Assignment 3

Question 1. [16 MARKS]

Given a list L, a *contiguous* sublist M of L is a sublist of L whose elements occur in immediate succession in L. For instance, $[4, 7, 2]$ is a contiguous sublist of $[0, 4, 7, 2, 4]$ but $[4, 7, 2]$ is not a contiguous sublist of $[0, 4, 7, 1, 2, 4]$.

We consider the problem of computing, for a list of integers L, a contiguous sublist M of L with maximum possible sum.

Algorithm 1 $MaxSublist(L)$

¡precondition¿: L is a list of integers.
¡postcondition¿: Return a contiguous sublist of L with maximum possible sum.

Part (1) [5 MARKS]

Using a divide-and-conquer approach, devise a recursive algorithm which meets the requirements of $MaxSublist$.

Solution:

Our algorithm will rely on the following subprocess.

Algorithm 2 $MaxSublistCrossing(L, i)$

¡precondition¿:

L is a list of integers and i is an integer index, $0 \leq i < |L| - 1$.

¡postcondition¿:

Return a contiguous sublist of L which crosses index i and achieves maximum possible sum subject to this requirement.

```
r ← i + 1
S ← 0
r_max ← i + 1
S_max ← 0
while r < |L| do
    S ← S + L[r]
    r ← r + 1
    if S > S_max then
        S_max ← S
        r_max ← r
    end if
end while

ℓ ← i − 1
S ← 0
ℓ_max ← i − 1
S_max ← 0
while ℓ > 0 do
    S ← S + L[ℓ − 1]
    ℓ ← ℓ − 1
    if S > S_max then
        S_max ← S
        ℓ_max ← k
    end if
end while

return L[ℓ_max : r_max]
```

Then our main algorithm may be defined as follows.

Algorithm 3 $MaxSublist(L)$

¡precondition¿: L is a list of integers.
¡postcondition¿: Return a contiguous sublist of L with maximum possible sum.

```
if |L| = 0 then
    return L
else if |L| = 1 and L[0] > 0 then
    return L
else if |L| = 1 and L[0] ≤ 0 then
    return [ ]
else
    n ← |L|
    L_1 ← MaxSublist(L[0 : ⌈n/2⌉])
    L_2 ← MaxSublist(L[⌈n/2⌉ : n])
    L_3 ← MaxSublistCrossing(L, ⌈n/2⌉)
    return L_i which has maximum sum
end if
```

Part (2) [8 MARKS]

Give a complete proof of correctness for your algorithm. If you use an iterative subprocess, prove the correctness of this also.

Solution:

First, we prove the correctness of the iterative subprocess $MaxSublistCrossing$.

We will handle each of the loops separately.

LOOP 1.

Define a loop invariant

$\boldsymbol{LI}_1(r, S, r_{max}, S_{max})$:

- $r \in \mathbb{N}$ satisfies $r \leq |L|$
- $S = \text{sum}(L[i+1 : r])$
- $r_{max} = \arg\max_{t: i+1 \leq t \leq r} \text{sum}(L[i+1 : t])$
- $S_{max} = \text{sum}(L[i+1 : r_{max}])$

Establish loop invariant

At the start of the first iteration of the first loop, $r = i+1$, $S = 0$ and $r_{max} \leftarrow i+1$. Hence,

- $S = 0 = \text{sum}(L[i+1 : i+1]) = \text{sum}(L[i+1 : r])$
- $r_{max} = i+1 = \arg\max_{t: i+1 \leq t \leq i+1} \text{sum}(L[i+1 : t]) = \arg\max_{t: i+1 \leq t \leq r} \text{sum}(L[i+1 : t])$
- $S_{max} = 0 = \text{sum}(L[i+1 : i+1]) = \text{sum}(L[i+1 : r_{max}])$

Therefore, $\boldsymbol{LI}_1(r, S, r_{max}, S_{max})$ holds.

