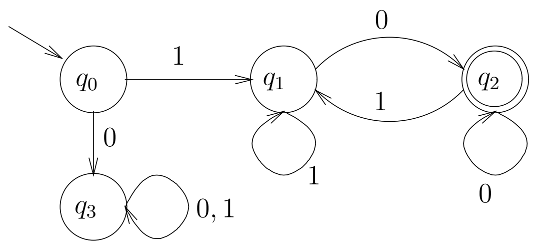
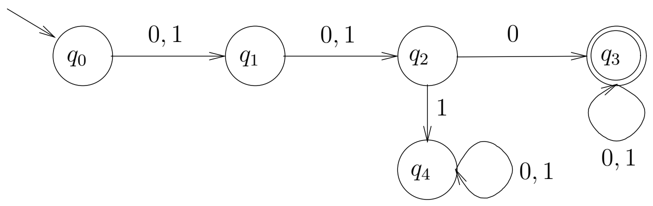
Solutions to Problem Set 1 (Revised)

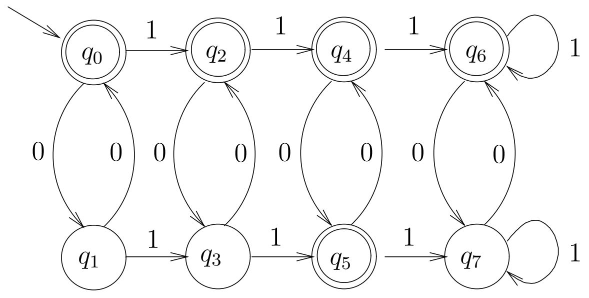
1.4 a). $L = \{w | w$ begins with a 1 and ends with a 0$\}$.



d). $L = \{w | w$ has length at least 3 and its third symbol is a 0$\}$.

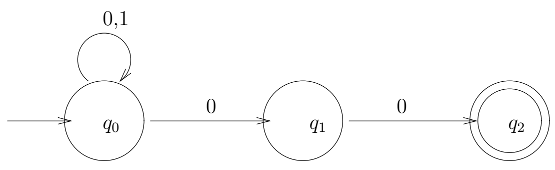


ℓ). $L = \{w | w$ contains an even number of 0's, or exactly two 1's $\}$.

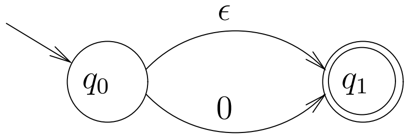


1.5 a). $L = \{w | w$ ends with 00$\}$ with three states.

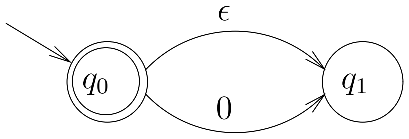
Notice that w only has to end with 00, and before the two zeros, there can be anything. Therefore, we can construct the following NFA to recognize L:



1.10 b). We need to give an example of NFA M (and corresponding language $C = L(M)$) such that, swapping the accept and non-accept states in M yields a NFA (say M') that does NOT recognize the complement of C. The example is the following: consider the language $C = \{ \epsilon, 0 \}$ recognized by the following NFA:



Clearly, swapping the accept and non-accept states gives the following NFA M'



But the language recognized by M' contains the empty word (ϵ) which already is in $C = L(M)$. Therefore, $L(M')$ cannot be the complement of $L(M)$ (otherwise it wouldn't accept ϵ).

1.12 To transform an NFA into a DFA, we start with finding the ϵ-closure of the original start state q_0 and make $E(q_0)$ the new start state. A conveninent way to save work and avoid mistakes is to compute transitions only for the new states appeared in previous computation, starting from the new start state. This lazy strategy is proved to be correct and the proof can be found in the first discussion section notes.
Second thing to bear in mind is that we're always computing the ϵ-closure of the transitions. Don't forget that!

a). The ϵ-closure of the original start state remains the same for this NFA. Please refer to Figure 1 for the final DFA.

States	a	b
1	1,2	2
1,2	1,2	1,2
2	$\emptyset$	1
$\emptyset$	$\emptyset$	$\emptyset$

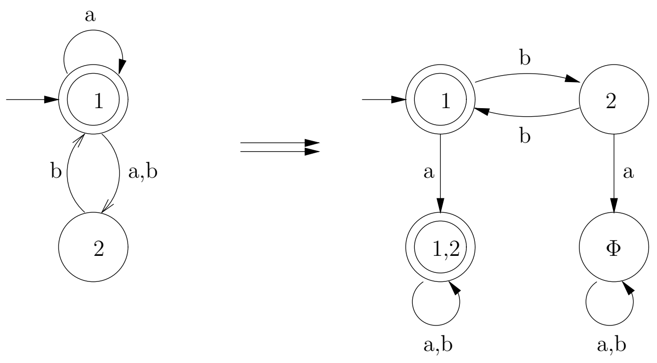


Figure 1: Problem 1.12 (a)

b). Notice that for this NFA, the ϵ-closure of its original start state is no longer the same. It is actually the new state $\{1,2\}$. So we'll start from here. Please refer to Figure 2 for the final DFA.

