. On all 3 tapes, we only move right.
If on one input tape we read a blank space (when x and y are not the same length), we treat it as 0. If on both input tapes we read a blank space, we halt.
The δ function is defined below. In all the transitions in the table, the movement is $\rightarrow$ (right) on all 3 tapes.

$$\delta(q_{\text{start}}, \triangleright, \triangleright, \triangleright) \coloneqq (q_0, \triangleright, \triangleright, \triangleright, \rightarrow, \rightarrow, \rightarrow)$$

	$0,0,\_$	$0,1,\_$	$1,0,\_$	$1,1,\_$	$\_,0,\_$	$\_,1,\_$	$0,\_,\_$	$1,\_,\_$
q_0	$q_0,0,0,0$	$q_0,0,1,1$	$q_0,1,0,1$	$q_1,1,1,0$	$q_0,\_,0,0$	$q_0,\_,1,1$	$q_0,0,\_,0$	$q_0,1,\_,1$
q_1	$q_0,0,0,1$	$q_1,0,1,0$	$q_1,1,0,0$	$q_1,1,1,1$	$q_0,\_,0,1$	$q_1,\_,1,0$	$q_0,0,\_,1$	$q_1,1,\_,0$

$$\delta(q_1, \_, \_, \_) \coloneqq (q_{\text{halt}}, \_, \_, \_, \circ, \circ, \circ), \qquad \delta(q_0, \_, \_, \_) \coloneqq (q_{\text{halt}}, \_, \_, \_, \circ, \circ, \circ)$$

Part 1b

Deciding $L_1 \coloneqq \{x \in \{0, 1\}^* : x \text{ contains } 001 \text{ as a substring}\}$.

Solution: We use a 1-tape DTM. States:

- q_0: looking for first 0
- q_1: looking for second 0
- q_2: looking for the 1

The starting state is q_{start}, the accepting state is q_{Y}.

$$\delta(q_{\text{start}}, \triangleright) \coloneqq (q_0, \triangleright, \rightarrow)$$

	0	1
q_0	$q_1, 0, \rightarrow$	$q_0, 1, \rightarrow$
q_1	$q_2, 0, \rightarrow$	$q_0, 1, \rightarrow$
q_2	$q_1, 0, \rightarrow$	$q_{\text{Y}}, 1, \circ$

Part 1c

Computing $Ax \oplus b$ where A is a binary $n \times n$ matrix, and x and b are n-long binary vectors

Solution: We use a 4-tape DTM. We assume that the input is given with A on tape 1, x on tape 2, b on tape 3. And that the encoding of A is by rows in succession.

Computation overview:

1. Read from tape 1 and tape 2; write to tape 4: $0,0 \to 0$, $0,1 \to 1$, $1,0 \to 1$, $1,1 \to 1$.
2. After reaching end of x, go back over tape 4 and compute $\oplus$, erasing as we go. The result is written to next empty cell on tape 4.
3. Repeat from step 1 until end of A.
4. At this point, tape 4 holds Ax.
5. Go over tape 3 and tape 4, computing $\oplus$ and writing the result on tape 4.

The definition of δ: We start in q_{start}, and transition to q_{Ax}:

$$\delta(q_{\text{start}}, \triangleright, \triangleright, \triangleright, \triangleright) \coloneqq (q_{Ax}, \triangleright, \triangleright, \triangleright, \triangleright, \rightarrow, \rightarrow, \circ, \rightarrow)$$

In q_{Ax}, if we have one or two 0's then we write 0. If we have two 1's then we write 1:

$$\begin{aligned}
\delta(q_{\text{start}}, \triangleright, \triangleright, \triangleright, \triangleright) &\coloneqq (q_{Ax}, \triangleright, \triangleright, \triangleright, \triangleright, \rightarrow, \rightarrow, \circ, \rightarrow) \\
\delta(q_{Ax}, 0, 0, \triangleright, _) &\coloneqq (q_{Ax}, 0, 0, \triangleright, 0, \rightarrow, \rightarrow, \circ, \rightarrow) \\
\delta(q_{Ax}, 0, 1, \triangleright, _) &\coloneqq (q_{Ax}, 0, 1, \triangleright, 0, \rightarrow, \rightarrow, \circ, \rightarrow) \\
\delta(q_{Ax}, 1, 0, \triangleright, _) &\coloneqq (q_{Ax}, 1, 0, \triangleright, 0, \rightarrow, \rightarrow, \circ, \rightarrow) \\
\delta(q_{Ax}, 1, 1, \triangleright, _) &\coloneqq (q_{Ax}, 1, 1, \triangleright, 1, \rightarrow, \rightarrow, \circ, \rightarrow)
\end{aligned}$$

