CS Theory (Fall '25) Assigned: Sept 9, 2025

Homework 1 Solutions

Instructors: *Toniann Pitassi and William Pires* Due: Sept 18, 2025 at 11am

Collaboration: Collaboration with other students in the class homework is allowed. However, you must write up solutions by yourself and understand everything that you hand in. You may also consult other reference materials, but you may not seek out answers from other sources or use AI tools. **You are required to list who you collaborated with on each problem, including any TAs or students you discussed the problems with in office hours. Also list any reference materials consulted other than the lectures and textbook for our class.** See the course webpage for more details.

Formatting: Write the solution to each part of each problem on a separate page. Be sure to correctly indicate which page each problem appears on in GradeScope. Please do not write your name on any page of the submission; we are using anonymous grading in GradeScope.

Grading: There are 80 total possible points, not including the Bonus problem which is optional and for extra credit. For all problems except for the Bonus problem, you have the option of answering "Don't know" on any parts of the question, and you will receive 20 percent of the total marks for those parts.

0 Exercises

Here are some recommended exercises. You should not turn in your solutions for these, and we will not grade them.

(a) From the Sipser textbook: Exercises 1.1, 1.2, 1.3, 1.5 (a,c)

(b) 1.6 (b,c,k)

(c) 1.7 (a,c)

(d) Are the following languages regular or not? (Just say "yes" or "no". Later we will show how to prove that a language is not regular.)

- $L = \{w \mid w = w^R\}$ (Notation: w^R is the string w written in reverse. For example if $w = 1101000$ then $w^R = 0001011$.)
- $L = \{0^n \mid n \text{ is even}\}$
- $L = \{0^n 1^m 0^n \mid m, n \geq 2\}$

(e) Draw an NFA using 5 states or less for the language L over $\Sigma = \{0, 1, 2\}$ where:

$$L = \{w \mid w \text{ contains the substring } 210\}.$$

(a) Solutions for 1.1, 1.2, and 1.5(a) are in the back of the book, starting on page 94.

- Solution for 1.3:

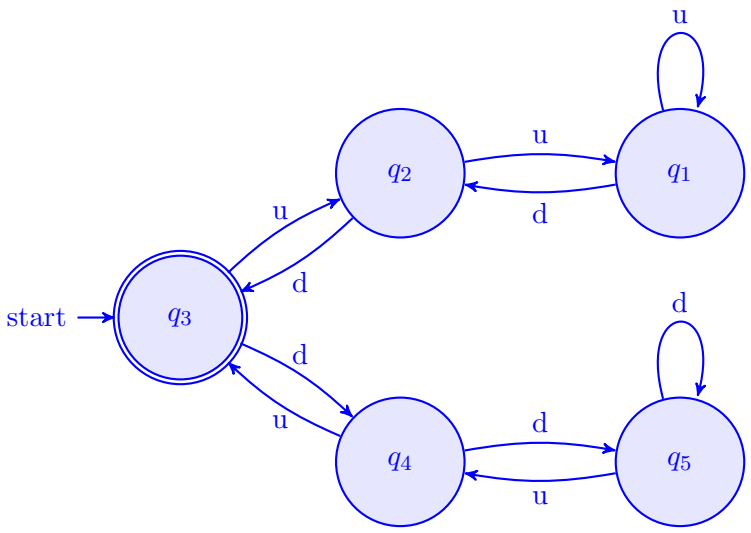


Figure 1: DFA for exercise 0(a) problem 1.3.

- Solution for 1.5c: The complement for the language $L = \{w \mid w \text{ contains neither the substrings } ab \text{ nor } ba\}$ is $\overline{L} = \{w \mid w \text{ contains the substrings } ab \text{ or } ba\}$. Figure 2 shows a DFA for the complement $\overline{L}$.

