COMP 2805 — Solved Problems

Question 1: Let p be a prime number. Prove that $\sqrt{p}$ is irrational.

Solution: The proof is basically the same as the one we did in class for $\sqrt{2}$. The proof is by contradiction. So we assume that $\sqrt{p}$ is a rational number. Then we can write

$$\sqrt{p} = a/b,$$

where a and b are integers and b is non-zero. We may assume that a and b are not both divisible by p. Taking squares, we get

$$a^2 = pb^2. \tag{1}$$

Hence, a^2 is divisible by p. Since p is a prime number, the integer a is also divisible by p. Hence, we can write

$$a = pc,$$

for some integer c. Substituting a into (1) gives

$$(pc)^2 = pb^2,$$

which simplifies to

$$pc^2 = b^2.$$

We see that b^2 is divisible by p. Again, since p is a prime number, it follows that b is divisible by p.

We have shown that a and b are both divisible by p, which is a contradiction. Therefore, $\sqrt{p}$ is not a rational number.

Question 2: Prove by induction that

$$\sum_{i=1}^{n} \frac{1}{i^2} < 2 - 1/n,$$

for every integer $n \geq 2$.

Solution: First observe that the claim is not true for $n = 1$; that is why we start the induction with $n = 2$.

Basis of the induction: If $n = 2$, then the left-hand side is equal to $1/1^2 + 1/2^2 = 5/4$, whereas the right-hand side is equal to $2 - 1/2 = 3/2$. Since $5/4 < 3/2$, the claim is true for this case.

Induction step: Let $n \geq 2$, and assume that

$$\sum_{i=1}^{n} \frac{1}{i^2} < 2 - 1/n. \tag{2}$$

We have to prove that

$$\sum_{i=1}^{n+1} \frac{1}{i^2} < 2 - 1/(n+1). \tag{3}$$

Using (11), we obtain

$$\sum_{i=1}^{n+1} \frac{1}{i^2} = \sum_{i=1}^{n} \frac{1}{i^2} + \frac{1}{(n+1)^2} < \left(2 - \frac{1}{n}\right) + \frac{1}{(n+1)^2}.$$

If we can show that

$$2 - \frac{1}{n} + \frac{1}{(n+1)^2} \leq 2 - \frac{1}{n+1}, \tag{4}$$

then the inequality in (12) holds, and we are done. So the final step is to prove that the inequality in (13) holds. We can rewrite (13) as

$$\frac{1}{n+1} - \frac{1}{n} + \frac{1}{(n+1)^2} \leq 0,$$

which can be rewritten as

$$\frac{-1}{n(n+1)} + \frac{1}{(n+1)^2} \leq 0,$$

which can be rewritten as

$$\frac{1}{(n+1)^2} \leq \frac{1}{n(n+1)},$$

which can be rewritten as

$$(n+1)^2 \geq n(n+1). \tag{5}$$

Since (5) is equivalent to (13), we need to prove that (5) is true. But this is easy to prove:

$$(n+1)^2 = (n+1)(n+1) \geq n(n+1).$$

In summary, we have proved that (5) holds. This implies that (13) holds. This, in turn, implies that (12) holds, completing the proof of the induction step.

Question 3: For each of the following languages, construct a DFA that accepts the language. In all cases, the alphabet is $\{0, 1\}$.

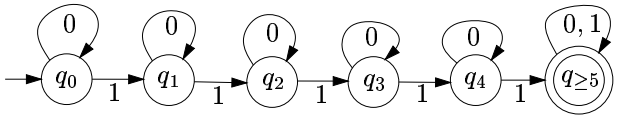
(3.1) $\{w : w$ contains at least five 1s$\}$.

Solution: We will use six states:

- q_0: no 1 has been read
- q_1: exactly one 1 has been read
- q_2: exactly two 1s have been read
- q_3: exactly three 1s have been read

- q_4: exactly four 1s have been read
- $q_{\geq 5}$: five or more 1s have been read

The start state is q_0; there is one accept state: $q_{\geq 5}$. Here is the state diagram:

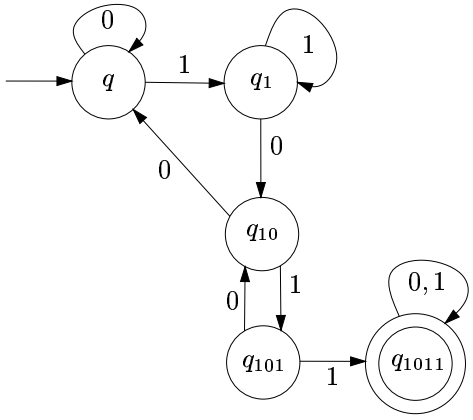


(3.2) $\{w : w$ contains the substring 1011, i.e., $w = x1011y$ for some strings x and $y\}$.

Solution: We will use five states:

- q: start state
- q_1: we have just seen 1 and hope that the next three symbols are 011.
- q_{10}: we have just seen 10 and hope that the next two symbols are 11.
- q_{101}: we have just seen 101 and hope that the next symbol is 1.
- q_{1011}: the input string contains 1011.

The start state is q; there is one accept state: q_{1011}. Here is the state diagram:



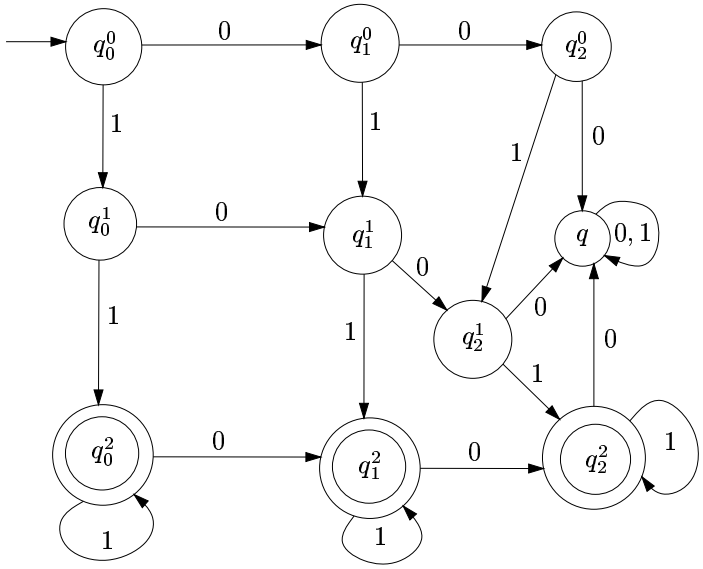
(3.3) $\{w : w$ contains at least two 1s and at most two 0s$\}$.

Solution: We will use ten states:

- q_0^0: we have seen zero 0s and zero 1s
- q_0^1: we have seen zero 0s and one 1
- q_0^2: we have seen zero 0s and at least two 1s
- q_1^0: we have seen one 0 and zero 1s

- q_1^1: we have seen one 0 and one 1
- q_1^2: we have seen one 0 and at least two 1s
- q_2^0: we have seen two 0s and zero 1s
- q_2^1: we have seen two 0s and one 1
- q_2^2: we have seen two 0s and at least two 1s
- q: we have seen at least three 0s

The start state is q_0^0; there are three accept states: q_0^2, q_1^2, and q_2^2. Here is the state diagram:

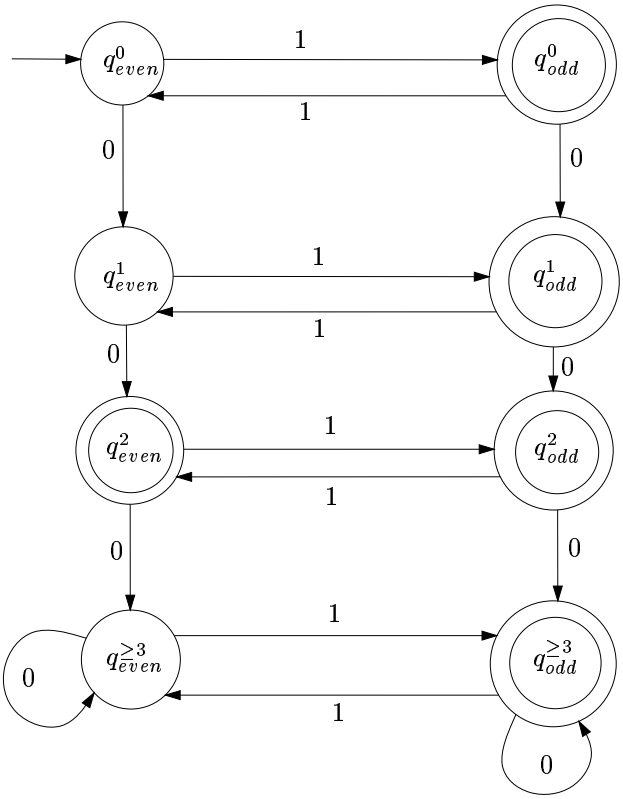


(3.4) $\{w : w$ contains an odd number of 1s or exactly two 0s$\}$.

Solution: We will use eight states; each state records

- that we have seen an even or odd number of 1s (given by the subscript) **and**
- that we have seen zero, one, two, or at least three 0s (given by the superscript).

Here is the state diagram:



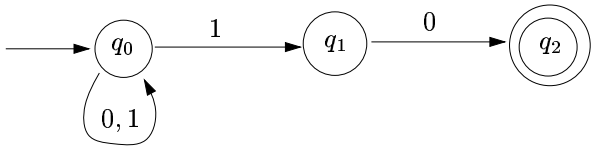
Question 4: For each of the following languages, construct an NFA, with the specified number of states, that accepts the language. In all cases, the alphabet is $\{0, 1\}$.

(4.1) The language $\{w : w \text{ ends with } 10\}$ with three states.

Solution: We will use the following states:

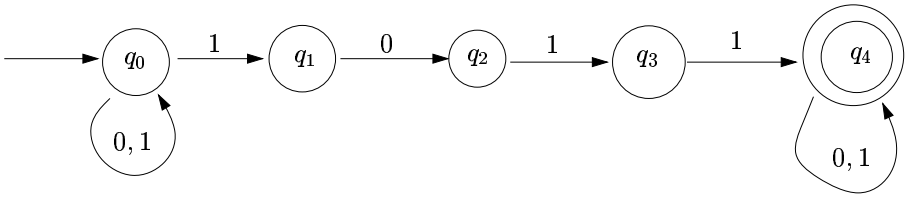
- q_0: we have not reached the rightmost two symbols in the input string.
- q_1: we have read the entire string except for the rightmost symbol (which we hope is 0); the last symbol read was 1.
- q_2: we have read the entire string and the last two symbols were 10.

The start state is q_0; there is one accept state: q_2. Here is the state diagram:



(4.2) The language $\{w : w \text{ contains the substring } 1011\}$ with five states.

