HW2CSE105W25: Sample solutions

CSE105W25 Team

Due: January 30th at 5pm, via Gradescope

In this assignment,

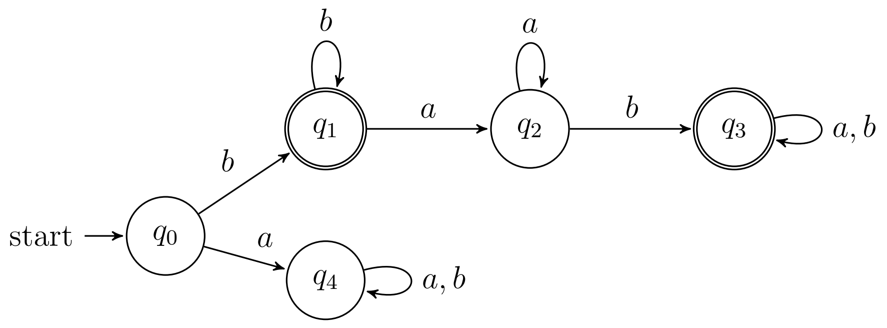
You will practice designing multiple representations of regular languages and working with general constructions of automata to demonstrate the richness of the class of regular languages.

Resources: To review the topics for this assignment, see the class material from Week 2 and Week 3. We will post frequently asked questions and our answers to them in a pinned Piazza post.

Reading and extra practice problems: Sipser Section 1.1, 1.2, 1.3. Chapter 1 exercises 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13, 1.14, 1.15, 1.16, 1.17, 1.18, 1.19, 1.20, 1.21, 1.22, 1.23. Chapter 1 problem 1.31, 1.36, 1.37.

Assigned questions

1. **Finite automata** (10 points): Consider the finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ whose state diagram is depicted below



(a) (*Graded for completeness*) [1] Write the formal definition of this automaton. In other words, give the five defining paramaters Q, Σ, δ, q_0, F so that they are consistent with the state diagram of M.

[1]This means you will get full credit so long as your submission demonstrates honest effort to answer the question. You will not be penalized for incorrect answers. To demonstrate your honest effort in answering the

Solution: Based on the state diagram,

- $Q = \{q_0, q_1, q_2, q_3, q_4\}$ (the collection of labels of nodes in the graph)
- $\Sigma = \{a, b\}$ (the collection of labels of arrow in the graph)
- $\delta : Q \times \Sigma \to Q$ is given by the table of values

	a	b
q_0	q_4	q_1
q_1	q_2	q_1
q_2	q_2	q_3
q_3	q_3	q_3
q_4	q_4	q_4

- The start state is q_0
- The set of accepting states is $F = \{q_1, q_3\}$

(b) (*Graded for correctness*) [2] Give a regular expression R so that $L(R) = L(M)$. In other words, we want a regular expression that describes the language recognized by this finite automaton. Justify your answer by referring to the definition of the semantics of regular expressions and computations of finite automata. Include an explanation for why each string in $L(R)$ is accepted by the finite automaton *and* for why each string not in $L(R)$ is rejected by the finite automaton.

Ungraded bonus: can you find more than one such regular expression?

Solution: The regular expression will trace the labels of paths from the start state to the accepting state.

$$R = b^+ \quad \cup \quad bb^*aa^*b(a \cup b)^*$$

To justify that this regular expression describes the language recognized by the finite automaton:

- Consider arbitrary string in $L(R)$: by definition of union concatenation, star, and $(a \cup b)^*$, this string can be written as

$$b^i a^j b w$$

for some integer $i > 0$, integer $j > 0$, and string w OR as b^i for some integer $i > 0$. In the first case: The computation of the automaton on this string starts

question, we expect you to include your attempt to answer *each* part of the question. If you get stuck with your attempt, you can still demonstrate your effort by explaining where you got stuck and what you did to try to get unstuck.

[2] This means your solution will be evaluated not only on the correctness of your answers, but on your ability to present your ideas clearly and logically. You should explain how you arrived at your conclusions, using mathematically sound reasoning. Whether you use formal proof techniques or write a more informal argument for why something is true, your answers should always be well-supported. Your goal should be to convince the reader that your results and methods are sound.

at q_0, then transitions to q_1 on reading the first b (there's at least one b because $i > 0$) and stays at q_1 while reading the next $i-1$ b's, then transitions to q_2 on reading the first a (there's at least one a because $j > 0$) and stays at q_2 while reading the next $j-1$ a's, then transitions to q_3 when reading the b, and then stays at q_3 while reading each character of w in turn. At the end of reading the string, the computation is at q_3, so accepts. In the second case: The computation of the automaton on this string starts at q_0, then transition to q_1 on reading the first b (there's at least one b because $i > 0$) and stays at q_1 while reading the next $i-1$ b's, and then the machine accepts the string. Thus, any string in $L(R)$ is accepted by the automaton.

- Consider arbitrary string accepted by the automaton. Since it's accepted, the computation of the automaton on the string must start at q_0 and end at q_1 or at q_3. The only paths in the state diagram that has that property are labelled by b to start, then some number of steps following the self loop at q_1, then end or continue by following an edge labelled by an a, then some number of steps following the self loop at q_2, then a b, then some number of steps following the self loop at q_3. In particular, the concatenation of labels of arrows followed in this path is bb^m or $bb^m aa^n bu$ for some integer $m > 0$, integer $n > 0$, and string u. This string exactly matches the pattern described in R.

Note: another regular expression we could have considered instead

$$b^+ \quad \cup \quad b(a \cup b)^* ab(a \cup b)^*$$

(c) (*Graded for completeness*) Keeping the same set of states Q, input alphabet Σ, same start state q_0, and same transition function δ, choose a new set of accepting states F_{new1} so that the new finite automaton $M_1 = (Q, \Sigma, \delta, q_0, F_{new1})$ that results recognizes a **proper superset** of $L(M)$, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your choice of F_{new1} *and* a precise and clear explanation of why every string that is accepted by M is also accepted by M_1 *and* an example of a string that is accepted by M_1 and is rejected by M; or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

Solution: We choose $F_{new1} = Q$, i.e. make all the states accepting. Since q_1 and q_3 are still accepting states, any string that would have been accepted by M is still accepted by M_1 because computations that reach q_1 or q_3 in M_1 are the same sequence of states as in M. However, the string a is accepted by M_1 (because its computation lands in q_4, which is in F_{new1}) but is rejected by M (because q_4 is not accepting in M).

(d) (*Graded for correctness*) Keeping the same set of states Q, input alphabet Σ, same start state q_0, and same transition function δ, choose a new set of accepting states F_{new2} so that the new finite automaton $M_2 = (Q, \Sigma, \delta, q_0, F_{new2})$ that results recognizes a **nonempty proper subset** of $L(M)$, or explain why there is no such example. A complete solution will include either (1) a precise and clear description of your choice of F_{new2} *and* an example string accepted by M_2 *and* a precise and clear explanation of why every string that is accepted by M_2 is also accepted by M *and* an example of a string that is accepted by M and is rejected by M_2; or (2) a sufficiently general and correct argument why there is no such example, referring back to the relevant definitions.

Solution: We choose $F_{new2} = \{q_1\}$, i.e. include only one of the two originally accepting states. An example string accepted by M_2 is *bb* (the computation of M_2 on this string is q_0, q_1, q_1). For each string accepted by M_2, the computation of M_2 starts at q_0 and then transitions to q_1 and stays there (because q_1 is the only accepting state in M_2, and the only incoming arrows to q_1 are the arrow from q_0 and the self-loop). Since the start state and transition function of M are the same as those of M_2, the computation of M on each of these strings also starts at q_0 and then transitions to q_1 and stays there. Thus, M accepts each of these strings because q_1 is in the set of accepting states of M. However, the string *bab* is accepted by M but is rejected by M_2 (because the computation of both machines on this string is q_0, q_1, q_2, q_3 and q_3 is in the set of accepting states for M but not for M_2).

