1 Regular Expression

Give regular expressions for the following two languages.

- $L_1 = \{w \in \{0,1\}^* \mid w \text{ does } \textit{not} \text{ have the substring } 111\}$. The string 11001 is in L_1 but not 111101.
- $L_2 = \{w \in \{0,1\}^* \mid \text{all 0 blocks of } w \text{ are of even length}\}$. Recall a 0 block is a maximal non-empty subtring of 0's in the string. The string 0000100 is in L_2 but not 001110.

Briefly explain your regular expressions.

Solution:

L_1: $\boxed{(\texttt{0} + \texttt{10} + \texttt{110})^*(\varepsilon + \texttt{1} + \texttt{11})}$ We cannot have more than two `1`s appear without immediately following with a `0` or the end of the string. This is the same as Lab 1½ Problem 2, except excluding `111` instead of `000`.

L_2: $\boxed{(\texttt{00} + \texttt{1})^*}$ The term `00` only allows an even number of `0`s at a time. ■

Rubric: 10 points: 5 points for each part.

- 1 points for a brief English explanation of your regular expression. This is how you argue that your regular expression is correct.
  - For longer expressions, you should explain each of the major components of your expression, and separately explain how those components fit together.
  - We do not want a *transcription*; don't just translate the regular-expression notation into English.
- 4 points for correctness.
  - -1 for a single mistake: one typo, excluding exactly one string in the target language, or including exactly one string not in the target language.
  - -2 for incorrectly including/excluding more than one but a finite number of strings.
  - -4 for incorrectly including/excluding an infinite number of strings.
- Regular expressions that are more complex than necessary may be penalized. Regular expressions that are significantly too complex may get no credit at all. On the other hand, minimal regular expressions are *not* required for full credit.

2 DFA Construction

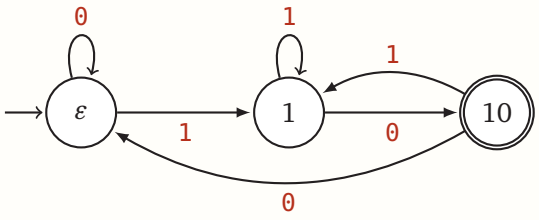
Draw or describe a DFA for the language defined below.

$$L = \{w \in \{0,1\}^* \mid w \text{ ends in } 10 \text{ and } |w| \text{ is odd}\}.$$

Your DFA must have at most 6 states. Label the states and explain them.

Solution: L is the intersection of the languages $L_1 = \{w \in \{0,1\}^* \mid w \text{ ends in } 10\}$ and $L_2 = \{w \in \{0,1\}^* \mid |w| \text{ is odd}\}$.

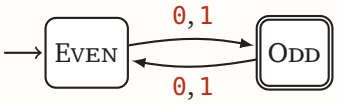
Here is a DFA M_1 for L_1. This is the same DFA seen in Homework 1 Problem 2 solutions.



- The state ε means no symbols have been read yet, only 0 has been read, or the last two symbols read were 00
- The state 1 means the last symbol read was 1
- The state 10 means the last two symbols read were 10

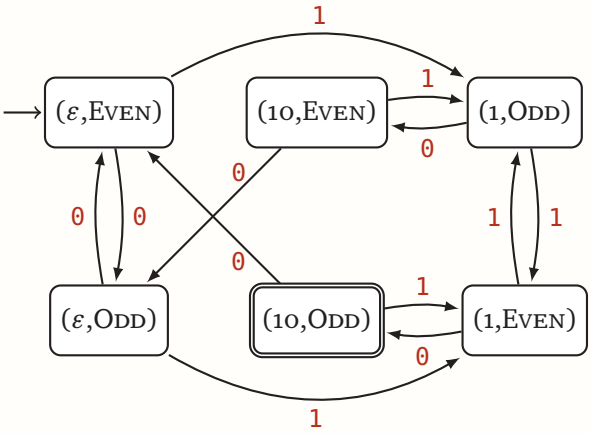
Equivalently, the state labeled x corresponds to the maximal prefix of 10 seen in the (up to) two most recently read symbols.

Here is a DFA M_2 for L_2:



- The state Even means that we have read an even number of characters
- The state Odd means that we have read an odd number of characters

Finally, the DFA M for L can be built by applying the product construction to M_1 and M_2. Note that the positions of the states have been rearranged to reduce edge crossings.



■

Rubric: 10 points. Standard DFA design rubric.