Solution: Here is the state diagram:



(4.3) The language $\{w : w$ contains an odd number of 1s or exactly two 0s$\}$ with six states.

Solution: First, we give a DFA (which is also an NFA) with two states that accepts all strings having an odd number of 1s. Then, we give an NFA with three states that accepts all strings having exactly two 0s. Finally, we apply the union construction to these NFA's.

Here is the DFA with two states that accepts all strings having an odd number of 1s. It has states

- q_0: we have seen an even number of 1s.
- q_1: we have seen an odd number of 1s.

q_0 is the start state, whereas q_1 is the accept state. Here are the transitions:

- When in state q_0 and reading 1: go to state q_1
- When in state q_0 and reading 0: go to state q_0
- When in state q_1 and reading 1: go to state q_0
- When in state q_1 and reading 0: go to state q_1

Here is the NFA with three states that accepts all strings having exactly two 0s. It has states

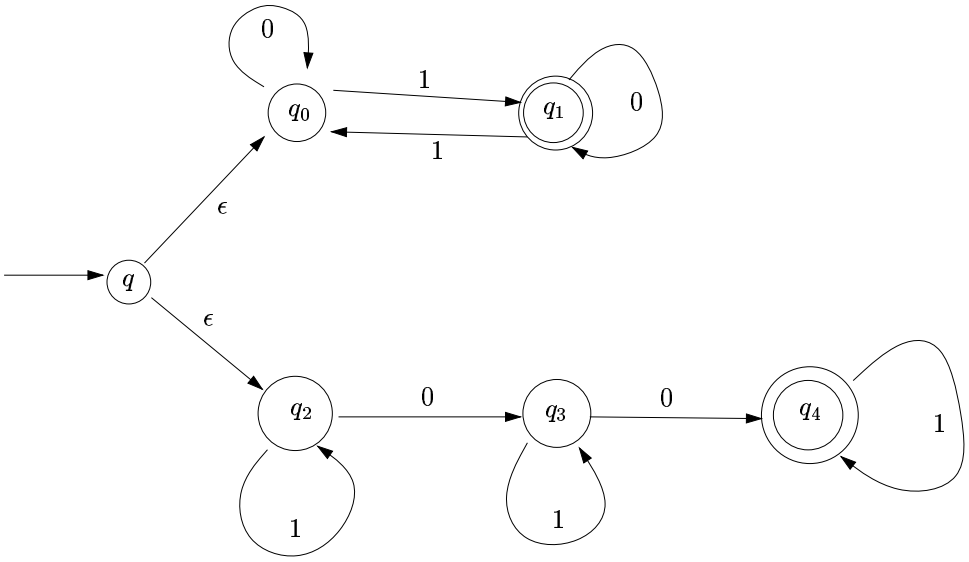
- q_2: we have not seen a 0 yet.
- q_3: we have seen exactly one 0.
- q_4: we have seen exactly two 0s.

q_2 is the start state, q_4 is the accept state. Here are the transitions:

- When in state q_2 and reading 1: go to state q_2
- When in state q_2 and reading 0: go to state q_3
- When in state q_3 and reading 1: go to state q_3
- When in state q_3 and reading 0: go to state q_4
- When in state q_4 and reading 1: go to state q_4

This automaton is nondeterministic, so there are no other arrows.

We now draw the state diagrams of these two automata; then we apply the union construction to them: Add a new start state q and give it ϵ-arrows to the start states of the two automata. Here is the result:



Question 5: Let Σ be a non-empty alphabet, and let L be a language over Σ, i.e., $L \subseteq \Sigma^*$. We define a binary relation R_L on $\Sigma^* \times \Sigma^*$, in the following way: For any two strings u and u' in Σ^*,

$$uR_Lu' \text{ if and only if } (\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L)\,.$$

Prove that R_L is an equivalence relation.

Solution: This question is actually easy, once you write down what has to be proven.

Here we go. We have to prove that the relation R_L satisfies the following three properties: R_L is reflexive, symmetric, and transitive.

Reflexive: First we show that R_L is reflexive. Let u be a string in Σ^*. We have to show that uR_Lu, i.e.,

$$\forall v \in \Sigma^* : uv \in L \Leftrightarrow uv \in L.$$

This is obviously true.

Symmetric: Next we show that R_L is symmetric. Let u and u' be strings in Σ, and assume that uR_Lu'. We have to show that $u'R_Lu$.

Since uR_Lu', we know that

$$\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L. \tag{6}$$

To prove that $u'R_Lu$, we have to show that

$$\forall v \in \Sigma^* : u'v \in L \Leftrightarrow uv \in L. \tag{7}$$

Since (6) and (7) are equivalent, and since (6) is true, it follows that (7) is also true. This means that $u' R_L u$.

Transitive: Finally, we show that R_L is transitive. Let u, u', and u'' be strings in Σ^*, and assume that $u R_L u'$ and $u' R_L u''$. We have to show that $u R_L u''$.

Since $u R_L u'$, we know that

$$\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L. \tag{8}$$

Since $u' R_L u''$, we know that

$$\forall v \in \Sigma^* : u'v \in L \Leftrightarrow u''v \in L. \tag{9}$$

To prove that $u R_L u''$, we have to show that

$$\forall v \in \Sigma^* : uv \in L \Leftrightarrow u''v \in L. \tag{10}$$

It is clear that (10) follows from (8) and (9). Hence, (10) is true, which means that $u R_L u''$.

Question 6: Let $\Sigma = \{0, 1\}$, let

$$L = \{w \in \Sigma^* : \ |w| \text{ is odd}\},$$

and consider the relation R_L defined in the previous question.
(6.1) Prove that for any two strings u and u' in Σ^*,

$$u R_L u' \Leftrightarrow |u| - |u'| \text{ is even.}$$

(6.2) Determine all equivalence classes of the relation R_L.

Solution: For this question, the main trick is to realize that

an even number minus an even number is an even number,

and

an odd number minus an odd number is an even number.

We start with the first part of the question. This first part asks for a $\Leftrightarrow$-proof, so we have to prove both directions. Let u and u' be strings in Σ^*.

$\Leftarrow$: Assume that $|u| - |u'|$ is even. We have to show that $u R_L u'$, i.e.,

$$\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L,$$

i.e.,

$$\forall v \in \Sigma^* : |uv| \text{ is odd } \Leftrightarrow |u'v| \text{ is odd .}$$

Let v be a string in Σ^*. We have

$$|uv| - |u'v| = (|u| + |v|) - (|u'| + |v|) = |u| - |u'|.$$

Since the right-hand side is even, it follows that $|uv|$ and $|u'v|$ are either both even or both odd. This means that

$$|uv| \text{ is odd } \Leftrightarrow |u'v| \text{ is odd } .$$

This proves that uR_Lu'.

$\Rightarrow$: We now assume that uR_Lu'. We have to show that $|u| - |u'|$ is even. Let v be a string in Σ^* such that $|uv|$ is odd. (If $|u|$ is even, then we can take for v any string whose length is odd. If $|u|$ is odd, then we can take for v any string whose length is even.) Since $|uv|$ is odd, we know that $uv \in L$. Since uR_Lu', this implies that $u'v \in L$. By the definition of L, it follows that $|u'v|$ is odd. Therefore,

$$|u| - |u'| = |uv| - |u'v| = \text{ odd minus odd } = \text{ even.}$$

This concludes the proof of the first part of the question. Next, we turn to the second part. We have to determine all equivalence classes of the relation R_L. I hope you see that the equivalence classes have something to do with being even or odd. Let

$$C_0 = \{w \in \Sigma^* : \ |w| \text{ is even } \},$$

and

$$C_1 = \{w \in \Sigma^* : \ |w| \text{ is odd } \}.$$

First we show that any two strings in C_0 are in the relation R_L: Let u and u' be strings in C_0. Then

$$|u| - |u'| = \text{ even minus even } = \text{ even.}$$

Hence, by the first part of the question, we have uR_Lu'.

Next we show that any two strings in C_1 are in the relation R_L: Let u and u' be strings in C_1. Then

$$|u| - |u'| = \text{ odd minus odd } = \text{ even.}$$

Hence, by the first part of the question, we have uR_Lu'.

It is clear that $C_0 \cup C_1 = \Sigma^*$ and $C_0 \cap C_1 = \emptyset$. Hence, C_0 and C_1 form a partition of Σ^*.

Conclusion: C_0 and C_1 are the two equivalence classes of the relation R_L.

Question 7: Let A be a regular language. Prove that there exists an NFA that accepts A and that has exactly one accept state. (*Hint:* There exists a DFA/NFA that accepts A. If this automaton has more than one accept state, modify it.)

Solution: Let $M = (Q, \Sigma, \delta, q, F)$ be a DFA that accepts A. If $|F| = 1$, then we are done, so we assume that F contains at least two states. We construct an NFA N that accepts A and that has exactly one accept state:

1. We make a copy of M.
2. We make each accept state of M a non-accept state of N.
3. We create a new state q', which will be the only accept state of N.

4. We add an ϵ-transition from every state of F to q'.

5. There are no transitions from q' to any state of N.

Formally, we define the NFA $N = (Q', \Sigma, \delta', q, F')$, where

- $Q' = Q \cup \{q'\}$,
- the start state q of N is the start state of M,
- $F' = \{q'\}$,
- for every $r \in Q'$ and every $a \in \Sigma_\epsilon$,

$$\delta'(r,a) = \begin{cases} \{\delta(r,a)\} & \text{if } r \in Q \text{ and } a \neq \epsilon, \\ \emptyset & \text{if } r \in Q \setminus F \text{ and } a = \epsilon, \\ \{q'\} & \text{if } r \in F \text{ and } a = \epsilon, \\ \emptyset & \text{if } r = q'. \end{cases}$$

The careful reader will ask whether this proof also works if the language A is empty. If $A = \emptyset$, then the set F of accept states of M could be empty as well. In this case, however, the above proof is correct.

Question 8: For any string $w = w_1 w_2 \dots w_n$, we denote by w^R the string obtained by reading w backwards, i.e., $w^R = w_n w_{n-1} \dots w_2 w_1$. For any language A, we define A^R to be the language obtained by reading all strings in A backwards, i.e.,

$$A^R = \{w^R : w \in A\}.$$

Let A be a regular language. Prove that the language A^R is also regular. (*Hint:* Use Question 1.)

Solution: Let $M = (Q, \Sigma, \delta, q, F)$ be an NFA that accepts A. By the previous question, we may assume that F consists of exactly one state, say $F = \{q'\}$.

If $w \in A$, then in the state diagram of M, there exists a path from q to q', such that by following this path, we read the string w. Observe that by traversing this path *backwards*, we read the string w^R.

We obtain an NFA for the language A^R, by

1. making a copy of M,
2. making q' the start state,
3. making q the accept state,
4. reversing all arrows in the state diagram of M.

