CSC236: Introduction to the Theory of Computation

By: Eric Koehli, Osama Sohail

August 3, 2018

Problem Set 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 the maximum sum of a contiguous sublist of L.

Part (1) [5 MARKS]

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

```
    def max_sublist_sum(lst: List[int]) -> int:
        """
        Pre: <lst> is a list of integers.
        Post: returns a contiguous sublist of <lst> with the
        maximum possible sum.
        Note: This could also be implemented with indices.
        """
        n = len(lst)

        # base case
        if n <= 1:
            return max(lst[0], 0)  # empty sublist is zero,
                                   # which is better than negative

        else:
            # divide the list
            mid = n // 2        # note: n is at least 2, so mid is at least 1

            # conqueur the list
            left_sum = max_sublist_sum(lst[:mid])   # from index 0...mid - 1
            right_sum = max_sublist_sum(lst[mid:])  # from index mid...n - 1
            cross_sum = max_crossing_sum(lst, mid, n)

            return max(left_sum, right_sum, cross_sum)
```

```python
def max_crossing_sum(lst: List[int], mid: int, n: int) -> int:
    """
    Parameter <mid> is the floor middle index of <lst>.
    Parameter <n> is the length of the input list <lst>.
    Pre: <lst> is a list of integers and len(lst) >= 2.
    Post: returns the maximum contiguous crossing sum
    starting from the middle of <lst>.
    """
    left_sum, right_sum, total = 0, 0, 0 # initialize values

    # max sum the left half
    k = mid - 1
    i = 0
    while i < mid:
        total += lst[k - i]
        i += 1
        if total > left_sum:
            left_sum = total

    i, total = 0, 0              # reset values
    # max sum the right half
    for i in range(mid, n):      # iterate from index mid...n - 1
        total += lst[i]
        if total > right_sum:
            right_sum = total

    # note: left_sum and right_sum are each at least zero
    crossing_sum = left_sum + right_sum
    return crossing_sum
```

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.

We will first prove the correctness of *max_crossing_sum*.

Proof. **Loop 1**

For this proof, let $A =$ `lst` and $m =$ `len(A[:mid])`. Define the loop invariant:

$$Inv(i, total, left\_sum) : 0 \leqslant i \leqslant m \wedge total = \sum_{j=0}^{i-1} A[k-j] \wedge left\_sum_i = \max(total_i, left\_sum_{i-1})$$

where we define $left\_sum_{-1}$ to be zero.

Base Case:

When the loop is reached, $i = 0$, $total = 0$, and $left\_sum_0 = 0$. So our invariant is

$$Inv(0, 0, 0) : 0 \leqslant 0 \leqslant m \wedge 0 = \sum_{j=0}^{-1} A[k-j] \wedge 0 = \max(0, 0)$$

which is true.

Inductive Step:

Assume that the invariant is true before an arbitrary iteration of the loop. Let i_0, $total_0$, $left\_sum_0$, i_1, $total_1$, $left\_sum_1$ be the values of i, $total$, and $left\_sum$ before and after an arbitrary loop, respectively. Assume $Inv(i_0, total_0, left\_sum_0)$ is true and that $i_0 < m$. We want to show that $Inv(i_1, total_1, left\_sum_1)$ is true. That is,

$$Inv(i_1, total_1, left\_sum_1) :$$

$$0 \leqslant i_1 \leqslant m \wedge total_1 = \sum_{j=0}^{i_1-1} A[k-j] \wedge left\_sum_1 = \max(total_1, left\_sum_0)$$

From the loop body, we have

(a) $i_1 = i_0 + 1 \implies i_0 = i_1 - 1$

(b) $total_1 = total_0 + A[k - i_0]$

(c) $left\_sum_1 = \max(total_1, left\_sum_0)$

Showing conjunct 1: $0 \leqslant i_1 \leqslant m$

The first part is true because $0 \leqslant i_0 \implies 1 \leqslant i_0 + 1 = i_1$. The second part is true because $i_0 < m \implies i_0 + 1 = i_1 \leqslant m$.

Showing conjunct 2: $total_1 = \sum_{j=0}^{i_1-1} A[k-j]$. From the loop body, we have

$$\begin{aligned}
total_1 &= total_0 + A[k-i_0] \\
&= \sum_{j=0}^{i_0-1} A[k-j] + A[k-i_0] \\
&= \sum_{j=0}^{i_0} A[k-j] \\
&= \sum_{j=0}^{i_1-1} A[k-j] && \text{(by (a))}
\end{aligned}$$

Showing conjunct 3: $left\_sum_1 = \max(total_1, left\_sum_0)$.