2. **Automata design** (12 points): As background to this question, recall that integers can be represented using base b expansions, for any convenient choice of base b. The precise definition is: for b an integer greater than 1 and n a positive integer, the **base b expansion of n** is defined to be

$$(a_{k-1} \cdots a_1 a_0)_b$$

where k is a positive integer, $a_0, a_1, \ldots, a_{k-1}$ are nonnegative integers less than b, $a_{k-1} \neq 0$, and

$$n = \sum_{i=0}^{k-1} a_i b^i$$

Notice: *The base b expansion of a positive integer n is a string over the alphabet $\{x \in \mathbb{Z} \mid 0 \leq x < b\}$ whose leftmost character is nonzero.*

An important property of base b expansions of integers is that, for each integer b greater than 1, each positive integer $n = (a_{k-1} \cdots a_1 a_0)_b$, and each nonnegative integer a less than b,

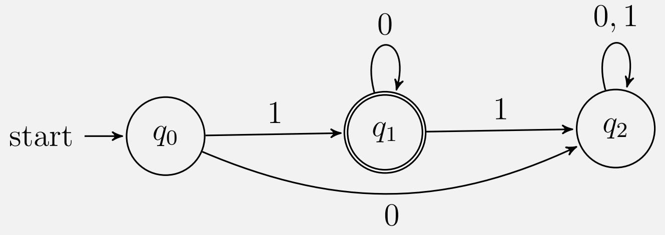
$$bn + a = (a_{k-1} \cdots a_1 a_0 a)_b$$

In other words, shifting the base b expansion to the left results in multiplying the integer value by the base. In this question we'll explore building deterministic finite automata that recognize languages that correspond to useful sets of integers.

(a) (*Graded for completeness*) Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are powers of 2. A complete solution will include the state diagram of your DFA and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Hints: (1) A power of 2 is an integer x that can be written as 2^y for some nonnegative integer y, (2) the DFA should accept the strings 100, 10 and 100000 and should reject the strings 010, 1101, and ε (can you see why?).

Solution:



This DFA accepts strings over $\{0, 1\}$ that start with exactly a single 1 and then have only 0s. Each state takes the following role:

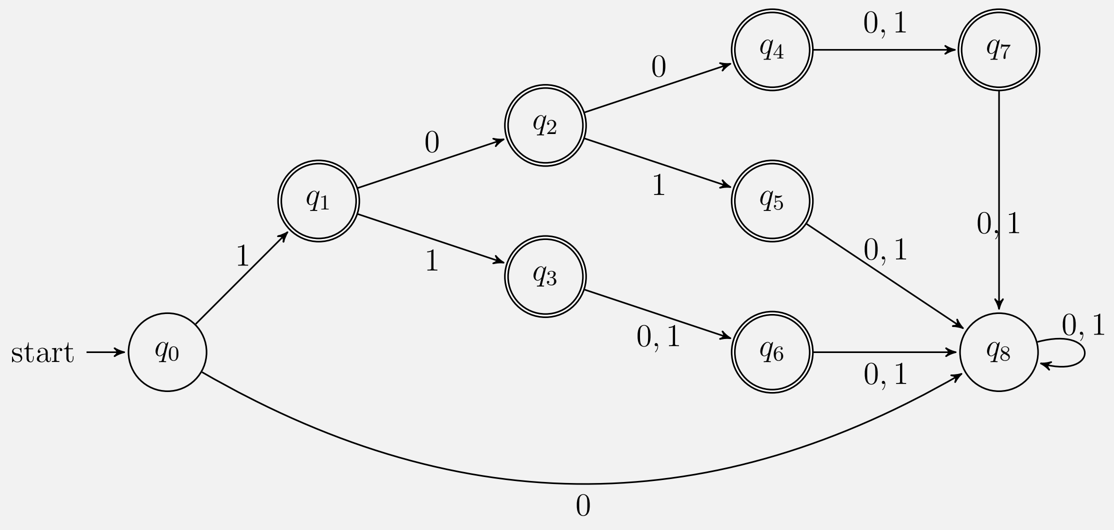
- q_0 is the starting state, and its role is to not accept the empty string by not being one of the accept states.
- q_1 is the state that makes sure the string starts with exactly one 1 and none comes after. We enter q_1 if the first character is a 1. If all we see afterward are zeros, we accept. Otherwise, we exit this only accept state.
- q_2 is the trap state. If the string starts with a 0 or has more than two 1s the computation enters this non-accept state and doesn't leave until we finish reading the string, at which point we reject.

Now, we need to show why strings starting with exactly a single 1 and than have only 0s are exactly the set of binary strings representing powers of 2.

- ($\subseteq$) Take any string s with a single 1 at the front and 0's afterward, its value is determined by that first 1 since all the zero coefficients do not contribute to the sum. Say $s = 1z_1z_2...z_n$ where $n \geq 0$, the value represented by s is $1 \cdot 2^n$, which is a power of 2.
- ($\supseteq$) Take any binary string s representing a power of 2. The first bit contributes 2^x for some $x \leq y$ to the sum. By induction, one can prove that $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ (left as an exercise). To put it another way, even if all digits of s were 1, the value it represents is $\sum_{i=0}^{x} 2^i = 2^{x+1} - 1$, which is one less than 2^{x+1}, the next power of 2 greater than 2^x. This means that none of the bits after the first can be a 1 or else s will not represent a power of 2. This completes the proof.

(b) (*Graded for correctness*) Design a DFA that recognizes the set of binary (base 2) expansions of positive integers that are less than 10. Your DFA must use **fewer than ten states**. A complete solution will include the state diagram of your DFA and a brief justification of your construction by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Solution:



By definition, the binary expansion of number 1 is written as string 1, since $1 = 1 \cdot 2^0$. The binary expansion of number 2 is written as string 10, since $2 = 1 \cdot 2^1 + 0 \cdot 2^0$. The binary expansion of number 3 is written as string 11, since $3 = 1 \cdot 2^1 + 1 \cdot 2^0$. Following the same reasoning, we can write out the set of binary expansions of the integers 1 through 9, which is $L = \{1, 10, 11, 100, 101, 110, 111, 1000, 1001\}$. The above DFA recognizes the set L, and each state takes the following role:

- q_0 is the starting state, and since we should reject the empty string, q_0 is not an accept state
- q_1 through q_7 are states that make sure only binary representations of positive integers less than 10 are accepted by the DFA. The computation on any string in L will end in one of the states in q_1 through q_7 which are the only accept states, and thus the strings in L are all accepted. Specifically,
  - q_1 makes sure that we accept string 1. The computation of the DFA on 1 is q_0, q_1 which ends in q_1, and since q_1 is an accept state, we accept 1.
  - q_2 makes sure that we accept string 10. The computation of the DFA on 10 is q_0, q_1, q_2 which ends in q_2, and since q_2 is an accept state, we accept the string 10.

- q_3 makes sure that we accept string 11. The computation of the DFA on 11 is q_0, q_1, q_3 which ends in q_3, and since q_3 is an accept state, we accept the string 11.
- Similarly, the computation on 100 will end in q_4, the computation on 101 will end in q_5, the computation on 110 and 111 will end in q_6, and the computation on 1000 and 1001 will end in q_7, which make sure that all these strings in L are accepted.

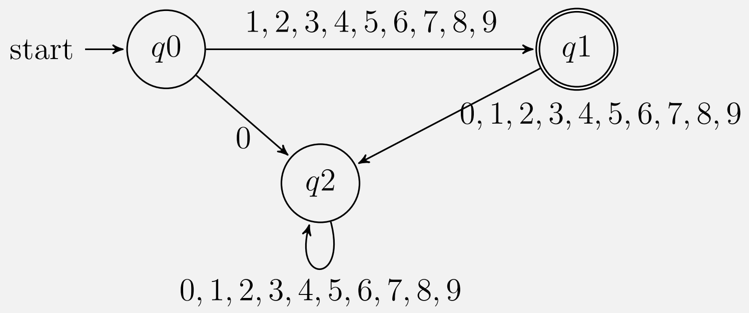
On the other hand, the computation on the empty string will end in non-accept state q_0, and the computation on any other strings not in L will end in non-accept state q_8. So all and only strings in L are accepted.

- q_8 is the trap state. If the string starts with a 0, or starts with a 1 but does not equal to any of the nine strings we want to accept, the computation enters this non-accept state and doesn't leave until we finish reading the string, at which point we reject.

(c) (*Graded for completeness*) Find a positive integer B greater than 1 so that there is a DFA that recognizes the set of base B expansions of positive integers that are less than 10 and it uses as few states as possible. A complete solution will include the state diagram of your DFA and a brief justification of your choice of base.