States	a	b
1,2	1,2,3	$\emptyset$
1,2,3	1,2,3	2,3
$\emptyset$	$\emptyset$	$\emptyset$
2,3	1,2	2,3

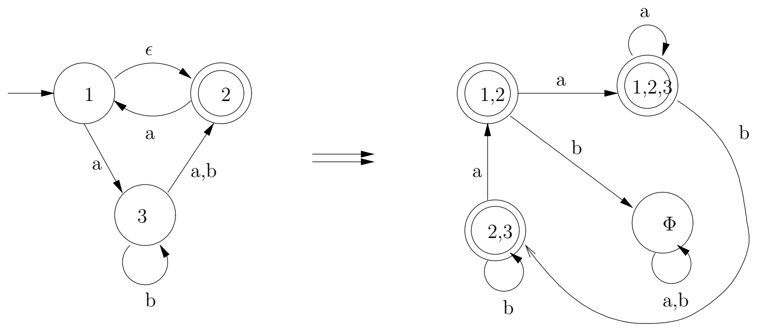


Figure 2: Problem 1.12 (b)

1.13 e). $L = \{w | w$ starts with 0 and has odd length, or starts with 1 and has even length$\}$.

Solution: Almost directly from the definition of L:

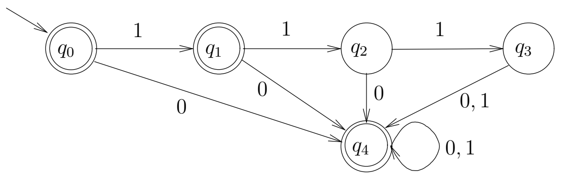
$$0(00 \cup 01 \cup 10 \cup 11)^* \cup 1((00 \cup 01 \cup 10 \cup 11)^*(0 \cup 1)) .$$

h). $L = \{w | w$ is any string except 11 and 111$\}$.

Solution: $(\epsilon \cup 1) \cup (0 \cup 10 \cup 110 \cup 1110 \cup 1111)(0 \cup 1)^*$.

Tip: How did we come up with this? First build the NFA or DFA that recognize the language (or the complement of the language sought; recall that, given a DFA recognizing L, the DFA recognizing $\overline{L}$ is given by swapping the accept and non-accept states in the DFA for L). Then, by simple inspection (or using the NFA-to-RE transformation procedure) obtain the corresponding RE.

For example, the DFA that recognizes L is:

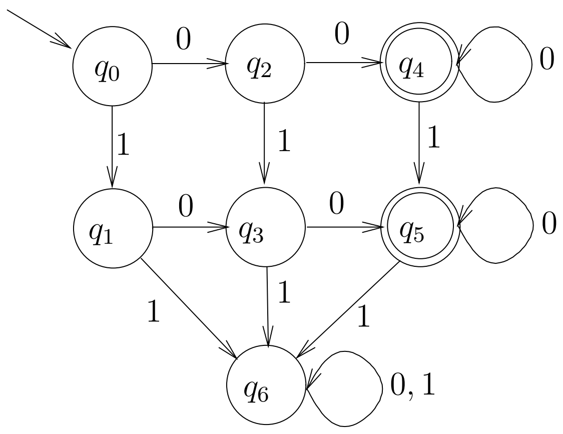


(it's not hard to see how to get to the regular expression from the above DFA).

j). $L = \{w | w$ contains at least two 0's and at most one 1 $\}$.

Solution: $000^* \cup (000^*1 \cup 010 \cup 100)0^*$.

Tip: The DFA that recognizes L is:



1.14 a). We convert the regular expression $(0 \cup 1)^*000(0 \cup 1)^*$ into an NFA by following the steps in Theorem 1.28 (shown in Figure 3).

Finally, for comparison, we also show an equivalent NFA built by observing patterns which is much simpler (shown in Figure 4.)

b). Same steps as above for the regular expression $(((00)^*(11)) \cup 01)^*$. Please refer to Figure 5.

1.16 a). We need to convert the NFA of Figure 6 (upper left automata) into a regular expression. We must follow the usual procedure:

1. Transform the NFA into a GNFA
2. Remove states of the GNFA one by one. For example, we first remove state 1 and then state 2.
3. When only the (single) start state and the (single) accept state are remaining, the resulting regular expression is the regular expression labeling the last arrow. This is $a^*b(a \cup ba^*b)^*$.

The procedure is depicted in Figure 6.

b). We need to convert the NFA of Figure 7 (upper left) into a regular expression. We follow the same procedure as before, which is depicted in Figure 7.

