Complexity Problem Set 2

Bayard Walsh

January 2024

1

1.1 A

Balanced Turing Machine:

$Q = \{q_0, q_1, q_2, q_3, q_{accept}, q_{reject}\}$
$\Sigma = \{0, 1\}^*$
$\Gamma = \{0, 1, \phi, \omega, \sqcup\}^*$
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

note: the tuple notation in the table, for example ϕ,(q_1, b, R), indicates write ϕ at current position and then go to (q_1, b, R). q_{accept} and q_{reject} step to themselves on anything because once either is reached the program terminates.

	q_0	q_1	q_2	q_3	q_{accept}	q_{reject}
0	ϕ,(q_1, b, R)	(q_1, b, R)	ω,(q_3, b, R)	(q_3, b, L)	q_{accept}	q_{reject}
1	ϕ,(q_2, b, R)	ω,(q_3, b, R)	(q_2, b, R)	(q_3, b, L)	q_{accept}	q_{reject}
$\sqcup$	$(q_{accept}, \sqcup, R)$	q_{reject}	q_{reject}	q_{reject}	q_{accept}	q_{reject}
ϕ	(q_0, b, R)	(q_1, b, R)	(q_2, b, R)	(q_0, b, R)	q_{accept}	q_{reject}
ω	(q_0, b, R)	(q_1, b, R)	(q_2, b, R)	(q_3, b, L)	q_{accept}	q_{reject}

Table 1: δ

1.2 B

The language starts at q_0, and then goes to either q_1 or q_2 depending on if the first character is 0 or 1 (an initial $\sqcup$ string is accepted as the empty string is technically balanced). This character is then overwritten as ϕ to show that the string up to that point is balanced other than that character and as a flag for future backwards iterations. q_1 and q_2 mirror the same behavior by looping right until they see the symbol that wasn't at q_0 initially, which is the opposite of the symbol we turned to ϕ. We use 2 states mirrored for this behavior to find 1 for 0 and 0 for 1. This opposite state is marked as ω, which indicates that

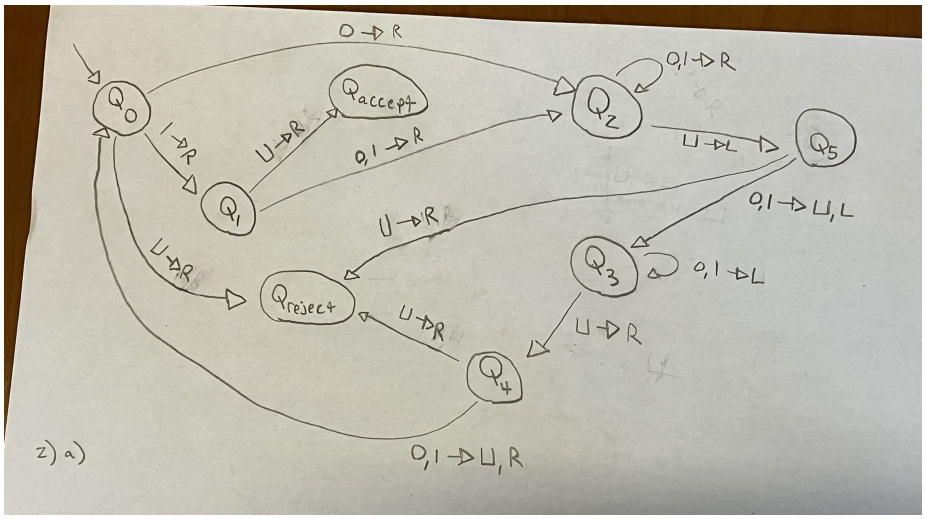
the given character has been crossed off in terms of balancing the overall string. Now we go to q_3. This state loops left until the first ϕ, which indicates that the string is balanced before that symbol and then returns to q_0. q_0 ignores any ω or ϕ while it loops, because these symbols have been crossed out, and if we are at q_0 we know that the string is balanced at the current configuration. Therefore we loop right on the tape and stay in q_0, and if we don't encounter any 0 or 1 we know that we have crossed off every symbol by setting contrasting values to ω and ϕ to balance each other out, meaning that the overall string is balanced and we can go to q_{accept} on the first $\sqcup$. If we encounter $\sqcup$ in any other state we go to q_{reject} as we know we currently do not have a balanced string and we have reached the end of the input string, meaning the string is not balanced and will not be accepted by our Turing machine.

