Problem 1 A *Turing machine with left reset* is similar to an ordinary Turing machine, but the transition function has the form

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, RESET\}$$

If $\delta(q, a) = (r, b, RESET)$, when the machine is in state q reading an a, the machine's head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

(Hint: Much like previous problems in this class, you'll need to describe some general construction that takes a Turing Machine-with-reset to an ordinary Turing Machine and visa-versa)

PART 1:
We can simulate an ordinary Turing Machine T with a left-reset-Turing Machine R or S.

$R =$ "On input w:

1. To simulate T when T moves its head to the right, R simply does the same.
2. To simulate T when T moves its head to the left, R first marks its current symbol and resets all the way to the left.
3. If the current cell is not a blank, R writes a blank over the symbol in the cell, and moves its head to the right. Otherwise, R moves its head to the right and repeats the current step (step 3).
4. If the symbol in the current cell does not have a mark, R writes the symbol from the previous cell over the symbol of the current cell. If the symbol in the current cell does have a mark, R writes the symbol from the previous cell over the symbol of the current cell, but with a mark. If the symbol in the current cell is a blank, R writes the previous symbol in the cell and continues to step 5. Otherwise it repeats the current step (step 4).
5. R resets its head, then scans right until it arrives at the marked symbol. If R needs to simulate a subsequent left-movement, it resets its head and goes to step 3. If R needs to simulate a subsequent right-movement, it removes the mark and moves its head to the right.
6. To simulate an additional right-movement, R goes to step 1. To simulate an additional left-movement, R goes to step 2.

R effectively simulates each of T's left movements by resetting and copying the entire tape one position to the right, using a mark to keep track of where to return its head to. Therefore, every string that is accepted by T is also accepted by R; every language that T recognizes is also recognized by R.

$S =$ "On input w:

1. To simulate T when T moves its head to the right, S simply does the same.
2. To simulate T when T moves its head to the left, S marks the current symbol and resets.
3. If the current symbol is not marked, go to 4. Otherwise, exit the procedure. Original location is on left edge of tape.
4. Mark the current symbol and reset.
5. Move the head to the first marked symbol, then move one more to the right.
6. If this symbol is unmarked, go to 7. Otherwise, the symbol is marked and this is where we started from at step 2. Unmark the symbol, reset, and move the head right until we get to the next marked symbol. This symbol is one to the left of our original location. Exit the procedure.

7. Mark the symbol, reset, advance to the next marked symbol, unmark it, move one to the right, and go to 6.

PART 2:
We can simulate a left-reset Turing machine R with an ordinary Turing machine T.

$T =$ "On input w:

1. To simulate R's right movement, T simply moves its head to the right.
2. To simulate R's reset movement, T moves its head left repeatedly until it reaches the left most end of the tape.

Since T can simulate R, every language that R recognizes is also recognized by T.

By describing the general constructions of ordinary Turing machines and Reset-Turing machines that simulate one another, we have shown that the two are equivalent in power. They recognize the same class of Turing-recognizable languages.

Problem 2 Give the informal descriptions for Turing machines that decide the following languages

a) $\{w \mid w$ contains twice as many 0s as 1s $\}$

$M =$ "On input string w:

1. Scan tape and mark the first unmarked 0. If no unmarked 0 is found, jump to step 4.
2. Continue to the right and mark the next unmarked 0, then return read-head to start of tape. If no unmarked 0 is found, *reject*.
3. Scan tape and mark the next unmarked 1. If no unmarked 1 is found, *reject*. Otherwise, go move read-head to start of tape, and go back to step 1.
4. Scan the tape to look for unmarked 1s. If there are any unmarked 1s left, *reject*. If there are not any unmarked 1s, *accept*."

b) $\{a + b = c \mid a, b, c \in \{0, 1\} *$ and the binary numbers represented by a and b sum to $c\}$

$M =$ "On input string w:

1. Scan tape to check if string is in the format: $\{0, 1\}^* + \{0, 1\}^* = \{0, 1\}^*$. If it isn't, *reject*. Move read-head back to start of tape.
2. Scan tape to check if all the 0s and 1s before the = symbol have been crossed off. If so, go to step 9. Scan right until either an X (X means crossed out symbol) or a + is found. Move left by one position. Cross off the symbol at the read-head. If the symbol was a 0, or no unmarked symbol was found, go to step 3. If the symbol was a 1, go to step 4.
3. Move past the + sign and scan until either an X or an = is found. Move left by one position. Cross off the symbol at the read-head. If the symbol was a 0 go to step 5. If the symbol was a 1, go to step 6. If no unmarked symbol was found between the + and the = sign, go to step 5.
4. Move past the + sign and scan until either an X or an = is found. Move left by one position. Cross off the symbol at the read-head. If the symbol was a 0 go to step 6. If the symbol was a 1, go to step 7. If no unmarked symbol was found between the + and the = sign, go to step 6 .
5. (case: $0 + 0 = 0$) Move past the = sign and scan until either an X or a *blank* is found. Move left by one position. Cross off the symbol at the read-head. If the symbol was a 0, move read-head back to start of tape, and go to step 2. If the symbol was a 1, or no unmarked symbol was found, *reject*.