But this is exactly what we have in the body of our loop for part (c). Note that $left\_sum_i$ is at least zero since we defined $left\_sum_{-1} = 0$. That is, it's base case is defined as zero. Thus we can conclude that $left\_sum$ is the maximum contiguous sum of integers starting from the middle of the given list and iterating to the left.

Loop 2

For the second loop, we define a new loop invariant. For this loop invariant, let $m = mid$ from the code:

$$Inv(i, total, right\_sum) : m \leqslant i \leqslant n \wedge total = \sum_{j=m}^{i-1} A[j] \wedge right\_sum_i = \max(total_i, right\_sum_{i-1})$$

where again, we define $right\_sum_{-1} = 0$.

Base Case:

When the second loop is reached, $i = m$, $total = 0$, $right\_sum = 0$. Then we have

$$Inv(m, 0, 0) : m \leqslant m \leqslant n \wedge 0 = \sum_{j=m}^{m-1} A[j] \wedge 0 = \max(0, 0)$$

which is true. Note that the sum above is equal to zero by definition of a sum over an empty range.

Inductive Step:

Assume that the invariant is true before an arbitrary iteration of the loop. Let i_0, $total_0$, $right\_sum_0$, i_1, $total_1$, $right\_sum_1$ be the values of i, $total$, and $right\_sum$ before and after an arbitrary loop, respectively. Assume $Inv(i_0, total_0, right\_sum_0)$ is true. Also assume that $i_0 < n$ so another iteration is necessary. We want to show that $Inv(i_1, total_1, right\_sum_1)$ is true. That is,

$$Inv(i_1, total_1, right\_sum_1) :$$

$$m \leqslant i_1 \leqslant n \wedge total_1 = \sum_{j=m}^{i_1-1} A[j] \wedge right\_sum_1 = \max(total_1, right\_sum_0)$$

From the second loop's body we have

(a) $i_1 = i_0 + 1$

(b) $total_1 = total_0 + A[i_0]$

(c) $right\_sum = \max(total_1, right\_sum_0)$

Showing conjunct 1: $m \leqslant i_1 \leqslant n$.

The first part is true since $m \leqslant i_0 < i_1$. The second part is also true by our assumption that $i_0 < n$. So it follows that $i_1 \leqslant n$ by (a) above.

Showing conjunct 2: $total_1 = \sum_{j=m}^{i_1-1} A[j]$.

From the loop body we have

$$\begin{aligned} total_1 &= total_0 + A[i_0] \\ &= \sum_{j=m}^{i_0-1} A[j] + A[i_0] && \text{(by I.H.)} \\ &= \sum_{j=m}^{i_0} A[j] \\ &= \sum_{j=m}^{i_1-1} A[j] && \text{(by (a))} \end{aligned}$$

Showing conjunct 3: $right\_sum = \max(total_1, right\_sum_0)$

But this is exactly what we have in property (c) from the loop body.

Step 4: Checking the post-condition

We will now check that the invariants and negation of their respective loop guards imply the post-condition. Loop 1's guard is $i < m$ and terminates when $i \geqslant m$. That is, the negation of the guard implies termination.

From loop 1, the invariant and negation of loop guard together:

$$Inv(i, total, left\_sum):$$

$$0 \leqslant i \leqslant m \wedge total = \sum_{j=0}^{i-1} A[k-j] \wedge left\_sum_i = \max(total_i, left\_sum_{i-1}) \wedge i \geqslant m$$

implies that $i = m$. Thus $total = \sum_{j=0}^{m-1} A[k-j]$ is the sum of all the integers in list A from index 0 up to index $m-1$. Therefore, $left\_sum$ is the maximum value of a contiguous sum of integers starting from the middle of the given list and iterating in reverse.

From loop 2, the invariant and negation of loop guard together:

$$Inv(i, total, right\_sum):$$

$$m \leqslant i \leqslant n \wedge total = \sum_{j=m}^{i-1} A[j] \wedge right\_sum_i = \max\left(total_i, right\_sum_{i-1}\right) \wedge i \geqslant n$$

implies that $i = n$. Thus $total = \sum_{j=m}^{n-1} A[j]$, the sum of the integers in the list A from index $m = mid$ up to index $n - 1$. Also, this implies that $right\_sum$ is the maximum value of a contiguous sum of integers starting from the middle of the given list and iterating to the right.