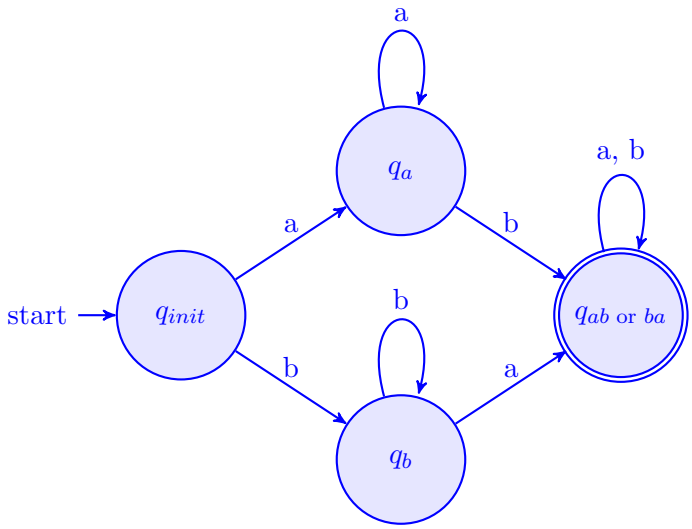


Figure 2: DFA for $\overline{L}$ for exercise 0(a) problem 1.5(c).

The intuition for constructing this DFA is that before seeing the substring ab or ba, the DFA will have read in only a's or only b's. Therefore we can use the states q_a and q_b to keep track of which character has been seen, then transition to $q_{ab \text{ or } ba}$ once both characters have been seen. The DFA for L can easily be constructed by taking the DFA for $\overline{L}$ and switching the accept and reject states, as seen below in 3.

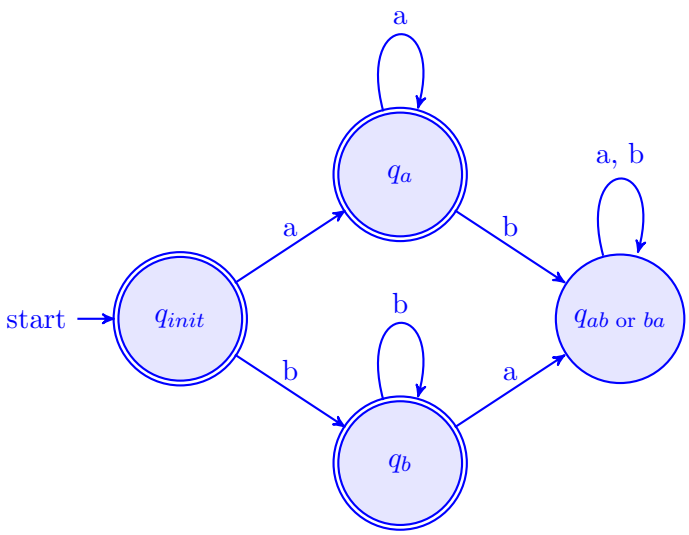


Figure 3: DFA for L for exercise 0(a) problem 1.5(c).

(b) • Solution for 1.6(b): A DFA recognizing $L = \{w \mid w \text{ contains at least three 1s}\}$ is the following.

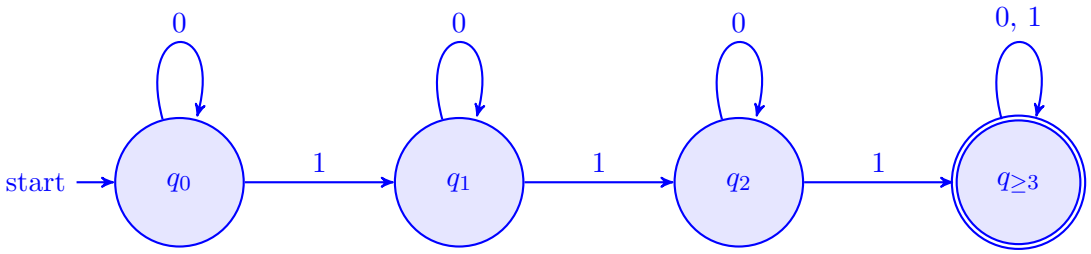


Figure 4: DFA for exercise 0(b) problem 1.6(b).

• Solution for 1.6(c): A DFA recognizing $L = \{w \mid w \text{ contains the substring } 0101\}$ is the following.

