1. (a) **Solution:** Figure 1 depicts an NFA for this language.

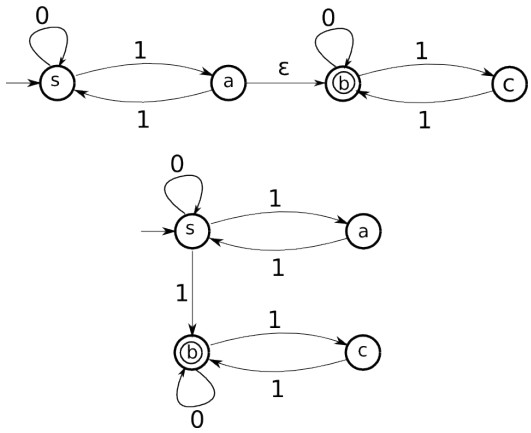

Figure 1. An NFA for the language that accepts strings where exactly one substring of 1s has odd length

To see the correctness of this construction, we first argue that a possible regular expression for this language is $(0 + 11)^*1(0 + 11)^*$. This expression captures all strings that start and end with strings that have blocks of 0s of arbitary length and blocks of 1s of even length, with a single 1 appended between the two. The 1 gets added to one block of 1s of even length, if one or both of the prefix and suffix strings ends with block(s) of 1s, and forms a single block of 1s of odd length. Else, if added between blocks of 0s, it forms a single block of 1s of odd length 1. All other blocks of 1s in the string belong entirely either in the prefix or the suffix string, hence have even length. Hence, every string accepted by the language described by the expression has a single block of 1s of odd length. Further, every string w that belongs in the language to be accepted can be written in the form $(0 + 11)^*1(0 + 11)^*$ as follows: the first 1 in the single block of 1s of odd length in w is the single 1, the substring to the left of this 1 forms the prefix string and that to its right the suffix. Both the prefix and suffix have blocks of 0 of arbitrary length, and blocks of 1s of even length, hence are captured by the expression $(0 + 11)^*$.

Now, an NFA for the language described by the regex is drawn. We draw two NFAs for the expression $(0 + 11)^*$, and combine them by (a) adding a transition edge reading a 1 between their (single) accept states, (b) considering the start state of the first NFA as the single start state, and (c) the accept state of the second NFA as the single accept state.

Another way to combine the separate NFAs is to add an epsilon-transition between the second state (named state a) of the first and the accept state of the next (state b), as the second state captures the strings $(0 + 11)^*1$, and defining the start and accept states as in the first conversion.

The NFA is shown in Figure 1. ■

(b) i. **Solution:** The diagram for the subsequent description is given in Figure 2.

Thompson's algorithm is used to construct the NFA from the given regular expression. NFAs for the three expressions $(010)^*$, $(01)^*$ and 0^* are drawn separately, then combined as per the algorithm. Accordingly, a new start state is drawn and connected to the start states of the three NFAs via ε-transitions, the accept states of all are connected to a new accept state via ε-transitions. This

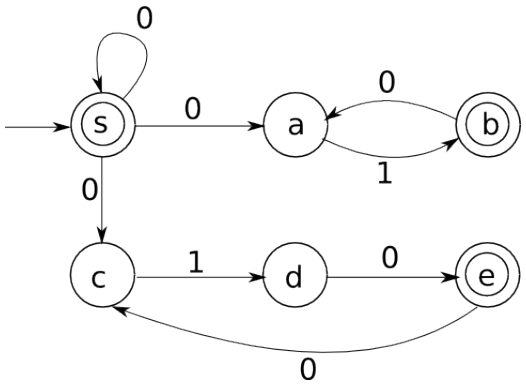

Figure 2. NFA for the given regular expression

NFA is then shortened by combining redundant states. As the start symbol of all three NFAs is the same, a new start state is not necessary here, instead the start states of all three NFAs can be combined. A new accept state too is not necessary and we take the union of the accept states of the separate NFAs as the set of accept states of the combined NFA. ■

ii. **Solution:** We use the incremental power set construction method to construct the equivalent DFA from the NFA of the previous part. The diagram is given in Figure 3. ■