When we reach the end of x, if it is not the end of tape 1, we go back to the start of tape 2 and tape 4, computing the $\oplus$ of tape 4. For $\sigma, \sigma' \in \{0,1\}$, write:

$$\begin{aligned}
\delta(q_{Ax}, \sigma', _, \triangleright, _) &\coloneqq (q_{\text{xor-4}}, \forall, _, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4}}, \forall, \sigma, \triangleright, 0) &\coloneqq (q_{\text{xor-4-0}}, \forall, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4}}, \forall, \sigma, \triangleright, 1) &\coloneqq (q_{\text{xor-4-1}}, \forall, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-0}}, \forall, \sigma, \triangleright, 0) &\coloneqq (q_{\text{xor-4-0}}, \forall, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-0}}, \forall, \sigma, \triangleright, 1) &\coloneqq (q_{\text{xor-4-1}}, \forall, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-1}}, \forall, \sigma, \triangleright, 0) &\coloneqq (q_{\text{xor-4-1}}, \forall, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-1}}, \forall, \sigma, \triangleright, 1) &\coloneqq (q_{\text{xor-4-0}}, \forall, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow)
\end{aligned}$$

When we reach the start of tape 2, we write the result of the $\oplus$,

$$\begin{aligned}
\delta(q_{\text{xor-4-0}}, \forall, \triangleright, \triangleright, \forall) &\coloneqq (q_{\text{write-0}}, \forall, \triangleright, \triangleright, \forall, \circ, \circ, \circ, \rightarrow) \\
\delta(q_{\text{xor-4-1}}, \forall, \triangleright, \triangleright, \forall) &\coloneqq (q_{\text{write-1}}, \forall, \triangleright, \triangleright, \forall, \circ, \circ, \circ, \rightarrow) \\
\delta(q_{\text{write-0}}, \forall, \triangleright, \triangleright, _) &\coloneqq (q_{Ax}, \forall, \triangleright, \triangleright, 0, \circ, \rightarrow, \circ, \rightarrow) \\
\delta(q_{\text{write-1}}, \forall, \triangleright, \triangleright, _) &\coloneqq (q_{Ax}, \forall, \triangleright, \triangleright, 1, \circ, \rightarrow, \circ, \rightarrow)
\end{aligned}$$

and continue in Ax.

If we reached a blank space on tape 1 and tape 2, we finished x and A. We need to compute the $\oplus$ of tape 4 one last time:

$$\begin{aligned}
\delta(q_{Ax}, _, _, \triangleright, _) &\coloneqq (q_{\text{xor-4-last}}, _, _, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-last}}, _, \sigma, \triangleright, 0) &\coloneqq (q_{\text{xor-4-0-last}}, _, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-last}}, _, \sigma, \triangleright, 1) &\coloneqq (q_{\text{xor-4-1-last}}, _, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-0-last}}, _, \sigma, \triangleright, 0) &\coloneqq (q_{\text{xor-4-0-last}}, _, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-0-last}}, _, \sigma, \triangleright, 1) &\coloneqq (q_{\text{xor-4-1-last}}, _, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-1-last}}, _, \sigma, \triangleright, 0) &\coloneqq (q_{\text{xor-4-1-last}}, _, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-1-last}}, _, \sigma, \triangleright, 1) &\coloneqq (q_{\text{xor-4-0-last}}, _, \sigma, \triangleright, _, \circ, \leftarrow, \circ, \leftarrow) \\
\delta(q_{\text{xor-4-0-last}}, _, \triangleright, \triangleright, _) &\coloneqq (q_{\text{write-0-last}}, _, \triangleright, \triangleright, _, \circ, \circ, \circ, \rightarrow) \\
\delta(q_{\text{xor-4-1-last}}, _, \triangleright, \triangleright, _) &\coloneqq (q_{\text{write-1-last}}, _, \triangleright, \triangleright, _, \circ, \circ, \circ, \rightarrow) \\
\delta(q_{\text{write-0-last}}, _, \triangleright, \triangleright, _) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, \triangleright, 0, \circ, \circ, \circ, \leftarrow) \\
\delta(q_{\text{write-1-last}}, _, \triangleright, \triangleright, _) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, \triangleright, 1, \circ, \circ, \circ, \leftarrow)
\end{aligned}$$