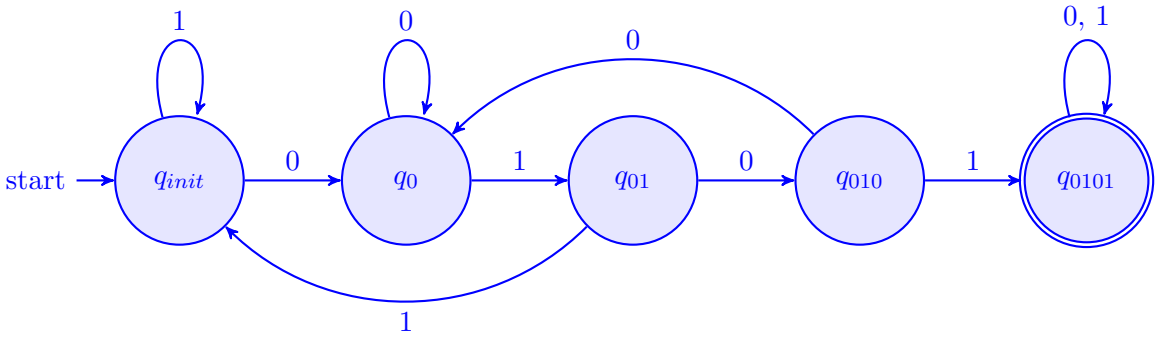


Figure 5: DFA for exercise 0(b) problem 1.6(c).

• Solution for 1.6(k): A DFA recognizing $L = \{\epsilon, 0\}$ is the following.

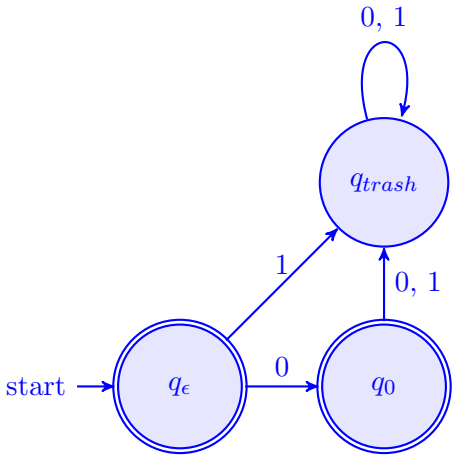


Figure 6: DFA for exercise 0(b) problem 1.6(k).

(c) See solution in the back of the Sipser textbook for 1.7(a).
Solution for 1.7(c): A NFA recognizing $L = \{w \mid w$ contains an even number of 0s, or contains exactly two 1s$\}$ with six states is the following.

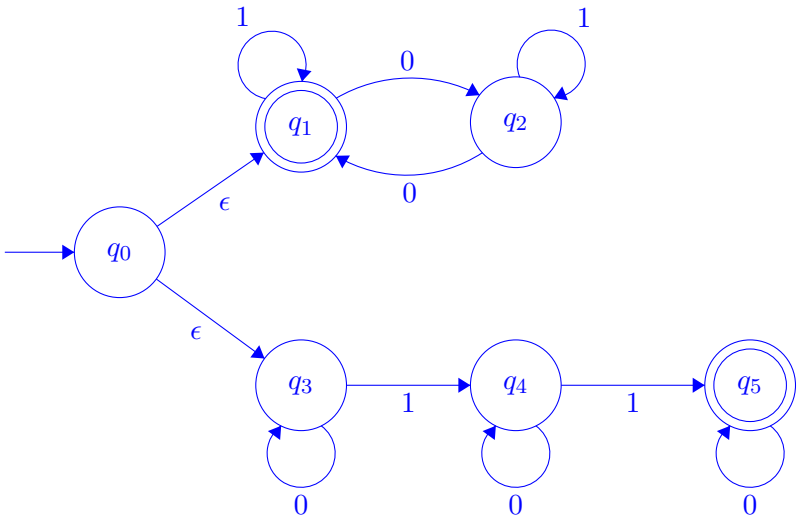


Figure 7: DFA for exercise 0(c) problem 1.7(c).

(d)
- $L = \{w \mid w = w^R\}$ is NOT regular.
- $L = \{0^n \mid n$ is even$\}$ is regular.
- $L = \{0^n 1^m 0^n \mid m, n \geq 2\}$ is NOT regular.