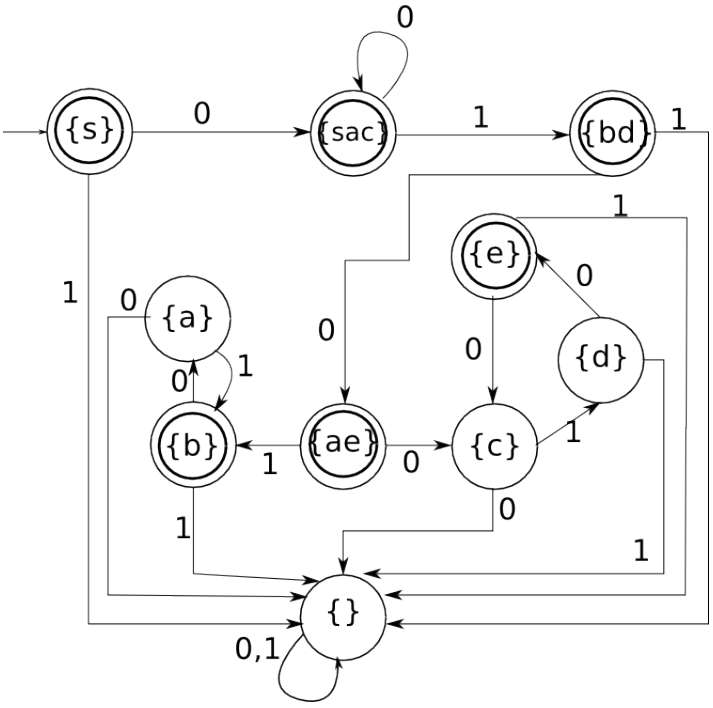

Figure 3. Corresponding DFA using power set construction

Rubric: 10 points.

(a) maximum points: 5
-5: completely incorrect NFA.
-1 for every small error (up to 2 transitions labelled incorrect, up to 1 edge missing, adding or deleting 1 state makes the construction correct) -1.5: missing justification.

(b.i) maximum points: 2
-2: completely incorrect NFA
-1: small errors (transitions labelled incorrect (should be 1, is 0 instead). If more than 2 transitions are mislabelled, consider NFA is completely incorrect, less than 2 edges missing)
-0.5: missing justification, if states are shortened. (Not necessary if Thompson's algorithm output drawn as is)

(b.ii) maximum points: 3
-3: completely incorrect NFA
-1: small errors (less than 3 transitions labelled incorrect, at most 1 state missing, at most 2 accept states not labelled as accept states)

2. (a) **Solution:** The approach is to consider r as one of the base cases of a regular expression and then to show that one can construct a regular expression r' such that $L(r') = \mathit{delete1}(L(r))$. Before we consider each case, we note that if a language L does not contain a string with a 1, then $\mathit{delete1}(L) = \emptyset$. This follows from the fact there does not exist strings x and y such that $x1y \in L$.

First, we start with $r = \emptyset$. We claim $r' = \emptyset$ satisfies $L(r') = \mathit{delete1}(L(r))$. Because $r = \emptyset$, $L(r) = \emptyset$. Thus, since $L(r) = \emptyset$, by the observation made above, $\mathit{delete1}(L(r)) = \emptyset$. As $L(r') = \emptyset$, we have $\mathit{delete1}(L(r)) = L(r')$.

Second, we consider $r = \varepsilon$. We claim $r' = \emptyset$ satisfies $L(r') = \mathit{delete1}(L(r))$. Because $r = \varepsilon$, $L(r) = \{\varepsilon\}$. Thus, since $L(r) = \{\varepsilon\}$ and thus does not contain any strings with a 1, by the observation made above, $\mathit{delete1}(L(r)) = \emptyset$. As $L(r') = \emptyset$, we have $\mathit{delete1}(L(r)) = L(r')$.

Third, we consider $r = a$ for some $a \in \Sigma$. Because $r = a$, $L(r) = \{a\}$. We now consider two cases. First, suppose $r = a$ where $a \neq 1$. We claim that, in this case, $r' = \emptyset$ satisfies $\mathit{delete1}(L(r)) = L(r')$. Because $L(r)$ does not contain any strings with a 1 in it, $\mathit{delete1}(L(r)) = \emptyset$. As $L(r') = \emptyset$, we have $\mathit{delete1}(L(r)) = L(r')$.