- 2 points for an unambiguous description of a DFA, including the states set Q, the start state s, the accepting states A, and the transition function δ.
  - **Drawings:** Use an arrow from nowhere to indicate s, and doubled circles to indicate accepting states A. If $A = \emptyset$, say so explicitly. If your drawing omits a junk/trash/reject state, say so explicitly. Draw neatly! If we can't read your solution, we can't give you credit for it.
  - **Text descriptions:** You can describe the transition function either using a 2d array, using mathematical notation, or using an algorithm.
  - **Product constructions:** You must give a complete description of each the DFAs you are combining (as either drawings, text, or recursive products), together with the accepting states of the product DFA.
- 2 points for briefly explaining the purpose of each state in English. This is how you argue that your DFA is correct.
  - For product constructions, explaining the states in the factor DFAs is both necessary and sufficient.
- 6 points for correctness.
  - -1 for a single mistake: a single misdirected transition, a single missing or extra accepting state, rejecting exactly one string that should be accepted, or accepting exactly one string that should be accepted.
  - -3 for incorrectly accepting/rejecting more than one but a finite number of strings.
  - -6 for incorrectly accepting/rejecting an infinite number of strings.
- DFAs that are more complex than necessary may be penalized. DFAs that are significantly more complex than necessary may get no credit at all. On the other hand, *minimal* DFAs are not required for full credit, unless the problem explicitly asks for them.
- Half credit for describing an NFA when the problem asks for a DFA.

3 Regular Expressions

Given a language L over alphabet Σ and string $x \in \Sigma^*$ we define the language insert(L, x) as the set of all strings w where w is obtained by taking a string $w' \in L$ and inserting x in w' at some position. More formally insert$(L, x) = \{uxv \mid uv \in L\}$. For example if $L = \{CS374\}$ and $x = not$ then insert$(L, x) = \{notCS374, CnotS374, CSnot374, CS3not74, CS37not4, CS374not\}$. In this problem, assuming that L is regular, you will derive an algorithm that generates a regular expression r' for insert(L, x) from regular expression r for L.

- For each of the base case of $r = \emptyset, \varepsilon$ and $r = a$, $a \in \Sigma$, write down the regular expression for insert$(L(r), x)$
- Assume $r = r_1 + r_2$ and that r'_1 and r'_2 are regular expressions for insert$(L(r_1), x)$ and insert$(L(r_2), x)$ respectively. Write a regular expression for insert$(L(r), x)$ terms of r_1, r_2, r'_1, r'_2, x (you do not have to use all of these).
- Assume $r = r_1 r_2$ and that r'_1 and r'_2 are regular expressions for insert$(L(r_1), x)$ and insert$(L(r_2), x)$ respectively. Write a regular expression for insert$(L(r), x)$ terms of r_1, r_2, r'_1, r'_2, x (you do not have to use all of these).
- Assume $r = r_1^*$ and that r'_1 is a regular expression for insert$(L(r_1), x)$. Write a regular expression for insert$(L(r), x)$ in terms of r_1, r'_1, x (you do not have to use all of these).

Solution:

- Base cases:
  - For $r = \emptyset$, $insert(\emptyset, x) = \emptyset$, so the regular expression is $\boxed{\emptyset}$
  - For $r = \varepsilon$, $insert(\{\varepsilon\}, x) = \{x\}$, so the regular expression is $\boxed{x}$
  - For $r = a$, $insert(\{a\}, x) = \{ax, xa\}$, so the regular expression is $\boxed{ax + xa}$
- $insert(L(r_1 + r_2), x) = insert(L(r_1) \cup L(r_2), x) = \{uxv \mid uv \in L(r_1) \cup L(r_2)\}$
  $= \{uxv \mid uv \in L(r_1)\} \cup \{uxv \mid uv \in L(r_2)\} = insert(L(r_1), x) \cup insert(L(r_2), x)$,
  so the regular expression for $insert(L(r_1 + r_2), x)$ is $\boxed{r'_1 + r'_2}$
- Assume r_1 and r_2 are not $\emptyset$, otherwise $r = \emptyset$. We can insert x inside of $L(r_1)$, inside of $L(r_2)$, or between two strings from each language. In summary, the regular expression for $insert(L(r_1 r_2), x)$ is $\boxed{r'_1 r_2 + r_1 x r_2 + r_1 r'_2}$ $r'_1 r_2$ and $r_1 r'_2$ allow for inserting x at the end of r_1 and the beginning of r_2, respectively, so the $r_1 x r_2$ term is redundant.
- Assume $r_1 \neq \emptyset$, otherwise $r_1^* = \varepsilon$. We can insert x inside of some instance of $L(r_1)$, or between two instances of $L(r_1)$. In summary, the regular expression for $insert(L(r_1^*), x)$ is $\boxed{r_1^*(r'_1 + x)r_1^*}$ Unlike the previous case, the $+x$ term is *not* redundant: we must allow for the possibility of inserting x when r_1^* produces ε. Accordingly, $\boxed{r_1^* r'_1 r_1^* + x}$ is an equivalent expression for $insert(L(r_1^*), x)$. ■