(e) See figure 8 below.

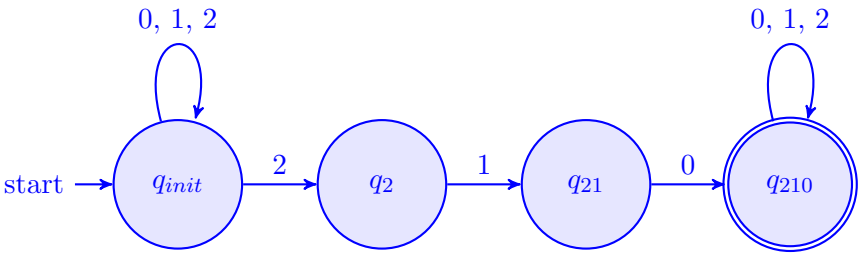


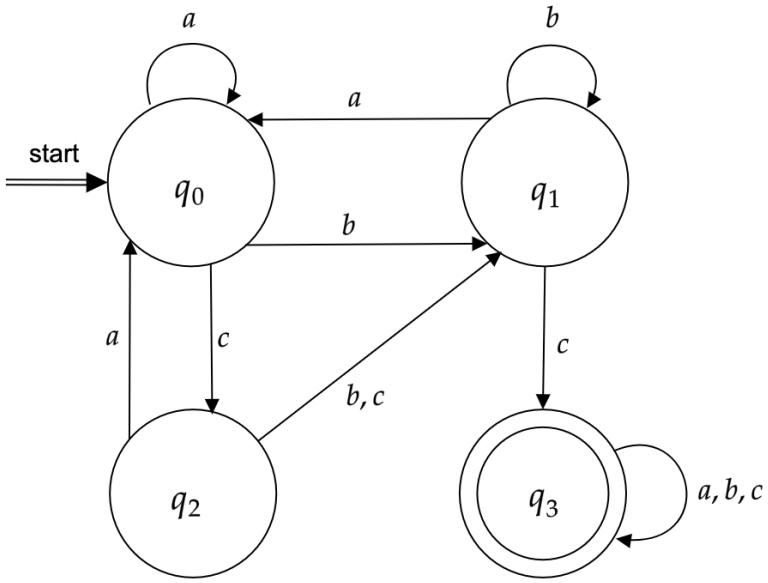
Figure 8: NFA for exercise 0(e).

If the input string contains 210, the NFA can stay in the start state until it is about to read in the 210 part, after which it can transition q_{210}, where it will stay and accept the string. If the input string does not contain 210, the NFA will never reach q_{210} and therefore will reject.

1 Problems

Problem 1: A DFA with four states (20 points)

Let D denote the following DFA over the alphabet $\{a, b, c\}$.



(a) Write out the formal definition of D (as a 5-tuple). Describe the transition function in a table.

(b) What is the language recognized by D? Give a short English description, then explain in a paragraph why your description is correct. Please give an explanation that is as short and simple as possible!

(a) States: $Q = \{q_0, q_1, q_2, q_3\}$.

Alphabet: $\Sigma = \{a, b, c\}$.

Transition function:

	a	b	c
q_0	q_0	q_1	q_2
q_1	q_0	q_1	q_3
q_2	q_0	q_1	q_1
q_3	q_3	q_3	q_3

Start state: q_0.

Accept states: $F = \{q_3\}$.

(b) The language is $L = \{w \mid w$ contains the substring bc or $ccc\}$.

Explanation: First we argue that any string containing bc or ccc as a substring is accepted. By inspection, if we haven't yet seen either the substring bc or ccc and the last symbol read is a then we are in q_0. Similarly if we haven't yet seen either substring bc or ccc and the last symbol read is b then we are in q_1. Therefore if $w = ubcv$, we are in q_1 after reading ub, and then move to q_3 on c and stay there. Thus any string containing bc as a substring is accepted. Now suppose bc doesn't occur as a substring, but ccc does. Then we can write $q = ucccv$ where either u is empty, or the last symbol of u is a (since if it was b, then bc would be a substring and if it was c then there is an earlier occurrence of ccc.) In either case, we are in q_0 after reading u, and then go to q_3 after reading ccc, and stay there. Thus we accept any string containing either the substring bc or ccc.

