

PSET 3 — 01/30/2023*Prof. Chakrabarti**Student: Amittai Siavava***Credit Statement**

I discussed ideas for this homework assignment with Paul Shin.

I also referred to the following books:

- (a) **Introduction to the Theory of Computation** by **Michael Sipser**.
- (b) **A Mathematical Introduction to Logic** by **Herbert Enderton**.

Problem 1.

Recall the CYCLE operation from Homework 1:

$$\text{CYCLE}(L) = \{yx : x, y \in \Sigma^* \text{ and } xy \in L\}$$

Prove that if L is regular, then so is $\text{CYCLE}(L)$.

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an DFA for L .

We design a new NFA N as follows:

$$N = (Q \times Q \times \{0, 1\}, \Sigma, \delta_N, s, F_N) \text{ such that } s \notin \Sigma, \text{ and:}$$

$$F_N = \{(q, q, 1) : q \in Q\} \tag{1.1}$$

$$\delta_N(s, \varepsilon) = \{(q, q, 0) : q \in Q\} \tag{1.2}$$

$$\delta_N((q, a, n), x) = \begin{cases} \{(q, \delta(a, x), n)\} & \text{if } q \in Q \text{ and } a \in \Sigma \\ \{(q, q_0, 1)\} & \text{if } a \in F \text{ and } a = \varepsilon \end{cases} \tag{1.3}$$

Claim 1.4. N accepts $\text{CYCLE}(L)$.

Proof. We need to prove that N accepts all strings in $\text{CYCLE}(L)$, and that any string N accepts is in $\text{CYCLE}(L)$.

Define two utility functions as follows:

$$\delta^*(q, x) = \begin{cases} q & \text{if } x = \varepsilon \\ \delta^*(\delta(q, x_1), x_2 \cdots x_n) & \text{if } x = x_1 x_2 \cdots x_n \text{ with each } x_i \in \Sigma \end{cases}$$

$$\delta_N^*(q, x) = \begin{cases} \delta_N(q, \varepsilon) \cup \{q\} & \text{if } x = \varepsilon \\ \delta_N^*(\delta_N(q, x_1), x_2 \cdots x_n) & \text{if } x = x_1 x_2 \cdots x_n \text{ with each } x_i \in \Sigma \end{cases}$$

(i) **Completeness:** $\mathcal{L}(N) \supseteq \text{CYCLE}(L)$

Let $a \in \Sigma_*$ be a string in $\text{CYCLE}(L)$ such that there exists y, x such that $x, y \in \Sigma^*$, $xy \in L$, and $a = yx$. The original DFA M accepts xy (since $xy \in L$). Let $p = p_1, \dots, p_m, p_{m+1}, \dots, p_{m+n}$ be the computational path of M on xy , such that $m+1 = |x|$, $n = |y|$, $p_1 = q_0$, each $\delta(p_i, s_i) = p_{i+1}$ for each $i \geq 0$, and $p_{m+n} \in F$. Consider the computation path of N on yx . We can make the ε -transition from s to $(p_{m+1}, p_{m+1}, 0)$. If we do, recall that p_{m+1}, \dots, p_{m+n} is the computational path of M on y , and the string a is equivalent to yx . Since δ_N reuses the original transition function, δ , to determine which state to move to on the current input symbol if it is in the alphabet, $\delta_N^*(s, y) = (p_{m+1}, p_{m+n}, 0)$.

However, p_{m+n} was an accepting state in M , so $p_{m+n} \in F$. Therefore, N has the option to take the ε -transition from $(p_{m+1}, p_{m+n}, 0)$ to $(p_{m+1}, q_0, 1)$. If we do, then consider then we start reading x while at $(p_{m+1}, q_0, 1)$. Since the computation path of M on x is p_1, \dots, p_m and δ_N reuses the original transition function, δ , to determine which state to move to on the current branch and phase when input the input symbol is in the alphabet, we have that $\delta_N^*((p_{m+1}, q_0, 1), x) = (p_{m+1}, p_{m+1}, 1)$.

However, $(p_{m+1}, p_{m+1}, 1) \in F_N$, so the string is accepted.