In $q_{Ax\oplus b}$ state, we first want to go to the start of tape 4. For $\sigma \in \{0, 1\}$, write:

$$\delta(q_{Ax\oplus b}, \_, \triangleright, \triangleright, \sigma) \coloneqq (q_{Ax\oplus b}, \_, \triangleright, \triangleright, \sigma, \circ, \circ, \circ, \leftarrow)$$

When we reach the start of tape 4, we want to compute its $\oplus$ with tape 3. We'll write the result on tape 4:

$$\begin{aligned}
\delta(q_{Ax\oplus b}, _, \triangleright, \triangleright, \triangleright) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, \triangleright, \triangleright, \circ, \circ, \rightarrow, \rightarrow) \\
\delta(q_{Ax\oplus b}, _, \triangleright, 0, 0) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, 0, 0, \circ, \circ, \rightarrow, \rightarrow) \\
\delta(q_{Ax\oplus b}, _, \triangleright, 0, 1) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, 0, 1, \circ, \circ, \rightarrow, \rightarrow) \\
\delta(q_{Ax\oplus b}, _, \triangleright, 1, 0) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, 1, 1, \circ, \circ, \rightarrow, \rightarrow) \\
\delta(q_{Ax\oplus b}, _, \triangleright, 1, 1) &\coloneqq (q_{Ax\oplus b}, _, \triangleright, 1, 0, \circ, \circ, \rightarrow, \rightarrow)
\end{aligned}$$

When we reach the end of tape 3 and tape 4, halt:

$$\delta(q_{Ax\oplus b}, \_, \triangleright, \_, \_) \coloneqq (q_{\text{halt}}, \_, \triangleright, \_, \_, \circ, \circ, \circ, \circ)$$

Problem 2

Prove or refute the equivalency of the following model to a standard TM:

Part 2a

A 1-tape TM where the tape is infinite to both sides.

Solution: A 1-tape TM where the tape is infinite to both sides (we'll call it a *double-sided TM*, or DSTM) is equivalent to a standard TM.

DSTM $\rightarrow$ TM: We can construct a 4-tape TM that, when given a DSTM, simulates an equivalent TM. Assume that the DSTM is given as a binary string on tape 1 and find the part that represents the input string. Move to the starting position.

1. From the starting position, move left, copying the contents of the input tape of the DSTM to tape 3.
2. Reverse tape 3.
3. Return to the starting position of tape 1 and copy the rest (to the right) to tape 2.

Now tape 2 and tape 3 hold the two halves of the original tape, split at the starting position. Tape 1 holds the encoding of the DSTM.
Proceed to execute the commands from the given DSTM. Since the DSTM starts with the read/write head on the part that is now on tape 2, the commands ignore tape 3.
If a command sends the read/write head to the beginning of tape 2 (meaning it should have moved to the part that is now on tape 3) we switch to states that ignore tape 2.
If, from these states, the read/write head gets sent to the start of tape 3 (meaning it should have moved to the part that is now on tape 2) we switch back to the commands that ignore tape 3.
At the end of execution, we copy the content of tape 3 (reversed) to tape 4, and concatenate the contents of tape 2 to tape 4. This simulates the final output of the DSTM.
As any k-tape TM is equivalent to a standard TM, this proves that a DSTM can be translated to a standard TM.

TM $\rightarrow$ DSTM: Any TM can be thought of as a DSTM, in which we only have input written to the right of the starting position and no moves send the read/write head to the left of the starting position.

