1 Regular Expression

Give a regular expression for the language defined below.

$$L = \{w \in \{0,1\}^* \mid w \text{ ends in } 01 \text{ and has odd length, or } w \text{ ends in } 11 \text{ and has even length}\}.$$

As an example 11 and 101 are in the language but 1101 and 011 are not.

Solution: We can represent L using the following regular expression:

$$((0+1)\,(0+1))^*(0+1)\,01 + ((0+1)\,(0+1))^*11.$$

This approach utilizes the fact that the expression $((0+1)\,(0+1))^*$ generates all binary strings of even length (including ϵ); in order to generate strings $w \in L$ of odd length, we simply add to this expression the subexpression $(0+1)$ to ensure that w has the correct parity.

[Note: explanation was not required for full credit on this problem. Other equivalent correct solutions exist.] ■

Rubric: 5 points are earned for correctly giving a regular expression that generates strings of odd length ending in 01, and 5 points are earned for correctly giving a regular expression that generates strings of even length ending in 11.

- -2.5 points (each occurrence) for incorrectly accounting for the parity of a string.
- -1 point each for minor errors.
- -3 points for trying to generate even-length strings by concatenating a string to itself.

2 DFA Construction

For a binary string z let $\text{num}(z)$ denote the number such that z is its binary representation. For example $\text{num}(0101) = 5$ and $\text{num}(1000) = 8$. For convenience we define $\text{num}(\epsilon) = 0$.

Draw a **DFA** for the language defined below.

$$L = \{w \in \{0,1\}^* \mid \text{num}(w) \text{ is divisible by 3 and } w \text{ has odd length}\}$$

For instance 011 (which is 3 in binary) and 01001 (9 in binary) are in the language but 0011 (3 in binary) and 00101 (5 in binary) are not. Your DFA must have at most 6 states. Label your states and explain them.

Solution: We will use product construction to get a DFA for the language L. We define two languages as follows.

$$L_1 = \{w \in \{0,1\}^* \mid \text{num}(w) \text{ is divisible by 3 }\}$$
$$L_2 = \{w \in \{0,1\}^* \mid w \text{ has odd length}\}$$

Note that $L = L_1 \cap L_2$.

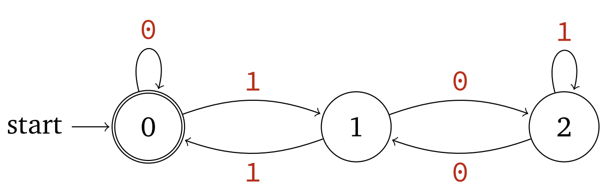
DFA_1 for L_1 is as given below. The states are labeled by an integer indicating the num of the string read so far modulo 3.

$$\begin{aligned} Q_1 &:= \{0,1,2\} \\ s_1 &:= 0 \\ A_1 &:= \{0\} \\ \delta_2(q,0) &:= (2q) \bmod 3 && \forall q \in Q_1 \\ \delta_2(q,1) &:= (2q+1) \bmod 3 && \forall q \in Q_1 \end{aligned}$$

Note that, for any integer n, we have that

$$\begin{aligned} (2n+1) \bmod 2 &= ((2n) \bmod 2 + 1) \bmod 3 \\ (2n) \bmod 2 &= (2(n \bmod 2)) \bmod 3 \end{aligned}$$

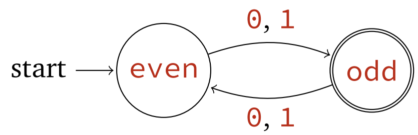
(this was covered in lectures for DFA Product Construction as well as in the lab problems). Here is a drawing of DFA_1.