Maintain loop invariant

Now let $r^0, S^0, r^0_{max}, S^0_{max}$ be the values of r, S, r_{max}, S_{max} at the start of an arbitrary iteration.

Let $r^1, S^1, r^1_{max}, S^1_{max}$ be the values of r, S, r_{max}, S_{max} at the end of that iteration.

Assume $\boldsymbol{LI}_1(r^0, S^0, r^0_{max}, S^0_{max})$. Note:

- Since r^0 did not satisfy the exit condition, then $r^0 < |L|$ and hence $r^1 = r^0 + 1$ satisfies $r^1 \leq 1$.
- Since $S^1 = S^0 + L[r^0]$ and $S^0 = \text{sum}(L[i+1 : r^0])$, we have that

$$S^1 = \text{sum}(L[i+1 : r^0 + 1]) = \text{sum}(L[i+1 : r^1])$$

- Since

$$r^0_{max} = \underset{t: i+1 \leq t \leq r^0}{\arg\max}\ \text{sum}(L[i+1 : t])$$

  then

$$\underset{t: i+1 \leq t \leq r^1}{\arg\max}\ \text{sum}(L[i+1 : t])$$

  is either r^0_{max} or r^1 according to which of $\text{sum}(L[i+1 : r^0_{max}])$ or $\text{sum}(L[i+1 : r^1])$ is greater. $MaxSublistCrossing$ identifies these cases appropriately so that

$$r^1_{max} = \underset{t: i+1 \leq t \leq r^1}{\arg\max}\ \text{sum}(L[i+1 : t])$$

- If $r^1_{max} = r^0_{max}$, then $\text{sum}(L[i+1 : r^1_{max}]) = \text{sum}(L[i+1 : r^0_{max}])$ which agrees with the fact that S^1_{max} is taken to be S^0_{max}. If $r^1_{max} \neq r^0_{max}$, then S^1_{max} is taken to be

$$S^1 = \text{sum}(L[i+1 : r^1]) = \text{sum}(L[i+1 : r^1_{max}])$$

Loop invariant and exit condition

The exit condition is $r \geq |L|$. Since r is an index of L, this is satisfied when $r = |L| - 1$. Assuming this together with $\boldsymbol{LI}(r, S, r_{max}, S_{max})$, we get that

$$r_{max} = \underset{t: i+1 \leq t \leq |L|-1}{\arg\max}\ \text{sum}(L[i+1 : t]) \tag{1}$$

Termination

Define the measure of progress as r. Since r increases by 1 with each iteration, and the algorithm terminates when $r > |L| - 2$, we may conclude that the algorithm terminates.

LOOP 2.

Following a similar argument, we may show that the second loop terminates at which point

$$\ell_{max} = \underset{t: 0 \leq t \leq i}{\arg\max}\, \text{sum}(L[t : i]) \tag{2}$$

POSTCONDITION

Putting together (1) and (2), we get that $L[\ell_{max} : r_{max}]$ is the maximum-sum sublist of L which includes index i. Since this is what is returned, the postcondition is obtained.

It remains to show the correctness of $MaxSublist$.

Base case

The base cases are where $|L| = 0$ or $|L| = 1$. It is easy to check that, on these inputs, the output satisfies the postcondition.

Recursive step

Consider the recursive case where $|L| \geq 2$.

Then recursive calls are made on inputs $L\left[0 : \lceil \frac{n}{2} \rceil\right]$ and $L\left[\lceil \frac{n}{2} \rceil : n\right]$ which both satisfy the algorithms precondition. Thereby, we may assume that L_1 and L_2 satisfy the postconditions of $MaxSublist\left(L\left[0 : \lceil \frac{n}{2} \rceil\right]\right)$ and $MaxSublist\left(L\left[\lceil \frac{n}{2} \rceil : n\right]\right)$ respectively.