(ii) **Soundness:** $\mathcal{L}(N) \subseteq \text{CYCLE}(L)$

Let b be a string that is accepted by N . This means $b = yx$ for some string $x, y \in \Sigma^*$ such that:

- (a) $\delta_N^*(s, y) = (q_y, q_f, 0)$ such that $q_f \in F$,
- (b) $\delta_N^*((q_y, q_f, 0), \varepsilon) = (q_y, q_0, 1)$ (since $q_f \in F$), and
- (c) $\delta_N^*((q_y, q_0, 1), x) = (q_y, q_y, 1) \in F_N$.

In short, y takes the NFA from some state q_y to some state q_f that would have been accepted in M , then, since that state is in the set of original accepting states, the NFA does a transition to the start state q_0 and then reads x . x then takes the NFA from q_0 to the exact state where it started, q_y . This means that the computation path of the corresponding DFA on y if it started execution at state q_y , make a complete path from q_0 to q_f in the DFA. Therefore, the string xy is accepted by the original DFA, thus the string $b = yx$ is a member of $\text{CYCLE}(L)$ since xy is a member of L .

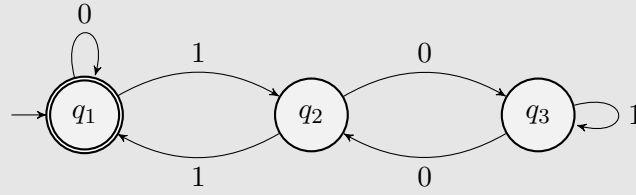
□

Problem 2.

- (a) Draw a 3-state DFA for the language

$$D_3 = \{x \in \{0, 1\}^* : \beta(x) \text{ is divisible by } 3\},$$

where $\beta(x)$ is the string x interpreted as a binary number.

FIGURE 1. DFA for D_3

- (b) Convert the DFA into a regular expression using the
- R_{ij}^k
- method.

To do this systematically, make a big table with 9 rows indexed by the pairs (i, j) and 4 columns indexed by the possible values of $k \in \{0, 1, 2, 3\}$. Fill each cell of the table with a regular expression that generates the corresponding R_{ij}^k . Sometimes, you'll obtain a complicated regular expression if you directly apply the equations from class. In such cases, show what the equations give you and only then simplify. You may use the shorthand X^+ to denote XX^* , where X is an arbitrary regular expression.

i	j	$k = 0$	$k = 1$	$k = 2$	$k = 3$
q_1	q_1	0	0^*	$0^* \cup 0^*1(10^*1)^*10^*$	$0^* \cup (0^*1(10^*1)^*10^*) \cup (0^*1(10^*1)^*0(1 \cup 0(10^*1)^*01^*)^*0(10^*1)^*10^*)$
q_1	q_2	1	$1 \cup 0^*1 = 0^*1$	$0^*1 \cup 0^*1(10^*1)^* = 0^*1(10^*1)^*$	$0^*1(10^*1)^* \cup (0^*1(10^*1)^*0(1 \cup 0(10^*1)^*0)^*0(10^*1)^*)$
q_1	q_3	\emptyset	\emptyset	$0^*1(10^*1)^*0$	$0^*1(10^*1)^*0 \cup 0^*1(10^*1)^*0(1 \cup 0(10^*1)^*01^*)^* = 0^*1(10^*1)^*0(1 \cup 0(10^*1)^*0)^*$
q_2	q_1	1	10^*	$10^* \cup (10^*1)^*10^* = (10^*1)^*10^*$	$(10^*1)^*10^* \cup (0(1 \cup 0(10^*1)^*0)^*)^*0(10^*1)^*10^*$
q_2	q_2	\emptyset	10^*1	$(10^*1)^*$	$(10^*1)^* \cup (0(1 \cup 0(10^*1)^*0)^*0)$
q_2	q_3	0	0	0	$0 \cup (0(1 \cup 0(10^*1)^*0)^*) = 0(1 \cup 0(10^*1)^*0)^*$
q_3	q_1	\emptyset	\emptyset	$0(10^*1)^*10^*$	$0(10^*1)^*10^* \cup ((1 \cup 0(10^*1)^*0)^*0(10^*1)^*10^*) = (1 \cup 0(10^*1)^*0)^*0(10^*1)^*10^*$
q_3	q_2	0	0	$0(10^*1)^*$	$0(10^*1)^* \cup ((1 \cup 0(10^*1)^*0)^*0(10^*1)^*) = (1 \cup 0(10^*1)^*0)^*0(10^*1)^*$
q_3	q_3	1	1	$1 \cup 0(10^*1)^*0$	$(1 \cup 0(10^*1)^*0)^*$