Hint: sometimes rewriting the defining membership condition for a set in different ways helps us find alternate representations of that set.

Solution:



We choose B to be 10. When $B = 10$, the alphabet we are working with is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. And by definition of base 10 expansion, the set of base 10 expansions of positive integers less than 10 is just the set $L = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. A DFA recognizing this language is shown above, where q_0 is the start state to make sure that we reject the empty string, q_1 is the only accept state that make sure that only strings in L are accepted, and q_2 is the trap state to make sure that we reject all other strings, i.e. strings starting with 0 or with length more than 1. Therefore, we have a DFA with 3 states to recognize the language.

For any choice of B, the start state cannot be an accept state since the empty string is not a valid representation of an integer. Also, there must be at least one accept state to accept all and only the strings that are base B expansions of positive integers that are less than 10. We also need at least one trap state (a non-accept state) distinct from the start state to indicate that the string starts with 0, or the symbols read so far already represent an integer greater than or equal to 10. Thus, any choice of B requires a DFA with at least 3 states. We have shown above that B = 10 achieves this minimal number of states.

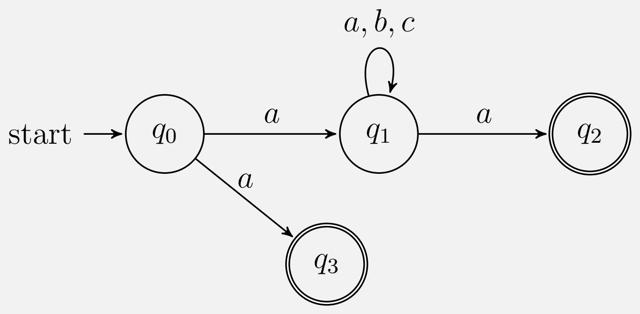
3. **Nondeterminism** (15 points): For this question, the alphabet is $\{a, b, c\}$.

(a) (*Graded for completeness*) Design a NFA that recognizes the language

$$L_1 = \{w \in \{a, b, c\}^* \mid w \text{ starts with } a \textbf{ and } \text{ends with } a\}$$

A complete solution will include the state diagram of your NFA and a brief justification of your construction that explains the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Solution: Here is the NFA N_1 that recognizes L_1:



- q_0 is the start state, it means we have not read any character yet.
- q_1 means the string starts with a. This state serves to process the characters following the starting a. Upon reading b or c, we can only stay in q_1. Upon reading a, we can either stay in q_1 or transit to q_2.
- q_2 means the string starts with a and ends with a and has length at least 2.
- q_3 serves to accept string a, which starts with a and ends with a.

Consider arbitrary string $s \in L_1$. The first case is when $s = a$. There exists a computation q_0, q_3 that processes the whole string and ends in an accept state, and thus s is accepted. The second case is when s has length at least 2. In this case, there also exists a computation that processes the whole string and ends in an accept state q_2: start from q_0, upon reading the first a transit to q_1, stay in q_1 when reading the following characters before the last character, and transit to q_2 upon reading the last character which is a. Therefore, all strings in L_1 are accepted by the NFA.

Consider arbitrary string $s \notin L_1$. If s is empty, the computation will end in q_0 which is a non-accept state. If s is non-empty and does not start with a, then all possible computation on s will stuck at q_0 at the beginning since there's no outgoing arrow labeled b or c from q_0. If s starts with a but does not end with a, the computation will either (1): upon reading the first a, transit to q_3, and then stuck, or (2): upon reading the first a, transit to q_1, and either (2.1): stay in q_1 to process the remaining characters thus end in q_1, or (2.2): transit to q_2 at some point if an a is seen and stuck afterwards. Therefore, there exist no computation that can process the whole string and end in an accept state. Thus all strings not in L_1 are rejected by the NFA.

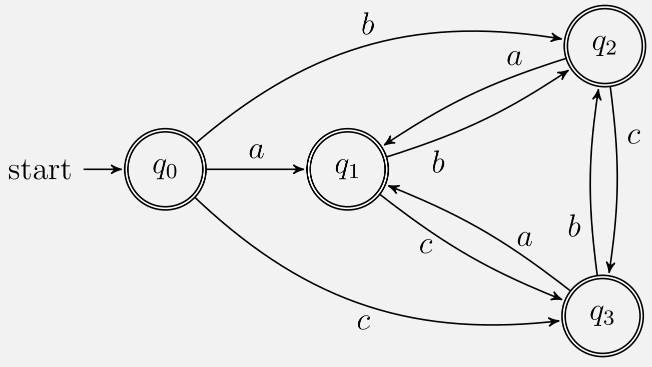
(b) (*Graded for correctness*) Design a NFA that recognizes the language

$$L_2 = \{w \in \{a,b,c\}^* \mid w \text{ has no consecutive repeated characters}\}$$

For example, the empty string, a, bac, and $abca$ are each elements of this language but aa and abb and $abbc$ are not elements of this language.

A complete solution will include the state diagram of your NFA and a brief justification of your construction that explains the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the specified language.

Solution: Here is the NFA N_2 that recognizes L_2:



- q_0 is the start state, meaning we have not read any character yet.
- q_1 means the last character we've read is a. Since from q_1, there is no outgoing arrow labeled a, it makes sure that the computation will stuck when encountering consecutive repeated a's.
- q_2 means the last character we've read is b. Since from q_2, there is no outgoing

arrow labeled b, it makes sure that the computation will stuck when encountering consecutive repeated b's.

- q_3 means the last character we've read is c. Since from q_3, there is no outgoing arrow labeled c, it makes sure that the computation will stuck when encountering consecutive repeated c's.

Consider arbitrary string $s \in L_2$. If $s = \varepsilon$, we accept s since q_0 is an accept state. If s is non-empty, since it has no consecutive repeated characters, computation will not get stuck, i.e. there exists a computation on s that can process the whole string and ends in q_1, q_2, or q_3, which are all accept states. Therefore, all strings in L_2 are accepted.

Consider arbitrary string $s \notin L_2$, i.e. s has consecutive repeated characters, i.e. it has aa, bb, or cc as a substring. If aa is the first consecutive repeated part in s, then the computation will stuck at q_1 when trying to process the repeated a. Similarly, if bb is the first consecutive repeated part in s, then the computation will stuck at q_2 when trying to process the repeated b, and if cc is the first consecutive repeated part in s, then the computation will stuck at q_3 when trying to process the repeated c. Therefore, there exist no computation that can fully process the string s, thus all strings not in L_2 are rejected.

(c) (*Graded for completeness*) Consider the language

$$L_1 \cup L_2 = \{w \in \{a,b,c\}^* \mid w \text{ starts with } a \text{ and ends with } a \text{ } \textbf{or} \text{ has no consecutive repeated characters}\}$$

Give at least two representations of this language among the following:

- A regular expression that describes $L_1 \cup L_2$
- A DFA that recognizes $L_1 \cup L_2$
- A NFA that recognizes $L_1 \cup L_2$

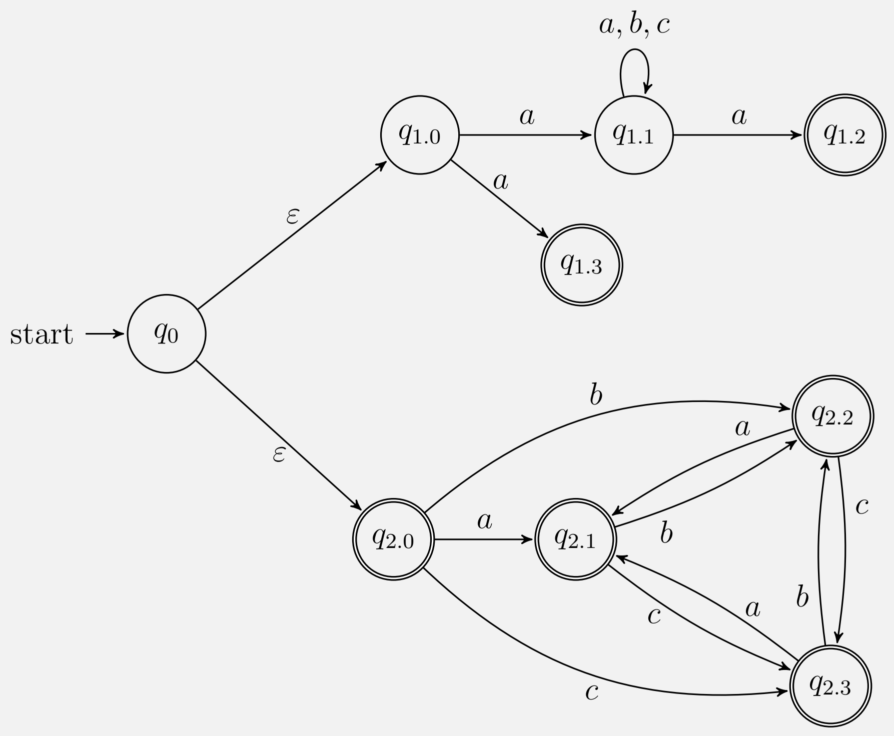
You can design your automata directly or use the constructions from class and chapter 1 in the book to build these automata from automata for the simpler languages.