DFA_2 for L_2 is as given below. The states are labeled odd or even, representing whether the string read so far is of odd or even length, respectively.

$$\begin{aligned} Q_2 &:= \{\texttt{odd}, \texttt{even}\} \\ s_2 &:= \texttt{even} \\ A_2 &:= \{\texttt{odd}\} \\ \delta_1(\texttt{odd}, a) &:= \texttt{even} && \forall a \in \{0,1\} \\ \delta_1(\texttt{even}, a) &:= \texttt{odd} && \forall a \in \{0,1\} \end{aligned}$$

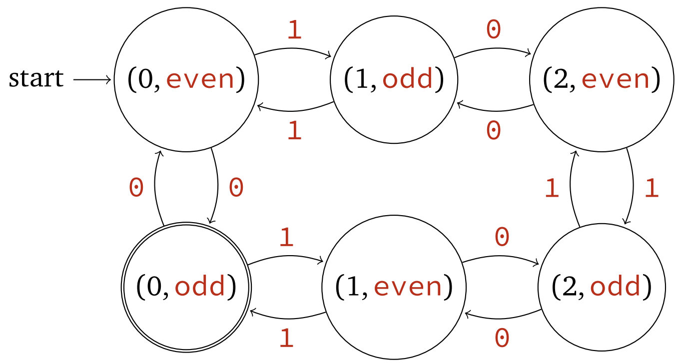
Here is a drawing of DFA_2.



DFA for L is given below, using product construction.

$$
\begin{aligned}
Q &:= Q_1 \times Q_2 = \{0,1,2\} \times \{\texttt{odd}, \texttt{even}\} \\
s &:= (s_1, s_2) = (0, \texttt{even}) \\
A &:= A_1 \times A_2 = \{(0, \texttt{odd})\} \\
\delta((q_1,q_2),a) &:= (\delta_1(q_1,a), \delta_2(q_2,a)) \qquad \forall (q_1,q_2) \in Q, \forall a \in \{\texttt{0},\texttt{1}\}
\end{aligned}
$$

Here is a drawing of the DFA. Note that it has 6 states as required.



■

Rubric: Cryptic and hard to interpret drawings, without any explanation, will receive a score of 0 (if we can't understand your solution, we can't give you credit for it).

- 6 points for the DFA.
  - −1 per small mistake (missing or wrong start/final state(s), missing/wrong transitions, etc).
  - If the final DFA is missing or wrong, then
    - ∗ 1 point for a DFA for accepting odd length strings.
    - ∗ 2 points for a DFA for accepting strings whose num is divisible by 3.
- 4 points for properly labeling and explaining the states.
  - −1 points for missing labels.
  - −3 points for not explaining states in either the individual DFAs or the product construction (−2 to −1 for improper explanations).

3 NFA Construction

Suppose you have constructed NFAs N_1, N_2, N_3 that accept the languages corresponding to the regular expressions r_1, r_2, r_3 respectively. You can assume that N_1, N_2, N_3 have the property that they have exactly one final state and that their start state is different from their final state. Formally $N_1 = (Q_1, \Sigma, \delta_1, s_1, \{f_1\})$, $N_2 = (Q_2, \Sigma, \delta_2, s_2, \{f_2\})$ and $N_3 = (Q_3, \Sigma, \delta_3, s_3, \{f_3\})$.

Describe a NFA N that accepts the language $(r_1 r_2 + r_3)^*$.

Solution: To start off, we can define the concatenation of r_1 and r_2 as

$$\begin{aligned}
Q &:= Q_1 \cup Q_2 \\
s &:= s_1 \\
A &:= \{f_2\} \\
\delta(f_1, \epsilon) &:= \{s_2\} \cup \delta_1(f_1, \epsilon) \\
\delta(f_1, c) &:= \delta_1(f_1, c) && c \neq \epsilon \\
\delta(q, c) &:= \delta_1(q, c) && q \in Q_1 \setminus \{f_1\}, c \in \Sigma \\
\delta(q, c) &:= \delta_2(f_1, c) && q \in Q_2, c \in \Sigma
\end{aligned}$$