Formally, we define $N = (Q, \Sigma, \delta', q', F')$, where

- Q is the set of states of M,
- the start state q' of N is the accept state of M,
- $F' = \{q\}$,
- for every $r \in Q$ and every $a \in \Sigma_\epsilon$,

$$\delta'(r, a) = \{r' \in Q : r \in \delta(r', a)\}.$$

You should convince yourself that this construction also works if the language A is empty.

Question 9: Let Σ be a non-empty alphabet, and let L be a language over Σ, i.e., $L \subseteq \Sigma^*$. In Assignment 1, you have seen the following equivalence relation R_L on $\Sigma^* \times \Sigma^*$: For any two strings u and u' in Σ^*,

$$uR_Lu' \text{ if and only if } (\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L)\,.$$

(9.1) Let L be a regular language, and let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts L. Let u and u' be strings in Σ^*. Let q be the state reached, when following the path in the state diagram of M, that starts in q_0 and that is obtained by reading the string u. Similarly, let q' be the state reached, when following the path in the state diagram of M, that starts in q_0 and that is obtained by reading the string u'.

Prove the following: If $q = q'$, then uR_Lu'.

(9.2) Prove that Question **(3.1)** implies the following: If L is a regular language, then the equivalence relation R_L has a finite number of equivalence classes.

Solution First Part: We assume that $q = q'$, and have to prove that uR_Lu', i.e.,

$$\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L.$$

Let $v \in \Sigma^*$, and assume that $uv \in L$. We have to show that $u'v \in L$.

Let P be the path in the state diagram of M that starts in q_0 and that is obtained by reading the string uv. Since $uv \in L$, this path ends in a state, say q'', of F. We can write P as P_1P_2, where

- P_1 is the path that starts in q_0 and that is followed when reading u (this path ends in the state q), and
- P_2 is the path that starts in q and that is followed when reading v (this path ends in the state q'').

Let P_1' be the path that starts in q_0 and that is followed when reading u'. By our assumption, this path ends in the state $q' = q$. Let P' be the path $P' = P_1'P_2$. This path starts in q_0, it is obtained when reading the string $u'v$, and it ends in the accept state q''. Hence, $u'v \in L$.

In a symmetric way, we can prove that $u'v \in L$ implies that $uv \in L$.

Solution Second Part: We assume that L is a regular language. Then there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ that accepts L.

For each state q of Q, we define L_q to be the set of all strings $w \in \Sigma^*$, such that the path in the state diagram of M that starts in q_0 and that is obtained by reading w, ends in the state q.

By the first part, we have uR_Lu' for all strings u and u' in L_q. In other words, all strings in L_q are in the same equivalence class of the relation R_L. This means that the number of equivalence classes is at most equal to the number of states in Q. Since Q is a finite set, this implies that the number of equivalence classes is finite.

Question 10: Let L be the language defined by

$$L = \{uu^R : u \in \{0,1\}^*\}.$$

In words, a string is in L if and only if its length is even, and the second half is the reverse of the first half.
(10.1) Let m and n be two distinct positive integers, and consider the two strings $u = 0^m1$ and $u' = 0^n1$. Prove that $\neg(uR_Lu')$.
(10.2) Use Questions **(4.1)** and **(3.2)** to prove that L is not a regular language.

Solution First Part: We consider the strings $u = 0^m1$ and $u' = 0^n1$, where $m \neq n$, and have to prove that $\neg(uR_Lu')$, i.e.,

$$\neg\left(\forall v \in \Sigma^* : uv \in L \Leftrightarrow u'v \in L\right).$$

This means that we have to show that there exists a binary string v such that $uv \in L$ and $u'v \notin L$.

Let $v = 10^m$, i.e., v is the reverse of u. Since

$$uv = uu^R,$$

the string uv is in L. Since

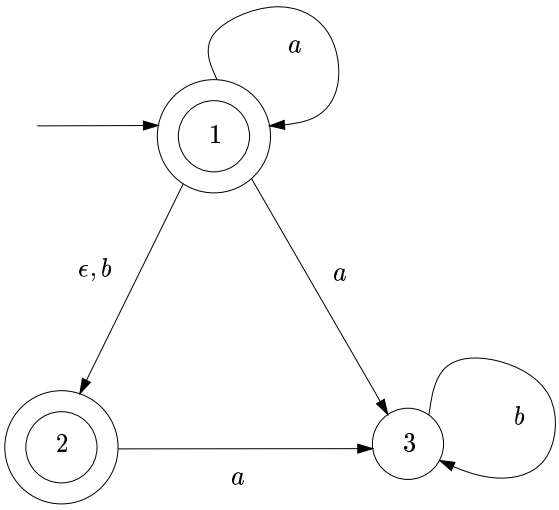
$$u'v = 0^n110^m$$

and since $m \neq n$, the string $u'v$ is not in L.

Solution Second Part: For each integer $m \geq 1$, let C_m be the equivalence class of the relation R_L that contains the string 0^m1.

By the first part, the equivalence classes $C_1, C_2, \ldots$, are distinct. This means that R_L has an infinite number of equivalence classes. But then it follows from Question **(3.2)** that L is not a regular language: If L were regular, R_L would have a finite number of equivalence classes.

Question 11: Use the construction given in class (and described in the notes) to convert the following NFA to an equivalent DFA.



Solution: Following the construction given in class, the DFA has the following eight states:

$$\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}.$$

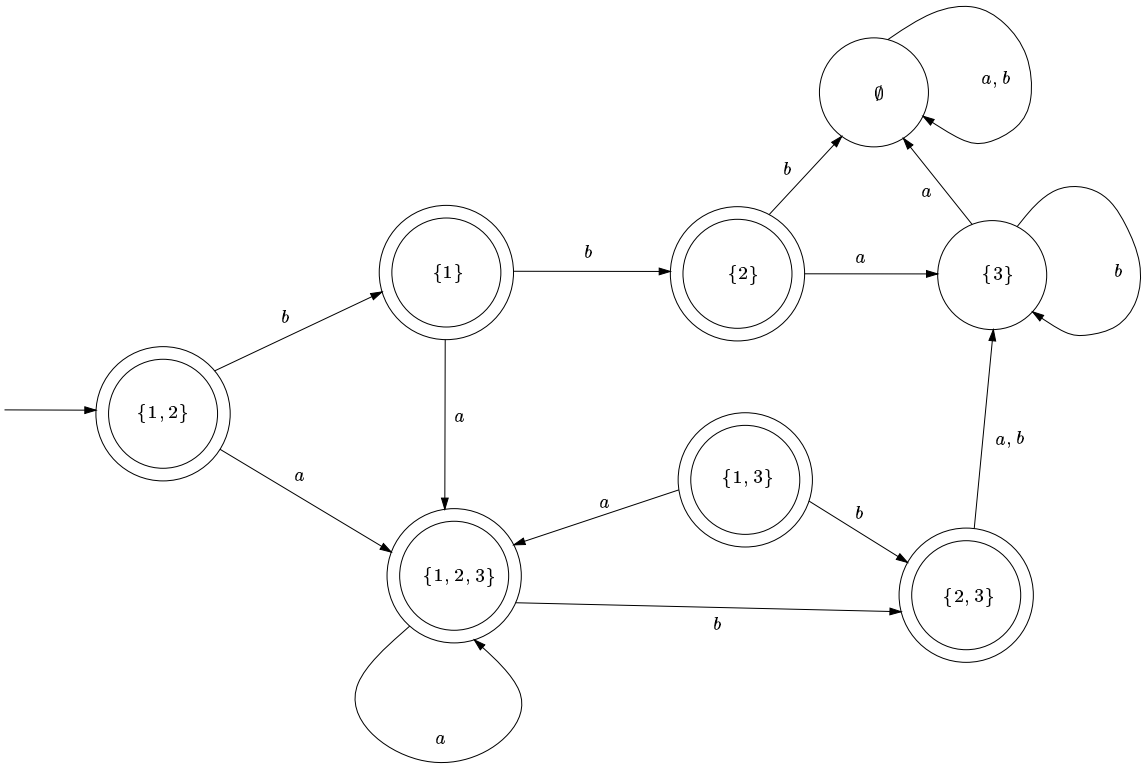
The start state of the DFA is the set of all states of the NFA that can be reached from the NFA's start state 1 by making zero or more ϵ-transitions. Hence, the start state of the DFA is $\{1,2\}$.

The set of accept states of the DFA consists of all states of the DFA that contain at least one accept state of the NFA. That is, the set of accept states of the DFA consists of all states of the DFA that contain 1 or 2. This gives the following six accept states:

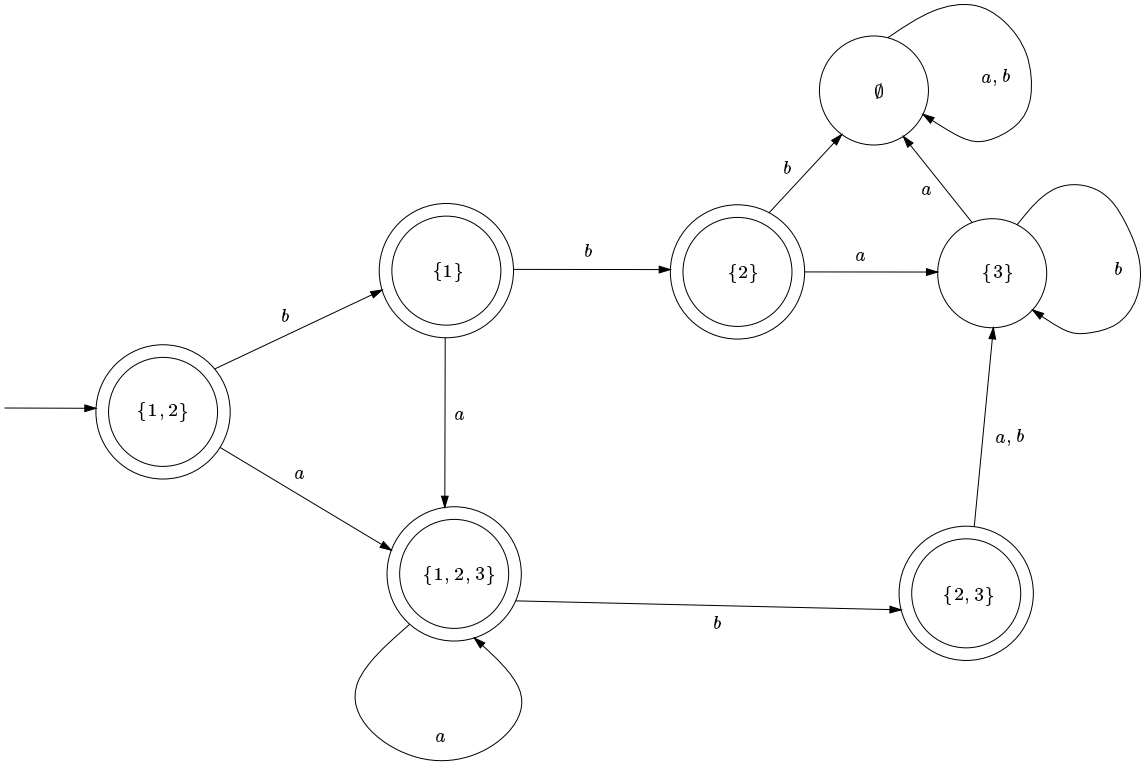
$$\{1\}, \{2\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}.$$

The transition function of the DFA is specified in the following state diagram:

ERROR IN FIGURE: When in state 1,2 and reading b, you switch to state 2. After you make this change, you will see that state 1 has no incoming edges and can be removed.