Since the program returns $left\_sum + right\_sum$, we can conclude that $max\_crossing\_sum$ returns the maximum contiguous crossing sum of integers starting from the middle and expanding outward in both directions. Therefore the post-condition is satisfied.

Step 5: Termination

We define the following loop variants for loops 1 and 2, respectively:

For loop 1, let $var_1(i) = m - i$, and let $var_2(i) = n - i$. Since i increases by 1 each iteration, $var_1(i)$, and $var_2(i)$ will decrease by 1 each iteration. Also, from the invariants we know that $i \leqslant m$ and $i \leqslant n$, so it follows that $m - i \geqslant 0$ and $n - i \geqslant 0$. Since each loop variant decreases on each iteration, but cannot decrease below zero, we can conclude that at some point each loop must terminate.

□

Proof of max_sublist_sum

Proof. Let L be the input list to *max_sublist_sum* and assume L satisfies the precondition.

Base Case:$pre \implies post$

If `len(L)` $\leqslant 1$, then there is only one item in the list and the max of that item and zero is returned. Since a contiguous sublist of L with maximum possible sum is either the value of the only item, or zero, the postcondition is satisfied. Note that an empty sublist has a maximum sum of zero, which is greater than a negative item.

Inductive Step:

$pre \implies pre$

Assume L satisfies the precondition and that `len(L)` > 1. Let $m =$ `len(L) // 2`, $L' = L[: m]$, and $L'' = L[m :]$. Further, we see that L' contains the first half of the items in L and L'' contain the second half of L. Since it is exactly L' and L'' that is passed into the two recursive calls, we can conclude that their preconditions are satisfied.

$post \implies post$

Let $S' = max\_sublist\_sum(L')$ and $S'' = max\_sublist\_sum(L'')$. Assume S' and S'' both satisfy the post condition; that is, they are each a contiguous sublist of their respective inputs with the maximum possible value. But since L' and L'' are the first and second halves of list L, then it follows that S' is a contiguous sublist with the maximum possible value of the first half of L. Likewise, S'' is a contiguous sublist with the maximum possible value of the second half of list L. But if the contiguous sublist with the maximum possible value of L is across the center of both both halves of L? In that case, let $CS = max\_crossing\_sum(L)$. Since we have proven the correctness of *max_crossing_sum*, we know CS is exactly the maximum value we are missing from the two recursive calls. Lastly, since the program returns the $\max(S', S'', CS)$ we can conclude that the postcondition of this algorithm is satisfied because the maximum is necessarily in one of those three cases.

Termination

We define a measure $m(L) =$ `len(L)`. Further, we will assume that $len(L') = \lceil\frac{len(L)}{2}\rceil$. Using the same symbols as above, we can see that each recursive call is made on a smaller list. We will show this is for L', but note that the exact same argument holds for L'' since L'' is at most half the size of L. Thus we have

$$\begin{aligned} m(L') &= len(L') \\ &= \lceil\frac{len(L)}{2}\rceil && \text{(by def of L')} \\ &< len(L) \\ &= m(L) \end{aligned}$$

Thus each recursive call is made on a smaller input list until eventually a base case is reached.

□

Part (3) [3 MARKS]

Analyze the running time of your algorithm.

Let $n = len(lst)$ be the length of the input list to *max_sublist_sum* and let $T(n)$ be the runtime. If $n = 1$, then $T(1) = c$ for some constant c. Otherwise there are two recursive calls made, each on a list of length $\frac{n}{2}$. Also, *max_crossing_sum* takes linear time since it iterates across the entire length of it's input list. So we end up with

$$T(n) = \begin{cases} c & \text{if } n \leqslant 1 \\ 2T(\frac{n}{2}) + dn + e & \text{otherwise} \end{cases}$$

where $2T(\frac{n}{2})$ is the cost of the two recursive calls and dn is the linear cost of *max_crossing_sum*, and finally e is a constant for all the constants (i.e. `return`, etc.). Note that $2T(\frac{n}{2}) = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor)$. Then by the Master Theorem, $a = 2$, $b = 2$, $f(n) = dn + e$. Since $k = 1$ and $\log_2 2 = 1$, so the Master Theorem gives

$$T(n) \in \mathcal{O}(n \log n)$$

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 2 *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.