Now suppose $r = 1$. We claim that $r' = \{\varepsilon\}$ satisfies $\mathit{delete1}(L(r)) = L(r')$. By definition of $\mathit{delete1}$, we have $\mathit{delete1}(L(r)) = \{xy \mid x1y \in \{1\}\}$. Taking $x = y = \varepsilon$, we see that the only string in $\mathit{delete1}(L(r))$ is ε. As $L(r') = \{\varepsilon\}$, we have $\mathit{delete1}(L(r)) = L(r')$. ■

(b) **Solution:** Assume $L(r_1') = \mathit{delete1}(L(r_1))$ and $L(r_2') = \mathit{delete1}(L(r_2))$. We want to find a regular expression r' for the language $\mathit{delete1}(L(r_1 + r_2))$ in terms of r_1, r_2, r_1', r_2'. We claim that $r' = r_1' + r_2'$ will satisfy this property. We proceed by proving $L(r_1' + r_2') = \mathit{delete1}(L(r_1 + r_2))$.

To start, we note that $L(r_1' + r_2') = L(r_1') \cup L(r_2')$. By assumption, $L(r_1') \cup L(r_2') = \mathit{delete1}(L(r_1)) \cup \mathit{delete1}(L(r_2))$. Furthermore, $\mathit{delete1}(L(r_1 + r_2)) = \mathit{delete1}(L(r_1) \cup L(r_2))$. Thus, it suffices to show $\mathit{delete1}(L(r_1)) \cup \mathit{delete1}(L(r_2)) = \mathit{delete1}(L(r_1) \cup L(r_2))$.

We first prove $delete1(L(r_1)) \cup delete1(L(r_2)) \subseteq delete1(L(r_1) \cup L(r_2))$. Let $w \in delete1(L(r_1)) \cup delete1(L(r_2))$. Suppose $w \in delete1(L(r_1))$. Then $w \in \{xy \mid x1y \in L(r_1)\}$. As $L(r_1) \subseteq L(r_1) \cup L(r_2)$, $w \in \{xy \mid x1y \in L(r_1) \cup L(r_2)\}$. The case when $w \in delete1(L(r_2))$ is argued the same. Thus, $w \in delete1(L(r_1) \cup L(r_2))$.

We next prove $delete1(L(r_1) \cup L(r_2)) \subseteq delete1(L(r_1)) \cup delete1(L(r_2))$. Let $w \in delete1(L(r_1) \cup L(r_2))$. Then $w = xy$ where $x1y \in L(r_1) \cup L(r_2)$. Suppose $x1y \in L(r_1)$. Then $w \in delete1(L(r_1))$. The case when $x1y \in L(r_2')$ is argued the same. Thus, $w \in delete1(L(r_1')) \cup delete1(L(r_2'))$. ■

(c) **Solution:** Assume $L(r_1') = delete1(L(r_1))$ and $L(r_2') = delete1(L(r_2))$. We want to construct a regular expression r' in terms of r_1, r_2, r_1', r_2' such that $L(r') = delete1(L(r_1 r_2))$. We claim that we can construct r' as $r' = r_1' r_2 + r_1 r_2'$. To prove the correctness of our construction, we will show $L(r_1' r_2 + r_1 r_2') = delete1(L(r_1 r_2))$. By the definitions of the addition and concatenation operations for regular expressions, the left-hand side can be written as $L(r_1')L(r_2) \cup L(r_1)L(r_2')$ and the right-hand side as $delete1(L(r_1)L(r_2))$. Thus, it suffices to prove $L(r_1')L(r_2) \cup L(r_1)L(r_2') = delete1(L(r_1)L(r_2))$.

We start by proving $L(r_1')L(r_2) \cup L(r_1)L(r_2') \subseteq delete1(L(r_1)L(r_2))$. Let $w \in L(r_1')L(r_2) \cup L(r_1)L(r_2')$. Suppose $w \in L(r_1')L(r_2)$. So $w = w_1 w_2$ where $w_1 \in L(r_1')$ and $w_2 \in L(r_2)$. By assumption, $L(r_1') = delete1(L(r_1))$, implying $w_1 \in delete1(L(r_1))$. So $w = x_1 y_1 w_2$ where $x_1 1 y_1 \in L(r_1)$ and $w_2 \in L(r_2)$. Thus, $w \in delete1(L(r_1)L(r_2))$. The case when $w \in L(r_1)L(r_2')$ is handled similarly. Therefore, $L(r_1')L(r_2) \cup L(r_1)L(r_2') \subseteq delete1(L(r_1)L(r_2))$.