Rubric: 10 points.

- 1 point for syntactically correct regular expressions in all parts.
- 1 point for each correct base case (all or nothing).
- 2 points for each correct recursive case. Minimal regular expressions are *not* required.
  - For the last part, half credit for missing $+x$ somewhere. All or nothing for all other parts.

4 NFAs

Let $N_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$ and $N_3 = (Q_3, \Sigma, \delta_3, s_3, A_3)$ be three NFAs. Let n_1, n_2, n_3 be the number of states in N_1, N_2, N_3 respectively. Describe an NFA $N = (Q, \Sigma, \delta, s, A)$ that accepts the language $(L(N_1) \cap L(N_2)) \cup L(N_3)$. For full credit your NFA N should have $O(n_1 n_2 + n_3)$ states.

- Give a high-level description of how you can construct N using procedures you have learnt in lecture or homework.
- Give a formal tuple notation for N. That is, you need to explain Q, δ, s, A in terms of the parameters of N_1, N_2, N_3.

Solution:

- We can apply the product construction for intersection from Homework 2 Problem 3 to N_1 and N_2 to obtain an NFA N_{12} for $L(N_1) \cap L(N_2)$. Then we can apply the construction for union in Thompson's algorithm to N_{12} and N_3 to obtain the desired NFA N for $(L(N_1) \cap L(N_2)) \cup L(N_3)$.
- Below we assume that $(Q_1 \times Q_2) \cap Q_3 = \emptyset$, so that we do not accidentally combine states in the definition of Q. We introduce a new starting state s' with ε transitions to the starting states of N_{12} and N_3. The transitions for $(q_1, q_2) \in Q_1 \times Q_2$ are lifted directly from Homework 2 Problem 3 solutions. To simplify the construction slightly, we will ignore the requirement in Thompson's algorithm of having exactly one accepting state.

$$
\begin{aligned}
Q &:= \{s'\} \cup (Q_1 \times Q_2) \cup Q_3 \\
s &:= s' \\
A &:= (A_1 \times A_2) \cup A_3 \\
\delta(s', \varepsilon) &:= \{(s_1, s_2), s_3\} \\
\text{For } (q_1, q_2) \in Q_1 \times Q_2, a \in \Sigma, \quad \delta((q_1, q_2), a) &:= \delta_1(q_1, a) \times \delta_2(q_2, a) \\
\text{For } (q_1, q_2) \in Q_1 \times Q_2, \quad \delta((q_1, q_2), \varepsilon) &:= \big(\delta_1(q_1, \varepsilon) \times \{q_2\}\big) \cup \big(\{q_1\} \times \delta_2(q_2, \varepsilon)\big) \\
\text{For } q_3 \in Q_3, \quad \delta(q_3, a) &:= \delta_3(q_3, a)
\end{aligned}
$$

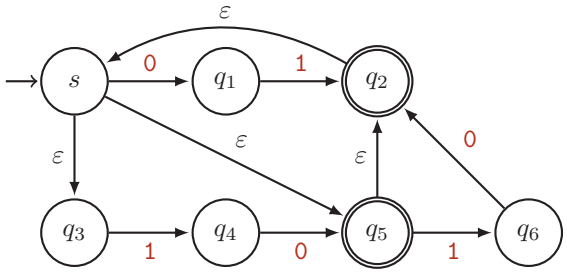
■

Rubric: 10 points: 5 points for each part.