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

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

Proof of correctness

Proof. **1. Define the loop invariant**

We define our loop invariant Inv(Z, i):

(a) $Z = \{I_0, I_1, \ldots, I_i\}$ is a subset of Y

(b) $0 \leqslant i \leqslant len(L)$

(c) Z may be extended to Z' such that

  (i) Z' covers points in L up to x_i

  (ii) Z' is the smallest such subset that covers points in L

<u>**Base Case**</u>:

2. Establish the loop invariant

Assume that the preconditions are true. Then when the loop is reached, we have

- $Z = \emptyset$
- $i = 0$

Thus conjunct (a) holds since $\emptyset \subseteq Y$, (even if Y was empty). Conjunct (b) is obviously true. Also, conjunct (c) holds because $i = 0$, so $x_i \notin L$ and so property (i) says that Z' covers points in L up to x_i, but since there are no points in L to cover yet, it is true that Z' covers points in L up to x_i (which is actually nothing yet). Similarly, $Z' = \emptyset$ is the smallest such subset that covers no points in L. Thus conjunct (c) (ii) also holds.

<u>**Inductive Step**</u>:

3. Maintain the Loop Invariant

We want to show that if the LI holds before an iterative step, then LI holds at the end of the iterative step.

Assume that the preconditions are satisfied and that the LI holds at the start of an arbitrary iteration. Let Z_0 and i_0 be the values of Z and i at the start of an arbitrary loop iteration. Also assume that $i_0 < len(L)$ so that another loop is necessary. Let Z_1 and i_1 be the values of Z and i at the end of that loop iteration. We are assuming that $Inv(Z_0, i_0)$ is true and we want to show that $Inv(Z_1, i_1)$ is also true.

By our assumption $Inv(Z_0, i_0)$, we know $Z_0 = \{I_0, \ldots, I_{i_0}\}$ can be extended to $Z_0' = \{I_0, \ldots, I_{i_0}, I_{i_1}\}$ that satisfies (i) and (ii). Further, from the body of the loop we know that

- $Z_1 = \{I_0, \ldots, I_{i_0}, I_{i_0+1}\}$, if Z_0' is extended to Z_1
- $Z_1 = \{I_0, I_1, \ldots, I_j\}$, otherwise
- $i_1 = i_0 + 1$

holds. By the first two bullet points above, in either case, it follows that $Z_1 \subseteq Y$, so conjunct (a) is true. By the third bullet point above, and by our assumption that $i_0 < len(L)$, it follows that $i_0 + 1 \leqslant len(L) \implies i_1 \leqslant len(L)$. Hence conjunct (b) is true. We now need to show conjunct (c): Z_1 may be extended to Z_1' such that

(i) Z_1' covers points in L up to x_{i_1}

(ii) Z_1' is the smallest such subset that covers points in L

If it's the case that Z_0' extends Z_1, then Z_1 already contains the interval $I_{i_0+1} = I_{i_1}$ being appended, so it follows that Z_1 covers all the points in L up to x_{i_1}, and conjunct (c) is satisfied.

On the other hand, suppose Z_0' doesn't extend Z_1 (bullet point two), then let I_j denote an interval in Z_0 such that $|I_{i_1}| < |I_j|$ and I_j covers up to the left-most point x_{i_1} in L. Our goal here is to show that the point x_{i_1} is covered in Z_1. Thus we know that the interval I_{i_1} is not appended because the point x_{i_1} is already covered by I_j which is contained in Z_0. Hence it follows that the interval I_j is also contained in Z_1. Hence Z_1 covers all the points in L up to x_{i_1} as desired. Further, since I_j is the interval that extends to the right as far as possible, it must extend at least as far to the right as I_{i_1}. Therefore I_j covers as many points as I_{i_1}, then it follows that Z_1 covers all the same points as Z_0' and since Z_0' is an optimal solution, it follows that Z_1 is optimal (the smallest such subset).

4. LI and exit condition $\implies$ postcondition

When the loop terminates, then $i = n$. Also suppose that $Inv(Z, i)$ holds. Then $Z = \{I_0, \ldots, I_m\}$ for $0 \leqslant m \leqslant n$ and Z covers all the points in L. Since L contains the same points as S, and since Z is the smallest such subset of Y that covers points in S, then we can conclude that the post condition is satisfied.

5. Termination

Let the measure of progress be defined as $m(L, i) = len(L) - i$. Since i increases by one after each iteration, we can see that $len(L) - i$ decreases by one each iteration. Further, from the invariant, we know that $i \leqslant len(L)$, so it follows that $len(L) - i \geqslant 0$. Then by our assumption $i = len(L)$, so we have $len(L) - len(L) = 0$ and the loop terminates.

