Sample solution to HW 4

(1) (i) From the initial state q_0, whatever it reads, the TM enters q_{acc}.

(ii) From the initial state q_0, whatever it reads, the TM enters q_{rej}.

(iii) It works as a DFA that accepts L_3. It always moves right, remembers in its state whether it reads 0 or 1. Say, we have two states p_0 and p_1, for remembering that it reads 0 and 1, respectively, in the previous cell. When it reads the blank symbol, from state p_0 it enters q_{acc}, and from state p_1 it enters q_{rej}.

More precisely, we have the following transitions in our Turing machine:

$$\begin{aligned}
(q_0, 0) &\to (p_0, 0, \texttt{Right}) \\
(q_0, 1) &\to (p_1, 1, \texttt{Right}) \\
(p_0, 0) &\to (p_0, 0, \texttt{Right}) \\
(p_0, 1) &\to (p_1, 1, \texttt{Right}) \\
(p_1, 0) &\to (p_0, 0, \texttt{Right}) \\
(p_1, 1) &\to (p_1, 1, \texttt{Right}) \\
(q_0, \sqcup) &\to (q_{\text{rej}}, 0, \texttt{Right}) \\
(p_0, \sqcup) &\to (q_{\text{acc}}, 0, \texttt{Right}) \\
(p_1, \sqcup) &\to (q_{\text{rej}}, 1, \texttt{Right})
\end{aligned}$$

Here we assume that $\epsilon \notin L_3$.

(iv) In general, description of pseudocode like the following is acceptable. Let N be the input number. On every pair (i, j), where $i, j \in \{2, \ldots, N-1\}$, check whether $i \times j = N$. If there is such a pair (i, j), output `False`. Otherwise, output `True`.

A more detailed explanation is as follows. The TM has five tapes.

- Tape 1 contains the input word w representing number N.
- In tape 2, it generates the number i.
  At the beginning the TM writes down $0^{|w|}$ on tape 2. To generate $i+1$ from i, the head goes to the rightmost cell of the tape. If the rightmost cell is 0, change it to 1. Otherwise, change the content of every cell from 1 to 0 until it sees the first 0 from the right direction. Change that 0 to 1.
- In tape 3, it generates the number j.
  It can be done in the same way as generating number i.
- Each time a new pair (i, j) is generated, it computes the number $i \times j$ on tapes 4, 5 and 6.
  It works as follows. Let $\lfloor i \rfloor$ and $\lfloor j \rfloor$ denote the binary representation of i and j, respectively. Say, the binary representation $\lfloor j \rfloor$ is $b_n \cdots b_0$. On tape 3, the TM has a "marker" that moves from b_n to b_0. When the marker is on b_k, if $b_k = 0$, it moves to b_{k-1}. If $b_k = 1$, it does the following.
  - The TM "copies" $\lfloor i \rfloor$ into tape 5.
  - After it finishes copying, the head on tape 5 in on the right cell of $\lfloor i \rfloor$, and it stays there.
  - On the other hand, the heads on tape 3 returns to the "marker," i.e., on position b_i. From there, the heads on tape 3 and tape 5 moves simultaneously to the right until tape 3 gets to the end of $\lfloor j \rfloor$. On each move, the head on tape 5 writes 0.

- After this, the content in tape 5 should be the binary representation of the $i \cdot 2^k$. TM performs addition between tape 4 and tape 5, and stores the result on tape 6.
- It then copies the result of the addition to tape 4.
  Intuitively, after this step, the content of tape 4 should be the binary representation of $i \cdot b_n \cdot 2^n + \cdots + i \cdot b_k \cdot 2^k$.
- The head on tape 3 moves the marker to the right by one step, i.e., moves it to b_{k-1}.

This "multiplication" process stops when the marker moves to the right of b_0.

(2) Both languages $\{0^n \mid n \geqslant 0\}$ and $\{1^n \mid n \geqslant 0\}$ are decidable. Now, either there is a parallel universe or there is not. In both cases L is a decidable language. So, L is decidable.

(3) In the original definition, an NTM $\mathcal{M}$ accepts an input word w, if there is an accepting run of $\mathcal{M}$ on w, and it rejects w, if *all* its runs are rejecting. With such definition, every NTM is equivalent to some DTM. More precisely, for every NTM $\mathcal{M}$, we can construct a DTM $\mathcal{M}'$ that on input w, $\mathcal{M}'$ generates all possible runs of $\mathcal{M}$ on w. $\mathcal{M}$ can then accept w if it finds an accepting run of $\mathcal{M}'$ on w. Thus, $\mathcal{M}'$ accepts w if and only if $\mathcal{M}$ accepts w.

With Bob's definition, the above construction of $\mathcal{M}'$ does not achieve the goal that $\mathcal{M}'$ accepts w if and only if $\mathcal{M}$ accepts w, since we still have to check that all runs are *not* rejecting.

Note: Explanation such as above is acceptable. In fact, you can go a bit further. You can also show that the following language is undecidable:

$$\{\lfloor \mathcal{M} \rfloor \$w \mid \text{all runs of } \mathcal{M} \text{ on } w \text{ are not rejecting}\}$$

Thus, Bob's definition does not imply simulation of NTM by DTM.