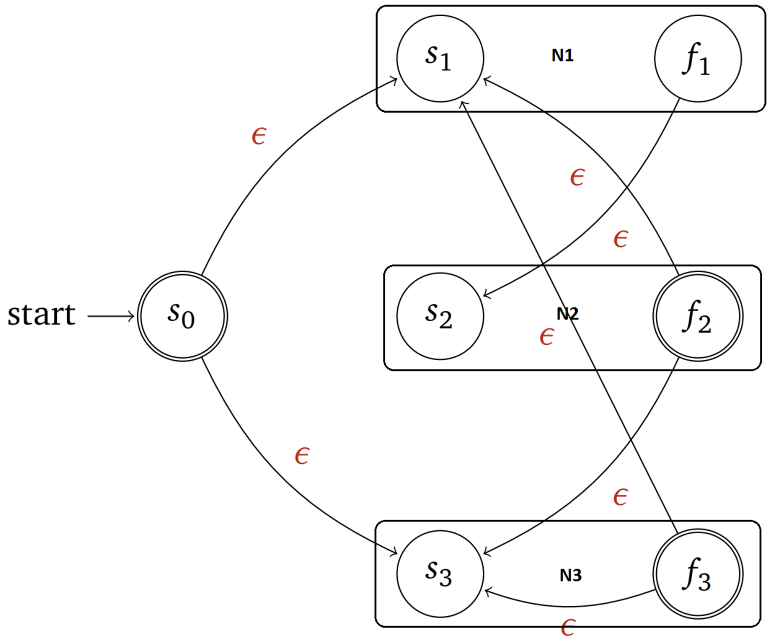
Using this, we can define the union of $r_1 r_2$ and r_3 as

$$\begin{aligned}
Q &:= Q_1 \cup Q_2 \cup Q_3 \cup \{s_0\} \\
s &:= s_0 \\
A &:= \{f_2, f_3\} \\
\delta(s_0, \epsilon) &:= \{s_1, s_3\} \\
\delta(f_1, \epsilon) &:= \{s_2\} \cup \delta_1(f_1, \epsilon) \\
\delta(f_1, c) &:= \delta_1(f_1, c) && c \neq \epsilon \\
\delta(q, c) &:= \delta_1(q, c) && q \in Q_1 \setminus \{f_1\}, c \in \Sigma \\
\delta(q, c) &:= \delta_2(q, c) && q \in Q_2, c \in \Sigma \\
\delta(q, c) &:= \delta_3(q, c) && q \in Q_3, c \in \Sigma
\end{aligned}$$

Finally, we can define the Kleene star of $r_1 r_2 + r_3$ as

$$\begin{aligned}
Q &:= Q_1 \cup Q_2 \cup Q_3 \cup \{s_0\} \\
s &:= s_0 \\
A &:= \{s_0, f_2, f_3\} \\
\delta(s_0, \epsilon) &:= \{s_1, s_3\} \\
\delta(f_1, \epsilon) &:= \{s_2\} \cup \delta_1(f_1, \epsilon) \\
\delta(f_2, \epsilon) &:= \{s_1, s_3\} \cup \delta_2(f_2, \epsilon) \\
\delta(f_3, \epsilon) &:= \{s_1, s_3\} \cup \delta_3(f_3, \epsilon) \\
\delta(f_1, c) &:= \delta_1(f_1, c) && c \neq \epsilon \\
\delta(q, c) &:= \delta_1(q, c) && q \in Q_1 \setminus \{f_1\}, c \in \Sigma \\
\delta(f_2, c) &:= \delta_2(f_2, c) && c \neq \epsilon \\
\delta(q, c) &:= \delta_2(q, c) && q \in Q_2 \setminus \{f_2\}, c \in \Sigma \\
\delta(f_3, c) &:= \delta_3(f_3, c) && c \neq \epsilon \\
\delta(q, c) &:= \delta_3(q, c) && q \in Q_3 \setminus \{f_3\}, c \in \Sigma
\end{aligned}$$

Alternatively, one possible drawing of the NFA is as follows.