2

2.1 A

Turing Machine: $|w|$ is odd and the middle bit of w is 1

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$
$\Sigma = \{0, 1\}^*$
$\Gamma = \{0, 1, \sqcup\}^*$
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$



2.2 B

We start in q_0, and reject if $\sqcup$, as $|\epsilon| = 0$ which is not odd. If 0 go to q_2, where we loop right until $\sqcup$, and then go to left one and go to q_5. Here we replace the final symbol with $\sqcup$ and then go to q_3, where we loop left until $\sqcup$, where we go right one position and then go to q_4, where we replace the first symbol with $\sqcup$ and then return to q_0. Therefore if our string starts with 0 we will abridge the first and last characters of the string. If our string starts with 1, we will instead check if the string is length 1 by checking if the next symbol is blank. If so we go to q_{accept} because we have a string of odd length with the middle symbol 1. If not, we will step to q_2 and begin the previously mentioned process of removing the first and last symbols. Therefore at every loop of our overall process we will either accept a string of length 1 equal to 1 or cut off the first and last character and loop again, or reject the string if we run out of string. All strings of odd length with a middle bit at 1 will eventually reach a string of length 1 equal to 1 following this approach. If we have an empty string at q_0,q_5 or q_4 we will reject the string as it doesn't follow the pattern in L, meaning it cannot be a string of odd length with a middle bit equal to 1. Therefore Turing machine M decides L.

3

Let L be some arbitrary finite language. As we know L is finite, the language must have some string (or strings) with maximum length. Therefore, let us have l' be a string in L with maximum length, where $|l'| = m$.

Definition:
We define our state space: $Q = \Sigma_0 \cup \Sigma_1 \cup \ldots \cup \Sigma_m \cup \{q_{\text{accept}}, q_{\text{reject}}\}$

The Turing Machine's state space is designed so it has a state corresponding to every possible symbol for every possible string in the input alphabet of L. For example, given w' such that $w' \in L$ and $w' = \sigma_1, \sigma_2, \cdots, \sigma_i$ (where $i \leq m$), the Turing Machine will have states $q_1, q_2, \cdots, q_i$ such that each q state in the Machine corresponds to a σ in w' and so the **NEXT** of each q is the next character in the string w'. More formally, for every arbitrary state q_j such that $q_0, q_1, \cdots, q_j, q_{j+1}, \cdots q_i$ we go **RIGHT** on the input string, such that $\delta(q_j, v)$ steps to (q_{j+1}, b, R). At q_i, the Turing Machine has that $\delta(q_i, \sqcup)$ steps to (q_{ACCEPT}, b, R).

The machine steps to (q_{REJECT}, b, R) whenever we encounter some character σ (note that σ can be $\sqcup$) such that σ does not occur at that position for any permutations of any strings of L. In other words, if at σ the sub string generated up to and including σ doesn't match any $s \in L$, step to (q_{REJECT}, b, R).

As we know that Σ is finite by the definition of an input alphabet and m is finite, we can construct this Turing Machine.

Proof of Correctness:
Say we have some arbitrary input w, and let C_i be the i-th configuration the Turing machine reaches when it is given w as input.

C_0: At the configuration of length 0 the only possible w string is ϵ.

Consider if $\epsilon \in L$. Then $\delta(q, \epsilon)$ steps to (q_{ACCEPT}, b, R), because for every $\in w$, we connected the states $q_0, q_1, \cdots, q_i$ corresponding to the characters in w and then step to (q_{ACCEPT}, b, R). Therefore if $\epsilon \in L$ it will also step to q_{ACCEPT}.

Consider if $\epsilon \notin L$. Then $\delta(q, \epsilon)$ steps to (q_{REJECT}, b, R), because we step to (q_{REJECT}, b, R) whenever we encounter some σ such that σ does not occur at that position for any permutations of any strings of L, and if $\epsilon \notin L$, then that σ does not occur at that position for any permutations of any strings of L and our Machine will step to q_{REJECT}.