TABLE 1. Regular Expressions Generated via the R_{ij}^k Method

The regular expression for the DFA is the union

$$\bigcup_{f \in F} R_{1n}^n.$$

Since $F = \{1\}$, the regular expression for the DFA is

$$R_{11}^3 = 0^* \cup (0^*1(10^*1)^*10^*) \cup (0^*1(10^*1)^*0(1 \cup 0(10^*1)^*01^*)^*0(10^*1)^*10^*).$$

Problem 3.

For each of the following languages, decide whether it is regular or not. If it is regular, give a finite automaton or a regular expression for it. If it is not regular, use any combination of the pumping lemma, closure properties, or results *proved* in class.

The pumping lemma stipulates that:

Lemma 3.5. *Pumping lemma for regular languages.*

Let $L \subseteq \Sigma^*$ be a regular language over an alphabet Σ . Then, there exists a positive integer p , called the pumping length, such that for all $s \in L$, $|s| > p$, there exists $u, v, w \in \Sigma^*$ such that:

- (i) $s = uvw$
- (ii) $|uv| \leq p$
- (iii) $|v| > 0$
- (iv) $uv^i w \in L$ for all $i \geq 0$

- (a) $L_1 = \{0^m 1^n 0^{m+n} : m, n \geq 0\}$.

The language is not regular.

Assuming L_1 was regular and p was the pumping length for L_1 , take $s = 0^p 1^k 0^{p+k}$ for some $k \geq 0$. Better yet, set $k = 0$, so that $s = 0^p 0^p$. Following the pumping lemma, take $u = 0^a$, $v = 0^b$, $w = 0^{p-a-b} 0^p$. The pumping lemma stipulates that:

- (i) $|uv| \leq p$.
- (ii) $|v| = b > 0$.

If we pump down the string, we see then $uw \in L_1$, so $0^a 0^{p-a-b} 0^p = 0^{p-b} 0^p \in L_1$. Therefore, $p - b = p$ (by definition of L_1), implying that $b = 0$. However, the pumping stipulated that $b > 0$, so we have a contradiction, meaning L_1 may not be regular.

- (b) $L_2 = \{xwx^R : x, w \in \{0, 1\}^*, |x| > 0 \text{ and } |y| > 0\}$.

The language is not regular.

Assuming L_2 was regular and p was the pumping length for L_2 , Take $s = 0^p 10^p$. Following the pumping lemma, take $u = 0^a$, $v = 0^b$, $w = 0^{p-a-b} 10^p$. The pumping lemma stipulates that:

- (i) $|uv| \leq p$.
- (ii) $|v| = b > 0$.

If we pump down the string, we see then $uw \in L_2$, so $0^a 0^{p-a-b} 10^p = 0^{p-b} 10^p \in L_2$. This implies that $p - b = p$, so $b = 0$, contradicting the pumping lemma. Therefore, L_2 may not be regular.

(c) $L_3 = \{0^m 1^n : m \text{ divides } n\}$.

The language is not regular.

Assuming L_3 was regular and p was the pumping length for L_3 , Take $s = 0^p 1^p$ so that $s \in L_3$. Following the pumping lemma, take $u = 0^a, v = 0^b, w = 0^{p-a-b} 1^p$. The pumping lemma stipulates that:

(i) $|uv| \leq p$.

(ii) $|v| = b > 0$.

If we pump up the string, we see then $uv^2w \in L_3$, so $0^a 0^{2b} 0^{p-a-b} 1^p = 0^{p+b} 1^p \in L_3$. This implies that $p + b$ divides p , so $b = 0$, contradicting the pumping lemma. Therefore, L_3 may not be regular.

(d) $L_4 = \{0^n 1^p : n \leq 4 \text{ or } p \text{ is prime (or both)}\}$.

First, note that non-regular languages are closed under reversal, i.e. if L is not regular, then L^R is also not regular. The intuition is as follows: Given a regular language L with a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes L , we can easily construct an NFA for L_R as follows:

(i) Reverse all the transitions in M .

(ii) Add a new start state $s \notin Q$ with ε -transitions from s to every state $f \in F$.