Part 2b

A 1-tape TM with a finite tape.

Solution: A 1-tape TM with a finite tape (we'll call it a *finite-tape TM*, or FTTM) is not equivalent to a standard TM, as the finite tape limits its power. Any FTTM with a tape of length m, can only take inputs of length m. Consider the language:

$$L \coloneqq \{0^n 1^n : n \in \mathbb{N}\}$$

We know that it can be decided by a standard TM. But an FTTM with a tape of length $2m$ will fail to handle any word $0^n 1^n$ with $n > m$.

Part 2c

A standard machine without a "stay" move (i.e. on every step, the head always moves to the left or to the right).

Solution: A standard machine without a "stay" move (we'll call it a *no-stay TM*, or NSTM) is equivalent to a standard TM. We'll assume that it has 1 tape.

TM $\rightarrow$ NSTM: Given a TM, we can construct an equivalent NSTM. For every command that has "stay":

$$\delta(q, \sigma) \coloneqq (\hat{q}, \hat{\sigma}, \circ)$$

we add the state $\hat{q}'$, and replace the command with two commands:

$$\delta(q, \sigma) \coloneqq (\hat{q}', \hat{\sigma}, \rightarrow)$$
$$\delta(\hat{q}', \forall) \coloneqq (\hat{q}, \forall, \leftarrow)$$

Every other command is unchanged.

NSTM $\rightarrow$ TM: An NSTM is just a regular TM that doesn't use the "stay" move in any command.

Problem 3

Suggest an alternative encoding of a Turing machine using binary rather than unary representation.

Solution: Similar to the encoding shown in the lecture, we can encode each of the numbers: $|\Gamma|, |Q|, k$ in binary, with the # separating them. Every state, character, and direction gets a serial number, and this number is encoded in binary. Every command is then the encodings of a state, char, state, char, direction, with #s separating them.

Problem 4

For some language $L \subseteq \Sigma^*$:

Part 4a

Prove or refute that if $L_1 \in \mathrm{R}$ and $L_2 \in \mathrm{R}$, then $L_1 \setminus L_2 \in \mathrm{R}$.

Solution: True.

Since $L_1, L_2 \in \mathrm{R}$, there exist Turing Machines M_1 and M_2 that are deciders for L_1 and L_2, respectively. This means they halt on all inputs. We construct a new TM that we define as M_{diff}, to decide if $L_1 \setminus L_2 \in \mathrm{R}$. Steps for M_{diff}:

1. Run M_1 on input w.
2. If M_1 rejects w, then $w \notin L_1$, so M_{diff} rejects w and halts.
3. If M_1 accepts w, then $w \in L_1$. Proceed to the next step.
4. Run M_2 on input w.
5. If M_2 accepts w, then $w \in L_2$. Since w is both in L_1 and L_2, M_{diff} rejects w and halts.
6. If M_2 rejects w, then $w \notin L_2$.Therefore $w \in L_1 \setminus L_2$. M_{diff} accepts w and halts.

Since M_1 and M_2 are deciders, they always halt. Therefore, M_{diff} will halt on any input w, making it a decider. We can conclude that $L_1 \setminus L_2 \in \mathrm{R}$.

Part 4b

For the language $\mathrm{UPF}(L) \coloneqq \{y \in \Sigma^* \mid \forall x \in \Sigma, xy \notin L\}$, prove or refute that:

$$L \in \mathrm{coRE} \iff \mathrm{UPF}(L) \in \mathrm{RE} \qquad \text{and} \qquad L \in \mathrm{R} \iff \mathrm{UPF}(L) \in \mathrm{R}$$

Solution: We will address each direction separately:

$L \in \text{R} \implies \text{UPF}(L) \in \text{R}$ is true. If $L \in \text{R}$, then there exists some TM M deciding L. Define M' on input y: For all $x \in \Sigma$ run $M(xy)$. If all outputs are 0, output 1. Else, output 0. M' decides $\text{UPF}(L)$, so $\text{UPF}(L) \in \text{R}$.