(4) The reduction is from CFL-Universality.

On input CFG $\mathcal{G}$:

- Construct a DFA $\mathcal{A}$ that accepts Σ^*.
- Output `True`, if $L(\mathcal{G}) = L(\mathcal{A})$. Otherwise, output `False`.

Question (5) is a bit more challenging. Its solution requires a few steps.

Step 1:

The following function is computable.

Function-All-or-Nothing	
Input:	A TM description $\lfloor \mathcal{M} \rfloor$ and a word w.
Task:	Output another TM description $\mathcal{M}'$ such that: • If $\mathcal{M}$ accepts w, $\mathcal{M}'$ accepts every word. • If $\mathcal{M}$ does not accept w, $\mathcal{M}'$ rejects every word.

Note the TM $\mathcal{M}'$ depends on $\mathcal{M}$ and w, so we will denote it by $\mathcal{M}_w$. It works as follows. On input v, $\mathcal{M}_w$ "deletes" v from its tape, writes w on its tape, and runs $\mathcal{M}$ on w. It goes to accepting state, if $\mathcal{M}$ goes to accepting state. That is, whatever the input v is, $\mathcal{M}_w$ accepts v if and only if $\mathcal{M}$ accepts w. An interesting question is: What is the language accepted by $\mathcal{M}_w$? The answer depends on whether $\mathcal{M}$ accepts w.

- If $\mathcal{M}$ accepts w, then $\mathcal{M}_w$ accepts every word.
- If $\mathcal{M}$ does not accept w, then $\mathcal{M}_w$ does not accept any word.

In particular, $\mathcal{M}$ accepts w if and only if the language accepted by $\mathcal{M}_w$ is infinite.

Now, the TM that computes Function-All-or-Nothing works by adding some appropriate transitions to $\mathcal{M}$. More precisely, on input $\lfloor\mathcal{M}\rfloor\$w$, it does the following:

- At the beginning let $\mathcal{M}_w$ contains exactly the same transitions as in $\mathcal{M}$.

  Let q_0 be the initial state of $\mathcal{M}$.
- Add a new tape symbol # and new transitions to $\mathcal{M}_w$ that upon reading the symbol #, it does the same thing as when reading the blank symbol $\sqcup$.

  That is, for every transition $(p, \sqcup) \to (q, \alpha, \beta)$ of $\mathcal{M}$, add a new transition $(p, \#) \to (q, \alpha, \beta)$ to $\mathcal{M}_w$. Intuitively, $\mathcal{M}_w$ treats the symbol # as if it is the blank symbol $\sqcup$.
- Add two new states s_0, s' (to $\mathcal{M}_w$) and s_0 is the initial state of $\mathcal{M}_w$.
- Add new transitions (to $\mathcal{M}_w$) such that from state s_0, $\mathcal{M}_w$ will move right "deleting" every input symbol by writing a symbol # on top of it until it reaches a blank symbol, upon which it enters state s'.

  More precisely, add the following transitions:

$$\begin{array}{rcl} (s_0, a) & \to & (s_0, \#, \texttt{Right}) \\ (s_0, \sqcup) & \to & (s', \#, \texttt{Stay}) \end{array}$$

- Add new transitions that from state s', $\mathcal{M}_w$ will go back to the leftmost cell of the tape and enter state p_0.
- Let $w = a_1 \dots a_n$.
- Add new states $p_1, \dots, p_{n+1}$ (to $\mathcal{M}_w$).
- Add the transitions (to $\mathcal{M}_w$) that from state p_i, $\mathcal{M}_w$ will write a_i into the tape, move right and enter state p_{i+1}.
- Add new transitions (to $\mathcal{M}_w$) that from state p_{n+1}, $\mathcal{M}_w$ will move right to the leftmost cell of the tape and enter state q_0.
- Finally, output $\lfloor\mathcal{M}_w\rfloor$.

Note that every transition of $\mathcal{M}$ is also a transition of $\mathcal{M}_w$. So, when $\mathcal{M}_w$ reaches the state q_0, it will use the transitions of $\mathcal{M}$ and the tape contains the word w.

Step 2:

Consider the following language:

$$\text{Finite-TM} \;:=\; \{\lfloor\mathcal{M}\rfloor \mid \text{the language accepted by } \mathcal{M} \text{ is finite}\}.$$

We will prove that Finite-TM is undecidable by showing HALT $\leqslant_T$ Finite-TM. The reduction works as follows. On input $\lfloor\mathcal{M}\rfloor\$w$:

- Compute $\lfloor\mathcal{M}_w\rfloor$ as explained in Step 1.
- Check whether $\lfloor\mathcal{M}\rfloor \in$ Finite-TM. If it is, output `False`. Otherwise, output `True`.

The correctness of the reduction follows from the fact that

- if $\mathcal{M}$ accepts w, then $\mathcal{M}_w$ accepts Σ^* which is infinite.
- if $\mathcal{M}$ does not accept w, the $\mathcal{M}_w$ does not accept any word, so the language accepted by $\mathcal{M}_w$ is empty set, which is finite.