Therefore our Turing Machine decides L at C_0.

C_{k+1}: Let $|w| = k + 1$. By assumption, our Turing Machine decides L on any arbitrary string up to the k-th configuration, C_k. Let $w = \sigma_0, \cdots \sigma_k, \sigma_{k+1}$, where each σ is a given character in the string.

Consider if $w \in L$. Therefore at configuration C_k, we are at some state q_k, and because our Turing Machine processes characters to the **RIGHT** at every configuration and we have a chain of states for every symbol $\in w$, because $w \in L$, we're at the symbol σ_k. At σ_k, we have $\delta(q_k, v) = (q_{k+1}, b, R)$,

where q_{k+1} is a state that represents the symbol σ_{k+1}, and then subsequently $\delta(q_{k+1}, v') = (q_{ACCEPT}, b', R)$. This is because $k+1$ is the last symbol of w, and by definition of our Turing Machine, we have a chain of states that corresponds to every σ in w, because $w \in L$, and we will step to q_{ACCEPT} after the final character. Therefore if $w \in L$ our Turing Machine will accept w.

Consider if $w \notin L$. If there is no $w' \in L$ such that w' begins with $\sigma_0, \cdots \sigma_k$, then we are already at q_{REJECT} because we defined the Turing Machine to q_{REJECT} whenever we encounter some character σ such that σ does not occur at that position for any permutations of any strings in L. As the Turing Machine rejects the wrong input, it decides L in this case. Next, consider if there is some $w' \notin L$ such that w' begins with w'', where $w'' \in L$ and $w'' = \sigma_0, \cdots \sigma_k$. Therefore we are at configuration C_k, at state q_k and at symbol σ_k. We know that at σ_k, we have $\delta(q, v) = (q_{REJECT}, b, R)$, because $|w| = k + 1$, $w \notin L$ and $\sigma_0, \cdots \sigma_k$ is a valid string up to that point of $w'' \in L$, meaning that at the $k + 1$ w deviates from any string $\in L$, so we step to (q_{REJECT}, b, R) by our Turing Machine definition. Therefore if $w \notin L$ our Turing Machine will reject w.

Therefore our Turing Machine decides L for any finite language L.

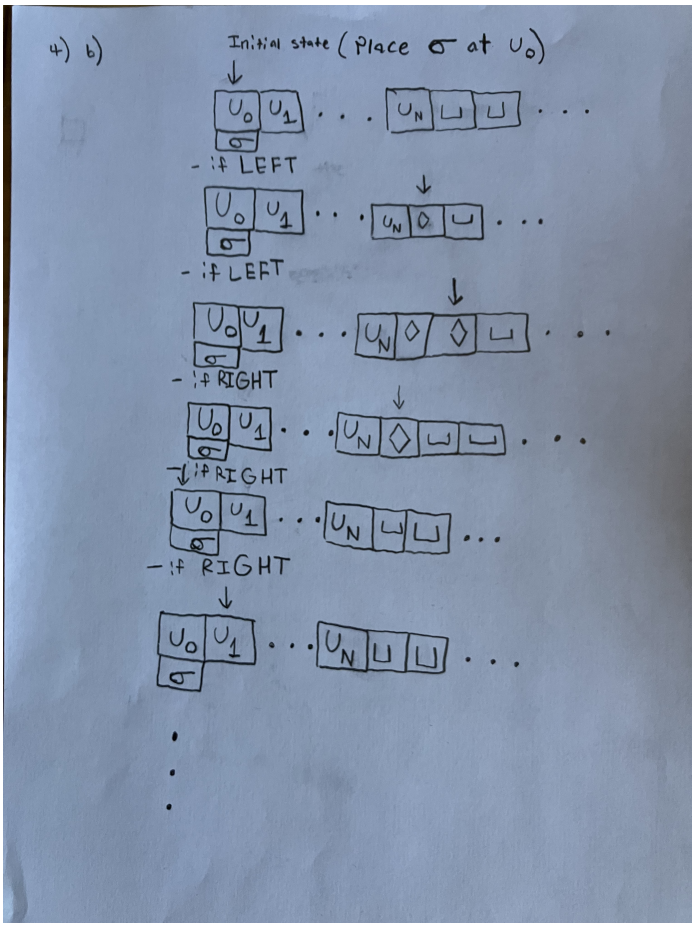
4

4.1 A

Formal definition of NEXT(uqv) for two-way-infinite Turing machine:
Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$
Let uqv be an arbitrary configuration of M, and let $u = u_1u_2 \dots u_n$ and $v = v_1v_2 \dots v_m$ where $u_i, v_i \in \Gamma$.