6. (case: $0 + 1 = 1$ or $1 + 0 = 1$) Move past the = sign and scan until either an X or a *blank* is found. Move left by one position. Cross off the symbol at the read-head. If the symbol at the read-head was a 1, move read-head back to start of tape and go to step 2. If the symbol was a 0, or no unmarked symbol was found, *reject*.
7. (case: $1 + 1 = 10$) Move past the = sign and scan until either an X or a *blank* is found. Move left by one position. Cross off the symbol at the read-head. If the symbol at the read-head was a 0, move to the left until arriving at the + sign. If the symbol at the read-head was a 1, or no unmarked symbol was found, *reject*.
8. Scan to the left and find the first unmarked symbol. If the symbol is a 0, write a 1 in its place. If the symbol is a 1, write a 0 in its place, and repeat this step(step 8). If no unmarked symbol is found, shift the contents of the entire tape to the right by one position, and fill the vacant position at the start of the tape with 1. Go to step 2.
9. Scan right to check if all symbols after the = have been crossed off. If they have, *accept*. Otherwise, *reject*."

Problem 3 Show that the Turing-decidable languages are closed under

a) **union:**

Let L be the union of two Turing-decidable languages, L_1 and L_2. A language is decidable if and only if some non-deterministic Turing machine decides it(Corollary 3.19). Therefore it follows that for languages L_1 and L_2, there exist Turing machines M_1 and M_2 that decide them.

$L = L_1 \cup L_2$ is Turing-decidable if and only if there is some non-deterministic Turing machine that decides it. Any non-deterministic Turing machine is automatically a deterministic Turing machine. So, to show that the Turing-decidable languages are closed under union, we describe a Turing machine M that decides L, where

$$L = \{w \mid w \in L_1 \cup L_2, M_1 \text{ and } M_2 \text{ decide } L_1 \text{ and } L_2\}$$

$M =$ "On input string w:

1. Run M_1 on w. If it accepts, *accept*.
2. Run M_2 on w. If it accepts, *accept*.
3. If both M_1 and M_2 halt and reject, *reject*."

Since M will always halt – end up in either an accept or reject state, L is decidable.

b) **intersection:**

Let L be the intersection of two Turing-decidable languages, L_1 and L_2. A language is decidable if and only if some non-deterministic Turing machine decides it(Corollary 3.19). Therefore it follows that for languages L_1 and L_2, there exist Turing machines M_1 and M_2 that decide them.

$L = L_1 \cup L_2$ is Turing-decidable if and only if there is some non-deterministic Turing machine that decides it. So, to show that the Turing-decidable languages are closed under intersection, we describe a Turing machine M that decides L, where

$$L = \{w \mid w \in L_1 \cap L_2, M_1 \text{ and } M_2 \text{ decide } L_1 \text{ and } L_2\}$$

$M =$ "On input string w:
1. Run M_1 on w. If it rejects, then reject.
2. If M_1 accepts the input, run M_2 on w. If M_2 rejects, then reject.
3. If both M_1 and M_2 accept the input, then accept.

M accepts w if both M_1 and M_2 accept it. If either M_1 or M_2 rejects w, then M rejects w. Since M_1 and M_2 are deciders, they will always halt. Consequently, M will always halt, and is therefore a decider.

c) **complement:**

For any Turing-decidable language L, there is a Turing machine M that decides it. To show that Turing-decidable languages are closed under complement, we construct a Turing machine M' that decides L', where

$$\overline{L} = \{w \mid w \in \overline{L} \text{ and } M \text{ decides } L\}$$

$\overline{M} =$ "On input w:
1. Run M on w. If it accepts, reject.
2. If M rejects, accept.

If M accepts w, $\overline{M}$ rejects it. If M rejects w, $\overline{M}$ accepts it. Since M is a decider, it will not loop. Therefore, $\overline{M}$ also will not loop and is a decider.

d) **set difference:**

Let L be the set-difference between two Turing-decidable languages, L_1 and L_2. A language is decidable if and only if some non-deterministic Turing machine decides it(Corollary 3.19). Therefore it follows that for languages L_1 and L_2, there exist Turing machines M_1 and M_2 that decide them.

