HW5 : Turing Machines **Sample Solutions**

CSE105Sp23

Assigned questions

1. Rep(**Describing a Turing Machine for** Rep) (18 points):
   Recall that for $L \subseteq \Sigma^*$ with $\Sigma = \{0,1\}$, we define the language over $\Gamma = \{0,1,2\}$

$$\begin{aligned}\text{Rep}(L) &:= \{w \in \Gamma^* \mid \text{between every pair of successive 2's in } w \text{ is a string in } L\} \\ &= \{w \in \Gamma^* \mid \text{for all } v \in \Sigma^* \text{ if } 2v2 \in \text{Substring}(\{w\}), \text{ then } v \in L\}\end{aligned}$$

where for all languages $K \subseteq \Gamma^*$ we let

$$\text{Substring}(K) := \{w \in \Gamma^* \mid \text{there exist } a, b \in \Gamma^* \text{ such that } awb \in K\}.$$

In this question, you will give three separate descriptions of a Turing machine which recognizes the language $\text{Rep}(\{0^n 1^n \mid n \geq 0\})$. This may seem like a somewhat tedious process, but we think that it is important to see all the different descriptions in action at least once for a single language.

   (a) (*Graded for completeness*) [1] High-level description: description of algorithm (precise sequence of instructions), without implementation details of machine. Your description can use data structures and refer to specific positions in the input strings (without specifying memory management).

Solution: We define the Turing machine as follows: "If there are zero or one 2's in the input string, accept. Otherwise, check that the string between the leftmost pair of successive 2's is some number of 0s followed by the same number of 1s. If not, reject. If so, check that the string between the next-leftmost pair of successive 2's has this pattern. If there is not another 2 to pair off, then accept."

[1] This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the question, we ask that you include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

(b) (*Graded for correctness*) [2] Implementation-level description: English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents.

Solution: M_1 = "On input string w

1. Scan the input tape from left to right until the first 2. If there is no 2, accept.
2. Continue scanning right until another 2 is reached, then return the head to the first character to the right of the previous 2. If there is not another 2, accept.
3. If the character is a 2, go to stage 2. If it is a 1, reject. If it is a 0, cross it off (by writing an "x" over it) and scan right to the first 1. If there are no 1s before the next 2, reject. Else, cross of the 1.
4. Scan left past any x's, representing crossed off 1s. If a 0 is reached, continue scanning left until the first 0 to the right of the last x is reached, and go to stage 3. If a 2 is read, meaning there are no more 0s to cross off, scan right past any x's. If any character besides a 2 or x is reached, reject. Else, once the next 2 is reached, return to stage 2."

(c) (*Graded for correctness*) Formal definition: Give the 7-tuple of parameters

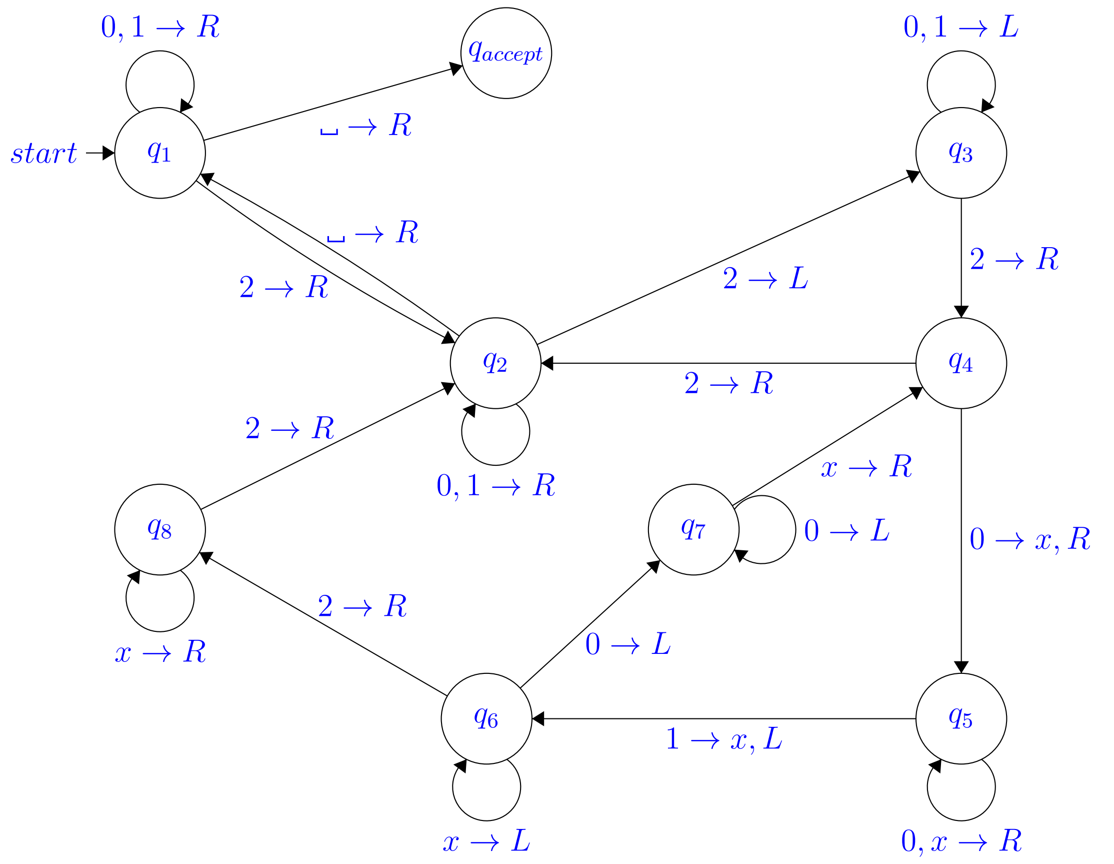
$$(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$$