The NFA above is correct, but, since the state $\{1, 3\}$ cannot be reached from the start state $\{1, 2\}$, it can be removed. This gives the final DFA:



Question 12: Give regular expressions describing the following languages. In all cases, the alphabet is $\{0, 1\}$.

- $\{w : w \text{ contains at least five 1s}\}$.

  Solution:
  $$0^*10^*10^*10^*10^*1(0 \cup 1)^*$$
  or
  $$(0 \cup 1)^*1(0 \cup 1)^*1(0 \cup 1)^*1(0 \cup 1)^*1(0 \cup 1)^*1(0 \cup 1)^*$$

- $\{w : w \text{ contains at least two 1s and at most one 0}\}$.

  Solution:
  $$111^* \cup 1^*(011 \cup 101 \cup 110)1^*$$

- $\{w : w \text{ contains an even number of 1s or exactly two 0s}\}$.

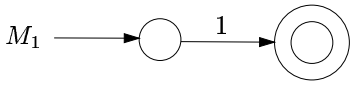
  Solution:
  $$0^* \cup (0^*10^*10^*)^* \cup 1^*01^*01^*$$

Question 13: Use the construction given in class (and described in the notes) to convert the regular expression
$$(((10)^*(00)) \cup 10)^*$$
to an NFA. The alphabet is $\{0, 1\}$.

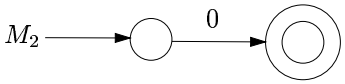
Solution: We first consider how the regular expression is "built":

- Take the regular expressions 1 and 0, and combine them into the regular expression 10.
- Take the regular expression 10, and turn it into the regular expression $(10)^*$.
- Take the regular expressions 0 and 0, and combine them into the regular expression 00.
- Take the regular expressions $(10)^*$ and 00, and combine them into the regular expression $(10)^*00$.
- Take the regular expressions $(10)^*00$ and 10, and combine them into the regular expression $(10)^*00 \cup 10$.
- Take the regular expression $(10)^*00 \cup 10$, and turn it into the regular expression $((10)^*00 \cup 10)^*$.

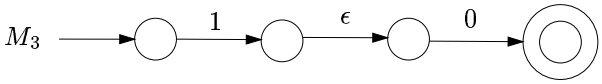
First, we construct an NFA M_1 that accepts the language described by the regular expression 1:



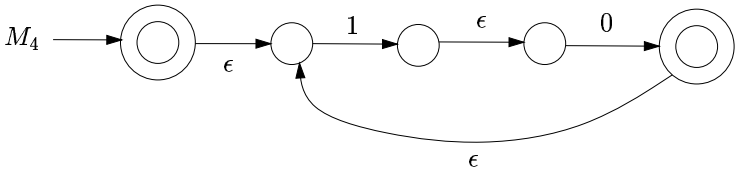
Next, we construct an NFA M_2 that accepts the language described by the regular expression 0:



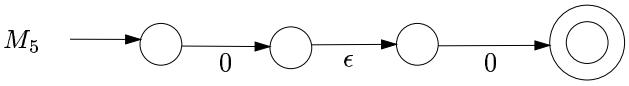
Next, we apply the concatenation construction to M_1 and M_2. This gives an NFA M_3 that accepts the language described by the regular expression 10:



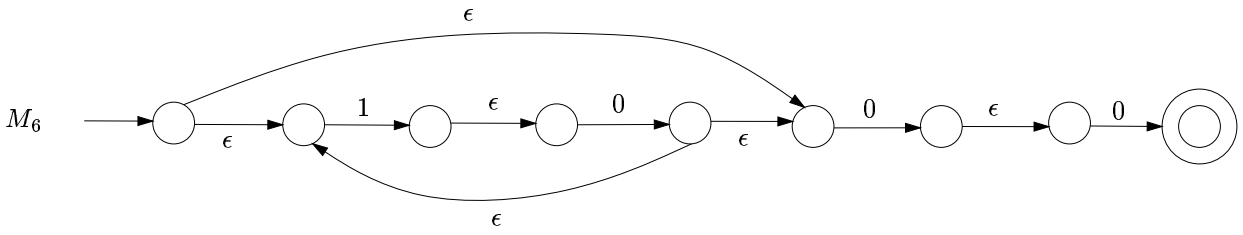
Next, we apply the star construction to M_3. This gives an NFA M_4 that accepts the language described by the regular expression $(10)^*$:



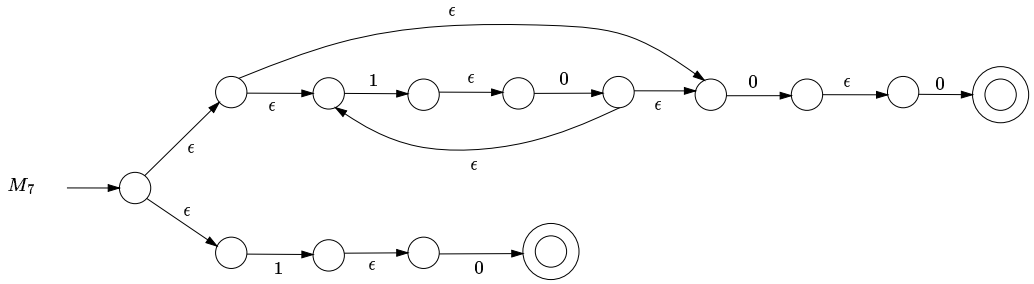
Next, we apply the concatenation construction to M_2 and M_2. This gives an NFA M_5 that accepts the language described by the regular expression 00:



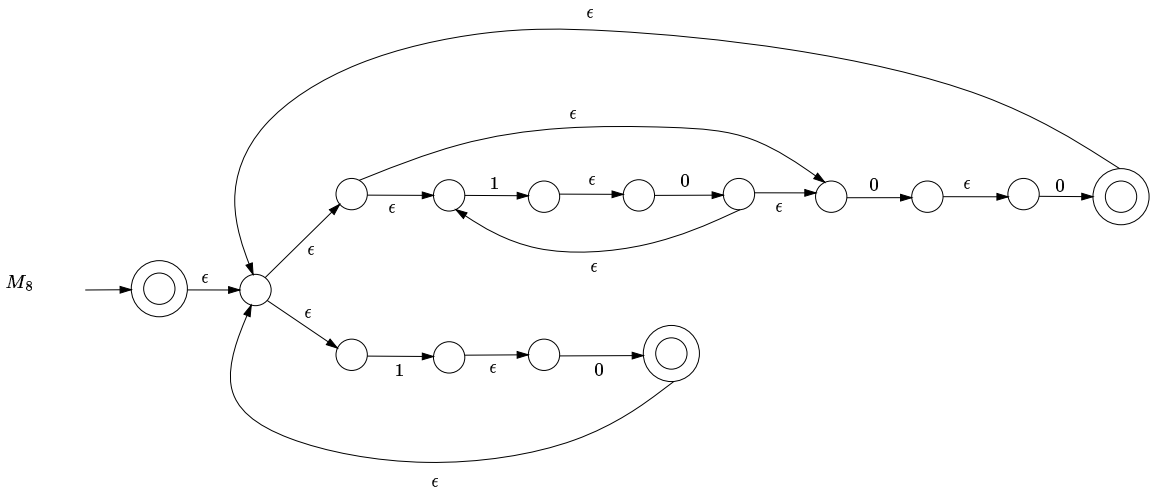
Next, we apply the concatenation construction to M_4 and M_5. This gives an NFA M_6 that accepts the language described by the regular expression $(10)^*00$:



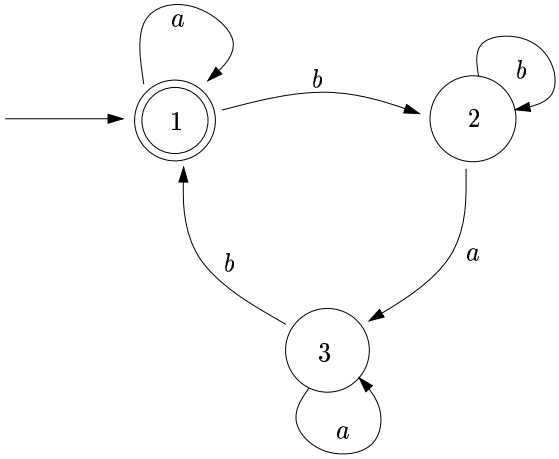
Next, we apply the union construction to M_6 and M_3. This gives an NFA M_7 that accepts the language described by the regular expression $(10)^*00 \cup 10$:



Finally, we apply the star construction to M_7. This gives an NFA M_8 that accepts the language described by the regular expression $((10)^*00 \cup 10)^*$:



Question 14: Use the construction given in class to convert the following DFA to a regular expression.



Solution: For each state $i = 1, 2, 3$, we define L_i to be the set of all strings w in $\{a, b\}^*$ such that the path in the state diagram that starts in state i and that corresponds to w ends in

the accept state 1. We obtain the following set of equations:

$$L_1 = \epsilon \cup aL_1 \cup bL_2 \quad (11)$$
$$L_2 = aL_3 \cup bL_2 \quad (12)$$
$$L_3 = aL_3 \cup bL_1 \quad (13)$$

Since 1 is the start state, we need a regular expression for L_1.

We use the following tool to solve these equations:

If $L = BL \cup C$ and $\epsilon \notin B$, then $L = B^*C$.

We solve the equations (11), (12), and (13), in the following way: From (13), we obtain

$$L_3 = a^*bL_1,$$

which we substitute into (12), giving

$$L_2 = aa^*bL_1 \cup bL_2,$$

which we rewrite as

$$L_2 = bL_2 \cup aa^*bL_1,$$

which solves to

$$L_2 = b^*aa^*bL_1,$$

which we substitute into (11), giving

$$L_1 = \epsilon \cup aL_1 \cup bb^*aa^*bL_1,$$

which we rewrite as

$$L_1 = (a \cup bb^*aa^*b)\, L_1 \cup \epsilon,$$

which solves to

$$L_1 = (a \cup bb^*aa^*b)^*\, \epsilon = (a \cup bb^*aa^*b)^*.$$

Hence, the regular expression describing the language accepted by the DFA is

$$(a \cup bb^*aa^*b)^*.$$

By the way,

$$(a \cup bb^*aa^*b)^*\, \epsilon$$

is also correct.