Furthermore, we have already shown the correctness of $MaxSublistCrossing$ so, because $(L, \lceil \frac{n}{2} \rceil)$ satisfies the precondition of $MaxSublistCrossing$, then L_3 satisfies the postcondition of $MaxSublistCrossing(L, \lceil \frac{n}{2} \rceil)$.

It follows that

- L_1 is the maximum-sum contiguous sublist of $L\left[0 : \lceil \frac{n}{2} \rceil\right]$
- L_2 is the maximum-sum contiguous sublist of $L\left[\lceil \frac{n}{2} \rceil : n\right]$
- L_3 is the maximum-sum contiguous sublist of L which includes index $\lceil \frac{n}{2} \rceil$.

Since the maximum-sum contiguous sublist of L must be one of these, returning whichever of L_1, L_2 or L_3 has maximum sum meets the postcondition.

Termination

i. Take the measure of input size to be $|L|$.

ii. Recursive calls are made when $|L|$ is at least 2 and are made on sublists of length at most $\lceil \frac{|L|}{2} \rceil$. Since $|L| \geq 2$, then $\lceil \frac{|L|}{2} \rceil < |L|$. Thus, the measure of input size goes down by at least one with each recursive call.

iii. When $|L| \leq 1$, then a base case is executed.

Part (3) [3 MARKS]

Analyze the running time of your algorithm.

Solution:

Treating each addition as a constant-time operation, we have that $MaxSublistCrossing(L)$ runs in linear time $O(n)$ where n is the length of L.

Then the running time of $MaxSublist$ on a list of length n may be given by the recurrence

$$T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n)$$

Hence, the appropriate parameters for application of the Master Theorem are $a = 2$, $b = 2$ and $k = 1$. Since $\log_b a = \log_2 2 = 1 = k$ we may conclude that $T(n) = O(n \cdot \log n)$.

Question 2. [18 MARKS]

For a point $x \in \mathbb{Q}$ and a closed interval $I = [a, b]$, $a, b \in \mathbb{Q}$, we say that I covers x if $a \leq x \leq b$. Given a set of points $S = \{x_1, \ldots, x_n\}$ and a set of closed intervals $Y = \{I_1, \ldots, I_k\}$ we say that Y covers S if every point x_i in S is covered by some interval I_j in Y.

In the "Interval Point Cover" problem, we are given a set of points S and a set of closed intervals Y. The goal is to produce a minimum-size subset $Y' \subseteq Y$ such that Y' covers S.

Consider the following greedy strategy for the problem.

Algorithm 4 $Cover(S, Y)$

¡precondition¿:

S is a finite collection of points in $\mathbb{Q}$. Y is finite set of closed intervals which covers S.

¡postcondition¿:

Return a subset Z of Y such that Z is the smallest subset of Y which covers S.

$L = \{x_1, \ldots, x_n\} \leftarrow S$ sorted in nondecreasing order
$Z \leftarrow \emptyset$
$i \leftarrow 0$
while $i < n$ **do**
  if x_{i+1} is not covered by some interval in Z **then**
    $I \leftarrow$ interval $[a, b]$ in Y which maximizes b subject to $[a, b]$ containing x_{i+1}
    Z.append(I)
  end if
  $i \leftarrow i + 1$
end while
return Z

Give a complete proof of correctness for *Cover* subject to its precondition and postcondition.

Solution:

Define the loop invariant

$\boldsymbol{LI}(Z, i)$:

- Z covers $\{x_1, \ldots, x_i\}$.
- There exists a set W, $Z \subseteq W \subseteq Y$, where W is a minimum-size subset of Y which covers S.

Establish loop invariant

At the start of the first iteration, $Z = \emptyset$ and $i = 0$. In this case, the first bullet of the loop invariant says that Z covers $\emptyset$. This is vacuously true. Furthermore, since Y covers S, there must exist some set minimum-size $W \subseteq$ which covers S. Since $Z = \emptyset$, then $Z \subseteq W$. Therefore, $\boldsymbol{LI}(Z, i)$.