Conversely we want to show that D rejects all other strings. D accepts if and only if it is in state q_3 at some point (if we end up there, we will never leave). We can only get to q_3 by being in q_1 and seeing c. If we are in q_1, this means that we either came from q_0, q_1, or q_2 and just saw b, or we came from q_2 and just saw c. In the first case, this means we saw the substring bc at some point. In the second case, we can only get to q_2 by seeing c, so we know that there were actually two consecutive c's at the step where we end up in q_1.

Problem 2: Constructing DFAs and NFAs (30 points)

Give a DFA, either as a 5-tuple or by drawing the deterministic finite automata for the following languages. No justification is required.

(a) Give a DFA for the language L over $\Sigma = \{0, 1\}$ consisting of all strings *not* containing the substring 111.

(b) Draw an NFA for the language L over $\Sigma = \{0, 1, 2\}$ where

$$L = \{w \mid w \text{ has length divisible by 3 } \textit{or } w \text{ contains the symbol 2 exactly once }\}.$$

Your NFA should have 7 states or less.

(c) Give an NFA that accepts the set of all strings over $\Sigma = \{0, 1\}$ with at least three symbols and such that the third symbol from the right end is 1. Try to give an NFA with the minimal number of states.

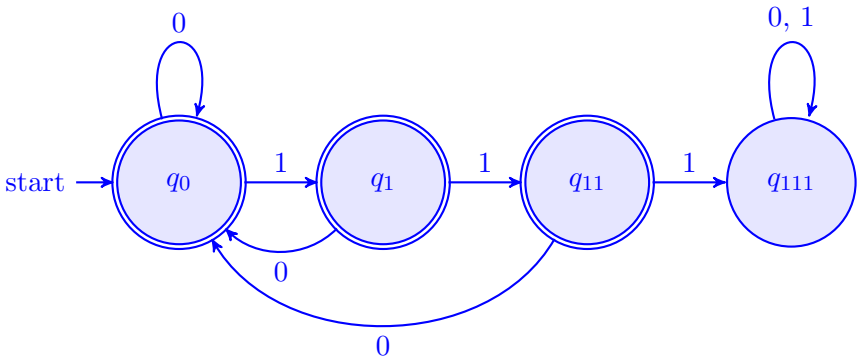


Figure 9: DFA for 2(a)

(a) The DFA in Figure 9 keeps track of how many consecutive 1's have been read. The state q_0 represents the state of not having read any consecutive 1's. The states q_1, q_{11}, and q_{111} represent having read one, two, and three consecutive 1's, respectively. All strings containing the substring 111 will reach q_{111} and never leave, while strings not containing 111 will never be in q_{111}.

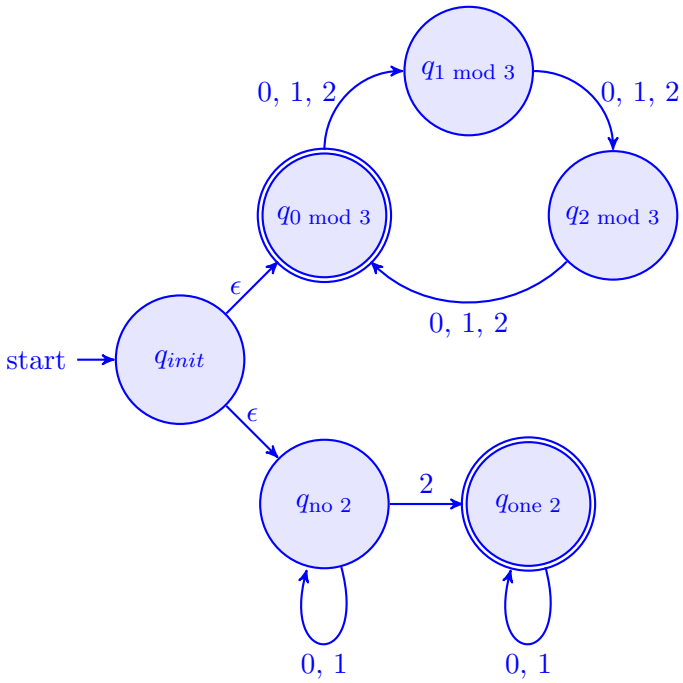


Figure 10: NFA for 2(b)