A complete solution will include at least two of the representations as well as a brief justification of each construction.

Solution:

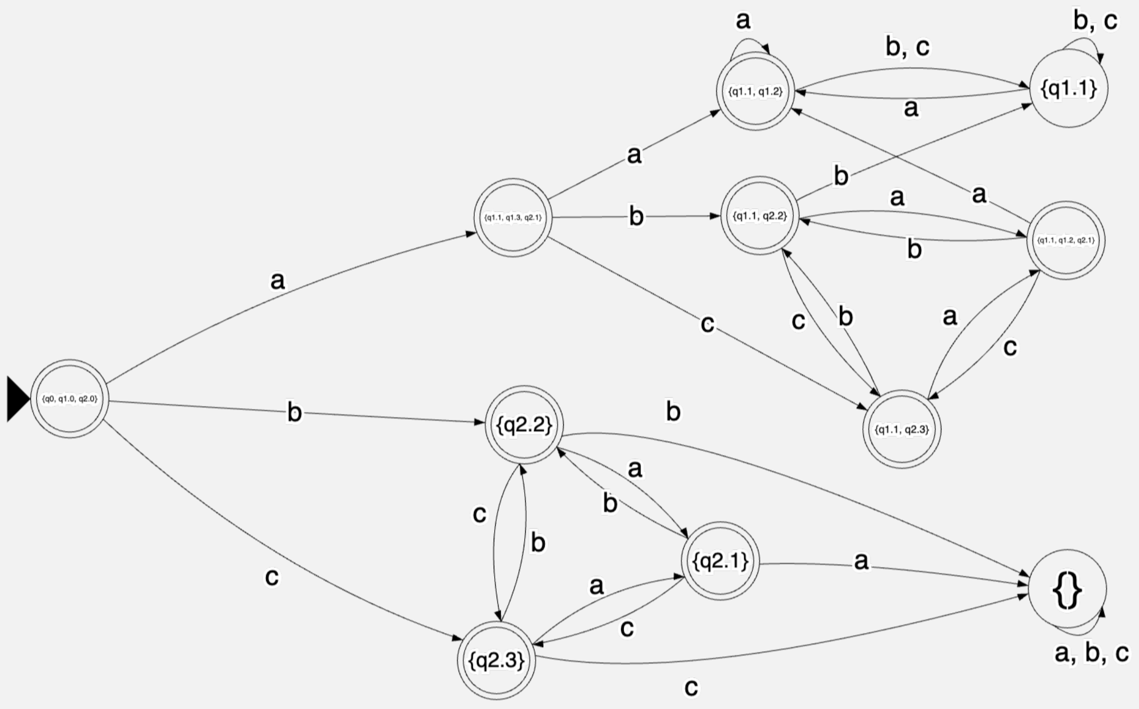
A NFA that recognizes $L_1 \cup L_2$:

Building NFA that recognizes $L_1 \cup L_2$ is quite straightforward. With N_1 (the NFA in part 3(a) that recognizes L_1) and N_2 (the NFA in part 3(b) that recognizes L_2), let's rename the states in N_1 from q_m to $q_{1.m}$ and rename the states in N_2 from q_n to $q_{2.n}$, and then we can apply the construction in week 2 notes page 11 (that we use to prove that the class of languages recognized by NFAs is closed under union) to construct the following NFA N_3 that recognizes $L_1 \cup L_2$:



A DFA that recognizes $L_1 \cup L_2$:

We can apply the "macro-states" construction in week 3 notes page 9 to convert the above NFA N_3 into a DFA below that recognizes the same language:

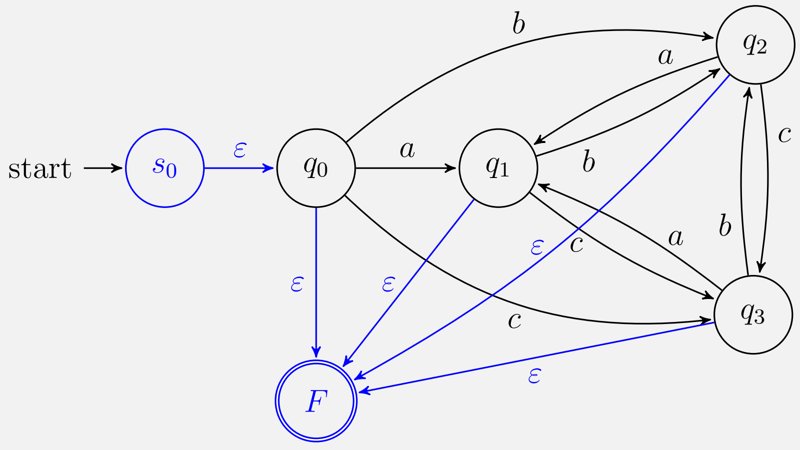


A regular expression that describes $L_1 \cup L_2$:

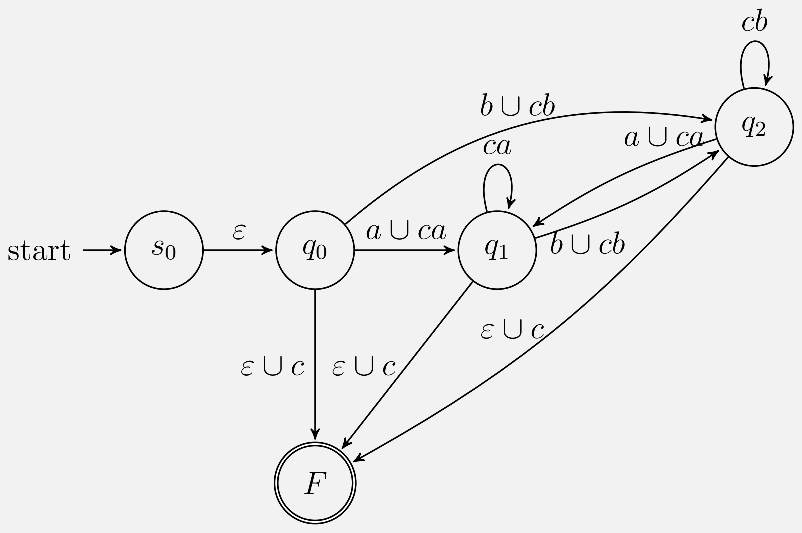
We can try to first get a regular expression R_1 that describes L_1, and then get a regular expression R_2 that describes L_2. The final result will be $R_1 \cup R_2$ by definition of union in regular expression. Let's start with R_1, which can be:

$$R_1 = a \ \cup \ a(a \cup b \cup c)^* a$$

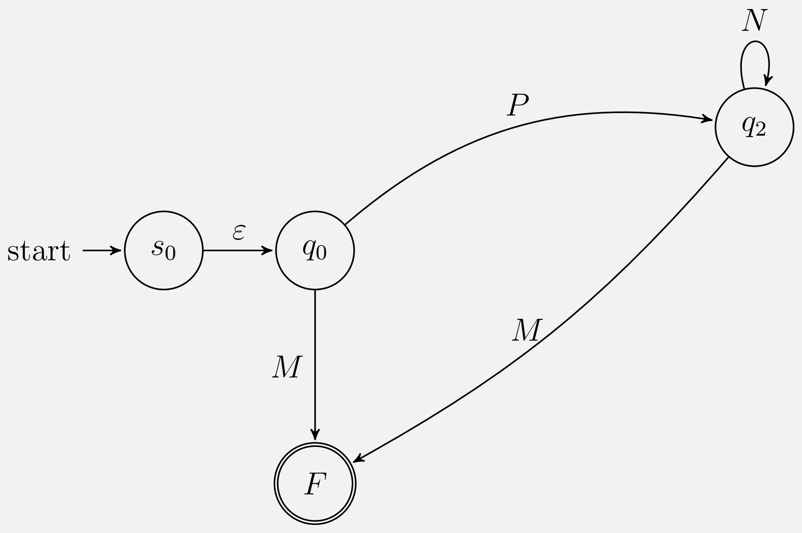
(Notice we take care of the edge case where the first and last character are actually just the only character in the string separately.)
Now let's work on R_2. We'll start with creating a GNFA from the NFA N_2 in part 3(b), then generating a regular expression from there. The initial GNFA will be:



After removing q_3:



After removing q_1:



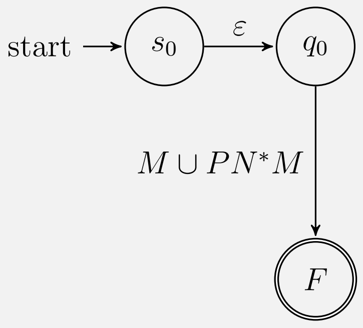
Where:

$$M = \varepsilon \cup c \cup (a \cup ca)(ca)^*(\varepsilon \cup c)$$

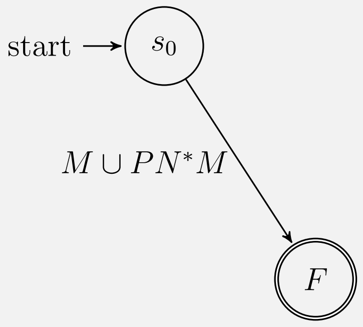
$$P = b \cup cb \cup (a \cup ca)(ca)^*(b \cup cb)$$

$$N = cb \cup (a \cup ca)(ca)^*(b \cup cb)$$

After removing q_2:



After removing q_0:



The regular expression that describes L_2, denoted as R_2, will be:

$$R_2 = M \cup PN^*M$$

i.e. $R_2 = (\varepsilon \cup c \cup (a \cup ca)(ca)^*(\varepsilon \cup c)) \cup (b \cup cb \cup (a \cup ca)(ca)^*(b \cup cb))(cb \cup (a \cup ca)(ca)^*(b \cup cb))^*(\varepsilon \cup c \cup (a \cup ca)(ca)^*(\varepsilon \cup c))$

The final regular expression that describes $L_1 \cup L_2$ will be

$$R_1 \cup R_2$$

i.e. $a \cup a(a \cup b \cup c)^*a \cup (\varepsilon \cup c \cup (a \cup ca)(ca)^*(\varepsilon \cup c)) \cup (b \cup cb \cup (a \cup ca)(ca)^*(b \cup cb))(cb \cup (a \cup ca)(ca)^*(b \cup cb))^*(\varepsilon \cup c \cup (a \cup ca)(ca)^*(\varepsilon \cup c))$

Wow! Seems like either an NFA or DFA would be a much cleaner way to recognize the language. Although regular expressions, NFA, and DFA are equally expressive, we can see that sometimes one model can be easier to build than others.

(d) (*Graded for completeness*) Consider the language

$$L_1 \cap L_2 = \{w \in \{a,b,c\}^* \mid w \text{ starts with } a \text{ and ends with } a \text{ \textbf{and} has no consecutive repeated characters}\}$$

Give at least two representations of this language among the following:

- A regular expression that describes $L_1 \cap L_2$

- A DFA that recognizes $L_1 \cap L_2$
- A NFA that recognizes $L_1 \cap L_2$

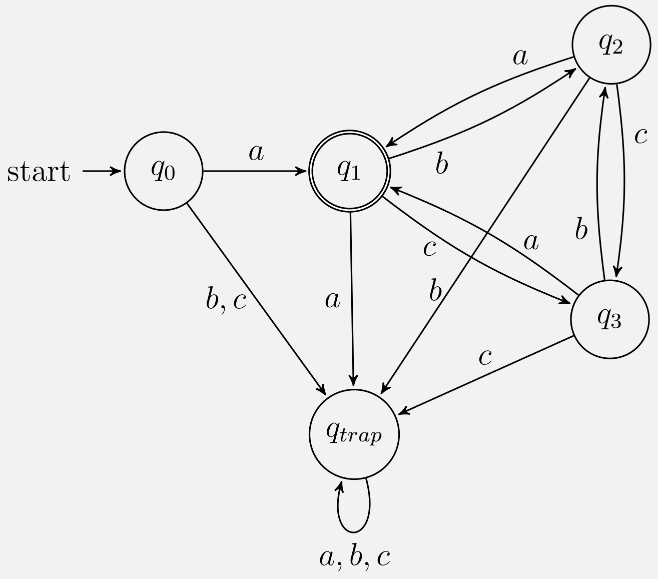
You can design your automata directly or use the constructions from class and chapter 1 in the book to build these automata from automata for the simpler languages.

A complete solution will include at least two of the representations as well as a brief justification of each construction.

Solution:

A DFA that recognizes $L_1 \cap L_2$:

One brute-force way to build this DFA is to first apply "macro-states" construction (week 3 notes page 9) to convert NFA N_1 into DFA D_1 and convert NFA N_2 into DFA D_2, and then apply the construction in Sipser Theorem 1.25 to build a new DFA that recognizes $L_1 \cap L_2$ using D_1 and D_2. But instead of taking these steps, we can notice some neat properties about the language and build the following DFA directly:

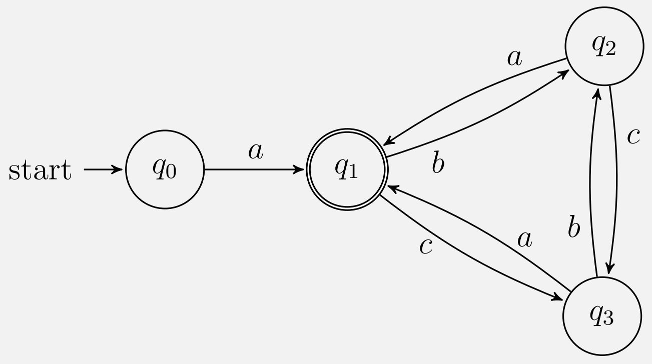


- q_0 is the start state, meaning we have not read any character yet.
- q_1 means the last character we've read is a, and there's no consecutive repeated characters yet. It is the only accept state because we want to make sure that accepted strings are all ended with a.
- q_2 means the last character we've read is b, and there's no consecutive repeated characters yet.
- q_3 means the last character we've read is c, and there's no consecutive repeated characters yet.

- q_{trap} means the string does not start with a (note that at q_0, upon reading b or c, we will transit to q_{trap}), or the string starts with a but has consecutive repeated characters (note that $\delta((q_1, a)) = q_{trap}$, $\delta((q_2, b)) = q_{trap}$, and $\delta((q_3, c)) = q_{trap}$).

A NFA that recognizes $L_1 \cap L_2$:

Note that the above DFA state diagram can also be a valid state diagram for an NFA. However, since the computations can get stuck in an NFA (we can have "missing arrows"), we can simplify the diagram a little bit by deleting the q_{trap}, to create our NFA N_4:



A regular expression that recognizes $L_1 \cap L_2$:

Similar to part 3(c), we'll start with creating a GNFA from the NFA N_4 that recognizes $L_1 \cap L_2$, then generating a regular expression from there (leave as an exercise). The final result is:

$$A \cup BC^*D$$

Where

$$A = \varepsilon \cup a(ca)^*(\varepsilon \cup c)$$

$$B = a(ca)^*(b \cup cb)$$

$$C = cb \cup (a \cup ca)(ca)^*(b \cup cb)$$

$$D = \varepsilon \cup c \cup (a \cup ca)(ca)^*(\varepsilon \cup c)$$

4. **General constructions** (13 points): In this question, you'll practice working with formal general constructions for automata and translating between state diagrams and formal definitions.