Maintain the loop invariant

Let Z_0, i_0 be the values of Z and i at the start of of an arbitrary iteration.

Let Z_1, i_1 be the values of Z and i at the end of of that iteration.

Assume $\boldsymbol{LI}(Z_0, i_0)$.

Then Z_0 covers $\{x_1, \ldots, x_{i_0}\}$ and there exists W_0, $Z \subseteq W_0 \subseteq Y$, which is a minimum-size subset of Y which covers S.

Case 1.

If Z_0 already covers x_{i_0+1}, then $Z_1 = Z_0$. In this case, Z_1 covers $\{x_1, \ldots, x_{i_0+1}\}$ and $Z_1 \subseteq W_0$, from which $\boldsymbol{LI}(Z_1, i_1)$ follows.

Case 2.

If Z_0 does not cover x_{i_0+1}, then Z_1 is obtained from Z_0 by adding an interval $I = [a, b] \in Y$ which maximizes b subject to being an interval which covers x_{i_0+1}.

W_0 must cover x_{i_0+1} by some interval $J = [c, d]$. Let $W_1 = (W_0 \setminus \{J\}) \cup \{I\}$ so that $Z_1 \subseteq W_1 \subseteq Y$. We want to show that W_1 covers S.

Note:

- $Z_0 \subseteq W_1$ implies that W_1 covers $\{x_1, \ldots, x_{i_0}\}$.
- Everything not covered by J is covered by $W_0 \setminus \{J\}$.
- Since $I = [a, b]$ was chosen to maximize b, then $J = [c, d]$ must satisfy $d \leq b$. Thus, for $k \geq i_0 + 1$, if x_k is covered by J, then x_k is covered by I.

Putting these facts together, we may conclude that W_1 covers S. Also, since W_1 contains the same number of intervals as W_0, and W_0 is a minimum subset which covers S, then W_1 is a minimum set which covers S.

Using the fact that $i_1 = i_0 + 1$, we may conclude that $\boldsymbol{LI}(Z_1, i_1)$ holds.

Loop invariant and exit condition imply post-condition

Assume $\boldsymbol{LI}(Z, i)$ and that the exit condition $i \geq n$ holds.

Then,

- Z covers $S = \{x_1, \ldots, x_n\}$;
- There exists a set W, $Z \subseteq W \subseteq Y$, where W is a minimum-size subset of Y which covers S.

Since W is a minimum-size subset of Y which covers S, then $Z \subseteq W$ implies that Z is a minimum-size subset of Y which covers S.

Since Z is what is returned by the algorithm, the postcondition is obtained.

Termination

i. Let i be the measure of progress.

ii. i increases by at least 1 with each iteration.

iii. When $i \geq |S|$, the loop terminates.

Question 3. [10 MARKS]

The first three parts of this question deals with properties of regular expressions (this is question 4 from section 7.7 of Vassos' textbook). Two regular expressions R and S are equivalent, written $R \equiv S$ if their underlying language is the same i.e. $\mathcal{L}(R) = \mathcal{L}(S)$. Let R, S, and T be arbitrary regular expression. For each assertion, state whether it is true or false and justify your answer.

Part (1) [2 MARKS]

$$\text{If } RS \equiv SR \text{ then } R \equiv S.$$

Solution: This assertion is false. Consider the following counter example where the underlying alphabet is $\Sigma = \{0, 1\}$: let $R = \epsilon$ and $S = 1$. Then $RS = 1 = SR$, but clearly $\mathcal{L}(R) = \{\epsilon\} \neq \{1\} = \mathcal{L}(S)$.

Part (2) [2 MARKS]

$$\text{If } RS \equiv RT \text{ and } R \not\equiv \emptyset \text{ then } S \equiv T.$$

Solution: This assertion is also false. Consider the following counter example (again the underlying alphabet is $\Sigma = \{0, 1\}$). Let $R = 0^*$, $S = \epsilon$, and $T = 0^*$. Observe that $RS = R = RT$. $\mathcal{L}(S) \neq \mathcal{L}(T)$ so $S \not\equiv T$.