(b) The NFA in Figure 10 nondeterministically guesses that the input string satisfies one of the two conditions for L, and uses ϵ transitions to perform parallel branches of computation for the two conditions. The "top" part of the NFA determines if the string has length divisible by 3; its states keep track of the number of characters read so far modulo three. The "bottom" part determines if the string contains exactly one 2 character; its states keep track of whether no 2's or one 2 has been seen. If a second 2 is seen, the computation terminates and the string is rejected.

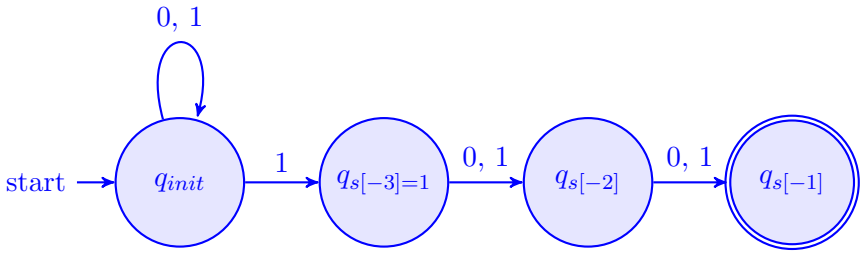
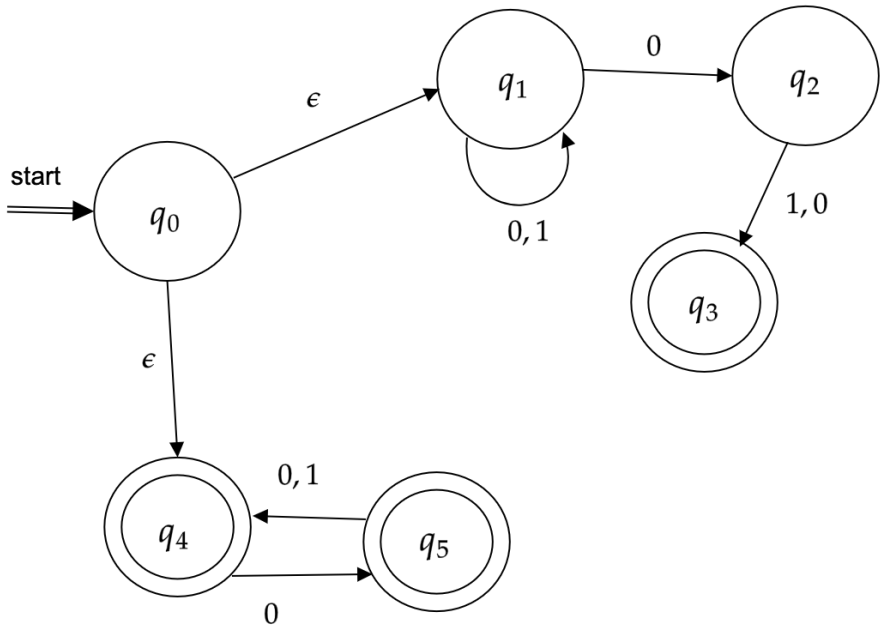


Figure 11: NFA for 2(c)

(c) The NFA in Figure 11 can stay in the start state until it reads in its third to last character. If this is 1, the NFA can transition to $q_{s[-3]=1}$. Afterwards, the NFA reads in the remaining two characters and arrives at the accept state $q_{s[-1]}$. This is the only case where the NFA can reach the accept state.

Problem 3: An NFA (20 points)

Let D denote the following NFA over the alphabet $\{0, 1\}$.



- What is the language recognized by D. Give a short English description, then explain in a paragraph why your description is correct. Please give an explanation that is as short and simple as possible!

At the start of its execution, D splits into the upper and lower branch. So, we'll define the language in two parts, L_1 for the strings accepted by the upper branch and L_2 for the strings accepted by the lower branch.