$\square$

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. $\mathcal{R} = \mathcal{S}$. 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]

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

Proof. Suppose this statement is true. Then lets define regular expressions for R and S over the alphabet $\{0, 1\}$. Let $R = (0 + 1)$ and let $S = (0 + 1)^*$. Then at least intuitively, we can see that $RS \equiv SR$ holds. But by our assignment of R and S, we know that $R \not\equiv S$. Thus we have reached a contradiction, so our initial assumption was incorrect and we can conclude that this statement is false. □

Part (2) [2 MARKS]

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

Proof. Let R, S, T be arbitrary regular expressions and assume that

(*i*) $RS \equiv RT$ and

(*ii*) $R \not\equiv \emptyset$.

By the first assumption, if we take the regular languages we get

$$\mathcal{L}(RS) \equiv \mathcal{L}(RT) \tag{1}$$
$$\mathcal{L}(R)\mathcal{L}(S) \equiv \mathcal{L}(R)\mathcal{L}(T) \tag{2}$$

Then equating both sides of (2) and using the fact that $\mathcal{L}(R) \not\equiv \emptyset$ we get

(a) $\mathcal{L}(R) \equiv \mathcal{L}(R)$ and

(b) $\mathcal{L}(S) \equiv \mathcal{L}(T)$

Property (b) says that the two regular languages (sets) represented by the regular expressions S and T are equivalent. Now we can use the property stated in the question such that if the underlying regular language is the same (i.e. equivalent), then it follows that the regular expressions are equivalent. That is, since $\mathcal{L}(S) \equiv \mathcal{L}(T)$, then $S \equiv T$. Therefore, we conclude that this statement is true. □

Part (3) [2 MARKS]

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

Let R, S be arbitrary regular expressions. Take any string from the LHS and note that it always begins with a character from the regular expression R and ends with a character from R. In between these characters, there is any number of characters from RS or R.

On the other hand, any string from the RHS also always starts with and ends with a character from R. Also, in between these characters, there is any number of characters from SR or R. As we have shown in **Part (1)**, it does not need to be the case that $R \equiv S$. Thus we can conclude that this statement is true.

Question 4. [16 MARKS]

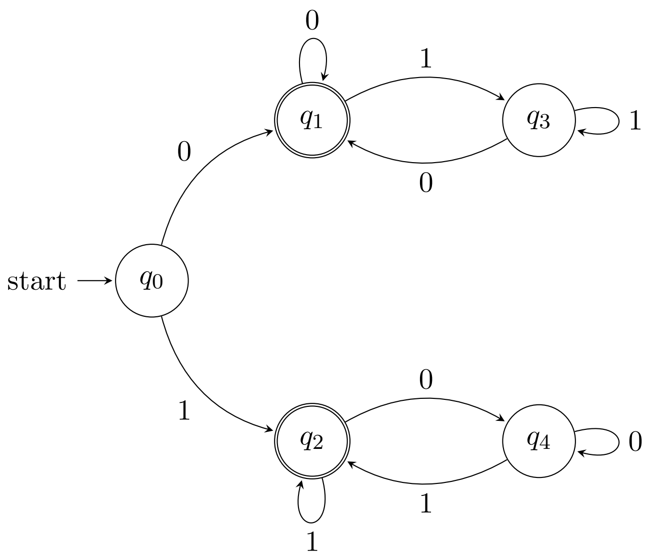
In the following, for each language L over the alphabet $\Sigma = \{0,1\}$ construct a regular expression R and a DFA M such that $\mathcal{L}(R) = \mathcal{L}(M) = L$. Prove the correctness of your DFA.

Part (1) [8 MARKS]

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

We define the regular expression $R = 0(0+1)^*0 + 1(0+1)^*1$ such that $\mathcal{L}(R) = L_1$. That is, L_1 is the regular language matched by the regular expression R.

Next, we define the DFA M:



Proof of Correctness

Proof. We define the predicate

$$P(w) := \delta^*(q_0, w) = \begin{cases} q_0 & \iff |w| < 1 \\ q_1 & \iff w \text{ starts with a zero and ends with a zero} \\ q_2 & \iff w \text{ starts with a one and ends with a one} \\ q_3 & \iff w \text{ starts with a zero and ends with a one} \\ q_4 & \iff w \text{ starts with a one and ends with a zero} \end{cases}$$

These are our five state invariants that we want to prove.

Base Case:

Show that the empty string ϵ satisfies the state invariant of the initial state: $w = \epsilon$.

In our case the initial state is q_0 and since the length of the empty string is zero, this invariant holds. Thus $P(\epsilon)$ is true.