Question 15: Using the pumping lemma, prove that the following languages are not regular.

1. $\{a^nb^mc^{n+m} : n \geq 0, m \geq 0\}$.
2. $\{a^nb^ma^n : n \geq 0, m \geq 0\}$.
3. $\{a^{2^n} : n \geq 0\}$. (Remark: a^{2^n} is the string consisting of 2^n many a's.)

Solution: First, we do

$$A = \{a^n b^m c^{n+m} : n \geq 0, m \geq 0\}.$$

Assume the language A is regular. Let p be the pumping length, as given by the pumping lemma. Let $s = a^p b^p c^{2p}$. Then $s \in A$ and $|s| = 4p \geq p$. Hence, by the pumping lemma, we can write $s = xyz$, where

1. $y \neq \epsilon$,
2. $|xy| \leq p$, and
3. $xy^i z \in A$, for all $i \geq 0$.

Since $|xy| \leq p$, the string y contains only as. Since $y \neq \epsilon$, the string y contains at least one a. Hence, the string xz contains (i) less than p many as, (ii) exactly p many bs, and exactly $2p$ many cs. That is, in the string xz, the number of as plus the number of bs is less than the number of cs. But this means that $xz = xy^0 z$ is not in the language A. This is a contradiction, because, by the pumping lemma, this string is an element of A. So we have a contradiction, and we can conclude that A is not regular.

Next, we do

$$B = \{a^n b^m a^n : n \geq 0, m \geq 0\}.$$

Assume the language B is regular. Let p be the pumping length, as given by the pumping lemma. Let $s = a^p b a^p$. Then $s \in B$ and $|S| = 2p + 1 \geq p$. Hence, by the pumping lemma, we can write $s = xyz$, where

1. $y \neq \epsilon$,
2. $|xy| \leq p$, and
3. $xy^i z \in B$, for all $i \geq 0$.

Since $|xy| \leq p$, the string y is contained in the leftmost a-block of the string s. We also know that y is non-empty. Consider the string $xz = xy^0 z$. This string starts with an a-block consisting of less than p many as, followed by one b, followed by an a-block consisting of exactly p many as. Hence, xz is not an element of B. This is a contradiction, because, by the pumping lemma, this string is an element of B. So we have a contradiction, and we can conclude that B is not regular.

Finally, we do

$$C = \{a^{2^n} : n \geq 0\}.$$

Assume the language C is regular. Let p be the pumping length, as given by the pumping lemma. Let $s = a^{2^p}$. Then $s \in C$ and $|S| = 2^p \geq p$. Hence, by the pumping lemma, we can write $s = xyz$, where

1. $y \neq \epsilon$,
2. $|xy| \leq p$, and

3. $xy^iz \in C$, for all $i \geq 0$.

Since $y \neq \epsilon$ and $|xy| \leq p$, the string y has the form $y = a^k$, for some integer k with $1 \leq k \leq p$. Consider the string

$$xy^2z = xyyz = a^{2^p+k}.$$

The length of this string is equal to $2^p + k$. Since $k \geq 1$, we have $2^p + k > 2^p$. Since $k \leq p$, we have $2^p + k \leq 2^p + p < 2^p + 2^p = 2^{p+1}$. Hence,

$$2^p < |xy^2z| < 2^{p+1},$$

that is, the length of the string xy^2z is strictly between two consecutive powers of 2. But this means that xy^2z is not in the language C. This is a contradiction, because, by the pumping lemma, this string is an element of C. So we have a contradiction, and we can conclude that C is not regular.

Question 16: Give context-free grammars that generate the following languages. In all cases, the set Σ of terminals is equal to $\{0, 1\}$.

1. $\{w : w$ contains at least three 1s$\}$.
2. $\{w : w$ starts and ends with the same symbol$\}$.
3. $\{w :$ the length of w is odd and its middle symbol is 0$\}$.

Solution: First, we do

$$\{w : w \text{ contains at least three 1s}\}.$$

Observe that this language is regular. Here is a DFA that accepts it: There are four states A_0, A_1, A_2, and A_3, where A_0 is the start state and A_3 is the accept state.

- When in state A_0 and read 0: go to state A_0
- When in state A_0 and read 1: go to state A_1
- When in state A_1 and read 0: go to state A_1
- When in state A_1 and read 1: go to state A_2
- When in state A_2 and read 0: go to state A_2
- When in state A_2 and read 1: go to state A_3
- When in state A_3 and read 0: go to state A_3
- When in state A_3 and read 1: go to state A_3

In class, it was shown how to convert a DFA to a context-free grammar:

- Set of variables: $\{A_0, A_1, A_2, A_3\}$

- Start variable: A_0
- Set of terminals: $\{0, 1\}$
- Rules:

$$\begin{array}{lll} A_0 \to 0A_0 & A_0 \to 1A_1 & \\ A_1 \to 0A_1 & A_1 \to 1A_2 & \\ A_2 \to 0A_2 & A_2 \to 1A_3 & \\ A_3 \to 0A_3 & A_3 \to 1A_3 & A_3 \to \epsilon \end{array}$$

Here is another explanation that this grammar generates the language:

- From A_3, we can generate all binary strings.
- From A_2, we can generate all strings that contain at least one 1.
- From A_1, we can generate all strings that contain at least two 1s.
- From A_0, we can generate all strings that contain at least three 1s.

By the way, here is another grammar that generates the same language (S is the start variable):

$$\begin{array}{lcl} S & \to & T1T1T1T \\ T & \to & \epsilon|0T|1T \end{array}$$

From S, we can derive the string $T1T1T1T$, whereas from T, we can derive all binary strings. Hence, from S, we can derive all strings of the form $u1v1w1x$, where u, v, w, and x are binary strings. These are exactly the binary strings that contain at least three 1s.

Next, we do

$$\{w : w \text{ starts and ends with the same symbol}\}.$$

This language is also regular. So one solution is to give a DFA that recognizes the language, and then to convert it to a context-free grammar. Here is a direct solution:

- Variables: S and T
- Start variable: S
- Terminals: 0 and 1
- Rules:

$$\begin{array}{l} S \to 0|1|0T0|1T1 \\ T \to \epsilon|0T|1T \end{array}$$

Here is the explanation why this works: From T, we can derive all binary strings. From S, we can derive the strings 0, 1, $0T0$, and $1T1$. Hence, from S, we can derive the following strings:

- 0,

- 1,
- $0(0 \cup 1)^*0$,
- $1(0 \cup 1)^*1$.

This is exactly the language we want.

Finally, we do

$$\{w : \text{ the length of } w \text{ is odd and its middle symbol is } 0\}.$$

- Variable: S
- Start variable: S
- Terminals: 0 and 1
- Rules: $S \to 0|0S0|0S1|1S0|1S1$

Here is the explanation why this works: Without using the rule $S \to 0$, we can derive from S all strings of the form uSv, where u and v have the same length.

If we add the rule $S \to 0$, then we can only replace, in the string uSv, the symbol S by 0. So we can generate all strings of the form $u0v$, where u and v have the same length. This is exactly the language we are supposed to generate.

Question 17: Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where $V = \{A, B, S\}$, $\Sigma = \{0, 1\}$, S is the start variable, and R consists of the rules

$$\begin{array}{lcl} S & \to & 0S|1A|\epsilon \\ A & \to & 0B|1S \\ B & \to & 0A|1B \end{array}$$

Define the following language L:

$$L := \{w \in \{0,1\}^* : \begin{array}{l} w \text{ is the binary representation of a non-negative integer} \\ \text{that is divisible by three } \} \cup \{\epsilon\} \end{array}$$

Prove that $L = L(G)$. (*Hint:* The variables S, A, and B are used to remember the remainder after division by three.)

Solution: By looking at the rules, you will notice that, from the start variable S, we can derive the empty string ϵ, and strings of the form wS, wA, and wB, where w is a non-empty binary string.

For any non-empty binary string w, let n_w be the non-negative integer whose binary represenation is w. For example,

- if $w = 1011$, then $n_w = 1 + 2 + 8 = 11$, and

- if $w = 0001011$, then $n_w = 1 + 2 + 8 = 11$.

We claim the following: Let w be a non-empty binary string. Then the following holds:

1. $S \stackrel{*}{\Rightarrow} wS$ if and only if $n_w \equiv 0 \bmod 3$.
2. $S \stackrel{*}{\Rightarrow} wA$ if and only if $n_w \equiv 1 \bmod 3$.
3. $S \stackrel{*}{\Rightarrow} wB$ if and only if $n_w \equiv 2 \bmod 3$.

If a non-empty string is in the language $L(G)$ of the grammar G, then, by definition, $S \stackrel{*}{\Rightarrow} w$. Since w does not contain any variable, the *last* step in the derivation of w must use the rule $S \to \epsilon$. Therefore, the derivation has the form

$$S \stackrel{*}{\Rightarrow} wS \Rightarrow w.$$

If we assume that the above claim is true, then we see that the following holds, for any non-empty binary string w:

$$\begin{array}{lll} w \in L(G) & \text{if and only if} & S \stackrel{*}{\Rightarrow} wS \Rightarrow w \\ & \text{if and only if} & S \stackrel{*}{\Rightarrow} wS \\ & \text{if and only if} & n_w \equiv 0 \bmod 3 \\ & \text{if and only if} & w \in L. \end{array}$$

In other words, if we can prove the claim made above, then we have also shown that $L = L(G)$.

It remains to prove the claim. The proof is by induction on the length of the string w. For the basis of the induction, we assume that $|w| = 1$. Then $w = 0$ or $w = 1$.

- If $w = 0$, then $n_w = 0$. We have $S \stackrel{*}{\Rightarrow} 0S = wS$ and $n_w \equiv 0 \bmod 3$.
- If $w = 1$, then $n_w = 1$. We have $S \stackrel{*}{\Rightarrow} 1A = wA$ and $n_w \equiv 1 \bmod 3$.

Hence, for the base case, the claim is true.

Let w be a non-empty binary string. The induction hypothesis is that the claim is true for w. What do we have to show?

- Let $w' = w0$, i.e., w' is the string obtained by adding 0 to the end of w. Then we have to prove that the claim holds for the string w'. Observe that $n_{w'} = 2n_w$.
- Let $w'' = w1$, i.e., w'' is the string obtained by adding 1 to the end of w. Then we have to prove that the claim holds for the string w''. Observe that $n_{w''} = 2n_w + 1$.