Recall the definitions of operations we've talked about that produce new languages from old: for each language L over an alphabet Σ, we have the associated sets of strings (also over Σ)

$$L^* = \{w_1 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in L\}$$

and

$$SUBSTRING(L) = \{w \in \Sigma^* \mid \text{there exist } x, y \in \Sigma^* \text{ such that } xwy \in L\}$$

and

$$EXTEND(L) = \{w \in \Sigma^* \mid w = uv \text{ for some strings } u \in L \text{ and } v \in \Sigma^*\}$$

Also, recall the set operations union and intersection: for any sets X and Y

$$X \cup Y = \{w \mid w \in X \text{ or } w \in Y\}$$

$$X \cap Y = \{w \mid w \in X \text{ and } w \in Y\}$$

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be DFA.
For simplicity, assume that $Q_1 \cap Q_2 = \emptyset$ and that $q_0 \notin Q_1 \cup Q_2$.
Consider the following definitions of new automata parameterized by these DFA:

- The NFA $N_\alpha = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma, \delta_\alpha, q_0, F_1 \cup F_2)$ with the transition function given by

$$\delta_\alpha(\ (q,x)\ ) = \begin{cases} \{q_1, q_2\} & \text{if } q = q_0,\ x = \varepsilon \\ \emptyset & \text{if } q = q_0,\ x \in \Sigma \\ \{\delta_1(\ (q,x)\ )\} & \text{if } q \in Q_1,\ x \in \Sigma \\ \{\delta_2(\ (q,x)\ )\} & \text{if } q \in Q_2,\ x \in \Sigma \\ \emptyset & \text{if } q \in Q_1 \cup Q_2,\ x = \varepsilon \end{cases}$$

Let's informally explain the transition function in plain English:

1. $\delta_\alpha(\ (q,x)\ ) = \{q_1, q_2\} \quad \text{if } q = q_0,\ x = \varepsilon$
   "add spontaneous moves from q_0 to q_1 (the start state of M_1) and q_2 (the start state of M_2) respectively"
2. $\delta_\alpha(\ (q,x)\ ) = \emptyset \quad \text{if } q = q_0,\ x \in \Sigma$
   "there should be no arrow from q_0 labeled by symbols in the alphabet (i.e. there are only spontaneous moves labeled by ε, specified in case 1)"
3. $\delta_\alpha(\ (q,x)\ ) = \{\delta_1(\ (q,x)\ )\} \quad \text{if } q \in Q_1,\ x \in \Sigma$
   "the original transitions in M_1 are all preserved"
4. $\delta_\alpha(\ (q,x)\ ) = \{\delta_2(\ (q,x)\ )\} \quad \text{if } q \in Q_2,\ x \in \Sigma$
   "the original transitions in M_2 are all preserved"
5. $\delta_\alpha(\ (q,x)\ ) = \emptyset \quad \text{if } q \in Q_1 \cup Q_2,\ x = \varepsilon$
   "for the original states in M_1 and M_2, no spontaneous moves (arrows labeled ε) are added"

- The NFA $N_\beta = (Q_1 \times Q_2, \Sigma, \delta_\beta, (q_1, q_2), F_1 \times F_2)$ with the transition function given by

$$\delta_\beta(\ (\ (r,s)\ , x\ )\ ) = \{(\ \delta_1(\ (r,x)\ ), \delta_2(\ (s,x)\ )\ )\}$$

  and

$$\delta_\beta(\ (\ (r,s)\ , \varepsilon\ )\ ) = \emptyset$$

  for $r \in Q_1, s \in Q_2, x \in \Sigma$.

- The NFA $N_\gamma = (Q_1 \cup \{q_0\}, \Sigma, \delta_\gamma, q_0, \{q \in Q_1 \mid \exists w \in \Sigma^*(\delta_1^*((q,w)) \in F_1)\})$, and

$$\delta_\gamma((q,a)) = \begin{cases} \{\delta_1((q,a))\} & \text{if } q \in Q_1,\ a \in \Sigma \\ \{q' \in Q_1 \mid \exists w \in \Sigma^*(\delta_1^*((q_1,w)) = q')\} & \text{if } q = q_0,\ a = \varepsilon \\ \emptyset & \text{if } q = q_0,\ a \in \Sigma \\ \emptyset & \text{if } q \in Q_1,\ a = \varepsilon \end{cases}$$

Let's informally explain the set of accept states of N_γ: "the new set of accept states are the original DFA's states which are reachable to the original set of accept states"

Let's informally explain the transition function in plain English:

1. $\delta_\gamma((q,a)) = \{\delta_1((q,a))\}$ if $q \in Q_1$, $a \in \Sigma$
   "the original transitions in M_1 are all preserved"

2. $\delta_\gamma((q,a)) = \{q' \in Q_1 \mid \exists w \in \Sigma^*(\delta_1^*((q_1,w)) = q')\}$ if $q = q_0$, $a = \varepsilon$
   "add spontaneous moves from q_0 to all states that are reachable from the original DFA's start state"

3. $\delta_\gamma((q,a)) = \emptyset$ if $q = q_0$, $a \in \Sigma$
   "there should be no arrow from q_0 labeled by symbols in the alphabet (i.e. there are only spontaneous moves labeled by ε, specified in case 2)"

4. $\delta_\gamma((q,a)) = \emptyset$ if $q \in Q_1$, $a = \varepsilon$
   "for the original states in the original DFA M_1, no additional spontaneous moves (arrows labeled ε) are added"

(a) (*Graded for correctness*) Illustrate the construction of N_α by defining a specific pair of example DFAs M_1 and M_2 and applying the construction above to create the new NFA N_α. Your example DFA should

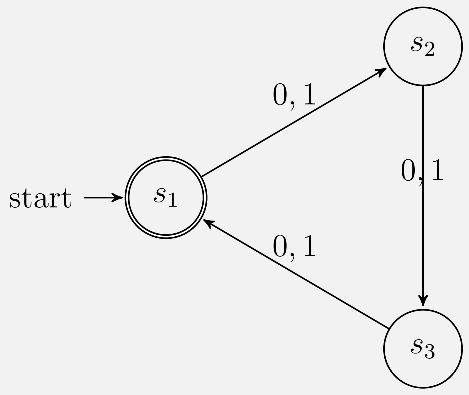
- Have the same input alphabet,
- each have exactly three states (all reachable from the respective start state),
- Accept at least one string and reject at least one string,
- Recognize different languages from one another, and
- Not have any states labelled q_0, and

- Not share any state labels.

Apply the construction above to create the new NFA. A complete submission will include the state diagrams of your example DFA M_1 and M_2 and the state diagram of the NFA N_α resulting from this construction and a precise and clear description of $L(M_1)$ and $L(M_2)$ and $L(N_\alpha)$, justified by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the language.

Solution:

Consider the alphabet as $\Sigma = \{0, 1\}$. The state diagram for DFA M_1 is the following:

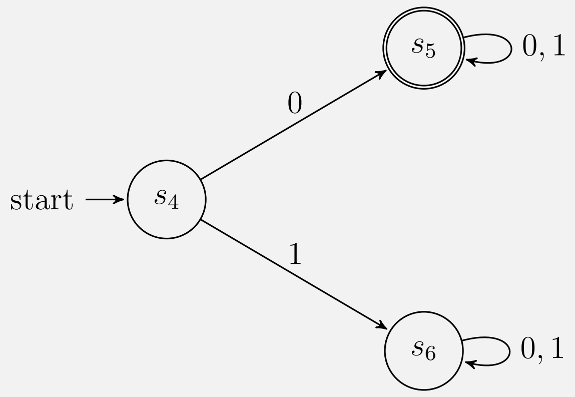


We claim that $L(M_1) = \{w \in \Sigma^* \mid |w| \text{ is a multiple of } 3\}$. The role of each state is:

- s_1 is the start state, and it also represents the number of characters we've read so far is $3n$ (where n is an integer and $n \geq 0$), i.e. a multiple of 3. It is an accept state.
- s_2 represents the number of characters we've read so far is $3n + 1$ (where n is an integer and $n \geq 0$). It is not an accept state.
- s_3 represents the number of characters we've read so far is $3n + 2$ (where n is an integer and $n \geq 0$). It is also not an accept state.