If $\delta(q, v_1) = (q', b, R)$, then NEXT$(uqv) = ubq'v_2v_3 \dots v_m$.
If $\delta(q, v_1) = (q', b, L)$, then NEXT$(uqv) = u_1u_2 \dots u_{n-1}q'u_nbv_2v_3 \dots v_m$.
Edge case: If $v = \epsilon$, then NEXT(uqv) = NEXT$(uq\sqcup)$.
Edge case: If $u = \epsilon$, then NEXT(uqv) = NEXT$(\sqcup qv)$.

This definition is similar to **NEXT(uqv)** for a one-way infinite Turing machine, however the properties of the infinite tape on the right are extended to the left, meaning the original edge case on the left (where $u = \epsilon$ so **NEXT** is $q'bv_2v_3 \cdots v_m$) can be removed, because there will always be some u' to the left (because the tape has infinite length) where the Turing Machine can step to. However, instead we add the edge case that mirrors the empty string behavior on the right side, where we treat ϵ as $\sqcup$. This is important because the configurations are not equal as mathematical objects but they represent the same scenario. These modifications let us simulate the two way infinite Turing Machine.



4.2 B

*Note: in this picture the Turing Machines tapes are sequential examples of the same tape after a given step (moving either **RIGHT** or **LEFT**) from top to bottom, following the commands given between each tape drawing*

We can simulate a Turing machine with an infinite two way tape through adding two symbols, say σ and $\diamond$, such that $\sigma \notin \Sigma$ and $\diamond \notin \Sigma$, but $\sigma \in \Gamma$ and $\diamond \in \Gamma$, and then following the procedure below:

Procedure:
At the very first configuration, say u_0, add σ to the square without overwriting u_0.

If at a sqaure with σ and given **LEFT**, go right until reaching the end of string and write $\diamond$ over the first $\sqcup$.

If at $\diamond$ and given **LEFT**, go to the right on the tape and write a $\diamond$.

If at $\diamond$ and given **RIGHT**, set the current $\diamond$ to $\sqcup$ and go to the left on the tape. Next: **If the new tape value is** $\diamond$: end there. **If the new tape value is not** $\diamond$: go left until reaching σ, then end.

In all other cases follow standard Turing Machine behavior.

Explanation:
Placing σ once at the very start functions as our start of Turing Machine flag and helps us return to that position in the future. If we are at the start of the Turing Machine, and we are given the command to go **LEFT**, to simulate an infinite empty string in the left directions we instead go **RIGHT** until we hit the end of the input string and mark the first $\sqcup$ as $\diamond$. From there, if we want to go **LEFT** on a $\diamond$, we instead go **RIGHT** and mark each $\sqcup$ as a $\diamond$. Because there are infinite $\sqcup$ positions on the **RIGHT** of the string, we can infinitely repeat this process of using blank steps **LEFT** with a $\diamond$ marking to simulate going **RIGHT** on infinite $\sqcup$ positions. However, if we are on a $\diamond$ and get the input to go **RIGHT**, we are attempting to move back towards the input string from the simulated left infinite $\sqcup$ string. In order to simulate this behavior towards the input string, we instead remove the $\diamond$ symbol and go **LEFT**. If the new position has a $\diamond$ symbol then we do nothing, because we still are on the simulated left tape created from attempting to go **LEFT** at the starting point. However if it is not on a $\diamond$, it means we have moved through our simulated left tape and should be at u_0, or the initial character. However, we aren't there because we are simulating the infinite string behavior on the space at the end of the input string, so instead we move to u_0 (marked by unique character σ). Otherwise we treat the string the same. Note that $\diamond$ placement is only used when simulating steps left of u_0 and only overwrites $\sqcup$, so the use of this symbol won't interfere with the rest of the tape behavior, while letting us simulate our infinite left tape.