- Full points for any correct high-level description of the construction that results in $O(n_1 n_2 + n_3)$ states.
  - Partial credit: 2 points for a correct construction with $\omega(n_1 n_2 + n_3)$ states.
- 5 points for correctly (formally) describing an NFA. Do not penalize for overly large NFAs; we already did that in the previous part.
  - +2.5 for accepting all strings in the target language
  - +2.5 for accepting only strings in the target language
  - −1 for a single mistake in the formal description (for example a typo)
  - Double-check correctness when the input language is $\emptyset$, or ε, or 0^*, or Σ^*.

5 NFAs and Subset Construction

Consider the NFA N shown below.



- Show that 1001 is accepted by N by describing an accepting walk in the NFA.
- What is $\delta^*(s, \varepsilon)$?
- What is the ε-closure of $\{q_3, q_5\}$?
- Consider the subset construction to create a DFA $M = (Q', \Sigma, \delta', s', A')$ from N. What is $\delta'(X, 0)$ where $X = \{q_1, q_2\}$?
- Argue that 100 is not accepted by N by computing $\delta^*(s, 100)$.

Solution:

- One possibility: $s \xrightarrow{\varepsilon} q_3 \xrightarrow{1} q_4 \xrightarrow{0} q_5 \xrightarrow{\varepsilon} q_2 \xrightarrow{\varepsilon} s \xrightarrow{0} q_1 \xrightarrow{1} q_2$, where q_2 is an accepting state.
  Another possibility: $s \xrightarrow{\varepsilon} q_5 \xrightarrow{1} q_6 \xrightarrow{0} q_2 \xrightarrow{\varepsilon} s \xrightarrow{0} q_1 \xrightarrow{1} q_2$, where q_2 is an accepting state.
- $\{s, q_2, q_3, q_5\}$
- $\{s, q_2, q_3, q_5\}$
- $\{q_1\}$
- $\delta^*(s, 100) = \{q_1\}$, which does not contain any accepting states.

■

Rubric: 10 points. 2 points for each part (all or nothing).

6 Non-regularity

Prove that the language $L = \{0^{3^n} \mid n \geq 0\}$ is not regular. You can use any proof technique you want. If you describe a fooling set for the language you need to justify the validity of the fooling set.

Solution: Following the idea for the language $\left\{0^{2^n} \;\middle|\; n \geq 0\right\}$:

Let $F = L = \left\{0^{3^n} \;\middle|\; n \geq 0\right\}$.

Let x and y be arbitrary elements of F.

Then $x = 0^{3^i}$ and $y = 0^{3^j}$ for non-negative integers i and j where $i \neq j$.

For $\left\{0^{2^n} \;\middle|\; n \geq 0\right\}$, we set $z = 0^{2^i}$, so that $xz = 0^{2^{i+1}}$. The analogous situation here would be $xz = 0^{3^{i+1}}$. To achieve this: Let $z = 0^{2 \cdot 3^i}$.

Then $xz = 0^{3^i} 0^{2 \cdot 3^i} = 0^{3 \cdot 3^i} = 0^{3^{i+1}} \in L$, and

$yz = 0^{3^j} 0^{2 \cdot 3^i} = 0^{3^j + 2 \cdot 3^i} \notin L$ since $i \neq j$.

Thus F is an infinite fooling set for L, so L cannot be regular. ■

Rubric: 10 points. Standard fooling set rubric:

- 4 points for the fooling set:
  - +2 for explicitly describing the proposed fooling set F.
  - +2 if the proposed set F is actually a fooling set for the target
  - No credit for the proof if the proposed set is not a fooling set.
  - No credit for the *problem* if the proposed set is finite.
- 6 points for the proof:
  - The proof must correctly consider *arbitrary* strings $x, y \in F$.
    - No credit for the proof unless both x and y are *always* in F.
    - No credit for the proof unless x and y can be *any* strings in F.
  - +2 for correctly describing a suffix z that distinguishes x and y.
  - +2 for proving either $xz \in L$ or $yz \in L$
  - +2 for proving either $yz \notin L$ or $xz \notin L$, respectively.

7 Regularity

Given a string w a *prefix* is any string x such that there is a string y such that $xy = w$. A proper prefix of w is a string x such that there is y with $|y| \geq 1$ such that $xy = w$. We will call a string x a proper-proper prefix of w if there is a string y such that $|y| \geq 2$ and $xy = w$. If $w = abcde$ then abc, ab, a, and ε are proper-proper prefixes of w. For a language

$$\text{PPPREFIX}(L) = \{x \mid x \text{ is proper-proper prefix of some string } w \in L\}$$

Alternatively,

$$\text{PPPREFIX}(L) = \{x \mid \exists y, |y| \geq 2, xy \in L\}.$$

Suppose L is regular. Prove that PPPREFIX(L) is regular. In particular, given a DFA $M = (Q, \Sigma, \delta, s, A)$ for L describe an NFA N that accepts the language PPPREFIX($L(M)$).