L is Turing-decidable if and only if there is some non-deterministic Turing machine that decides it. Any non-deterministic Turing machine is automatically a deterministic Turing machine. So, to show that the Turing-decidable languages are closed under set-difference, we describe a Turing machine M that decides L, where

$$L = \{w \mid w \in L_1 \text{ and } w \notin L_2, M_1 \text{ and } M_2 \text{ decide } L_1 \text{ and } L_2\}$$

$M =$ "On input string w:
Let w be any string that contains symbols in languages L_1 and L_2. Then,
1. Run M_1 on w. If it rejects, then reject.
2. If M_1 accepts the input, run M_2 on w. If M_2 accepts, then reject.
3. If M_1 accepts w and M_2 rejects w, then accept.

M accepts w if M_1 accepts and M_2 rejects. If M_1 rejects w or if M_2 accepts it, M rejects. M will always end up in either an accept or reject state, therefore it is a decider.

Problem 4 ***Show that the Turing-recognizable languages are closed under concatenation.***
This problem requires providing constructions that take individual Turing machines and combines them into a new machine that recognizes *the new language. Remember, this is about Turing-recognizable languages not just decidable so that there's a possibility of non-termination.*

Let L be the concatenation of two Turing-recognizable languages, L_1 and L_2. A language is Turing-recognizable if some Turing machine recognizes it(Definition 3.5). Therefore it follows that for languages L_1 and L_2, there exist Turing machines M_1 and M_2 that recognize them.
$L = L_1 \circ L_2$ is Turing-recognizable if and only if there is some non-deterministic Turing machine that recognizes it(Corollary 3.18). So, to show that the Turing-recognizable languages are closed under concatenation, we describe a Turing machine M that recognizes L, where

$$L = \{w \mid w \in L_1 \circ L_2,\ M_1 \text{ and } M_2 \text{ recognizes } L_1 \text{ and } L_2\}$$

$M =$ "On input string w:

1. Divide string w into two parts, w_1 and w_2.
2. Run M_1 on w_1. If it rejects, *reject*. If it accepts, move on to step 3.
3. Run M_2 on w_2. If it rejects, *reject*. If it accepts, *accept*."

At steps 2 and 3, if either M_1 or M_2 ends up looping – not ending up in an accept or reject state, M will consequently also loop forever. However, if w is divided in such a way that M_1 and M_2 accepts w_1 and w_2, M will halt and accept w. Therefore M recognizes L.

Problem 5 For each of the following Turing machine variants determine if the machine is more powerful, equivalent, or less powerful than a single-tape Turing machine. If less powerful describe the class of languages recognized by the machine. Explain your answers.

a) A Turing Machine that can only make moves to the right and never left.

A turing machine that can only move right can only read the input once. It may be able to write to the tape, but it can never go back and read what it wrote, so the tape to the left of the read head cannot act as memory. Therefore, it is not equivalent to a regular Turing Machine. Since this "read-right only" Turing machine has no functional memory, it can only recognize languages that a Finite State Machine can: Regular Languages.

b) A Turing Machine that can move right one space or move left two spaces.

This machine is equivalent to a normal, single-tape Turing machine. You can simulate a regular turing machine with the Left-2 turing machine by moving right once after every left-2, effectively simulating a left-1 move.

You can simulate a Left-2 machine with a regular Turing machine by simply moving left twice when moving left.

c) A Turing Machine that never writes over a space on the tape that already contains a symbol.

This "write once" turing machine is equivalent to a regular turing machine. Because we cannot modify or mark any of the symbols, we need to format the input before loading it into the tape. The way we will format it is inserting a blank space after each character of the input string. When the string is loaded onto the tape, we will insert a blank space to the right of every symbol. We can use this blank space to behave as a "marked" symbol.

Essentially, since we cannot directly modify any symbol, any time we need to, we copy the entire tape and place it to the right of the last symbol of the original input. We should use a symbol not in $\sum$ to separate the strings, like #. However, in order to copy, we need to be able to mark the current symbol we want to copy, place the copied symbol after the #, and return to where we left off.

When we grab a symbol from the original input string, we move one to the right and insert a marked version of that symbol in the blank space we inserted before loading in the string. Then we move left until we reach the # symbol, move one right and add the symbol. Then we move back until we see the first marked symbol, move one right, and repeat. For each following symbol to copy, we move right until we see 2 blanks in a row (we need to copy the string with the format "a_b_c_d...."). When we see 2 blanks, write in the symbol to be copied, and repeat until we hit the # symbol.