Part (3) [2 MARKS]

$$(RS + R)^*R \equiv R(SR + R)^*.$$

Solution: This is true! Consider any string w that is in $\mathcal{L}\left((RS + R)^*R\right)$. $w = a_1 a_2 \cdots a_{n-1} b_{n,1}$ where $a_i \in \mathcal{L}\left(RS + R\right)$ for $i \in [n-1]$ and $b_{n,1} \in \mathcal{L}(R)$. We can further decompose each a_i as follows:

$$a_i = \begin{cases} b_{i,1}b_{i,2} & \text{where } b_{i,1} \in R, b_{i,2} \in S \text{ if } a_i \in RS \\ b_{i_1} & \text{where } b_{i,1} \in R \text{ if } a_i \in \mathcal{L}(R) \end{cases}$$

We show that $w \in \mathcal{L}(R(SR + R)^*)$ by considering an alternate decomposition of w. Note that $b_{1,1} \in R$. If a_i has block $b_{i,2}$, then $b_{i,2}b_{i+1,1} \in \mathcal{L}(SR)$ remark $b_{i+1,1}$ exists since $i \in [n-1]$. Thus $w \in \mathcal{L}(R(SR + R)^*)$ demonstrating that $(RS + R)^*R \equiv R(SR + R)^*$.

Part (4) [4 MARKS]

Prove or disprove the following statement: for every regular expression R, there exists a FA M such that $\mathcal{L}(R) = \mathcal{L}(M)$. *Note*: even if you find the proof of this somewhere else, please try to write up the proof in your own words. Just citing the proof is *NOT* enough.

Solution: This statement is true. Observe that a regular expression is defined recursively, so it is natural that We will prove this using structural induction. Our predicate is

$$P(R) := \text{"there exists an FA } M \text{ such that } \mathcal{L}(R) = \mathcal{L}(M)\text{"}.$$

We will show that $P(R)$ is true for all regular expressions R.

Suppose the underlying alphabet is In the base case we need to come up with DFAs M_0, M_1, M_2 for regular expressions $R_0 = \emptyset$, $R_1 = \{\epsilon\}$, and $R_2 = \{a\}$. These can be seen in Figure 1, Figure 2, and 3 respectively.

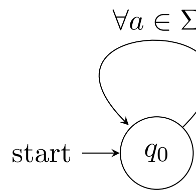

Figure 1: M_0.

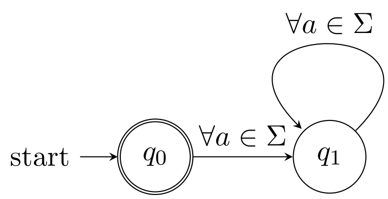

Figure 2: M_1.

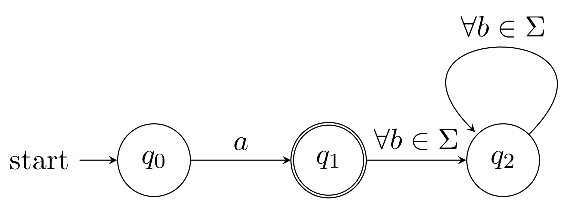

Figure 3: M_2.