We then move on to showing $delete1(L(r_1)L(r_2)) \subseteq L(r_1')L(r_2) \cup L(r_1)L(r_2')$. Let $w \in delete1(L(r_1)L(r_2))$. So $w = xy$ where $x1y \in L(r_1)L(r_2)$. Thus, we can write $x1y$ as $w_1 w_2$ where $w_1 \in L(r_1)$ and $w_2 \in L(r_2)$. Suppose that the added 1 in $x1y$ is in w_1. That is, we can write w as $x_1 y_1 w_2$ where $x_1 1 y_1 \in L(r_1)$. This implies $x_1 y_1 \in L(r_1')$ as, by assumption, $L(r_1') = delete1(L(r_1))$. Thus, as $w = x_1 y_1 w_2$, $w \in L(r_1')L(r_2)$. The other case when the added 1 in $x1y$ is in w_2 is handled similarly and would imply $w \in L(r_1)L(r_2')$. Thus, $delete1(L(r_1)L(r_2)) \subseteq L(r_1')L(r_2) \cup L(r_1)L(r_2')$.

■

(d) **Solution:** Assume $L(r_1') = delete1(L(r_1))$. We want to construct a regular expression r' in terms of r_1 and r_1' such that $L(r') = delete1(L((r_1)^*))$. We claim that letting $r' = r_1^* r_1' r_1^*$ will satisfy this property. To show this, we proceed by proving $L(r_1^* r_1' r_1^*) = delete1(L((r_1)^*))$.

Note that the left-hand side is equal to $L(r_1)^* L(r_1') L(r_1)^*$ and the right-hand side is equal to $delete1(L((r_1)^*)) = delete1(L(r_1)^*)$. Thus, it suffices to prove $L(r_1)^* L(r_1') L(r_1)^* = delete1(L(r_1)^*)$.

We start by showing $L(r_1)^* L(r_1') L(r_1)^* \subseteq delete1(L(r_1)^*)$. Assume that $w \in L(r_1)^* L(r_1') L(r_1)^*$. Thus, we can write w as $w_1 w_2 w_3$ where $w_1, w_3 \in L(r_1^*)$ and $w_2 \in L(r_1')$. By assumption, $L(r_1') = delete1(L(r_1))$. So $w_2 \in delete1(L(r_1))$. This means that we can write w_2 as $x_2 y_2$ where $x_2 1 y_2 \in L(r_1)$. Thus, $w = w_1 x_2 y_2 w_3$. Then because $x_2 1 y_2 \in L(r_1)$, we have have $w_1 x_2 1 y_2 w_3 \in L(r_1)^*$ as $w_1, w_3 \in L(r_1)$. This implies $w_1 x_2 1 y_2 w_3 \in L(r_1)^*$. As $w = w_1 x_2 y_2 w_3$, by definition of $delete1$, we have

$$w \in \{xy \mid x1y \in L((r_1)^*)\} = delete1(L(r_1)^*).$$

Next, we prove $delete1(L(r_1)^*) \subseteq L(r_1)^* L(r_1') L(r_1)^*$. Let $w \in delete1(L(r_1)^*)$. By

definition of *delete*1, we can write w as xy where $x1y \in L(r_1)^*$. Thus, we can write $x1y$ as $z_1 z_2 \cdots z_n$ for some positive integer n, where $z_i \in L(r_1)$ for $i = 1, 2, \ldots, n$. Suppose that the added 1 in the string $x1y$ is in the string z_i. Then we can write z_i as $x_i 1 y_i$. Therefore, we have

$$w = z_1 z_2 \cdots z_{i-1} x_i y_i z_{i+1} \cdots z_n.$$

Note that $z_1 z_1 \cdots z_{i-1}, z_{i+1} z_{i+1} \cdots z_n \in L(r_1)^*$. By definition of *delete*1, we have $x_i y_i \in \mathit{delete}1(L(r_1))$. By assumption, $\mathit{delete}1(L(r_1)) = L(r_1')$. Thus, $x_i y_i \in L(r_1')$. Therefore, as $w = z_1 z_2 \cdots z_{i-1} x_i y_i z_{i+1} \cdots z_n$, we have that $w \in L(r_1)^* L(r_1') L(r_1)^*$. ■