We next observe that

- $n_w \equiv 0 \bmod 3$ if and only if $n_{w'} \equiv 0 \bmod 3$,
- $n_w \equiv 1 \bmod 3$ if and only if $n_{w'} \equiv 2 \bmod 3$,

- $n_w \equiv 2 \bmod 3$ if and only if $n_{w'} \equiv 1 \bmod 3$,
- $n_w \equiv 0 \bmod 3$ if and only if $n_{w''} \equiv 1 \bmod 3$,
- $n_w \equiv 1 \bmod 3$ if and only if $n_{w''} \equiv 0 \bmod 3$,
- $n_w \equiv 2 \bmod 3$ if and only if $n_{w''} \equiv 2 \bmod 3$.

Using these observations, and using the induction hypothesis, if follows that

$$\begin{array}{lll} S \stackrel{*}{\Rightarrow} w'S & \text{if and only if} & S \stackrel{*}{\Rightarrow} wS \\ & \text{if and only if} & n_w \equiv 0 \bmod 3 \\ & \text{if and only if} & n_{w'} \equiv 0 \bmod 3. \end{array}$$

In a similar way, we obtain

$$\begin{array}{lll} S \stackrel{*}{\Rightarrow} w'A & \text{if and only if} & S \stackrel{*}{\Rightarrow} wB \\ & \text{if and only if} & n_w \equiv 2 \bmod 3 \\ & \text{if and only if} & n_{w'} \equiv 1 \bmod 3. \end{array}$$

From these two examples, you can verify the remaining cases:

1. $S \stackrel{*}{\Rightarrow} w'B$ if and only if $n_{w'} \equiv 2 \bmod 3$,
2. $S \stackrel{*}{\Rightarrow} w''S$ if and only if $n_{w''} \equiv 0 \bmod 3$,
3. $S \stackrel{*}{\Rightarrow} w''A$ if and only if $n_{w''} \equiv 1 \bmod 3$,
4. $S \stackrel{*}{\Rightarrow} w''B$ if and only if $n_{w''} \equiv 2 \bmod 3$.

Question 18: Let A and B be context-free languages over the same alphabet Σ.
(18.1) Prove that the union $A \cup B$ of A and B is also context-free.
(18.2) Prove that the concatenation AB of A and B is also context-free.
(18.3) Prove that the star A^* of A is also context-free.

Solution: Since A is context-free, there is a context-free grammar $G_1 = (V_1, \Sigma, R_1, S_1)$ that generates A. Similarly, since B is context-free, there is a context-free grammar $G_2 = (V_2, \Sigma, R_2, S_2)$ that generates B. We assume that $V_1 \cap V_2 = \emptyset$. (If this is not the case, then we rename the variables of G_2.)

First, we show that $A \cup B$ is context-free. Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where

- $V = V_1 \cup V_2 \cup S$, where S is a new variable, which is the start variable of G,
- $R = R_1 \cup R_2 \cup \{S \to S_1 | S_2\}$.

From the start variable S, we can derive the strings S_1 and S_2. From S_1, we can derive all strings of A, whereas from S_2, we can derive all strings of B. Hence, from S, we can derive all strings of $A \cup B$. In other words, the grammar G generates the union of A and B. Therefore, this union is context-free.

Next, we show that AB is context-free. Let $G = (V, \Sigma, R, S)$ be the context-free grammar, where

- $V = V_1 \cup V_2 \cup S$, where S is a new variable, which is the start variable of G,
- $R = R_1 \cup R_2 \cup \{S \to S_1 S_2\}$.

From the start variable S, we can derive the string $S_1 S_2$. From S_1, we can derive all strings of A, whereas from S_2, we can derive all strings of B. Hence, from S, we can derive all strings of the form uv, where $u \in A$ and $v \in B$. In other words, the grammar G generates the concatenation of A and B. Therefore, this concatenation is context-free.

Finally, we show that A^* is context-free. Let $G = (V, \Sigma, R, S_1)$ be the context-free grammar, where

- $V = V_1$,
- S_1 is the start variable (hence, we do not introduce a new start variable),
- $R = R_1 \cup \{S \to \epsilon | SS\}$.

From the start variable S, we can derive all strings S^n, where $n \geq 0$. From S, we can derive all strings of A. Hence, from S, we can derive all strings of the form $u_1 u_2 \ldots u_n$, where $n \geq 0$, and each string u_i $(1 \leq i \leq n)$ is in A. In other words, the grammar G generates the star of A. Therefore, A^* is context-free.

Question 19: Define the following two languages A and B:

$$A = \{a^m b^n c^n : m \geq 0, n \geq 0\}$$

and

$$B = \{a^m b^m c^n : m \geq 0, n \geq 0\}.$$

(19.1) Prove that both A and B are context-free, by constructing two grammars, one that generates A and one that generates B.

(19.2) We have seen in class that the language

$$\{a^n b^n c^n : n \geq 0\}$$

is not context-free. Explain why this implies that the intersection of two context-free languages is not necessarily context-free.

(19.3) Use De Morgan's Law to conclude that the complement of a context-free language is not necessarily context-free.

Solution: First, the first part of the question. Here is a context-free grammar that generates A (S_1 is the start variable):

$$S_1 \to aS_1|T_1$$
$$T_1 \to \epsilon|bT_1c$$

From S_1, we can derive all strings of the form a^mT_1, where $m \geq 0$. From T_1, we can derive all strings of the form b^nc^n, where $n \geq 0$. Hence, from S_1, we can derive all strings of the form $a^mb^nc^n$, where $m \geq 0$ and $n \geq 0$. In other words, this grammar generates the language A.

Here is a context-free grammar that generates B (S_2 is the start variable):

$$S_2 \to S_2c|T_2$$
$$T_2 \to \epsilon|aT_2b$$

This grammar generates the language B.

By the way, you can also prove that A is context-free, by constructing a pushdown automaton that accepts A:

- It reads the input string from left to right.
- As long as an a is read, the head moves one cell to the right on the input tape, and the stack is not changed.
- When the first b is read, it changes to the state "now I am in the block of bs". For every b read, a symbol S is pushed onto the stack. If a is read in this state, the automaton loops forever.
- When the first c is read, it changes to the state "now I am in the block of c's". For every c read, the top symbol S is popped from the stack. If a or b is read in this state, the automaton loops forever. If the top symbol of the stack is \$ and c is read, then it loops forever (because there are more cs than bs). If the top symbol of the stack is S and $\square$ is read on the tape, then it loops forever (because there are more bs than cs). If the top symbol of the stack is \$ and $\square$ is read on the tape, then we make the stack empty (and, hence, accept).

You should be able to give a formal definition of this pushdown automaton. In a similar way, you can define a pushdown automaton that accepts the language B.

Now we do the second part of the question. We know that A and B are both context-free. The intersection of A and B is

$$C = \{a^nb^nc^n : n \geq 0\}.$$

We have seen in class that C is not context-free. Therefore, the intersection of two context-free languages is not necessarily context-free.

Now the last part of the question. We have to use De Morgan's Law to show that the complement of a context-free language is not necessarily context-free.

The proof is by contradiction. So we assume that the complement of every context-free language is context-free.

Let A and B be as above. Both are context-free, so by our assumption, their complements $\overline{A}$ and $\overline{B}$ are also context-free. Then, by question 4.1, $\overline{A} \cup \overline{B}$ is context-free. Then, by our assumption, the complement of $\overline{A} \cup \overline{B}$, i.e., $\overline{\overline{A} \cup \overline{B}}$ is context-free. But, the latter language is equal to $A \cap B$, and we know from the second part in this question, that $A \cap B$ is not context-free. This is a contradiction.

Conclusion: The complement of a context-free language is not necessarily context-free.

Question 20: Give (deterministic or nondeterministic) pushdown automata that accept the following languages.

1. $\{w \in \{0,1\}^* : w \text{ contains more 1s than 0s}\}$.
2. $\{w \in \{0,1\}^* : w \text{ is a palindrome}\}$. (A string w is a palindrome if $w = w^R$, i.e., reading w from left to right gives the same result as reading w from right to left.)

Solution: I will give two deterministic pushdown automata for the language

$$\{w \in \{0,1\}^* : w \text{ contains more 1s than 0s}\}.$$

Here is the first solution. The idea is to walk along the input string from left to right, and use the stack to keep track of the number of 1s minus the number of 0s seen so far. (How do we do this: for each 1, push a symbol S onto the stack; for each 0, pop the top symbol S.) The problem is that even though if the input string is in the language, an initial segment of the string may have more 0s than 1s. In this case, the stack would become empty before we have seen the entire string. So this doesn't work. The next idea is to use the stack to keep track of the number of 0s minus the number of 1s seen so far. This leads to a similar problem as above. The trick is to combine these two ideas.

- Tape alphabet $\Sigma = \{0,1\}$.
- Stack alphabet $\Gamma = \{\$, S\}$.

We use three states:

- q_0: It is the start state. If we are in this state, then
  - the number of 0s seen is at least the number of 1s seen, and
  - the number of S-symbols on the stack is equal to the number of 0s seen minus the number of 1s seen.
- q_1: If we are in this state, then
  - the number of 1s seen is at least the number of 0s seen, and
  - the number of S-symbols on the stack is equal to the number of 1s seen minus the number of 0s seen.

- q_2: If we are in this state, then we have read the entire input string (so the tape head is on the blank symbol immediately to the right of the input string). If we are in this state, then we know that the input string has more 1s than 0s (so the string belongs to the language). What remains to be done is to pop all symbols from the stack.

The instructions are as follows.

- $q_0 0\$ \to q_0 R\$S$
- $q_0 0S \to q_0 RSS$
- $q_0 1\$ \to q_1 R\$S$
- $q_0 1S \to q_0 R\epsilon$
- $q_0 B\$ \to q_0 N\$$
  - Explanation: We have read the entire string; since the stack does not contain any S-symbols, the string contains as many 1s as it contains 0s. Therefore, the string is not in the languge. In this instruction, we don't make any changes, so we loop forever and don't accept the string.
- $q_0 BS \to q_0 NS$
  - Explanation: We have read the entire string; since we are in state q_0 and the stack contains at least one S-symbol, the string contains more 0s than 1s. Therefore, the string is not in the languge. In this instruction, we don't make any changes, so we loop forever and don't accept the string.
- $q_1 0\$ \to q_0 R\$S$
- $q_1 0S \to q_1 R\epsilon$
- $q_1 1\$ \to q_1 R\$S$
- $q_1 1S \to q_1 RSS$
- $q_1 \square\$ \to q_1 N\$$
  - Explanation: We have read the entire string; since the stack does not contain any S-symbols, the string contains as many 1s as it contains 0s. Therefore, the string is not in the languge. In this instruction, we don't make any changes, so we loop forever and don't accept the string.
- $q_1 \square S \to q_2 N\epsilon$
  - Explanation: We have read the entire string; since we are in state q_1 and the stack contains at least one S-symbol, the string contains more 1s than 0s. Therefore, the string is in the languge. We switch to state q_2.