Next, for the inductive hypothesis, we will suppose that for regular expressions S, T, $P(S)$ and $P(T)$ are true. That is there exists finite automaton M_S and M_T such that $\mathcal{L}(S) = \mathcal{L}(M_S)$ and $\mathcal{L}(T) = \mathcal{L}(M_T)$. We will show that for regular expressions $S + T$, ST, and S^* we can find equivalent FA $M_{(S+T)}$, $M_{(ST)}$, and M_{S^*}. You might find it easer to use the formal definition of finite automaton. Suppose M_S and M_T are defined as follows:

$$M_S = (Q_S, \Sigma, \delta_S, b_S, F_S) \text{ and } M_T = (Q_T, \Sigma, \delta_T, b_T, F_T).$$

$M_{(S+T)}$: Let $\delta_+ : (Q_S \cup Q_T \cup \{q_0\}) \times \Sigma \to (Q_S \cup Q_T)$ such that

$$\delta_+(q, a) = \begin{cases} \{b_S, b_T\} & \text{if} q = q_0 \\ \delta_s(q, a) & \text{if } q \in Q_S \\ \delta_t(q, a) & \text{if } q \in Q_T \end{cases}$$

Then $M_{(S+T)}$ can be defined as

$$M_{(S+T)} = (Q_S \cup Q_T \cup \{q_0\}, \Sigma, \delta_+, q_0, F_S \cup F_T).$$

See Figure 4 for the state diagram.

$M_{(ST)}$: Let $\delta_\cdot : (Q_S \cup Q_T) \times \Sigma \to (Q_S \cup Q_T)$ such that

$$\delta_\cdot(q, a) = \begin{cases} b_T & \text{if } q = f_S \\ \delta_s(q, a) & \text{if } q \in Q_S \\ \delta_t(q, a) & \text{if } q \in Q_T \end{cases}$$

Then $M_{(ST)}$ can be defined as

$$M_{(ST)} = (Q_S \cup Q_T, \Sigma, \delta_\cdot, b_S, F_T).$$

See Figure 5 for the state diagram.

M_{S^*}: Let $\delta_* : (Q_S \cup \{q_0\}) \times \Sigma \to (Q_S \cup \{q_0\})$ such that

$$\delta_*(q, a) = \begin{cases} q_0 & \text{if } q = f_S, \text{ and} \\ \delta_s(q, a) & \text{if } q \in Q_S \end{cases}$$

Then M_{S^*} can be defined as

$$M_{S^*} = (Q_S \cup \{q_0\}, \Sigma, \delta_*, q_0, F_S).$$

See Figure 6 for the state diagram.

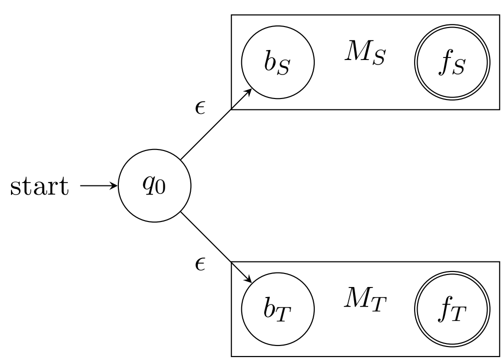

Figure 4: $M_{(S+T)}$.

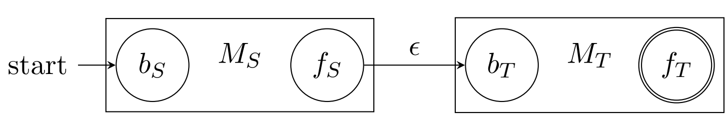

Figure 5: $M_{(ST)}$.

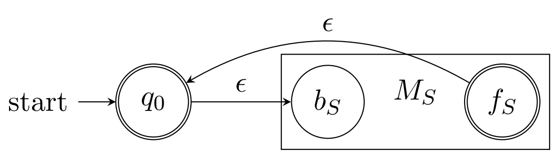

Figure 6: M_{S^*}.

Note: this is actually part of the proof to show the equivalence of regular expressions and finite automaton. In particular we will show DFA $\rightarrow$ regular expression $\rightarrow$ NDFA $\rightarrow$ DFA. If you see that a student did not do this part well, can you please put a comment to the effect of: *please look at the solution guide, this is important for an in-class proof.*

Question 4. [16 MARKS]

In the following, for each language L over the alphabet $\Sigma = \{0, 1\}$ construct a regular expression R and a DFA M such that $\mathcal{L}(R) = \mathcal{L}(M) = L$. Prove the correctness of your DFA.