describing the Turing Machine. Represent the transition function δ by drawing the state diagram of the Turing machine. You may use the following conventions: omit the reject state from the diagram; any missing transitions in the state diagram are assumed to go to the reject state.

Solution: $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{rej}})$:

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_{\text{accept}}, q_{\text{rej}}\}$
- $\Sigma = \{0, 1, 2\}$
- $\Gamma = \{0, 1, 2, x, \sqcup\}$
- We describe δ with the state diagram below
- $q_0 = q_1$
- The accept and reject states are q_{accept} and q_{rej}, respectively.

[2]This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.



Note: Notation is modeled from the textbook. See Sipser pg. 145 for details of the conventions used.

2. **This Turing Machine is broken... or is it?** (12 points):
Let's consider a variant of a Turing Machine which models computation where the data keeps getting corrupted. A Corrupted Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ has some of the usual Turing Machine components: set of states (Q); input alphabet (Σ); transition function (δ); start state (q_0); accepting state (q_{acc}), rejecting state (q_{rej}). Unlike a normal Turing Machine, the tape alphabet $\Gamma \supseteq \Sigma$ has two (rather than one) special characters $\sqcup$ and $\lightning$:

- $\sqcup \in \Gamma$, $\sqcup \notin \Sigma$: This is the usual blank symbol.
- $\lightning \in \Gamma$, $\lightning \notin \Sigma$: This symbol indicates a cell that has been corrupted. The machine cannot write over any corrupted cell; that is, for each $q \in Q$,

$$\delta(q, \lightning) \in \{(r, \lightning, D) \mid r \in Q, D \in \{L, R\}\}$$

Computation in the Corrupted Turing Machine proceeds as normal except that sometimes when writing a tape symbol as intended, a $\lightning$ symbol is written instead. Thankfully, the pattern of corruption is predictable: the first write is corrupted, and then every other write thereafter is corrupted. That is, the first, third, fifth,... etc. writes are corrupted.

(a) (*Graded for correctness*) Prove that for every **regular** language L, there exists a Corrupted Turing Machine M that recognizes L.

Solution: *Intuition:* A Turing machine, by definition, has a finite control (as captured by the state diagram) and a tape. Control provides a deterministic flow between states, which is equivalent to finite automata with an additional capability of reading/writing to the tape on every transition. For a regular language L, we can define a Turing Machine which recognizes this language, simply not using the capability of the tape.

Formal: We know that every regular language is recognized by a DFA. The DFA transitions from one state to another reading one input symbol at a time. An equivalent Turing Machine has the same control. There are two issues to address:

i. How does the TM simulate the computation of a DFA; i.e., reading the input one symbol at a time
ii. When does the TM accept/reject?