(iii) Change all the accepting states to non-accepting states.

(iv) Change the old start state to not be a start state, but make it an accepting state.

Claim 3.6. L_4^R is not regular, therefore L_4 is not regular.

Proof. Note that $L_4^R = \{1^m 0^n : m \text{ is prime and } n \geq 4\}$.

Suppose p is the pumping length for L_4^R , pick $s = 1^p 0^5$. By the pumping lemma, take $u = 1^a, v = 1^b, w = 1^{p-a-b} 0^5$. The pumping lemma stipulates that $uv^k w \in L_4^R$ for all $k \geq 0$. However, note that:

$$\begin{aligned} uv^k w &= 1^a 1^{kb} 1^{p-a-b} 0^5 \in L_4^R \\ &\implies 1^{p+(k-1)b} 0^5 \in L_4^R \\ &\implies p + (k-1)b \text{ is prime.} \end{aligned}$$

Set $k = p + 1$, then $p + (k-1)b = p + pb = p(1+b)$. However, $b > 0$ by the pumping lemma, so $p(1+b)$ cannot be prime. This contradiction implies that L_4^R is not regular. Therefore, L_4 cannot be regular. \square

- (e) The infinite union $\bigcup_{n \geq 1} A_i$, where each A_i is a regular language. *The question should be interpreted as asking whether the union is **guaranteed** to be regular no matter how the sets A_i are chosen.*

No, the union is not guaranteed to be regular. For a counter-example, set $A_i = 0^i 1^i$ for all $i \geq 1$. Then, $\bigcup_{n \geq 1} A_i = \{0^n 1^n : n \geq 1\}$, which is not a regular language.

- (f) The infinite intersection $\bigcap_{n \geq 1} A_i$, where each A_i is a regular language. *The question should be interpreted as asking whether the intersection is **guaranteed** to be regular no matter how the sets A_i are chosen.*

No, the language is not guaranteed to be regular. For a counter-example: Let $P' = \{0, 1, 4, 6, 8, 9, 10, \dots\}$ be the set of all composite numbers. Let P_i refer to the i 'th element in the sequence.

Define $\Sigma = \{1\}$ and $A_i = 1^* - \{1^{P_i}\}$ for all $i \geq 1$. Then, define the language L as follows:

$$L = \bigcap_{n \geq 1} A_i = \{1^p : p \text{ is prime.}\}$$

As we proved in class, the language L is not regular.

Problem 4.

- (a) For a language A over alphabet Σ , define the relation \equiv_A on strings in Σ^* as follows: “ $x \equiv_A y$ ” means:

$$\forall w \in \Sigma^* (xw \in A \iff yw \in A).$$

Formally (and concisely) prove that \equiv_A is an equivalence relation.

If \equiv_A is an equivalence relation, then it must be reflexive, symmetric, and transitive.

(i) **Reflexivity:**

It is trivial to show that \equiv_A is reflexive: For any string x , if extending x with an arbitrary string w makes it a member of A , then the same extension will always make x a member of A . Thus, $x \equiv_A x$.

(ii) **Symmetry:**

Suppose $x \equiv_A y$. Then, for *all* strings $w \in \Sigma^*$, extending x with w makes it a member of A *if and only if* extending y with w makes it a member of A . The reverse must also hold: extending y with a string $w \in \Sigma^*$ makes it a member of A *if and only if* extending x with w also makes it a member of A . Therefore, $\equiv_A x$ whenever $x \equiv_A y$.

- (iii) **Transitivity:** Let $a, b, c \in \Sigma^*$ be such that $a \equiv_A b$ and $b \equiv_A c$. By definition of \equiv_A , $aw \in A \iff bw \in A$, and $bw \in A \iff cw \in A$.

Take w to be an arbitrary extension of a such that $aw \in A$, then we also have that $bw \in A$. However, since $b \equiv_A c$, this also means that $cw \in A$. Therefore, $aw \in A \iff cw \in A$, so $a \equiv_A c$.

- (b) The equivalence \equiv_A is called the *left equivalence relation* of the language A . An equivalence relation on a set partitions the set into disjoint subsets called equivalence classes in the following way: two elements belong to the same class iff they are related by the equivalence relation. Thus, \equiv_A , which is a relation on Σ^* , partitions Σ^* into equivalence classes: these are called the left equivalence classes of the language A .