Part (1) [8 MARKS]

Let $L_1 = \{x \in \{0,1\}^* :$ the first and last charactes of x are the same$\}$. Note: $\epsilon \notin L$ since ϵ does not have a first or last character.

Solution: Let R_1 be the following regular expression:

$$R_1 = 0 + 1 + 0(0+1)^*0 + 1(0+1)^*1$$

and M_1 be the DFA shown in Figure 7. We claim that $L_1 = \mathcal{L}(M_1)$.

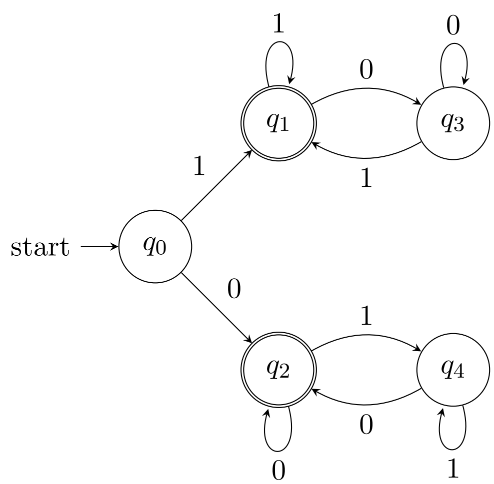

Figure 7: Solution DFA for L_1.

Claim 1. $L_1 = \mathcal{L}(M_1)$.

Proof. We will prove the claim by structural induction. First we definite our predicate:

$$P(x) := \delta^*(q_0, x) = \begin{cases} q_0, & \text{if } x = \epsilon \\ q_1, & \text{if } x \text{ starts and ends with a one} \\ q_2, & \text{if } x \text{ starts and ends with a zero} \\ q_3, & \text{if } x \text{ starts with a one and ends with a zero} \\ q_4, & \text{if } x \text{ starts with a zero and ends with a one} \end{cases}$$

We claim that $P(x)$ is true for all $x \in \{0,1\}^*$.

In the base case $x = \epsilon$. By inspection, $P(x)$ holds. In the Inductive step $x = ya$ for some $y \in \{0,1\}^*$ and $a \in \{0,1\}$. We assume that $P(y)$ holds. There are two cases to consider:

$a = 0$ We need to consider every possible value of $\delta^*(q_0, y)$. Observe that

$$\delta^*(q_0, x) = \delta(\delta^*(q_0, y), 0) = \begin{cases} q_2, & \text{if } \delta^*(q_0, y) \in \{q_0, q_2, q_4\} \\ q_3, & \text{if } \delta^*(q_0, y) \in \{q_1, q_3\} \end{cases}$$

By the induction hypothesis we know that $y \in \{q_0, q_2, q_4\}$ if and only if $y = \epsilon$ or y starts with a 0 and $y \in \{q_1, q_3\}$ if and only if y starts with a 1. Thus $\delta^*(q_0, x) \in q_2$ if and only if x starts with a zero and ends in a zero and $\delta^*(q_0, x) \in q_3$ if and only if x starts with a one (and ends in a zero).

$a = 1$ Again we need to consider all possible values of $\delta^*(q_0, y)$. The details will be omitted since this is similar to the previous case.

Since the state invariants are disjoint and exhaustive, we have shown that $L_1 = \mathcal{L}(M_1)$ as required. □

Part (2) [8 MARKS]

Let a *block* be a maximal sequence of identical characters in a finite string. For example, the string 0010101111 can be broken up into blocks: 00, 1, 0, 1, 0, 1111. Let $L_2 = \{x \in \{0,1\}^* : x \text{ only contains blocks of length at least three}\}$.