Consider any string s where $|s|$ is a multiple of 3. The computation of M_1 on s ends in s_1, which is an accept state, meaning that s is accepted by M_1. Therefore, M_1 accepts each string with length of multiple of 3. On the other hand, consider any string p where $|p|$ is not a multiple of 3, i.e. $|p| = 3n + 1$ or $|p| = 3n + 2$ for some non-negative integer n. By the above state roles, the computation of M_1 on p ends in s_2 or s_3, which are not accept states, meaning that p is rejected by M_1. Thus we proved that $L(M_1) = \{w \in \Sigma^* \mid |w| \text{ is a multiple of } 3\}$.

The state diagram for DFA M_2 is the following:

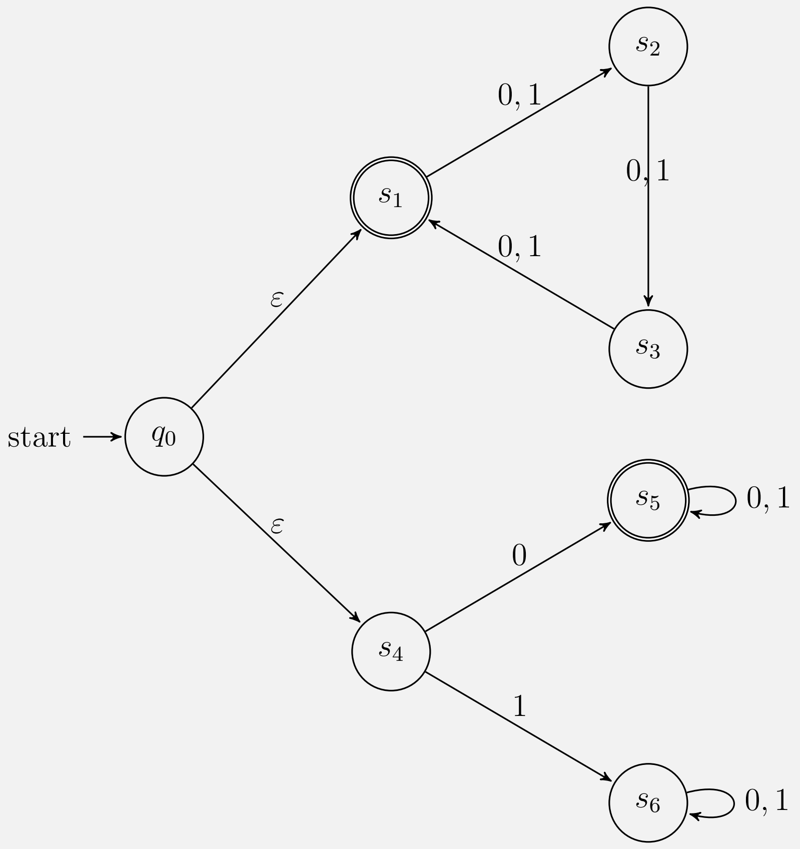


We claim that $L(M_2) = \{w \in \Sigma^* \mid w \text{ starts with } 0\}$. The role of each state is:

- s_4 is the start state. It is not an accept state because we want to reject the empty string.
- s_5 means that the input string starts with 0. We enter to s_5 if the first character is 0, and stay in the state until we finish processing the string, at which point we accept.
- s_6 means that the input string starts with 1. We enter to s_6 if the first character is 1, and stay in the state until we finish processing the string, at which point we reject.

Based on the above description of state roles, any string starting with a 0 will be accepted by M_2, and any string not starting with a 0 (i.e. empty string or a string starting with a 1) will be rejected by M_2. Therefore, $L(M_2) = \{w \in \Sigma^* \mid w \text{ starts with } 0\}$.

Applying the construction, the state diagram for NFA N_α is the following:



We claim that $L(N_\alpha) = \{w \in \Sigma^* \mid |w| \text{ is a multiple of 3 } \textbf{or } w \text{ starts with } 0\}$. q_0 is the start state. From the start state q_0, we can spontaneously move to s_1 (from where we can start simulating M_1), or to s_4 (from where we can start simulating M_2). States s_1 through s_6 have the same meaning as described previously.

For any string s with length of multiple of 3, the computation of M_1 on s has the sequence of states $u_1, u_2, ..., u_k$ ($k \geq 1$) where $u_1 = s_1$ and $u_k = s_1$. Thus, there exists a computation of N_α on s which is $q_0, u_1, u_2, ..., u_k$ that processes the whole string and ends in s_1, at which point we accept. And for any string that starts with a 0, the computation of M_2 on p has the sequence of states $u_1, u_2, ..., u_k$ ($k > 1$) where $u_1 = s_4$ and $u_k = s_5$. Thus, there exists a computation of N_α on p which is $q_0, u_1, u_2, ..., u_k$ that processes the whole string and ends in s_5, at which point we accept. On the other hand, for any string q whose length is not multiple of 3 **and** does not start with 0, the computation of M_1 should be $u_1, u_2, ..., u_k$ ($k > 1$) where $u_1 = s_1$ and u_k is s_2 or s_3. The computation of M_2 should be $l_1, l_2, ..., l_j$ ($j \geq 1$) where $l_1 = s_4$ and l_j is s_4 or s_6. Therefore, the computations of N_α on q that process the whole string have two possibilities: $q_0, u_1, u_2, ..., u_k$, or $q_0, l_1, l_2, ..., l_j$. Neither of them ends in an accept state, which means that we reject q. Therefore, $L(N_\alpha) = \{w \in \Sigma^* \mid |w| \text{ is a multiple of 3 } \textbf{or } w \text{ starts with } 0\}$.

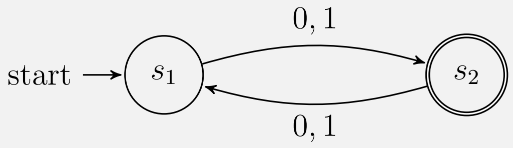
(b) (*Graded for correctness*) Illustrate the construction of N_β by defining a specific pair of example DFAs M_1 and M_2 and applying the construction above to create the new NFA N_β. Your example DFA should

- Have the same input alphabet,
- each have exactly two states (all reachable from the respective start state),
- Accept at least one string and reject at least one string,
- Recognize different languages from one another, and
- Not have any states labelled q_0, and
- Not share any state labels.

Apply the construction above to create the new NFA. A complete submission will include the state diagrams of your example DFA M_1 and M_2 and the state diagram of the NFA N_β resulting from this construction and a precise and clear description of $L(M_1)$ and $L(M_2)$ and $L(N_\beta)$, justified by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the language.

Solution:

Consider the alphabet as $\Sigma = \{0, 1\}$. The state diagram for DFA M_1 is the following:



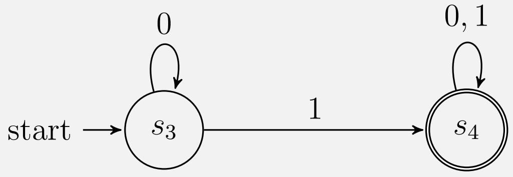
We claim that $L(M_1) = \{w \in \Sigma^* \mid w \text{ has odd length}\}$. The role of each state is:

- s_1 is the start state, and it also represents the number of characters we've read so far is even, i.e. $2n$ (where n is an integer and $n \geq 0$). It is not an accept state.

- s_2 represents the number of characters we've read so far is odd, i.e. $2n + 1$ (where n is an integer and $n \geq 0$). It is an accept state.

The computation of M_1 on each string with odd length will end in s_2, meaning we accept all odd length strings. On the other hand, the computation of M_1 on each string with even length will end in s_1, meaning we reject all even length strings. Thus we proved that $L(M_1) = \{w \in \Sigma^* \mid w \text{ has odd length}\}$.