- $q_2 \square S \to q_2 N \epsilon$
- $q_2 \square \$ \to q_2 N \epsilon$
  - Explanation: Now the stack is empty, so we accept the string.

Here is the second solution. The idea is similar, but now we use only two states. In the first solution, we used the S-symbol to keep track of the absolute value of the difference in 1s and 0s seen so far. We used two states to indicate whether we had seen more 1s than 0s, or more 0s than 1s.

In this second solution, the stack will store only 0s (plus \$ at the bottom), if we have seen more 0s than 1s. It will store only 1s (plus \$ at the bottom), if we have seen more 1s than 0s.

- Tape alphabet $\Sigma = \{0, 1\}$
- Stack alphabet $\Gamma = \{\$, 0, 1\}$.

We use two states:

- q: This is the start state. If we are in this state, then the following holds: Let a be the number of 1s read so far, let b be the number of 0s read so far.

  If $a \geq b$, then the stack contains $a - b$ many 1s (plus \$ at the bottom); the stack does not contain any 0s.

  If $a \leq b$, then the stack contains $b - a$ many 0s (plus \$ at the bottom); the stack does not contain any 1s.

- q': If we are in this state, then we have read the entire input string w, and we know that w has more 1s than 0s. The tape head is on the blank symbol immediately to the right of w. The stack contains only 1s plus the \$ at the bottom. The purpose of this state is to make the stack empty (because we want to accept the string).

Here are the instructions:

- $q0\$ \to qR\$0$
- $q00 \to qR00$
- $q01 \to qR\epsilon$
- $q1\$ \to qR\$1$
- $q10 \to qR\epsilon$
- $q11 \to qR11$
- $qB\$ \to qN\$$
  - Explanation: The string contains equal number of 1s and 0s, so we loop forever.

- $q\Box 0 \to qN\$$
  - Explanation: The string contains more 0s than 1s, so we loop forever.
- $q\Box 1 \to q'N\epsilon$
- $q'\Box 1 \to q'N\epsilon$
- $q'\Box\$ \to q'N\epsilon$
  - Explanation: The stack is empty now, so we terminate and accept.

Finally, we give a nondeterministic pushdown automaton for the language

$$\{w \in \{0,1\}^* : w \text{ is a palindrome}\}.$$

We have to be careful:

- If a palindrome is of odd length, then it can be written as $u0v$ or as $u1v$, where v is the reverse of u.
- If a palindrome is of even length, then it can be written as uv, where v is the reverse of u.

At the start, we "guess" whether the input string has odd or even length.

If we guessed for odd length, then we do the following: While walking along the string from left to right, we "guess" that we have reached the middle symbol. All symbols to the left of the middle are pushed onto the stack. After we have reached the middle, we check if the contents of the stack is the same as the remaining part of the input string.

If we guessed for even length, then we do the following: While walking along the string from left to right, we "guess" that we have just entered the second half of the input string. All symbols in the first half are pushed onto the stack. After we have entered the second half, we check if the contents of the stack is the same as the remaining part of the input string.

- tape alphabet $\Sigma = \{0,1\}$ and
- stack alphabet $\Gamma = \{\$, 0, 1\}$.

We will use five states:

- q_0: start state.
- q_1: we have guessed that the input string has odd length; we have not guessed yet the position of the middle symbol.
- q_2: we have guessed that the input string has odd length; we have guessed already the position of the middle symbol.

- q_3: we have guessed that the input string has even length; we have not guessed yet that we have entered the second half of the input string.
- q_4: we have guessed that the input string has even length; we have guessed already that we have entered the second half of the input string.

Here are the instructions:

- $q_0 0\$ \to q_1 N\$$
- $q_0 0\$ \to q_3 N\$$
- $q_0 1\$ \to q_1 N\$$
- $q_0 1\$ \to q_3 N\$$
- $q_0 \Box\$ \to q_0 N\epsilon$ (input string is empty, therefore accept)
- $q_1 0\$ \to q_1 R\$0$
- $q_1 0\$ \to q_2 R\$$
- $q_1 1\$ \to q_1 R\$1$
- $q_1 1\$ \to q_2 R\$$
- $q_1 00 \to q_1 R00$
- $q_1 00 \to q_2 R0$
- $q_1 01 \to q_1 R10$
- $q_1 01 \to q_2 R1$
- $q_1 10 \to q_1 R01$
- $q_1 10 \to q_2 R0$
- $q_1 11 \to q_1 R11$
- $q_1 11 \to q_2 R1$
- $q_1 \Box\$ \to q_1 N\$$ (loop forever)
- $q_1 \Box 0 \to q_1 N0$ (loop forever)
- $q_1 \Box 1 \to q_1 N1$ (loop forever)
- $q_2 0\$ \to q_2 N\$$ (loop forever)
- $q_2 1\$ \to q_2 N\$$ (loop forever)

- $q_2 00 \rightarrow q_2 R\epsilon$
- $q_2 01 \rightarrow q_2 N1$ (loop forever)
- $q_2 10 \rightarrow q_2 N0$ (loop forever)
- $q_2 11 \rightarrow q_2 R\epsilon$
- $q_2 \square\$ \rightarrow q_2 N\epsilon$ (accept)
- $q_2 \square 0 \rightarrow q_2 N0$ (loop forever)
- $q_2 \square 1 \rightarrow q_2 N1$ (loop forever)
- $q_3 0\$ \rightarrow q_3 R\$0$
- $q_3 1\$ \rightarrow q_3 R\$1$
- $q_3 00 \rightarrow q_3 R00$
- $q_3 00 \rightarrow q_4 N0$
- $q_3 01 \rightarrow q_3 R10$
- $q_3 01 \rightarrow q_4 N1$
- $q_3 10 \rightarrow q_3 R01$
- $q_3 10 \rightarrow q_4 N0$
- $q_3 11 \rightarrow q_3 R11$
- $q_3 11 \rightarrow q_4 N1$
- $q_3 \square\$ \rightarrow q_3 N\$$ (loop forever)
- $q_3 \square 0 \rightarrow q_3 N0$ (loop forever)
- $q_3 \square 1 \rightarrow q_3 N1$ (loop forever)
- $q_4 0\$ \rightarrow q_4 N\$$ (loop forever)
- $q_4 1\$ \rightarrow q_4 N\$$ (loop forever)
- $q_4 00 \rightarrow q_4 R\epsilon$
- $q_4 01 \rightarrow q_4 N1$ (loop forever)
- $q_4 10 \rightarrow q_4 N0$ (loop forever)
- $q_4 11 \rightarrow q_4 R\epsilon$
- $q_4 \square\$ \rightarrow q_4 N\epsilon$ (accept)

- $q_4 \square 0 \to q_4 N 0$ (loop forever)
- $q_4 \square 1 \to q_4 N 1$ (loop forever)

Question 21: Prove that the following languages are not context-free:

1. $\{a^n b^n a^n b^n : n \geq 0\}$. The alphabet is $\{a, b\}$.
2. $\{w\#x :$ w is a substring of x; and $w, x \in \{a, b\}^*\}$.

   The alphabet is $\{a, b, \#\}$. The string $aba\#abbababbb$ is in the language, whereas the string $aba\#baabbaabb$ is not in the language.

Solution: First, we prove that the language

$$A = \{a^n b^n a^n b^n : n \geq 0\}$$

is not context-free.

Assume that A is context-free. By the pumping lemma, there is an integer $p \geq 1$, such that for all strings $s \in A$ with $|s| \geq p$, the following holds: We can write $s = uvxyz$, where

1. vy is non-empty,
2. vxy has length at most p,
3. the string $uv^i xy^i z$ is in A, for all $i \geq 0$.

Consider the pumping length p. We choose $s = a^p b^p a^p b^p$. Then s is a string in A, and the length of s is $4p$, which is at least p. So we can write $s = uvxyz$ such that 1., 2., and 3. above hold. Since $|vxy| \leq p$, there are three cases:

Case 1: vxy lives in the leftmost half of s, that is, it lives in $a^p b^p$.

Case 2: vxy lives in the rightmost half of s, that is, it lives in $a^p b^p$.

Case 3: vxy lives in the two middle quarters of s, that is, it lives in $b^p a^p$.

Since all these cases are very similar, I will only consider Case 1. (You should go through the other cases yourself.)

So we assume that vxy lives in the leftmost half of s. Consider the string $s' = uvvxyyz$. By the pumping lemma, s' is contained in A. The string s' contains the rightmost half of s. That is, s' ends with $a^p b^p$. The part of s' that is to the left of the rightmost half of s is not equal to $a^p b^p$. Therefore, the string s' is not in A. This is a contradiction. It follows that the language A is not context-free.

Next we prove that the language

$$B = \{w\#x :$$ w is a substring of x; and $w, x \in \{a, b\}^*\}$$$

is not context-free. Remember that the alphabet for this language is equal to $\{a, b, \#\}$.

Observe the following: if w and x are equal, then the string $w\#x$ is in the language B.

Assume the language B is context-free. By the pumping lemma, there is an integer $p \geq 1$, such that for all strings s in B with $|s| \geq p$, the following holds: We can write $s = uvxyz$, where

1. vy is non-empty,

2. vxy has length at most p,

3. the string uv^ixy^iz is in B, for all $i \geq 0$.

Consider the pumping length p. We choose $s = a^pb^p\#a^pb^p$. Then s is a string in B, and the length of s is $4p+1$, which is at least p. So we can write $s = uvxyz$ such that 1., 2., and 3. above hold.

Case 1: vxy is completely to the left of the #-symbol.

Consider the string $uvvxyyz$. In this string, the part to the left of the #-symbol is longer than the part to the right of the #-symbol. Hence, the left part cannot be a substring of the right part. Therefore, $uvvxyyz$ is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Case 2: vxy is completely to the right of the #-symbol.

Consider the string uxz. In this string, the part to the right of the #-symbol is shorter than the part to the left of the #-symbol. Hence, the left part cannot be a substring of the right part. Therefore, uxz is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Case 3: v or y contains the #-symbol.

Then the string $uvvxyyz$ contains two #-symbols, hence, it is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Alternatively, the string uxz contains no #-symbol, hence, it is not in B. But, by the pumping lemma, it must be in B. This is a contradiction.

Case 4: x contains the #-symbol. (This is the remaining case.)