Solution: Let R_2 be the regular expression:

$$R_2 = (000(0)^* + 111(1)^*)^*$$

and M_2 be the DFA shown in Figure 8. We claim that $L_2 = \mathcal{L}(M_2)$.

Claim 2. *$L_2 = \mathcal{L}(M_2)$.*

Proof. We will prove the claim by structural induction. First we definite our predicate:

$$P(x) := \delta^*(q_0, x) = \begin{cases} q_0, & \text{if } x = \epsilon \\ q_1, & \text{if the last block of } x \text{ consists of one zero} \\ q_2, & \text{if the last block of } x \text{ consists of two zeros} \\ q_3, & \text{if the last block of } x \text{ consists of at least three zeros} \\ q_4, & \text{if the last block of } x \text{ consists of one one} \\ q_5, & \text{if the last block of } x \text{ consists of two ones} \\ q_6, & \text{if the last block of } x \text{ consists of at least three ones} \\ g, & \text{otherwise} \end{cases}$$

We claim that $P(x)$ is true for all $x \in \{0,1\}^*$.

In the base case $x = \epsilon$. By inspection, $P(x)$ holds. In the Inductive step $x = ya$ for some $y \in \{0,1\}^*$ and $a \in \{0,1\}$. We assume that $P(y)$ holds. There are two cases to consider:

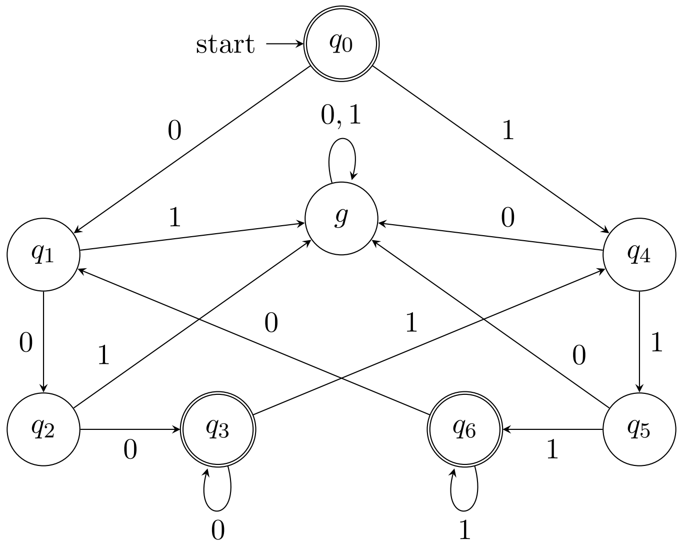

Figure 8: Solution DFA for L_2.

$a = 0$ We need to consider every possible value of $\delta^*(q_0, y)$. Observe that

$$\delta^*(q_0, x) = \delta(\delta^*(q_0, y), 0) = \begin{cases} q_1, & \text{if } \delta^*(q_0, y) \in \{q_0, q_6\} \\ q_2, & \text{if } \delta^*(q_0, y) = q_1 \\ q_3, & \text{if } \delta^*(q_0, y) \in \{q_2, q_3\} \\ g, & \text{otherwise} \end{cases}$$

By the induction hypothesis $y \in \{q_0, q_6\}$ if and only if $y = \epsilon$ or y ends with a block of at least three 0. Thus $x = y0$ in this case is a string whose only/last block consists of one 0. Similarly $\delta^*(q_0, y) = q_1$ and $\delta^*(q_0, y) \in \{q_2, q_3\}$ if and only if y ends with a block of one 0, two 0s and at least three 0s respectively. Thus $x = y0$ ends with a block of two 0s or at least three 0s respectively. In all other cases $x = y0$ will either contain a block of 1s of size at most two or a block of 0s of size at most two.

$a = 1$ This is identical to the previous case except that all instances of one and zero swap places.

Since the state invariants are disjoint and exhaustive (garbage state g covers all cases not previously specified), we have shown that $L_2 = \mathcal{L}(M_2)$ as required. □