$\text{UPF}(L) \in \text{R} \implies L \in R$ is false. If we define:

$$L \coloneqq \{xy \in \Sigma^* \mid (x = 0) \vee (x = 1 \wedge y \in L')\}, \qquad L' \notin \text{R}$$

then $\text{UPF}(L) = \emptyset \in \text{R}$.

$L \in \text{coRE} \implies \text{UPF}(L) \in \text{RE}$ is true: If $L \in \text{coRE}$, then $\overline{L} \in \text{RE}$, so there exists some TM M recognizing $\overline{L}$. Note that we can write:

$$\text{UPF}(L) = \{y \mid \forall x : xy \in \overline{L}\}$$

So we can build a recognizer M' for $\text{UPF}(L)$ as follows: For all $x \in \Sigma$ dovetail $M(xy)$. If all of them accept, then accept. If at least one rejects, then reject. Otherwise (at least one loops) then loop. If $\forall x : xy \notin L$, then $M'(y) = 1$. Otherwise, $M'(y) \in \{0, \infty\}$. So $\text{UPF}(L) \in \text{RE}$.

$\text{UPF}(L) \in \text{RE} \implies L \in \text{coRE}$ is false. If we define:

$$L \coloneqq \{xy \in \Sigma^* \mid (x = 0) \vee (x = 1 \wedge y \in L')\}, \qquad L' \notin \text{coRE}$$

then $\text{UPF}(L) = \emptyset \in \text{RE}$, but $L' \notin \text{coRE}$.

Part 4c

Prove that $\text{REV} = \{\langle M \rangle \mid$ if M accepts x then it accepts also $x^R\} \notin \text{R}$.

Solution: Rice's Theorem states that any non-trivial property of the language of a Turing machine is undecidable. Define the property:

$$P_{\text{REV}} = \{\langle M \rangle \mid \forall x \in \Sigma^* : \; x \in L(M) \implies x^R \in L(M)\}.$$

This property is non-trivial: For existence, take the machine M_{accept} which accepts every input. In this case, $x \in L(M_{\text{accept}}) \implies x^R \in L(M_{\text{accept}})$, so $L(M_{\text{accept}}) \in P_{\text{REV}}$. For non-universality, take the machine M_{01} that accepts only the single string 01. Here, $x \in L(M_{01}) \implies x = 01 \implies x^R = 10 \implies x^R \notin L(M_{01})$. Therefore, $L(M_{01}) \notin P_{\text{REV}}$. We conclude that P_{REV} is a non-trivial property.

This property is a property of a language: assume for a contradiction that there are two TMs M_1, M_2 which accept the same language, but $M_1 \in P_{\text{REV}}$ while $M_1 \notin P_{\text{REV}}$. This means that there exists some $x \in L(M_2)$ such that $x^R \notin L(M_2)$. But $L(M_1) = L(M_2)$, so this implies that there exists some $x \in L(M_1)$ such that $x^R \notin L(M_1)$, a contradiction to the assumption that $M_1 \in P_{\text{REV}}$.

So by Rice's Theorem, this property is undecidable, meaning $\text{REV} \notin \text{R}$.

Problem 5

Part 5a

Prove or refute: $A_{\text{TM}} = \{\langle M, x \rangle \mid M \text{ accepts } x\}$ is an RE-complete language.

Solution: This statement is true. To be RE-complete, a language L must satisfy two conditions:

1. $L \in \text{RE}$.
2. L must be RE-hard, meaning every other language in RE can be reduced to it.

First, as proven in the lectures, $A_{\text{TM}} \in \text{RE}$. We recall to proof: it is sufficient to construct a recognizer for A_{TM}: the universal TM U, which on input $\langle M, x\rangle$ simulates $M(x)$. If M accepts x, U accepts $\langle M, x\rangle$. Otherwise, U rejects or doesn't halt.

Second, A_{TM} is RE-hard. Let $L' \in \text{RE}$, and let $M_{L'}$ be a TM recognizing L'. We can reduce L' to A_{TM} with:

$$f(w) = \langle M_{L'}, w\rangle.$$

This works since $w \in L'$ if and only if $M_{L'}$ accepts w, which by definition means $\langle M_{L'}, w\rangle \in A_{\text{TM}}$.

Since A_{TM} is in RE and is RE-hard, it is indeed RE-complete.