For the former issue, we just always move right on the tape, reading each symbol in turn. For the latter issue, notice that DFA could have many accept states, but by definition a TM only has a single accept state. Furthermore, we only want to accept once we're done reading the input string (not just the first time we reach an accept state). To do this, we introduce a separate q_{accept} and q_{reject} state in the control of the TM. We introduce new transitions in the TM that move from an accept state to q_{accept} whenever we're done reading the input (i.e., the tape head is at a blank symbol on the tape); and similarly, we create a transition from each non-accept state in the DFA to q_{reject} whenever we're done reading the input.
Formally, for DFA $(Q, \Sigma, \delta, q_0, F)$, we define the Corrupted TM

$$(Q', \Sigma, \Gamma, \delta', q_0, q_{accept}, q_{reject})$$

with

$$Q' = Q \cup \{q_{accept}, q_{reject}\},$$
$$\Gamma = \Sigma \cup \{\sqcup, \lightning\}$$

and start state, accept state, and reject state as labeled in the 7-tuple, and δ' is defined as follows:

$$\delta'(q_i, a) = (q_j, a, R) \text{ if } q_i \in Q,\ a \in \Sigma, \text{ and } \delta(q_i, a) = q_j$$
$$\delta'(q, \sqcup) = (q_{accept}, \sqcup, R) \text{ for each } q \in F$$
$$\delta'(q, \sqcup) = (q_{reject}, \sqcup, R) \text{ for each } q \in Q - F$$
$$\delta'(q, x) = (q_{reject}, \sqcup, R) \text{ otherwise}$$

For each string $w \in \Sigma^*$, if the computation of the given DFA on w is $r_0, \ldots, r_{|w|}$, then the computation of the Corrupted TM we defined on w is $r_0, \ldots, r_{|w|}, q_{accept}$ exactly when $\delta^*(q_0, w) = r_{|w|} \in F$ and is $r_0, \ldots, r_{|w|}, q_{reject}$ exactly when $\delta^*(q_0, w) = r_{|w|} \notin F$. Thus, this Corrupted Turing Machine recognizes the language recognized by the given DFA. Notice that the "corrupted" aspect of the corrupted TM never comes into play because we always move right and never read any symbol we've previously written.

(b) (*Graded for completeness*) It will turn out that the Corrupted Turing Machine is no less powerful than our usual definition of a Turing Machine. Let's break the proof into a few steps. First, define the alphabet $\Sigma_{\text{pairs}} := \Sigma \times \Sigma = \{(a, b) \mid a \in \Sigma, b \in \Sigma\}$ of pairs of symbols in Σ. Give the construction of a Corrupted Turing Machine that takes input $a = a_1 a_2 \cdots a_n \in \Sigma^*$ and rewrites it as pairs of symbols interspersed by corrupted symbols. To be precise, the Corrupted Turing Machine should take starting tape configuration

a_1	a_2	$\ldots$	a_{n-1}	a_n	$\sqcup$	$\sqcup$	$\ldots$

to the tape configuration

$\lightning$	(a_1, a_2)	$\lightning$	(a_3, a_4)	$\lightning$	$\ldots$	$\lightning$	(a_{n-1}, a_n)	$\lightning$	$\sqcup$	$\sqcup$	$\ldots$

with the head once again pointing to the first cell. For simplicity, you may assume that n is even.

Solution: *Intuition:* Each time we read a symbol, we "remember" it in the finite control portion of the TM. This way, when we go to write a symbol which we know to be corrupted (recall, we know every other write is corrupted), we can simply choose not to make a meaningful write. We wait until the next write and then write two symbols at that tape location at the same time. We do this by writing down a symbol in Σ_{pairs}.

Formal: Let's create a Corrupted Turing machine with $4 + |\Sigma|$ states which have labels q_0, q_1, q_2, q_3 and $q^{(a)}$ for each $a \in \Sigma$. We preprocess the input as follows: From $i = 1$, (Always move head position to the right)

1. If i is odd, read the symbol on the current tape position. If the symbol is not blank, record it in memory (it doesn't matter what we write because it will be corrupted. I.e., $\delta(q_0, a) = (q^{(a)}, \lightning, R)$ for $a \in \Sigma$. If the symbol is blank, move to step 3; i.e., $\delta(q_0, \sqcup) = (q_1, \lightning, L)$.
2. If i is even, use the already stored symbol in the current state and combine with this symbol and write symbol in Σ_{pairs}. Go back to step 1. I.e., $\delta(q^{(a)}, b) = (q_0, (a, b), R)$.
3. Move the head to the left until we reach the beginning. There's a trick to notice that we're at the beginning of the tape. Namely we expect to read a corrupted symbol every other read. However, if we're at the beginning of the tape, the first character is corrupted so we will keep reading the corrupted symbol. (To formalize this we can use the states q_1 and q_2 to keep track of

which step we're on, and transition to q_4 when we've read two $\lightning$ symbols in a row.)