For example, consider the language $C = \{x \in 0, 1^* : |x| \text{ is even}\}$ over the alphabet $\{0, 1\}$. Convince yourself that any two even-length strings are related by \equiv_C , as are any two odd-length strings. Also, no odd-length string is related by \equiv_C to an even-length string. Thus, C has exactly two left equivalence classes: (1) odd-length strings, i.e., $\{0, 1\}^* - C$, and (2) even-length strings, i.e., C .

Similarly, convince yourself that the language $B = (01)^*$ over the alphabet $\{0, 1\}$ has three equivalence classes, which are: (1) B , (2) $(01)^*0$, and (3) $\{0, 1\}^* - (B \cup (01)^*0)$.

Describe the left equivalence classes of each of the following languages (no proofs required):

- (i) $L_1 = \{a, aa, aaa, b, ba, baa\}$ over the alphabet $\{a, b\}$.

L_1 has four equivalence classes:

- (1) $\{a, b\}$,
- (2) $\{aa, ba\}$,
- (3) $\{aaa, baa\}$, and
- (4) $\{a, b\}^* - L_1$

- (ii) $L_2 = a^*b^*c^*$ over the alphabet $\{a, b, c\}$.

L_2 has four equivalence classes:

- (1) a^* ,
- (2) a^*b^+ ,
- (3) $a^*b^*c^+$, and
- (4) $\{a, b\}^* - L_2$

- (iii) $L_3 = (ab \cup ba)^*$ over the alphabet $\{a, b\}$.

L_3 has three equivalence classes:

- (1) $(ab \cup ba)^* = L_3$,
- (2) $(ab \cup ba)^*a$,
- (3) $(ab \cup ba)^*b$,
- (4) $\{a, b\}^* - ((ab \cup ba)^* \cup ((ab \cup ba)^*(a \cup b)))$

(iv) $L_4 = \{0^n 1^n : n \geq 0\}$ over the alphabet $\{0, 1\}$.

L_4 has infinite equivalence classes:

- | | |
|--|---|
| (1) 0^n for any $n \geq 0$ | (each n generates an equivalence class), |
| (2) $0^n 1^k$ for any $n > 0$ and $0 < k < n$ | (each $(n - k)$ generates an equivalence class) |
| (3) L_4 | (A single equivalence class) |
| (4) $\{0, 1\}^* - (L_4 \cup 0^n \cup 0^n 1^k)$ | (A single equivalence class) |

Problem 5.

Now, apply the notion of left equivalence to the theory of regular languages.

- (a) For a language A over alphabet Σ and a string $x \in \Sigma$, let $[x]_A$ denote the equivalence class (of A) to which x belongs. For instance, consider the set $X_1 = (01)^*$, $X_2 = (01)^*0$, and $X_3 = \{0, 1\}^* - (X_1 \cup X_2)$, then:
- (i) $[010]_B$ denotes the set X_2 ,
 - (ii) $[01010]_B$ also denotes the same set X_2 , and
 - (iii) $[\varepsilon]_B$ and $[0101]_B$ both denote the set X_1 .

There are rarely one unique way way to write an equivalence class $[x]_A$ — there are usually multiple choices for x . These choices are called *representatives* of the equivalence class.

Prove that for any $x \in \Sigma^*$ and $a \in \Sigma$, the class $[xa]_A$ is completely determined by the class $[x]_A$ and the alphabet symbol a — that is, prove that the particular x we pick as a representative for the class $[x]_A$ is inconsequential.

Recall the definition of equivalence classes: two strings x, y are in the same equivalence class iff $x \equiv_A y$, i.e. $xw \in A \leftrightarrow yw \in A$ for all $w \in \Sigma^*$. Consequently, we have that if $xw \in A$ for any string x in A , then $yw \in A$ for every other element y in the equivalence class. Therefore, the choice of a member of the equivalence class as a representative does not uniquely determine the set of transitions since every other member of the equivalence class would make the same set of accepting and rejecting transitions.

- (b) Suppose a language A over alphabet Σ has finitely many equivalence classes; $[x_1]_A, [x_2]_A, \dots, [x_n]_A$ for some $n \geq 1$. Prove that A is regular.