(e) **Solution:** We want to apply the above construction to the regular expression $r = 0^* + (01)^* + 011^*0$. That is, we want to find the regular expression r' for $\mathit{delete}1(r)$ that is given by the above construction. To simplify the explanation, let $C(r')$ denote the regular expression obtained from the above construction for $\mathit{delete}1(r')$.

From part (b), $C(0^* + (01)^* + 011^*0) = C(0^*) + C((01)^*) + C(011^*0)$. We find these summands separately.

We have

$$\begin{aligned} C(0^*) &= 0^* C(0) 0^* && \text{(apply part (d))} \\ &= 0^* \emptyset 0^* && \text{(apply part (a))} \\ &= \emptyset. \end{aligned}$$

For the second summand,

$$\begin{aligned} C((01)^*) &= (01)^* C(01)(01)^* && \text{(apply part (d))} \\ &= (01)^* (C(0)1 + 0C(1))(01)^* && \text{(apply part (c))} \\ &= (01)^* (\emptyset 1 + 0\varepsilon)(01)^* && \text{(apply part (a) twice)} \\ &= (01)^* 0 (01)^*. \end{aligned}$$

For the third summand,

$$\begin{aligned} C(011^*0) &= C(01)1^*0 + 01C(1^*0) && \text{(apply part (c))} \\ &= 01^*0 + 01C(1^*0) && (C(01) = 0) \\ &= 01^*0 + 01(C(1^*)0 + 1^* C(0)) && \text{(apply part (c))} \\ &= 01^*0 + 01(1^* C(1) 1^* 0 + 1^* \emptyset) && \text{(apply part (a) and part (d))} \\ &= 01^*0 + 01(1^* \varepsilon 1^* 0) && \text{(apply part (a))} \\ &= 01^*0 + 011^*0 \\ &= 01^*0. \end{aligned}$$

Therefore, $C(r) = (01)^* 0 (01)^* + 01^*0$. ■

Rubric: 10 points.

(a) maximum points: 1.5
-0.5 for an incorrect construction and justification of any of the base cases

(b) maximum points: 2.5
-1.5 for incorrect construction
-1 for incorrect justification

(c) maximum points: 2.5
same as part (b)

(d) maximum points: 2.5
same as part (b)

(e) maximum points: 1
-1 for not using construction
-0.5 for errors in applying construction

3. (a) This solution uses Jeff Erickson's definition of ε-reach. More details can be found at https://courses.engr.illinois.edu/cs374/fa2018/notes/models/04-nfa.pdf. We define ε-reach of a state $q \in Q$ as the set of all the states $r \in Q$ such that r can be reached from q via a finite sequence of ϵ-transitions. See the notes for a formal definition.

Solution: First we construct the DFA $M_3 = (Q_3, \Sigma, \delta_3, s_3, A_3)$ that is the subset construction of N_2.

$$\begin{aligned} Q_3 &= 2^{Q_2} (\text{ or equivalently} P(Q_2)) \\ \delta_3(X,a) &= \bigcup_{p \in X} \bigcup_{r \in \delta_2(p,a)} \varepsilon\text{-reach}(r) \qquad \text{for all } X \subseteq Q \text{ and } a \in \Sigma \\ s_3 &= \{\varepsilon\text{-reach}(s_2)\} \\ A_3 &= \{S \subseteq Q_3 \mid S \cap A_2 \neq \emptyset\} \end{aligned}$$

Now we construct DFA $M = (Q, \Sigma, \delta, s, A)$ that is the product construction of M_1 and M_3.