(c) (*Graded for completeness*) Starting from the tape configuration we created in the previous step, describe the implementation of a Corrupted Turing Machine which simulates the computation of any normal/uncorrupted Turing Machine.

Solution: *Intuition:* After the pre-processing we did, the tape consists of symbol $\lightning$ and Σ_{pairs}. Upon reading tape symbol $(a_1, a_2) \in \Sigma_{pairs}$, implies that we have now read two symbols a_1 and a_2 on the original uncorrupted tape. So, in the corrupted Turing Machine using the same original state diagram, the machine should take two steps every time it reads a tape symbol in Σ_{pairs}.

Formal: Let M_{orig} be the Turing machine that will be simulated. We define the Corrupted Turing machine which first pre-processes the input as specified in part (b). Then, for the input string $w = a_1a_2a_3a_4a_5\dots a_n$, the tape configuration which we created in part(b), we give the following implementation-level description of the Corrupted Turing Machine:

1. Move the tape head right, bypassing the first $\lightning$ symbol
2. On scanning an input symbol $(a, b) \in \Sigma_{pairs} \cup \{(\sqcup, \sqcup)\}$ simulates the computation of M_{orig} by transitioning to the state that M_{orig} would transition to after reading a and then b and then move the read/write head according to one of the following cases:
   a. If the read/write of M_{orig} on a would move to the Right, and the read-/write head of M_{orig} on b would move Left, we stay in the same position on the tape. ($R \to L =$ no movement on the tape)
   b. If the transition of M_{orig} on both a and b move Right, move the head Right *twice*, bypassing the $\lightning$ before the next symbol in Σ_{pairs}.
   c. If the transition of M_{orig} on a moves Left, the tape head should be moved to the Left *twice*. This is the one case where you're only processing one symbol from the original tape.

Challenge; not graded: A key idea in the construction above was to increase the number of symbols in our tape alphabet. Can you do the same construction *without* increasing the size of the tape alphabet? That is, for every language L recognized by a normal Turing machine over alphabet Σ and tape alphabet $\Sigma \cup \{\sqcup\}$, is there a Corrupted Turing Machine with tape alphabet $\Sigma \cup \{\sqcup, \lightning\}$ that recognizes L?

Solution: No. Consider any language that requires you to memorize an entire string such as $L = \{ww \mid w \in \Sigma^*\}$. Claim: if the head of the corrupted Turing machine has visited the nth cell, then at least $\lceil n/2 \rceil$ of the first n cells have been corrupted; furthermore, at least every other cell must be corrupted.

Based on the claim, let's give an upper bound on the total number of configurations of the tape once the head has visited the nth cell. There are two possibilities in terms of whether or not the even-indexed cells are corrupted or the odd-indexed cells are corrupted (by the claim, we must be in one of these two configurations). There are at most $n/2$ cells which have not been corrupted in this way. For each of those cells, there are $(|\Sigma| + 2)$-many choices—the cell can be some symbol in Σ or it might be blank or corrupted (this is actually overcounting the number of configurations a bit, but that's okay). Therefore, we get an upper bound of $2(|\Sigma| + 2)^{n/2}$ possible configurations of the tape.

Notice however, that there are $|\Sigma|^n$ length-n strings. Dividing the number of initial configurations ($|\Sigma|^n$) by an upper bound on the number of possible tape configurations after the corrupted Turing machine has read the first n symbols ($2(|\Sigma| + 1)^{n/2}$), we get (setting $|\Sigma| = 3$ for concreteness)

$$\frac{1}{2} \cdot \frac{3^n}{5^{n/2}} = \frac{1}{2}(3/\sqrt{5})^n,$$

which is a lower bound on the number of distinct length-n strings that nevertheless have the same tape contents. That is, since $3/\sqrt{5} \approx 1.342$, the number of such configurations grows with n. The only way to keep track of these distinctions is in state memory, but n is arbitrary and there is only constant state memory. Therefore, the Corrupted Turing Machine must fail to accept some $2n$-length string $ww \in L$.

3. **True/False enumerator** (20 points):
For each of the following statements, determine if it is true or false. Clearly label your choice by starting your solution with **True** or **False** and then provide a brief (3-4 sentences or so) justification for your answer.

(a) (*Graded for correctness*) Every enumerator enumerates an infinite language.