Inductive Step:

For each transition $q \overset{\alpha}{\to} r$, show that if a string w satisfies the invariant of state q, then the string $w\alpha$ satisfies the invariant of r:

There are ten transition states we must show that this is true. Note that the structure of our DFA is symmetric with respect to the transition states, so we will show that the invariants of the top half hold without loss of generality. The bottom half is shown similarly with the 0's and 1's switched (which we show in brackets).

1. Transition states $q_0 \overset{0}{\to} q_1$ and $q_0 \overset{1}{\to} q_2$:

Suppose that an arbitrary string w satisfies the invariant of state q_0 (i.e. it's just $w = \epsilon$ since q_0 is the initial state). Appending a 0 (1) from the string gives us $w0 = \epsilon 0 = 0$ ($w1 = \epsilon 1 = 1$). Thus $w0$ ($w1$) starts and ends with the same character since it has been the only character read so far. Hence $w0$ ($w1$) satisfies the invariant of q_1 (q_2). Thus

$$\begin{aligned}\delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_0, 0) && \text{(by assumption)} \\ &= q_1\end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_2)$.

2. Transition states $q_1 \overset{0}{\to} q_1$ and $q_2 \overset{1}{\to} q_2$:

Assume an arbitrary string w satisfies the invariant of state q_1 (q_2). Appending a 0 (1) to the string doesn't change the first character, and since the first and last character was already a zero (one) by assumption, appending a zero (one) doesn't change the last character. Hence $w0$ ($w1$) satisfies the invariant of q_1 (q_2). Thus

$$\begin{aligned}\delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_1, 0) && \text{(by assumption)} \\ &= q_1\end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_2)$.

3. Transition states $q_1 \xrightarrow{1} q_3$ and $q_2 \xrightarrow{0} q_4$:

Assume an arbitrary string w satisfies state q_1 (q_2). Appending a 1 (0) doesn't change the first character, but does change the last character to a 1 (0). Hence $w1$ ($w0$) satisfies the invariant of q_3 (q_4). That is,

$$\begin{aligned}\delta(q_0, w1) &= \delta(\delta(q_0, w), 1) \\ &= \delta(q_1, 1) && \text{(by assumption)} \\ &= q_3\end{aligned}$$

$(\delta(q_0, w0) = \delta(\delta(q_0, w), 0) = q_4)$.

4. Transition states $q_3 \xrightarrow{0} q_1$ and $q_4 \xrightarrow{1} q_2$:

Suppose an arbitrary string w satisfies state q_3 (q_4). Appending a 0 (1) to w only changes the last character to a 0 (1). Hence $w0$ ($w1$) satisfies the invariant of q_1 (q_2). That is,

$$\begin{aligned}\delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_3, 0) && \text{(by assumption)} \\ &= q_1\end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_2)$.

5. Transition states $q_3 \xrightarrow{1} q_3$ and $q_4 \xrightarrow{0} q_4$:

Suppose an arbitrary string w satisfies state q_3 (q_4). Appending a 1 (0) to w only changes the last character to a 1 (0). Thus $w1$ ($w0$) satisfies the invariant of q_3 (q_4). Hence

$$\begin{aligned}\delta(q_0, w1) &= \delta(\delta(q_0, w), 1) \\ &= \delta(q_3, 1) && \text{(by assumption)} \\ &= q_3\end{aligned}$$

$(\delta(q_0, w0) = \delta(\delta(q_0, w), 0) = q_4)$.