■

Rubric: 10 points

- -2 points for each minor error (e.g. missing or incorrect start/final state(s), missing/wrong transitions, etc.)
- -5 points for major error (not including kleene star, concatenation, or union in your NFA)

4 Closure Properties 1

Let $L_{374} = \{w \mid w \text{ contains CS374 as a substring}\}$ and $L_{473} = \{w \mid w \text{ contains CS473 as a substring}\}$. Let

$$L = \{w \mid \text{all substrings of CS374 in } w \text{ occur before any substring of CS473 in } w\}.$$

The string "CS374blahCS473blahCS374" is not in the language while ϵ, "CS374blahCS374blahCS473blah", and "CS473" are in the language.

Express L using L_{374}, L_{473}, Σ^* and using only the operations union, intersection, complement, concatenation, set difference and Kleene star. Briefly explain your expression.

Solution: L_1, L_2 and L_3 are defined as follows:

- $L_1 = (L_{374} \setminus L_{473}).(L_{473} \setminus L_{374})$ contains all strings w which contain *both* CS374 *and* CS473 as substrings where the concatenation forces it to satisfy the condition stated.
- $L_2 = (L_{374} \setminus L_{473}) \cup (L_{473} \setminus L_{374})$ contains all strings w which contain *either* only CS374 *or* CS473 as substrings and thus implicitly satisfy our condition.
- $L_3 = (\overline{L_{374}} \cap \overline{L_{473}})$ contains all strings w which contain *neither* CS374 *nor* CS473 as substrings and therefore implicitly satisfy our condition.

These subcases are exhaustive and thus

$$L = L_1 \cup L_2 \cup L_3$$

■

Solution: (Alternate)
Notation: $A = (L_{374} \setminus L_{473})$ and $B = (L_{473} \setminus L_{374})$

$$L = (\overline{L_{374}} \cup A) \, . \, (\overline{L_{473}} \cup B)$$

■

Rubric: 10 points.

- -3.5 for missing strings belonging to L_1
- -2.5 for missing strings belonging to L_2
- -2.5 for missing strings belonging to L_3
- -1.5 for missing explanation

Scaled to 8 points for using kleene star incorrectly.

5 Non-regularity

For a binary string z let num(z) denote the number such that z is its binary representation. For example num$(0101) = 5$ and num$(1000) = 8$. For convenience we define num$(\epsilon) = 0$. Let

$$L = \{x\#y \mid x, y \in \{0,1\}^*, \text{num}(x) > \text{num}(y)\}.$$

For example the string 0011#1 is in L while the string 110#01000 is not in the language.

- Prove that the language L is *not* regular.

You can use any proof technique you wish including providing a fooling set or via closure properties. If you use a fooling set argument, you need to only provide a justification for the validity of your fooling set.

Solution: Let F be the language 1^*.
Let x and y be distinct arbitrary strings in F.
Without loss of generality, $x = 1^i$ and $y = 1^j$ for some $j > i \geq 0$.
Let $z = \#1^i$.
Then $xz = 1^i\#1^i \notin L$ because $\text{num}(1^i) = \text{num}(1^i)$.
On the other hand, $yz = 1^j\#1^i \in L$ since $\text{num}(1^j) > \text{num}(1^i)$, because $j > i$.
Thus, z distinguishes x and y. F is an infinite fooling set for L, so L cannot be regular. ■

Rubric: 10 points.

- 3 points for selecting a valid fooling set.
- 3 points for z which distinguishes x, y
- 3 points for explaining why $xz \in L, yz \notin L$, or vice versa
- 1 point for using an arbitrary x, y
- -1 point for small mistakes

6 CFG construction

Assume $\Sigma = \{0, 1\}$ for this problem.

- Describe a CFG for the language $\{0^n 1^m \mid n > 0, m > 2n\}$.
- Describe a CFG for the language $\{0^n 1^m \mid n > 0, 0 \le m < 2n\}$.
- Describe a CFG for the *complement* of the language $\{0^n 1^{2n} \mid n \ge 0\}$.

In order to get full credit you should briefly explain how your grammar works, and the role of each nonterminal.

Solution: Intuitively we can parse any string w as follows. First, remove first k $\mathtt{0}$s and $2k$ $\mathtt{1}$s for the largest possible value of k. Then we can modify the remaining strings to make sure it fulfill the requirement of the language.