Solution: False. The language $L(M)$ of an enumerator M is defined as the set of strings $w \in \Sigma^*$ such that M prints w after some finite number of steps. The enumerator does not halt, but this does not mean its language is infinite. As a (trivial) example we define an enumerator that enumerates the empty language $\emptyset$:

```
E = 1. For $i = 1, 2, \ldots$
    2.         increment $i$
```

E does not print any string so $L(E) = \emptyset$, a finite language.

(b) (*Graded for correctness*) Let E be any enumerator and M be any Turing machine. If $L(E) = L(M)$, then M enters the reject state for all strings not in $L(E)$.

Solution: False. Consider the Turing machine M defined as

$$M = (\{q0, qacc, qrej\}, \Sigma, \Sigma \cup \{\sqcup\}, \delta, q0, qacc, qrej)$$

with

$$\delta(q, x) = (q_0, x, L)$$

for all $q \in \{q0, qacc, qrej\}$ and $x \in \Sigma \cup \{\sqcup\}$. Notice that on every input, M just loops in its start state (which is neither an accept nor a reject state). Thus $L(M) = \emptyset$. Let E be an enumerator that does not print any strings (see part (a) for its high-level description). Clearly, $L(E) = L(M) = \emptyset$. However, for any string not in $L(E)$ (which in this case is every string in Σ^*), M never enters the reject state.

(c) (*Graded for correctness*) Let E be any enumerator over Σ. Suppose $a, b \in \Sigma^*$ and $a, b \in L(E)$. If E prints a before b, then $|a| \leq |b|$.

Solution: False. There are many counterexamples examples you could give, but here is one:

Define the enumerator

```
E_1 = 1. Print 11
      2. Print 1
      3. for i = 1, 2, ...
      4.         increment i
```

Then $L(E_1) = \{1, 11\}$ and $|1| < |11|$ but 11 is printed before 1.

(d) (*Graded for correctness*) Let M decide language L over Σ such that M halts on all inputs $w \in \Sigma^*$ in $|w|^{2023}$ steps. There exists an enumerator E with the following properties: $L(E) = L(M)$; and if $a, b \in L$ and $|a| < |b|$, then E prints a before b.

Solution: True.
Let $s_1, s_2, \ldots$ be a list of all strings in Σ^* in string (shortlex) order.
Given a Turing machine M with the property in the statement. Define the enumerator E_M = "Ignore any input. Repeat the following for $i = 1, 2, 3 \ldots$
1. Run M on input s_i
2. If M accepts s_i, print out s_i"

First, notice that $L(E_M) = L(M)$ because, if a string is in $L(M)$ then that string is s_x for some nonnegative integer x and running E_M, each iteration for values of i from $i = 0$ to $i = x - 1$ takes finitely many steps (because step 1 is a simulation of a computation of M and we assumed that each such computation halts in finitely many steps), so in the xth iteration, E_M simulates the computation of M on s_x, and since M accepts the string, E_M will print it; conversely, if a string is not in $L(M)$ then E_M will not print it because E_M only prints strings that are accepted by M. Moreover, let a and b be strings such that $|a| < |b|$ and each of the strings is accepted by M. By definition of string order, strings are ordered first by length and then alphabetically. Since $|a| < |b|$, $a = s_{i_1}$ and $b = s_{i_2}$ for $i_1 < i_2$. Thus, in the computation of E_M, a is considered in the i_1th iteration, which is before the iteration where b is considered. In this i_1th iteration, the computation of M on $s_{i_1} = a$ is simulated, and $s_{i_1} = a$ is printed. In the i_2th iteration, the computation of M on $s_{i_2} = b$ is simulated, and $s_{i_2} = b$ is printed. Thus, a is printed by E_M before b.

(e) (*Graded for correctness*) Let N be a nondeterministic Turing machine. There is an enumerator E that enumerates the set of all and only strings accepted by N that have odd length.

Solution: True.
Let N be a nondeterministic Turing machine. Theorem 3.17 in Sipser states that for every nondeterministic Turing machine, there is a (deterministic) Turing machine that recognizes the same language.
Thus, there exists a deterministic Turing machine N' such that $L(N) = L(N')$.
Define the Turing machine

```
$M_{inter}$ = "On input $x$
    1. If $|x|$ is even, reject.
    2. Otherwise, run $N'$ on $x$.
    3.     If it accepts, accept; if it rejects, reject."
```

$L(M_{inter}) = \{x \mid |x| \text{ is odd and } x \in L(N') = L(N)\}$. Thus, M_{inter} recognizes the set of all and only strings accepted by N that have odd length. By Theorem 3.21 in Sipser, since this set is recognizable, there is some enumerator that enumerates it.