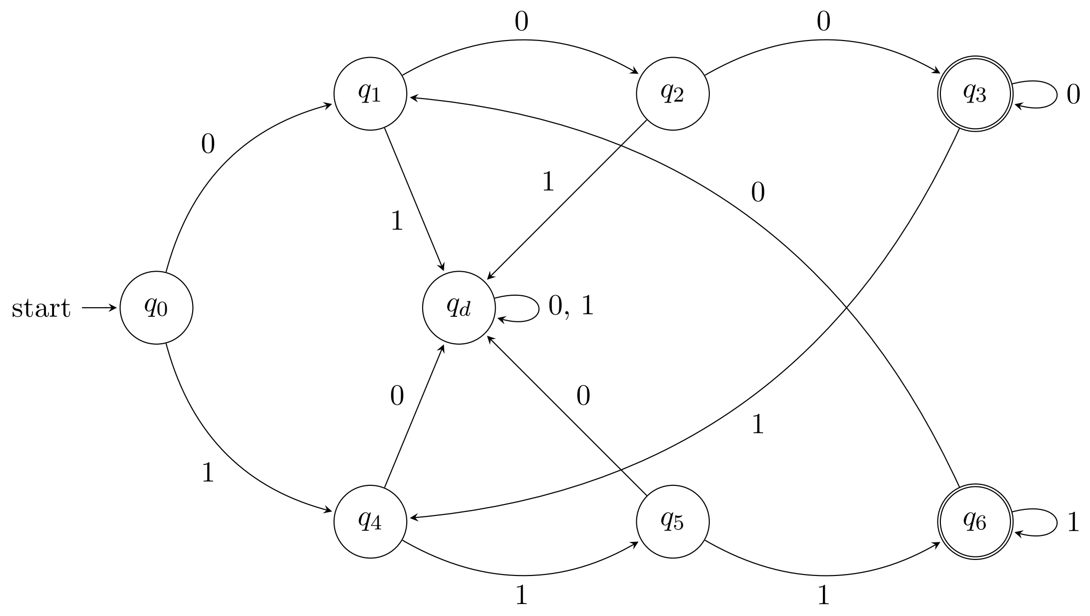
Thus $P(w)$ holds for each transition state, and this completes the induction step. Having shown the base case and induction step, we can conclude that the DFA M is true for all strings w over $\{0, 1\}^*$. Therefore, $\mathcal{L}(R) = \mathcal{L}(M) = L_1$. $\square$

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 $L_2 = \{x \in \{0,1\}^* : x \text{ only contains blocks of length at least three}\}$.

We define the regular expression $R = (0^3 0^* + 1^3 1^*)^*$ such that $\mathcal{L}(R) = L_2$. That is, L_2 is the regular language matched by the regular expression R.

Next, we define the DFA M:



Proof of Correctness

Proof. We define the predicate

$$P(w) := \delta^*(q_0, w) = \begin{cases} q_0 & \iff |w| < 1 \\ q_1 & \iff w \text{ ends with a block of } 0^1 \\ q_2 & \iff w \text{ ends with a block of } 0^2 \\ q_3 & \iff w \text{ only contains blocks of length at least three and ends with a block of } 0^n \\ q_4 & \iff w \text{ ends with a block of } 1^1 \\ q_5 & \iff w \text{ ends with a block of } 1^2 \\ q_6 & \iff w \text{ only contains blocks of length at least three and ends with a block of } 1^n \\ q_d & \iff w \text{ contains at least one segment block }^1 \end{cases}$$

for $n \geqslant 3 \in \mathbb{N}$. These are our eight state invariants that we want to prove.

[1]We define a segment block as a sequence such as $0^j 1^k$ or $1^j 0^k$, where $0 < j < 3$ and $k \in \mathbb{Z}^+$.

<u>**Base Case**</u>:

Show that the empty string ϵ satisfies the state invariant of the initial state: $w = \epsilon$.

In our case the initial state is q_0 and since the length of the empty string is zero, this invariant holds. Thus $P(\epsilon)$ is true.

<u>**Inductive Step**</u>:

For each transition $q \xrightarrow{\alpha} r$, show that if a string w satisfies the invariant of state q, then the string $w\alpha$ satisfies the invariant of r:

There are fifteen transition states to consider. Note that, once again, the structure of our DFA is symmetric with respect to the transition states, so we will show that the invariants of the top half hold without loss of generality. The bottom half is shown similarly with the 0's and 1's switched (which we show in brackets).

1. Transition states $q_0 \xrightarrow{0} q_1$ and $q_0 \xrightarrow{1} q_4$:

Suppose that an arbitrary string w satisfies the invariant of state q_0 (i.e. it's just $w = \epsilon$ since q_0 is the initial state). Appending a 0 (1) from the string gives us $w0 = \epsilon 0 = 0$ ($w1 = \epsilon 1 = 1$). Thus $w0$ ($w1$) ends with a block of 0^1 (1^1). Hence $w0$ ($w1$) satisfies the invariant of q_1 (q_4). Thus

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_0, 0) && \text{(by assumption)} \\ &= q_1 \end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_4)$.

2. Transition states $q_1 \xrightarrow{0} q_2$ and $q_4 \xrightarrow{1} q_5$:

Assume an arbitrary string w satisfies the invariant of state q_1 (q_4). Appending a 0 (1) to w adds another zero (one) to the end of w. Hence $w0$ ($w1$) ends with a block of 0^2 (1^2). That is,

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_1, 0) && \text{(by assumption)} \\ &= q_2 \end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_5)$.