So, $\mathcal{M}\$w \in \mathsf{HALT}$ if and only if $\lfloor \mathcal{M}_w \rfloor \notin \mathsf{Finite\text{-}TM}$.

Step 3:

Let us denote by Complement-is-CFL the language representing the problem in question (5). That is,

$$\mathsf{Complement\text{-}is\text{-}CFL} \;:=\; \{\lfloor \mathcal{G} \rfloor \mid \Sigma^* - L(\mathcal{G}) \text{ is CFL}\}$$

Here $\lfloor \mathcal{G} \rfloor$ denotes the CFG $\mathcal{G}$ written as a string.

Now, an accepting run of a TM $\mathcal{M}$ on w can be written as:

$$C_0 \vdash C_1^R \vdash C_2 \vdash C_3^R \vdash C_4 \cdots\cdots \qquad (\dagger)$$

where C_i^R denotes the reversal of the configuration C_i. From what we learn in the class, we can construct a CFG $\mathcal{G}$ that generates every string that is *not* an accepting run of $\mathcal{M}$ of the form (†).

We will show that $\mathsf{Finite\text{-}TM} \leqslant_m \mathsf{Complement\text{-}is\text{-}CFL}$ by the following reduction. On input $\lfloor \mathcal{M} \rfloor$:

- Construct a CFG that generates strings that are not accepting runs of $\mathcal{M}$ of the form (†). Let us denote the CFG by $\mathcal{G}_\mathcal{M}$.
- Check whether $L(\mathcal{G}) \in \mathsf{Complement\text{-}is\text{-}CFL}$. If it is, output `True`. Otherwise, output `False`.

The correctness of our reduction follows from the claim:

$$\lfloor \mathcal{M} \rfloor \;\in\; \mathsf{Finite\text{-}TM} \quad \text{if and only if} \quad \lfloor \mathcal{G}_\mathcal{M} \rfloor \;\in\; \mathsf{Complement\text{-}is\text{-}CFL}.$$

The "only if" direction is as follows. Let $\mathcal{M} \in \mathsf{Finite\text{-}TM}$. This means that there are only finitely many words that $\mathcal{M}$ accepts, which then implies that there are only finitely many accepting runs of $\mathcal{M}$.

Furthermore, since $\mathcal{G}_\mathcal{M}$ generates all strings that are *not* accepting runs of $\mathcal{M}$, this means $\Sigma^* - L(\mathcal{G}_\mathcal{M})$ consists of all the accepting runs of $\mathcal{M}$ of the form (†), that is, strings of the form:

$$C_0 \vdash C_1^R \vdash C_2 \vdash C_3^R \vdash C_4 \cdots\cdots \vdash C_N,$$

where C_N is an accepting configuration. Since there are only finitely many accepting runs of $\mathcal{M}$, the language $\Sigma^* - L(\mathcal{G}_\mathcal{M})$ is finite. Since every finite language is CFL, the language $\Sigma^* - L(\mathcal{G}_\mathcal{M})$ is CFL.

The "if" direction is as follows. Let $\mathcal{G}_\mathcal{M}$ be such that $\lfloor \mathcal{G}_\mathcal{M} \rfloor \in \mathsf{Complement\text{-}is\text{-}CFL}$. This means that $\Sigma^* - L(\mathcal{G}_\mathcal{M})$ is CFL. As we note earlier, $\Sigma^* - L(\mathcal{G}_\mathcal{M})$ denotes the language that consists of all strings of the form:

$$C_0 \vdash C_1^R \vdash C_2 \vdash C_3^R \vdash C_4 \cdots\cdots \vdash C_N,$$

where C_N is an accepting configuration. That is, strings in $\Sigma^* - L(\mathcal{G}_\mathcal{M})$ represent the accepting runs of $\mathcal{M}$. Let us denote by $\mathcal{G}_0$ the CFG that generates $\Sigma^* - L(\mathcal{G}_\mathcal{M})$.

We will use pumping lemma to show that $L(\mathcal{G}_0)$ *cannot* be infinite. If $L(\mathcal{G}_0)$ is infinite, we can take long enough a string u that can be pumped. Suppose u is the following word:

$$u \;:=\; C_0 \vdash C_1^R \vdash C_2 \vdash C_3^R \vdash C_4 \cdots\cdots \vdash C_N$$

When we apply pumping lemma on u, there are only two parts of u that can be pumped simultaneously, say, for example, on C_i and C_j^R, which will then violate the conditions for the string to be an accepting run. Thus, $L(\mathcal{G}_0)$ must be finite.

Now, since $\mathcal{G}_0$ generates all the accepting runs of $\mathcal{M}$, this means that there are only finitely many accepting runs of $\mathcal{M}$. In turn, this means that there are only finitely many words accepted by $\mathcal{M}$. Thus, $\lfloor \mathcal{M} \rfloor \in \mathsf{Finite\text{-}TM}$.