The upper branch accepts when it goes to q_2 and then to q_3 with the last two bits of the input. In order to get to q_2 with the second-to-last bit, that bit must be 0. The last bit can be anything. There can be any number of bits preceding the last two; the NFA can stay in q_1 and repeatedly "guess" that it may be on the last two bits every time it sees a 0. Therefore, $L_1 = \{w \mid w$ has length at least two, and the penultimate bit is $0\}$. More formally, $L_1 = \{w \mid |w| \geq 2, w_{|w|-1} = 0\}$.

The lower branch accepts if and only if it doesn't die (because both states are accept states). Therefore, all we need to ensure that the lower branch accepts is to make sure that we always make it out of q_4 by seeing a 0. We are in q_4 every other bit, so we need all bits in odd positions to be 0. Therefore, $L_2 = \{w \mid$ all bits in odd positions are $0\}$. Slightly more formally, $L_2 = \{w \mid w_{2k+1} = 0$ for $k \in \mathbb{Z}_{\geq 0}\}$.

We conclude that $L = L_1 \cup L_2$.

Problem 4: Never seen before (20 points)

Consider the following language over $\Sigma = \{0, 1, 2\}$:

$$L = \{w \in \{0,1,2\}^* \mid \text{ at least one of the alphabet symbols does not occur in } w.\}$$

For instance $00, 0111$ and 21122 are in L, but $0012, 1201$ aren't in L.

(a) Give a DFA as a 5-tuple or by drawing it for the above language.

(b) Give a short explanation (2 to 4 sentences) explaining why your construction is correct. [1]

(a) We give the state diagram in Figure 12 below.