Since vxy has length at most p, this implies that v contains only b's, and y contains only a's. So we can write

$$v = b^k \text{ for some } k \geq 0$$

and

$$y = a^\ell \text{ for some } \ell \geq 0.$$

Since vy is not empty, at least one of k and ℓ must be strictly positive. Consider the string uxz. This string is equal to

$$uxz = a^pb^{p-k}\#a^{p-\ell}b^p.$$

By the pumping lemma, this string is in B, so a^pb^{p-k} must be a substring of $a^{p-\ell}b^p$. This implies that $\ell = 0$ (otherwise, there are more a's in a^pb^{p-k} than there are a's in $a^{p-\ell}b^p$).

Consider the string $uvvxyyz$. This string is equal to

$$uvvxyyz = a^pb^{p+k}\#a^{p+\ell}b^p.$$

By the pumping lemma, this string is in B, so a^pb^{p+k} must be a substring of $a^{p+\ell}b^p$. This implies that $k = 0$ (otherwise, there are more b's in a^pb^{p+k} than there are b's in $a^{p+\ell}b^p$).

So we have shown that both k and ℓ are zero. This is a contradiction.

Conclusion: In each of the four cases, we get a contradiction. Therefore, the language B is not context-free.

Question 22: Construct a Turing machine with one tape that decides the language

$$\{w \in \{0,1\}^* : \ w \text{ contains twice as many 0's as 1's}\}.$$

Assume that, at the start of the computation, the tape head is on the leftmost symbol of the input string. Explain the meaning of the states that you use.

First solution: The idea is to walk along the string from left to right and delete two 0's and one 1. Then walk back to the left, and repeat. If we delete a symbol (that means, replace it by $\square$), then there are "holes" in the string, so it is not clear any more how to find the leftmost and rightmost symbols of the string. One solution is to shift the piece to the right of a $\square$ one position to the left. I will use a different approach: At the beginning, we write a \$ to the left and to the right of the string. Then we can use these \$'s to find the start and end of the string. So after deleting two 0's and one 1, I will just leave the "holes" in the string.

Here is an outline of the algorithm:
Stage 1: Write \$ to the left of the leftmost symbol of the input string w.
Stage 2: Walk to the right and write \$ to the right of the rightmost symbol of w.
Stage 3: Walk back to the leftmost \$.
Stage 4: Walk to the right, find two 0's and one 1, and delete them. Then walk to the leftmost \$ again.

Repeat stage 4 until the string is empty. If nothing "strange" happens, accept; otherwise, reject.

We use the following states:

- q_0: start state; make one step to the left.
- q_1: we are at the first empty cell to the left of the bitstring.
- q_2: the leftmost \$ has been written; we walk to the first empty cell to the right of the bitstring.
- q_3: we walk to the leftmost \$.
- q_4: start of stage 4; we walk to the right until we see a 0 or a 1.
- q^0: we are in stage 4; one 0 has been deleted.
- q^{00}: we are in stage 4; two 0's have been deleted.
- q^1: we are in stage 4; one 1 has been deleted.
- q^{01}: we are in stage 4; one 0 and one 1 have been deleted.
- Comment: if two 0's and one 1 have been deleted, we switch to state q_3.
- q_{accept}

- q_{reject}

Here are the instructions:

$$q_0 0 \to q_1 0L \qquad q_1\square \to q_2\$R \qquad q_2 0 \to q_2 0R \qquad q_3 0 \to q_3 0L$$
$$q_0 1 \to q_1 1L \qquad\qquad\qquad q_2 1 \to q_2 1R \qquad q_3 1 \to q_3 1L$$
$$q_0\square \to q_{accept} \qquad\qquad\qquad q_2\square \to q_3\$L \qquad q_3\$ \to q_4\$R$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad q_3\square \to q_3\square L$$

$$q_4 0 \to q^0\square R \qquad q^0 0 \to q^{00}\square R \qquad q^{00} 0 \to q^{00} 0R$$
$$q_4 1 \to q^1\square R \qquad q^0 1 \to q^{01}\square R \qquad q^{00} 1 \to q_3\square L$$
$$q_4\square \to q_4\square R \qquad q^0\square \to q^0\square R \qquad q^{00}\square \to q^{00}\square R$$
$$q_4\$ \to q_{accept} \qquad q^0\$ \to q_{reject} \qquad q^{00}\$ \to q_{reject}$$

$$q^1 0 \to q^{01}\square R \qquad q^{01} 0 \to q_3\square L$$
$$q^1 1 \to q^1 1R \qquad q^{01} 1 \to q^{01} 1R$$
$$q^1\square \to q^1\square R \qquad q^{01}\square \to q^{01}\square R$$
$$q^1\$ \to q_{reject} \qquad q^{01}\$ \to q_{reject}$$

Second solution: Rearrange the input string so that all 0's are to the left of all 1's. Then test whether the new string is of the form $0^{2n}1^n$.
Stage 1: Starting at the leftmost symbol, walk to the right until you see the first 1. Go to Stage 2.
Stage 2: At the start of this stage, the head is on the cell containing the leftmost 1. Replace this 1 by $\square$. Then walk to the right until you see the first 0; replace this 0 by 1. Then walk to the left until you see the first $\square$, and replace it by 0.

Hence, in this second stage, we find a 1 and a 0, such that the 1 is to the left of the 0, and we change their order. Of course, we repeat Stage 2 until we don't find a 0 to the right of the leftmost 1.
Stage 3: Repeatedly remove the two leftmost 0's and the rightmost 1.

Question 23: Construct a Turing machine with one tape, that gets as input an integer $x \geq 1$, and returns as output the integer $x - 1$. Integers are represented in binary.

Start of the computation: The tape contains the binary representation of the input x. The tape head is on the leftmost bit of x, and the Turing machine is in the start state.

End of the computation: The tape contains the binary representation of the number $x - 1$. The tape head is on the leftmost bit of $x - 1$, and the Turing machine is in the final state.

The Turing machine in this question does not have an accept state or a reject state; instead, it has a final state. As soon as this final state is entered, the Turing machine terminates. At termination, the contents of the tape is the output of the Turing machine.

Solution: The Turing machine will do the following:

Stage 1: Walk to the rightmost bit of the input string.
Stage 2: Walk to the left and replace each 0 by a 1. When the first 1 is reached, replace it by a 0. At that moment, $x - 1$ has been computed. Then walk to the leftmost bit.

We use the following states:

- q_0: start state; we are in stage 1.
- q_1: final state.
- q_2: we are in stage 2. Until now, we have only encountered 0's.
- q_3: $x - 1$ has been computed; we walk to the leftmost bit.

Here are the instructions:

$$\begin{array}{lll} q_0 0 \to q_0 R & q_2 0 \to q_2 1 L & q_3 0 \to q_3 0 L \\ q_0 1 \to q_0 R & q_2 1 \to q_3 0 L & q_3 1 \to q_3 1 L \\ q_0 \Box \to q_2 L & & q_3 \Box \to q_1 \Box R \end{array}$$

Question 24: Let n be a fixed positive integer, and let k be the number of bits in the binary representation of n. (Hence, $k = 1 + \lfloor \log n \rfloor$.) Construct a Turing machine with one tape, tape alphabet $\{0, 1, \Box\}$, and exactly $k + 1$ states $q_0, q_1, \ldots, q_k$, that does the following:

Start of the computation: The tape is empty, i.e., every cell of the tape contains $\Box$, and the Turing machine is in the start state q_0.

End of the computation: The tape contains the binary representation of the integer n, the tape head is on the rightmost bit of the binary representation of n, and the Turing machine is in the final state q_k.

The Turing machine in this question does not have an accept state or a reject state; instead, it has a final state q_k. As soon as state q_k is entered, the Turing machine terminates.

Solution: This Turing machine only works for this single integer n. Basically, we will encode the binary representation of n in the states of this Turing machine.

Let the binary representation of the (fixed) integer n be $a_0 a_1 \ldots a_{k-1}$, so that

$$n = a_{k-1} + a_{k-2} 2 + a_{k-3} 2^2 + a_{k-4} 2^3 + \ldots + a_1 2^{k-2} + a_0 2^{k-1}.$$

The instructions are as follows:

$$\begin{array}{lcl} q_0 \Box & \to & q_1 a_0 R \\ q_1 \Box & \to & q_2 a_1 R \\ \vdots & \vdots & \vdots \\ q_{k-2} \Box & \to & q_{k-1} a_{k-2} R \\ q_{k-1} \Box & \to & q_k a_{k-1} N \end{array}$$

Question 25: Give an informal description (in plain English) of a Turing machine with three tapes, that gets as input the binary representation of an arbitrary integer $m \geq 1$, and returns as output the unary representation of m.

Start of the computation: The first tape contains the binary representation of the input m. The other two tapes are empty (i.e., contain only $\Box$s). The Turing machine is in the start state.

End of the computation: The third tape contains the unary representation of m, i.e., a string consisting of m many ones. The Turing machine is in the final state.

The Turing machine in this question does not have an accept state or a reject state; instead, it has a final state. As soon as this final state is entered, the Turing machine terminates.

Hint: Use the second tape to maintain a string of ones, whose length is a power of two.

Solution: Let the binary representation of m be $b_{k-1}b_{k-2}\ldots b_0$, so that

$$m = b_0 + b_1 2 + b_2 2^2 + b_3 2^3 + \ldots + b_{k-2}2^{k-2} + b_{k-1}2^{k-1}.$$

Stage 1: On the first tape, walk to the rightmost bit (i.e., b_0). On the second tape, write 1.

Stage 2: In this stage, there is a loop, in which the following *invariant* is maintained:

1. The head of the first tape is at b_i.
2. The second tape contains a string consisting of 2^i many ones.
3. The third tape contains a string consisting of $\sum_{j=1}^{i-1} b_j 2^j$ many ones.

At the start of Stage 2, this invariant holds for $i = 0$. In one iteration, we do the following:

1. If $b_i = 1$, then copy the contents of the second tape to the third tape.
2. Double the contents of the second tape (i.e., if initially, the second tape contains a string consisting of ℓ many ones, then afterwards, it contains a string consisting of 2ℓ many ones.)
3. On the first tape, move one cell to the left (this corresponds to the instruction $i := i+1$).

You should verify that the invariant is correctly maintained. The loop terminates as soon as the head of the first tape reads $\Box$; this means that $i = k$, and Stage 2 is completed.

How do we do the "doubling" step on the second tape:

1. Start at the leftmost 1 on the second tape. Walk to the right until you see the first $\Box$, and replace this $\Box$ by the symbol α. Then walk back to the leftmost 1 of the string.
2. Replace the leftmost 1 by \$, walk to the leftmost $\Box$, and replace this $\Box$ by 1. Then walk back to the leftmost 1, and repeat.
3. If all 1s to the left of α are gone, the second tape contains the string $\$^\ell\alpha 1^\ell$. Now replace all \$s by 1s. This results in the string $1^\ell\alpha 1^\ell$. Finally, delete the symbol α, and shift the second string 1^ℓ one position to the left. This results in the string $1^{2\ell}$.