Part 5b

Prove or refute: $\text{EQ}_{\text{TM}} = \{\langle M, M'\rangle \mid L(M) = L(M')\}$ is an RE-complete language.

Solution: This statement is false, as $\text{EQ}_{\text{TM}} \notin RE$. To see why it's not in RE, we reduce from $\overline{A_{\text{TM}}}$, which is known not to be in RE. Given an input $\langle M, w\rangle$, we build two machines M_1 and M_2:

- M_2: Rejects every input, so $L(M_2) = \emptyset$.
- M_1: Ignores its input and runs M on w. If M accepts w, M_1 accepts; otherwise, it rejects.

If M does not accept w, then M_1 also rejects every input, so $L(M_1) = \emptyset$ and $L(M_1) = L(M_2)$. In this case, $\langle M_1, M_2\rangle \in \text{EQ}_{\text{TM}}$.

If M does accept w, then M_1 accepts every string ($L(M_1) = \Sigma^*$), which is different from $L(M_2) = \emptyset$, so $\langle M_1, M_2\rangle \notin \text{EQ}_{\text{TM}}$.

This means:

$$\langle M, w\rangle \in \overline{A_{\text{TM}}} \quad \iff \quad \langle M_1, M_2\rangle \in \text{EQ}_{\text{TM}}$$

Since $\overline{A_{\text{TM}}} \notin \text{RE}$, EQ_{TM} cannot be in RE either, and therefore it cannot be RE-complete.

Part 5c

Prove or refute: $L = \{0^*1^*\}$ is an R-complete language.

Solution: This statement is true. $L \in \text{R}$ as it is a regular language. We can simply scan the input from left to right and check that no 0 appears after a 1. If this condition holds, accept; otherwise, reject.

L is also R-hard. We prove this claim by showing that any language in R can be reduced to L. Let $A \in \text{R}$, and let M_A be a decider for it. Define the function f: on input x, simulate $M_A(x)$. If $M_A(x) = 1$, output 01. Otherwise, output 10. Note that as A is decidable, M_A always halts. So:

$$x \in A \iff f(x) = 01 \iff f(x) \in L \quad \text{and} \quad x \notin A \iff f(x) = 10 \iff f(x) \notin L$$

meaning $A \leq L$. So L is R-hard.

Since $L \in \text{R}$ and L is R-hard, L is R-complete.

Problem 6

To which minimal class of R, RE, coRE, $\overline{\text{RE} \cup \text{coRE}}$ do each of the following languages belong:

Part 6a

$$L_1 \coloneqq \{\langle M, M' \rangle : M, M' \text{ are TMs}, \quad L(M) \subseteq L(M'), \quad |L(M') \setminus L(M)| = 1\}$$

This language is not in RE, because even if we run with dovetailing, we can never know if we have yet to find a counterexample for the requirement $L(M) \subseteq L(M')$ or $|L(M') \setminus L(M)| = 1$. It is also not in coRE, because verifying the complement of the language requires us to check if there is some $w \in L(M)$ such that $w \notin L(M')$, or if $|L(M') \setminus L(M)| \neq 1$. And while we can get a positive answer if $|L(M') \setminus L(M)| = 0$ or if $\exists w \in L(M) : w \notin L(M')$, if $L(M) \subseteq L(M')$ but $|L(M') \setminus L(M)| > 1$, we have no guarantee that we will ever get a confirmation for the latter. Formally:

To show $L_1 \notin$ coRE, we reduce HALT to L_1: Given $\langle M, x \rangle$ we build TMs A, B:

- Let s be some string.
- Define $A \coloneqq M_{\text{reject}}$, so $L(A) = \emptyset$
- Define B: for input y, simulate $M(x)$. If $M(x) \neq \infty$ and $y = s$ then accept. Else, loop forever.

Then:

- If $M(x)$ halts, then $L(B) = \{s\}$, so $L(A) \subseteq L(B)$ and $|L(B) \setminus L(A)| = 1$. Meaning $\langle A, B \rangle \in L_1$.
- If $M(x)$ doesn't halt, then $L(B) = \emptyset$, so $|L(B) \setminus L(A)| = 0$. Meaning $\langle A, B \rangle \notin L_1$.