Solution: We will think of PPPREFIX(L) as the set of strings obtained by deleting a string of length at least two from the end of a string in L. Following the examples of the various *delete* problems seen before, we will simulate reading in the input, and then guessing transitions corresponding to reading a string of length at least two.

For each $q \in Q$, let $X_q \subseteq Q$ be the set of states of M that can be reached from q by following two or more transitions. Equivalently, $X_q = \{\delta^*(q, y) \mid y \in \Sigma^* \text{ such that } |y| \geq 2\}$. We will create two copies of M labeled BEFORE and AFTER. For each $q \in Q$, add ε-transitions from the BEFORE copy of q to the AFTER copies of the elements of X_q. The AFTER copy has no transitions, and we accept only if we land on a copy of an accepting state.

Intuitively, the NFA reads some amount of the input to get to some state (q, BEFORE), and then guesses some string y of length at least 2 and takes an ε-transition to $(\delta^*(q, y), \text{AFTER})$.

Formally, we create an NFA $N = (Q', \Sigma, \delta', s', A')$ as follows:

$$\begin{aligned}
Q' &:= Q \times \{\text{BEFORE}, \text{AFTER}\} \\
s' &:= (s, \text{BEFORE}) \\
A' &:= \{(q, \text{AFTER}) \mid q \in A\} \\
\delta'((q, \text{BEFORE}), a) &:= \begin{cases} \{(\delta(q, a), \text{BEFORE})\} & a \in \Sigma \\ \{(\delta^*(q, y), \text{AFTER}) \mid y \in \Sigma^*, |y| \geq 2\} = X_q \times \{\text{AFTER}\} & a = \varepsilon \end{cases} \\
\delta'((q, \text{AFTER}), a) &:= \varnothing
\end{aligned}$$

■

Solution: Recall that for PREFIX($L(M)$), one possible solution is to compute a set X consisting of states that can reach an accepting state, and then set X to be the new set of accepting states. We will adapt this idea to solve PPPREFIX.

Let $X \subseteq Q$ be the set of states of M that can reach an accepting state by following a walk of length *at least two*. Equivalently, $X = \{q \in Q \mid \exists y, |y| \geq 2 \text{ and } \delta^*(q, y) \in A\}$.

Observe that $\delta^*(s, x) \in X$ if and only if there is some string y of length at least two such that $\delta^*(s, xy) = \delta^*(\delta^*(s, x), y) \in A$. So all we need to do is change the set of accepting states to X.

So in fact the **DFA** $\boxed{N = (Q, \Sigma, \delta, s, X)}$ accepts PPPREFIX($L(M)$). ■

Solution: Notice that $\text{PPPREFIX}(L) = \textsc{Reverse}(\text{PS}\textsc{uffix}(\text{PS}\textsc{uffix}(\textsc{Reverse}(L))))$. Thus given M, we can apply the construction given in lecture/Jeff's notes for REVERSE, then apply the construction from Homework 2 Problem 2 for PSUFFIX *twice*, and finally apply the construction for REVERSE once more to obtain an NFA N for $\text{PPPREFIX}(L)$. Strictly speaking, the constructions given in lecture/notes and Homework apply to DFAs, not NFAs, but we can run the incremental subset construction in between as necessary to convert the intermediary NFAs into DFAs before feeding them into the next construction. ■

Rubric: 10 points. Standard langage transformation rubric:

- + 2 for a formal, complete, and unambiguous description of the output automaton, including the states, the start state, the accepting states, and the transition function, as functions of an arbitrary input DFA.
  - – No points for the rest of the problem if this is missing.
- + 2 for a brief English explanation of the output automaton. We explicitly do not want a formal proof of correctness, or an English transcription, but a few sentences explaining how your machine works and justifying its correctness. What is the overall idea? What do the states represent? What is the transition function doing? Why these accepting states?
- + 6 for correctness
  - – +3 for accepting all strings in the target language
  - – +3 for accepting only strings in the target language
  - – −1 for a single mistake in the formal description (for example a typo)
  - – Double-check correctness when the input language is $\emptyset$, or ε, or 0^*, or Σ^*.