$$\begin{aligned} Q &= Q_1 \times Q_3 = \{(q,S) \mid q \in Q_1 \text{ and } S \subseteq Q_2\} \\ \delta((q_1', q_2'), a) &= (\delta_1(q_1', a), \delta_3(q_2', a)) \\ s &= (s_1, s_3) = (s_1, \varepsilon\text{-reach}(s_2)) \\ A &= A_1 \times (Q_3 \setminus A_3) = \{(q,S) \mid q \in A_1 \text{ and } S \notin A_3\} = \{(q,S) \mid q \in A_1 \text{ and } S \cap A_2 = \emptyset\} \end{aligned}$$

M should accept only if M_1 accepts and M_3 does not accept and hence the definition of A as $A_1 \times (Q_3 \setminus A_3)$. ■

(b) We will assume that all states of M are reachable from the start state s. Otherwise we can modify M to remove the states not reachable from s and this will not affect the behaviour of M. Let $X \subseteq Q$ be the set of states of M that are reachable from s

on a non-empty string. That is $X = \{q \in Q \mid \delta^*(s,w) = q, |w| \geq 1\}$. Every state is reachable from s and M is a DFA so if $q \neq s$ then $q \in X$ (the only way to reach a state $q \neq s$ is via a non-empty string. If $q = s$ we then we need to check whether there is a non-empty string w such that $\delta^*(s,w) = s$. There is such a string if there is a state $q \neq s$ that can reach s, or if s has a self-loop on some symbol a. In any case we can determine X and $X = Q$ or $X = Q \setminus \{s\}$.

We now define an NFA N from M such that $L(N) = \text{PSUFFIX}(L(M))$. The NFA N has one additional state t which becomes its start state.

Solution: Define NFA $N = (Q', \Sigma, \delta', s', A')$ in terms of $M = (Q, \Sigma, \delta, s, A)$.

$$\begin{aligned} Q' &= Q \cup \{t\} \\ \delta'(q,a) &= \{\delta(q,a)\} \quad \text{for all } q \in Q \text{ and } a \in \Sigma \\ \delta'(t,\varepsilon) &= X \\ s' &= t \\ A' &= A \end{aligned}$$

The NFA N starts in t and jumps to a state $q \in X$ and then simulates M starting from q. Suppose N accepts a string w. Since $t \notin A'$, N can accept w only by jumping to some $q \in X$ and then reaching an an accept state of M by simulating M on w from q. This implies that there is a a state $q \in X$ such that $\delta^*_M(q,w) \in A$. Since $q \in X$ there is a non-empty string $u \in \Sigma^*$ such that $\delta^*_M(s,u) = q$. Hence $\delta^*_M(s,uw) \in A$ which implies that $uw \in L(M)$. Therefore $w \in \text{PSUFFIX}(L(M))$. Conversely if $w \in \text{PSUFFIX}(L(M))$ then there is a non-empty string $u \in \Sigma^*$ such that $uw \in L(M)$. Let $q = \delta^*_M(s,u)$. We have $q \in X$. Hence there is an accepting path for NFA N on w where N jumps from t to q and then simulates M on w. Thus $w \in L(N)$. This shows that $L(N) = \text{PSUFFIX}(L(M))$. ■

Rubric: 10 points.

(a) 5 points:

+ 2.5 for correctly applying subset construction to N_2
+ 2.5 for correctly applying product construction to their new DFA and M_1.

For both parts of (a).

-.5 for missing Q, δ, s, A
-.5 for incorrect Q.
-.5 for incorrect δ.
-.5 for incorrect s.
-.5 for incorrect A.

Note: There may be more than one way to do the constructions. For cases where the student combines the power and subset constructions, combine the corresponding point values (total $= 5$, each mistake $= -1$).

(b) 5 points

+ 3 for correctly constructing N.

-.5 for missing Q, δ, s, A
-.5 for incorrect Q.
-.5 for incorrect δ.
-.5 for incorrect s.
-.5 for incorrect A.

Note: Judge correctness based on if NFA N is does accept PSUFFIX(L) for M accepting L.

-.5 for not creating the described NFA AND not creating a NFA that satisfies PSUFFIX(L).

Note: This means we give them this .5 if their NFA matches their description, even if the description is wrong OR if their description is wrong, but the NFA is correct.

+ 2 for describing how to construct N.

-1 for missing description, but NFA is flawless.
-2 for missing description and flawed NFA.
-1 for incorrect description.