3. Transition states $q_2 \xrightarrow{0} q_3$ and $q_5 \xrightarrow{1} q_6$:

Assume an arbitrary string w satisfies the invariant of state q_2 (q_5). Appending a 0 (1) to w adds another zero (one) to the end of w. Hence $w0$ ($w1$) ends with a block of 0^3 (1^3). That is,

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_2, 0) && \text{(by assumption)} \\ &= q_3 \end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_6)$.

4. Transition states $q_3 \xrightarrow{0} q_3$ and $q_6 \xrightarrow{1} q_6$:

Assume an arbitrary string w satisfies the invariant of state q_3 (q_6). Appending a 0 (1) to w adds another zero (one) to the end of w. Hence $w0$ ($w1$) ends with a block of at least three. That is, $w0$ ($w1$) ends with a block of 0^n (1^n), where $n \geqslant 3 \in \mathbb{N}$. Hence,

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_3, 0) && \text{(by assumption)} \\ &= q_3 \end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_6)$.

5. Transition states $q_3 \xrightarrow{1} q_4$ and $q_6 \xrightarrow{0} q_1$:

Assume an arbitrary string w satisfies the invariant of state q_3 (q_6). Appending a 1 (0) to w adds a one (zero) to the end of w. Hence $w1$ ($w0$) ends with a block of 1^1 (0^1). Hence,

$$\begin{aligned} \delta(q_0, w1) &= \delta(\delta(q_0, w), 1) \\ &= \delta(q_3, 1) && \text{(by assumption)} \\ &= q_4 \end{aligned}$$

$(\delta(q_0, w0) = \delta(\delta(q_0, w), 0) = q_1)$.

6. Transition states $q_4 \xrightarrow{0} q_d$ and $q_1 \xrightarrow{1} q_d$:

Assume an arbitrary string w satisfies the invariant of state q_4 (q_1). Appending a 0 (1) to w adds another zero (one) to the end of w. Since w ends with a block of 1^1 (0^1) by our assumption, then $w0$ ($w1$) ends with a block of 0^1 (1^1). But by our assumption of w, we also know that the block before appending a 0 (1) to the end of w only contained a block of 1^1 (0^1). Then since $w0$ ($w1$) contains at least one block segment (as define above $j < 3$), so we can conclude that $w0$ ($w1$) satisfies the state invariant of q_d. That is,

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_4, 0) && \text{(by assumption)} \\ &= q_d \end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_d)$.

7. Transition states $q_5 \xrightarrow{0} q_d$ and $q_2 \xrightarrow{1} q_d$:

Assume an arbitrary string w satisfies the invariant of state q_5 (q_2). Appending a 0 (1) to w adds a zero (one) to the end of w. Then $w0$ ($w1$) ends with a block of 0^1 (1^1). But by our assumption of w, we also know that the block before appending a 0 (1) to the end of w only contained a block of 1^2 (0^2). Then since $w0$ ($w1$) contains at least one block segment (as define above $j < 3$), so we can conclude that $w0$ ($w1$) satisfies the state invariant of q_d. That is,

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_5, 0) && \text{(by assumption)} \\ &= q_d \end{aligned}$$

$(\delta(q_0, w1) = \delta(\delta(q_0, w), 1) = q_d)$.

8. Transition state $q_d \xrightarrow{0,1} q_d$:

Assume an arbitrary string w satisfies the invariant of state q_d. Appending a 0 or 1 to w adds a zero or one to the end of w. But by our assumption of w, we know that w contains at least one segment block such that $0^j 1^k$ or $1^j 0^k$ and $0 < j < 3$. So appending a 0 or 1 to w doesn't change what we already know about w, (i.e. we still know that w contains at least one segment block) Thus we can conclude that $w0$ and $w1$ satisfy the state invariant of q_d. That is,

$$\begin{aligned} \delta(q_0, w0) &= \delta(\delta(q_0, w), 0) \\ &= \delta(q_d, 0) && \text{(by assumption)} \\ &= q_d \end{aligned}$$

Also,

$$\begin{aligned} \delta(q_0, w1) &= \delta(\delta(q_0, w), 1) \\ &= \delta(q_d, 1) && \text{(by assumption)} \\ &= q_d \end{aligned}$$

Thus $P(w)$ holds for each transition state, and this completes the induction step. Having shown the base case and induction step, we can conclude that the DFA M is true for all strings w over $\{0, 1\}^*$. Therefore, $\mathcal{L}(R) = \mathcal{L}(M) = L_2$. $\square$