The state diagram for DFA M_2 is the following:

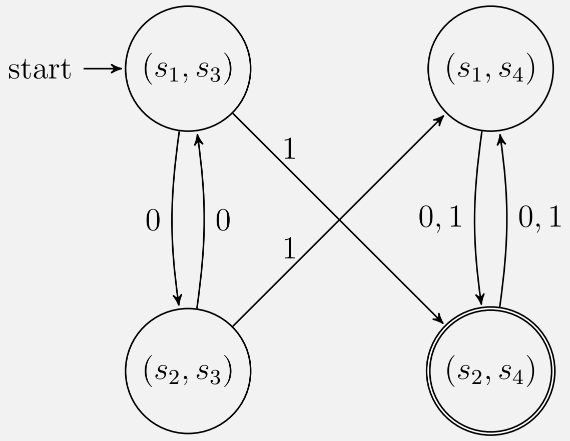


We claim that $L(M_2) = \{w \in \Sigma^* \mid w \text{ contains at least one } 1\}$. The role of each state is:

- s_3 is the start state. It means that we have not seen a 1 so far. It is not an accept state.
- s_4 means that the input string contains a 1. Upon reading a 1, we transit to s_4 and stays here until we finish process the whole string, at which point we accept.

The computation of M_2 on each string containing 1 will end in s_4, at which point we accept. On the other hand, the computation of M_2 on each string with no 1 will end in s_3, meaning we reject them. Thus we proved that $L(M_2) = \{w \in \Sigma^* \mid w \text{ contains at least one } 1\}$.

Applying the construction, the state diagram for NFA N_β is the following:



We claim that $L(N_\beta) = \{w \in \Sigma^* \mid w \text{ has odd length } \textbf{and } w \text{ contains at least one } 1\}$. The role of each state is:

- (s_1, s_3) is the start state, which means the number of characters we've read is even and we have not seen a 1 yet
- (s_1, s_4) means the number of characters we've read is even and we have seen a 1
- (s_2, s_3) means the number of characters we've read is odd and we have not seen a 1 yet
- (s_2, s_4) means the number of characters we've read is odd and we have seen a 1

Only (s_2, s_4) is an accept state, which means that all and only the strings whose length is odd **and** contain 1 are accepted, while all other strings are rejected. Thus, $L(N_\beta) = \{w \in \Sigma^* \mid w \text{ has odd length } \textbf{and } w \text{ contains at least one } 1\}$.

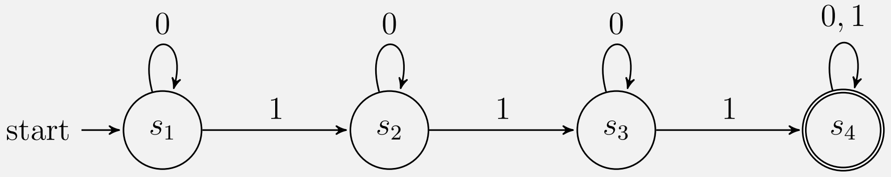
(c) (*Graded for correctness*) Illustrate the construction of N_γ by defining a specific example DFA M_1 and applying the construction above to create the new NFA N_γ. Your example DFA should

- Have exactly four states (all reachable from the respective start state),
- Accept at least one string and reject at least one string,
- Not have any states labelled q_0.

Apply the construction above to create the new NFA. A complete submission will include the state diagram of your example DFA M_1 and the state diagram of the NFA N_γ resulting from this construction and a precise and clear description of $L(M_1)$ and $L(N_\gamma)$, justified by explaining the role each state plays in the machine, as well as a brief justification about how the strings accepted and rejected by the machine connect to the language.

Solution:

Consider the alphabet as $\Sigma = \{0, 1\}$. The state diagram for DFA M_1 is the following:

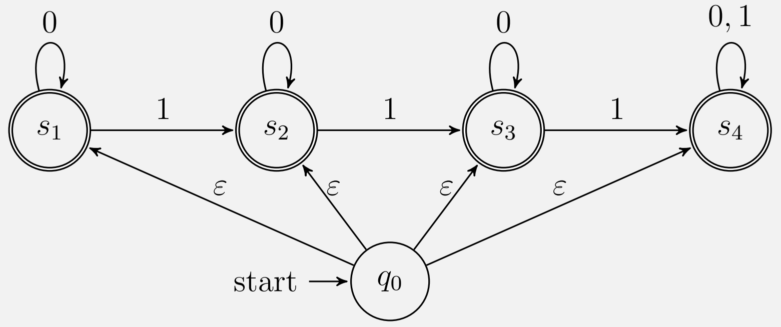


We claim that $L(M_1) = \{w \in \Sigma^* \mid w \text{ contains at least three 1's}\}$. The role of each state is:

- s_1 is the start state. It means that we have not seen a 1 so far.
- s_2 means that we have seen exactly one 1's.
- s_3 means that we have seen exactly two 1's.
- s_4 means that we have seen at least three 1's.

The computation of M_1 on any string with at least three 1's will end in s_4, at which point we accept. Computations on all other strings (i.e. strings with less than three 1's) will end in s_1, s_2, or s_3, meaning we reject them. Thus we proved that $L(M_1) = \{w \in \Sigma^* \mid w \text{ contains at least three 1's}\}$.

Applying the construction, the state diagram for NFA N_γ is the following:



We claim that $L(N_\gamma) = \Sigma^*$. The meaning of s_1 through s_4 remains the same as above, but now we make all of them accept states. We also add a q_0 as our new start state, from where we can spontaneously jump to any state in s_1 through s_4. Note that we can directly go to s_4 from the beginning, and for any input string in Σ^*, s_4 is able to process the whole string. Thus, for each string, there exists a computation that process the whole string and ends in an accept state (q_4). We proved that $L(N_\gamma) = \Sigma^*$.

(d) (*Graded for completeness*) If possible, associate each construction above with one of the operations whose definitions we recalled at the start of the question. For example, is it the case that (for all choices of DFA M_1 and M_2) $L(N_\alpha) = L(M_1) \cup L(M_2)$? or $L(N_\alpha) = L(M_1) \cap L(M_2)$? etc.

A complete solution will consider each of the constructions $N_\alpha, N_\beta, N_\gamma$ in turn, and for each, either name the operation that's associated with the construction (and explain why) or explain why none of the operations mentioned is associated with the construction.

Solution:

- $L(N_\alpha) = L(M_1) \cup L(M_2)$
  The construction for N_α is similar to the construction in week 2 notes page 11 (that we use to prove languages recognized by NFAs are closed under union), with the difference that we are now taking two DFAs instead of two NFAs. Note that each string accepted by M_1 will also be accepted by N_α by first taking the spontaneous move from q_0 to q_1 and then simulate M_1 from there. Similarly, each string accepted by M_2 will also be accepted by N_α by first taking the spontaneous move from q_0 to q_2 and then simulate M_2 from there. On the other hand, for each string rejected by both M_1 and M_2, there are only two possible computations in N_α that can process the whole string: one is first taking the spontaneous move from q_0 to q_1 and then simulate M_1, the second is first taking the spontaneous move from q_0 to q_2 and then simulate M_2. Both of these computations will end in a non-accept state. So there is no computation that processes the whole string and ends in an accept state, thus strings not in $L(M_1) \cup L(M_2)$ are all rejected by N_α. Therefore, we proved that $L(N_\alpha) = L(M_1) \cup L(M_2)$.

- $L(N_\beta) = L(M_1) \cap L(M_2)$
  The construction for N_β is similar to the construction in week 3 notes page 4, with the difference that the new accept states are $F_1 \times F_2$, and the output is an NFA. For each string accepted by both M_1 and M_2, by definition of the transition function δ_β, there exist a computation of N_β on it that processes the whole string and would end in (r, s) where $r \in F_1$ and $s \in F_2$, i.e. $(r, s) \in F_1 \times F_2$ is an accept state, so N_β will accept it. On the other hand, for each string not accepted by both M_1 and M_2, there is only one computation of N_β on it that can process the whole string, but it will end in $(r', s') \notin F_1 \times F_2$, so N_β rejects the string. Therefore, we proved that $L(N_\beta) = L(M_1) \cap L(M_2)$.

- $L(N_\gamma) = SUBSTRING(L(M_1))$
  According to the definition of N_γ (and the informal description on page 18 in this solution document), the new set of accept states of N_γ are the original DFA M_1's states which are reachable from the original set of accept states. We also add spontaneous moves from q_0 to all states that are reachable from the original DFA's start state. From this definition, we can see that N_γ accepts all strings that are each a substring of some string in $L(M_1)$, and rejects all other strings. This is exactly $SUBSTRING(L(M_1))$ by the definition of $SUBSTRING$.