Thus HALT $\leq L_1$. And since HALT $\notin$ coRE, $L_1 \notin$ coRE.

To show $L_1 \notin$ RE, we reduce $\overline{\text{HALT}}$ to L_1: Given $\langle M, x \rangle$ we build TMs A, B:

- Let s be some string.
- Define A: for input y, simulate $M(x)$. If $M(x) \neq \infty$ and $y = s$ then accept. Else, loop forever.
- Define B to accept only s, so $L(B) = \{s\}$

Then:

- If $M(x)$ halts, then $L(A) = \{s\}$, so $|L(B) \setminus L(A)| = 0$. Meaning $\langle A, B \rangle \notin L_1$.
- If $M(x)$ doesn't halt, then $L(A) = \emptyset$, so $L(A) \subseteq L(B)$ and $|L(B) \setminus L(A)| = 1$. Meaning $\langle A, B \rangle \in L_1$.

Thus $\overline{\text{HALT}} \leq L_1$. And since $\overline{\text{HALT}} \notin$ RE, $L_1 \notin$ RE.

Therefore:

$$L_1 \in \overline{\text{RE} \cup \text{coRE}}$$

Part 6b

$$L_2 \coloneqq \{\langle M \rangle : \exists M' \text{ such that } L(M') \subseteq L(M), \text{ where } M, M' \text{ are TMs}\}$$

This language is in R. In fact, it is actually the language $\{\langle M \rangle : M \text{ is a TM}\}$, as for any TM M, the TM M_{reject}, which rejects every input and thus $L(M_{\text{reject}}) = \emptyset$, can be the required M'. And since for any input we can check if it is a valid TM, this language is in R.

Part 6c

$$L_3 \coloneqq \{\langle M, M' \rangle : |L(M') \cap L(M)| > 1 \text{ where } M, M' \text{ are TMs}\}$$

This language is in RE. We describe an enumerator for it: Dovetailing, enumerate all pairs of TMs and simulate every word on both of them, looking for pairs that have more than one word that they both accept.

It is not in coRE, because even if we dovetail, we will never get an answer if $|L(M') \cap L(M)| = 0$. Formally, we prove this by reducing HALT to L_3:

Given $\langle M, x \rangle$, build TMs A, B:

- Let s, s' be some strings.
- Define $A \coloneqq M_{\text{accept}}$ which accepts all inputs, so $L(A) = \Sigma^*$.
- Define B: on input y, if $y = s$ accept. Else, simulate $M(x)$. If it halts and $y = s'$, accept. Else, loop forever.

Then:

- If $M(x)$ halts, then $L(B) = \{s, s'\}$, so $|L(A) \cap L(B)| = 2$. Meaning $\langle A, B \rangle \in L_3$.
- If $M(x)$ doesn't halt, then $L(B) = \{s\}$, so $|L(A) \cap L(B)| = 1$. Meaning $\langle A, B \rangle \notin L_3$.

Thus HALT $\leq L_3$. And since HALT $\notin$ coRE, $L_3 \notin$ coRE. Therefore:

$$L_3 \in \text{RE} \setminus \text{coRE}$$

Part 6d

$$L_4 \coloneqq \{\langle M \rangle : M \text{ is a 1-tape TM, and the head never visits cell number } a\}$$

This language is in R, because for every M and a, the a restricts the number of possible configurations. Specifically, there are $N \coloneqq |\Gamma|^a \cdot |Q| \cdot a$ configurations: $|\Gamma|^a$ is the number of strings that the tape can hold (as after cell a, the string is irrelevant), $|Q|$ is the number of states, and a is the number of head positions. We can iterate over all x such that $|x| \leq a$, and for each one simulate $M(x)$ for keeping track of the configurations we see.

- If we reach a configuration that has the head at cell a, we halt and reject.
- If $M(x)$ halts without the head reaching cell a, we accept.
- If we reach a configuration that we have already seen, that is a loop.

In the third case, if the head hasn't been to cell a yet, then it never will and we can accept.
For any $M(x)$ and a, one of these cases will happen within $N + 1$ steps, so we will always halt.