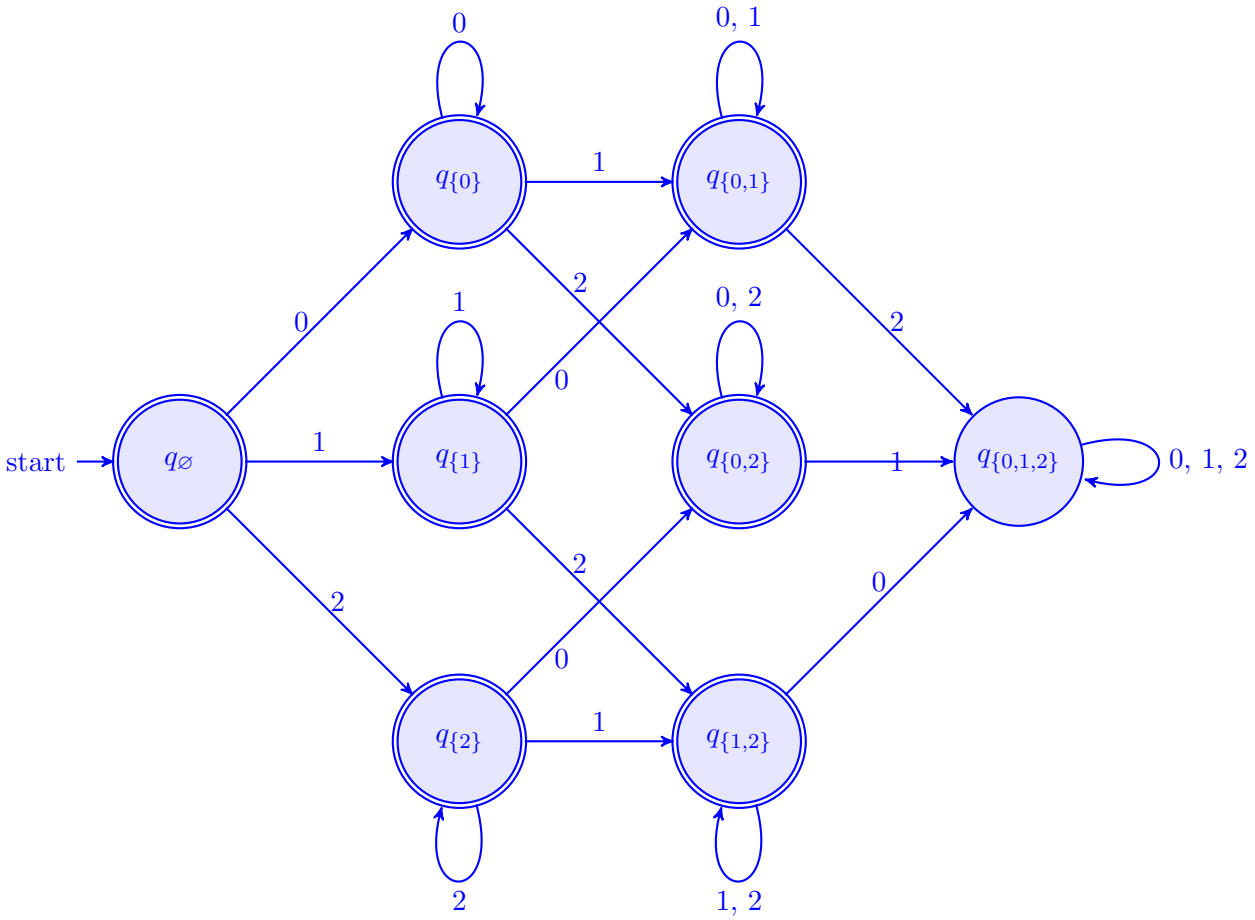


Figure 12: DFA state diagram for 4(a)

(b) Each state in the DFA represents the subset of characters that have been seen so far. For example, if the DFA has read in 001010 so far, it will be in $q_{\{0,1\}}$ because it saw the characters 0 and 1, but not 2. If the DFA is in state q_S corresponding to subset of characters $S \subset \{0, 1, 2\}$, and reads in

[1] To make the explanation easier, your state's names should reflect what information you're keeping track of. I.e. you could have names like $q_{\text{last symbol seen }=1,}$

character c, the DFA will transition to $q_{S\cup\{c\}}$. Therefore the DFA will be in the reject state $q_{\{0,1,2\}}$ if and only if it has seen all three characters in the input string. Otherwise, the DFA will accept.

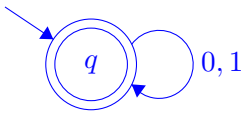
2 Bonus Problem

Let L_1, L_2 be languages over $\Sigma = \{0, 1\}$ such that: M_1 is a minimal-state DFA for L_1 with s_1 states and M_2 is a minimal state DFA for L_2 with s_2 states. Since regular languages are closed under union, there is a generic construction of a DFA for $L_1 \cup L_2$ using at most $s' = s_1 \cdot s_2$ states.(See Theorem 1.25 in Sipser book.)

(a) Give an example where the generic construction is not the best possible. That is, give an example of languages L_1,L_2 over $\Sigma = \{0, 1\}$ such that $L_1 \cup L_2$ has a 1-state DFA, but L_1 requires at least two states. You do not need to formally prove that L_1 requires at least two states.

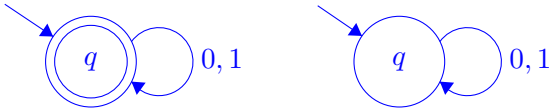
(b) Generalize your construction from part (a) to show: for any $s > 1$, there exists L_1, L_2 such that $L_1 \cup L_2$ has a 1-state DFA, but L_1 requires at least s states. You do not need to formally prove that L_1 requires at least s states.

(a) Our trick will be to define L_1 and L_2 such that $L_1 \cup L_2 = \Sigma^*$ (all strings). This means that $L_1 \cup L_2$ will have the 1-state DFA



(the DFA that always accepts).

Let $L_1 = \{00\}$ and $L_2 = \Sigma^* \setminus \{00\}$. Then, L_1 requires at least 2 states. To see this, observe that there are exactly two 1-state DFAs:



The first one always accepts and the second one never accepts. Therefore, any language besides Σ^* or $\emptyset$ requires a DFA of at least two states.

(b) Let $L_1 = \{0^s\}$ (the string $\underbrace{0 \ldots 0}_{s}$) and $L_2 = \Sigma^* \setminus \{0^s\}$. Again, $L_1 \cup L_2$ has a 1-state DFA (the one that always accepts).

However, L_1 requires at least s states. We will see a proof of this later in the course!