Let A be a language with finitely many equivalence classes. As shown in part (a) above, for any equivalence class $[x]_A$, the equivalence class $[xa]_A$ is not dependent on the choice of x as a representative. Therefore, we can capture each unique equivalence class $[x]_A$ with a single state in an automaton and, since A has finitely many equivalence classes, we can construct an automaton with finite states that accepts A .

Problem 6.

The connection between left equivalence and regularity is even deeper.

- (a) Let A be a regular language over alphabet Σ . Prove (formally and rigorously) that A has finitely many distinct left equivalence classes. The function δ^* we defined in class might be useful.

Let A be a regular language over an alphabet Σ .

Claim 6.7. *A has finitely many distinct left equivalence classes.*

Proof. Since A is regular, there exists a DFA

$$M = (Q, \Sigma, \delta, q_0, F),$$

where Q is a finite set, that recognizes A .

Define the function $\delta^* : Q \times \Sigma^* \rightarrow Q$ as follows:

$$\delta^*(q, x) = \begin{cases} q & \text{if } x = \varepsilon \\ \delta^*(\delta(q, x_1), x_2 \cdots x_n) & \text{if } x = x_1 x_2 \cdots x_n \text{ with each } x_i \in \Sigma \end{cases}$$

First, we show that each distinct equivalence class can be fully captured by a single state. Take any equivalence class $[x]_A \in A$. Then if any string $w \in A$ extends a member of $[x]_A$ to a member of A , then w extends *all members* of $[x]_A$ to a member of A . Therefore, we can fully capture $[x]_A$ by a single state q in M , since all members of $[x]_A$ have matching sequences of transitions out of q to accepting / rejecting states.

Next, we show that no two distinct equivalence classes can be captured by the same state. Take $[x]_A$ and $[y]_A$ as two distinct equivalence classes in A , such that $x_w \in A$ and $y_w \notin A$ for some $w \in \Sigma^*$. Suppose the two classes, $[x]_A$ and $[y]_A$, are captured by the same state q in M . Since M is a DFA, it always makes the same sequence of transitions out of q so it would be impossible for it to accept the string xw and reject the string yw . Therefore, $[x]_A$ and $[y]_A$ cannot be captured by the same state q in M .

Finally, we tie up the above conditions to prove that A has a finite number of distinct equivalence classes. Recall that A is regular, and M recognizes M . As shown above, each state $q \in Q$ may fully capture a distinct equivalence class, but it may not capture *more than one* distinct equivalence class. This means that:

Corollary 6.8. *If L is a regular language and $M_L = (Q_L, \Sigma_L, \delta_L, q_0, F_L)$ is a DFA that recognizes L , then number of states in M_L is greater than or equal to the number of equivalence classes in L , i.e. if $\{\#[x]_L : x \in L\}$ is the set of **distinct** equivalence classes in L ,*

$$|Q_L| \geq |\{\#[x]_L : x \in L\}|$$

By the corollary, the number of equivalence classes of a language A may not be more than the number of states in a DFA that recognizes A . However, M , our original DFA recognizing has a finite number of states, so A must have a finite number of distinct left equivalence classes. \square

- (b) Using one or more of the results proved above, give alternate proofs that the following languages are not regular:

- (i) $L_4 = \{0^n 1^n : n \geq 0\}$.

Suppose L_4 is regular, and M_4 is a DFA that recognizes L_4 . By corollary 6.8, M_4 must have at least as many states as L_4 has equivalence classes. But, as shown in problem 4.2, part (iv) L_4 has infinite equivalence classes, meaning M_4 must have infinite states, and DFAs must have a *finite* number of states.

Therefore, no DFA can recognize L_4 , so L_4 is not regular.

- (ii) $L = \{x \in \{0, 1\}^* : x = x^R\}$.

We start by noting that L has infinite equivalence classes. After reading a string x , we only accept if:

- (i) $x \in L$
- (ii) we read a string $w = x^R$
- (iii) We read some string w such that $xw = yy^R$ for some $y \in \{0, 1\}^*$.

Since $\{0, 1\}^*$ is infinite, there are infinite such strings x , therefore infinite equivalence classes.

Now, suppose L is regular, and M is a DFA that recognizes L . By corollary 6.8, M must have at least as many states as L has equivalence classes. But, as shown above, L has infinite equivalence classes, meaning M must have infinite states, and DFAs must have a *finite* number of states.

Therefore, no DFA can recognize L , so L is not regular.