1. $n > 0, m > 2n$. The remaining string should be $\mathtt{1}^+$. And we need to make sure that there exists at least one $\mathtt{0}$.

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{11} \mid \mathtt{0}A\mathtt{11} & \qquad \{\mathtt{0}^n\mathtt{1}^m \mid m > 2n, n > 0\} \\ A &\to A\mathtt{1} \mid \mathtt{1} & \mathtt{1}^+ \end{aligned}$$

2. $n > 0, m < 2n$. The remaining string should be $\mathtt{0}^+$ or $\mathtt{0}^+\mathtt{1}$.

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{11} \mid B & \qquad \{\mathtt{0}^n\mathtt{1}^m \mid 0 \le m < 2n, n > 0\} \\ B &\to \mathtt{0}B \mid \mathtt{0} \mid \mathtt{01} & \mathtt{0}^+ \text{ or } \mathtt{0}^+\mathtt{1} \end{aligned}$$

3. Complement of $\left\{\mathtt{0}^n\mathtt{1}^{2n} \,\middle|\, n \ge 0\right\}$ is the union of two modified previous cases and $(\mathtt{1}+\mathtt{0})^*\mathtt{10}(\mathtt{1}+\mathtt{0})^*$. Notice that $n \ge 0$, CFG from previous two cases need to be modified.

$$\begin{aligned} S &\to \mathtt{0}S\mathtt{11} \mid A \mid B \mid X & \qquad \{\mathtt{0}^n\mathtt{1}^m \mid m \ne 2n, n \ge 0\} \text{ or } (\mathtt{1}+\mathtt{0})^*\mathtt{10}(\mathtt{1}+\mathtt{0})^* \\ A &\to A\mathtt{1} \mid \mathtt{1} & \mathtt{1}^+ \\ B &\to \mathtt{0}B \mid \mathtt{0} \mid \mathtt{01} & \mathtt{0}^+ \text{ or } \mathtt{0}^+\mathtt{1} \\ X &\to Z\mathtt{10}Z & (\mathtt{1}+\mathtt{0})^*\mathtt{10}(\mathtt{1}+\mathtt{0})^* \\ Z &\to \varepsilon \mid \mathtt{1}Z \mid \mathtt{0}Z & (\mathtt{1}+\mathtt{0})^* \end{aligned}$$

■

Rubric: 10 points

- 2 points: correct $m > 2n$ grammar
- 1 point: how $m > 2n$ grammar works and the role of nonterminals
- 2 points: $m < 2n$ grammar
- 1 point: how $m < 2n$ grammar works and the role of nonterminals
- 2 points: complement grammar
- 2 points: how complement grammar works and the role of nonterminals

7 Closure Properties 2

For a language L we define a language

$$\text{CutOutMid}(L) = \{xz \mid x, y, z \in \Sigma^* \text{ and } xyz \in L\}.$$

For example if $L = \{abcd\}$ then CutOutMid$(L) = \{abcd, bcd, cd, d, \epsilon, acd, ad, a, abd, ab, abc\}$.

- Prove that for any regular language L, the language CutOutMid(L) is regular. If you use a NFA construction explain your formal construction and give a description of your construction. If you use closure properties or regular expression explain your reasoning.

Recall that PREFIX$(L) = \{x \mid x, y \in \Sigma^* \text{ and } xy \in L\}$ is the set of prefixes of strings in L, and SUFFIX$(L) = \{y \mid x, y \in \Sigma^* \text{ and } xy \in L\}$ is the set of suffixes of strings in L. You can convince yourself that CutOutMid$(L) \neq$ PREFIX$(L) \cdot$ SUFFIX(L) to avoid going down an incorrect path.

Solution (NFA Construction): Let L be a regular language. Suppose $M = (Q, \Sigma, \delta, s, A)$ is a DFA for L. We construct an NFA $M' = (Q', \Sigma, \delta', s', A')$ that accepts the language CutOutMid(L). For a state $q \in Q$, we define $R_q = \{p \in Q \mid \text{There exists } w \in \Sigma^* \text{ such that } \delta^*(q, w) = p\}$. Let $Q' = Q \times \{\text{before}, \text{after}\}$, $s' = (s, \text{before})$, $A' = A \times \{\text{after}\}$, and for all $q \in Q$, $r \in \{\text{before}, \text{after}\}$ and $a \in \Sigma$. We define:

$$\delta'((q, r), a) = \begin{cases} \{(\delta(q, a), \text{before})\} \cup R_q \times \{\text{after}\} & r = \text{before} \\ \{(\delta(q, a), \text{after})\} & r = \text{after} \end{cases}$$

The NFA M' basically consists of two copies of M such that the first copy has ϵ transitions to the second. Given a string $w = xz$ such that $xyz \in L$ for some $y \in \Sigma^*$, the NFA starts in the first copy and reads x in the first copy of M. Then at some point, M' non-deterministically guesses that the string x is over and also guesses the state in which M would have landed if it had been given the string xy. Then M' continues to read z from that state in the second copy of M. In the end it lands in an accepting state of the second copy of M, which is, by construction, an accepting state of M'. Conversely, if M' accepts a string w, it has to make exactly one ϵ transition from the first copy of M to the second copy of M. Suppose it makes the ϵ transition from (p, before) to (q, after). Let x correspond to the part of the string that was read before this transition and z be the rest. This ϵ transition corresponds to a string y that would have taken M from state p to q. Since xz takes M' to an accepting state, by construction of M', the string xyz would have taken M to the corresponding state in M. Therefore $xyz \in L$. ■

Rubric: 10 points:

- Allowing multiple skips: -7
- Bad notation: -1 to -3
- Making assumptions about length of y: -3
- Does not check that the state after the "skip" is reachable from the current state in the original automata: -3

Solution (Regular Expressions): We know that for a regular language L, the languages PREFIX(L) and SUFFIX(L) are regular. For a regular expression R, let $P(R)$ and $S(R)$ denote a regular expression for the languages PREFIX$(L(R))$ and SUFFIX$(L(R))$. We prove by induction on the length of regular expressions that for every regular expression R, there exists a regular expression R' describing CutOutMid$(L(R))$.

- **Induction base**:
  - If $L(R) = \emptyset$, then CutOutMid$(L(R)) = \emptyset$ which is a regular set and is described by the empty regular expression.
  - If $R = \epsilon$, the CutOutMid$(L(R)) = \{\epsilon\}$. Therefore $R' = \epsilon$ describes CutOutMid$(L(R))$.
  - If $R = a$ for some $a \in \Sigma$, then CutOutMid$(L(R)) = \{a, \epsilon\}$. Thus, $R' = a + \epsilon$ is a regular expression for CutOutMid$(L(R))$.
- **Induction hypothesis**: Suppose to every regular expression R shorter that n characters, corresponds a regular expression R' such that $L(R') =$ CutOutMid$(L(R))$.
- **Induction step**: Suppose R is a regular expression of length $n > 1$. There are three cases:
  - If $R = R_1 \cup R_2$, then by induction hypothesis, there exist regular expressions R'_1 and R'_2 describing CutOutMid$(L(R_1))$ and CutOutMid$(L(R_2))$, respectively. In this case $R' = R'_1 + R'_2$ describes CutOutMid$(L(R))$.
  - If $R = R_1R_2$, then by induction hypothesis, there exist regular expressions R'_1 and R'_2 describing CutOutMid$(L(R_1))$ and CutOutMid$(L(R_2))$, respectively. In this case $R' = R'_1R_2 + P(R_1)S(R_2) + R_1R'_2$ describes CutOutMid$(L(R))$.
  - If $R = R_1^*$, then by induction hypothesis, there exists a regular expression R'_1 describing CutOutMid$(L(R_1))$. In this case $R' = R_1^*(R'_1 + P(R_1)S(R_1))R_1^*$ describes CutOutMid$(L(R))$.

■

Rubric: 10 points:

- 1 point for each of the three base cases.
- 2 points for union and concatenation.
- 3 points for Kleene star.