The equivalent regular expression is

$$\epsilon \cup ((a \cup b)a^*b)(b \cup a(a \cup b))a^*b)^*(\epsilon \cup a)\,.$$

1.17 b). $A_2 = \{www | w \in a, b*\}$
Proof: Same as Example 1.40 on Pg. 81 of the Sipser book. ▌

c). $A_3 = \{a^{2^n} | n \geq 0\}$. (Here, a^{2^n} means a string of 2^n a's)
Proof: Assume to the contrary that A_3 is regular. By the Pumping Lemma, any string in it longer than the pumping length should be pumpable. Let the pumping length be p. We choose $a^{2^p} \in A_3$ as the string that we will pump. Let $w = xyz = a^{2^p}$. By condition 3 of the lemma, $|xy| \leq p$. This means that the pumping part, which has to be non-zero in length cannot be of length greater than p.

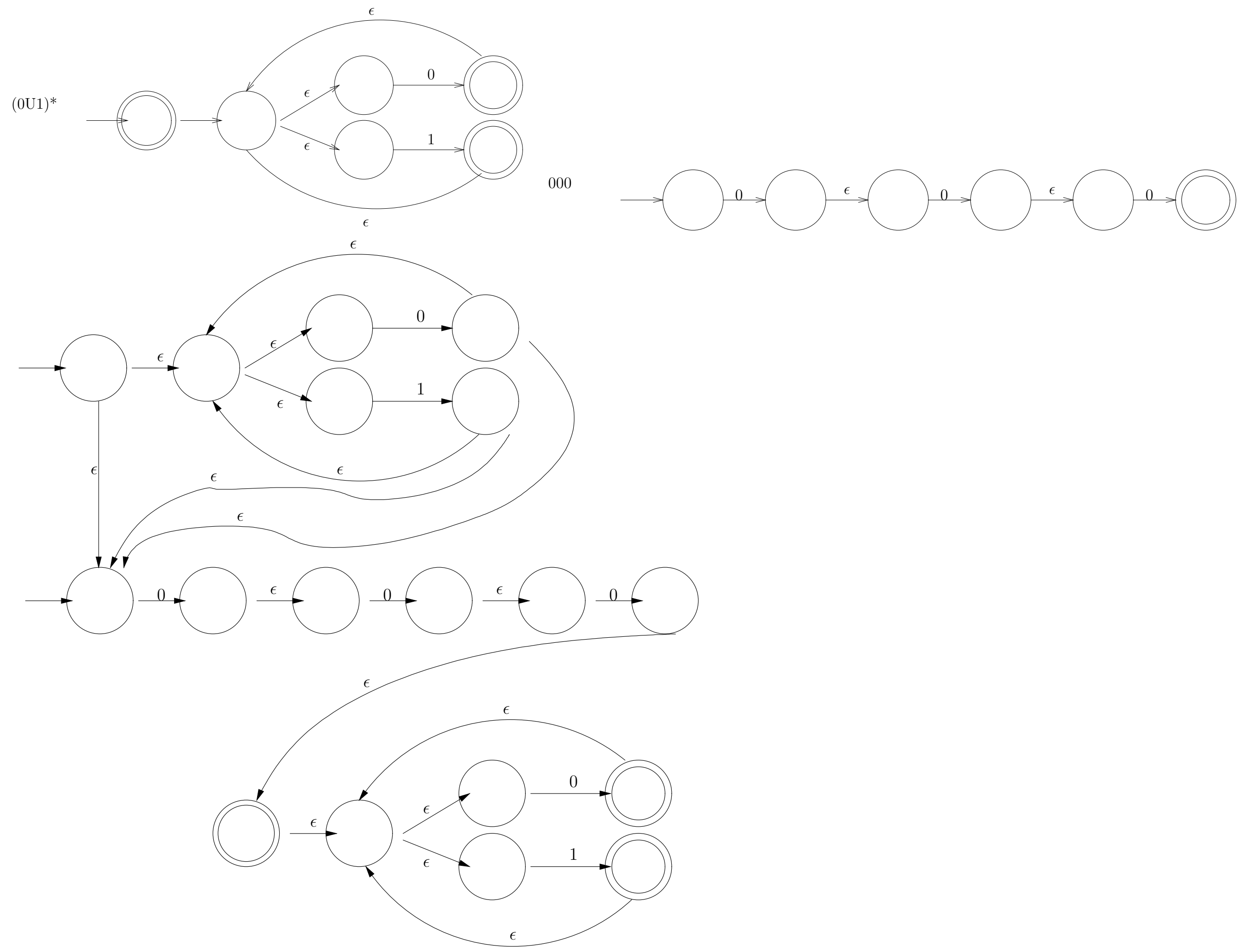


Figure 3: Problem 1.14 (a) NFA from following steps in Theorem 1.28

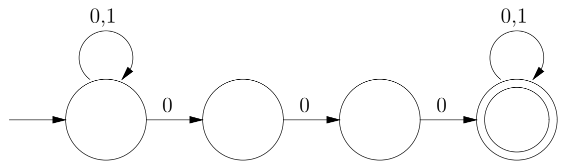


Figure 4: Problem 1.14 (a) Simpler NFA by inspection

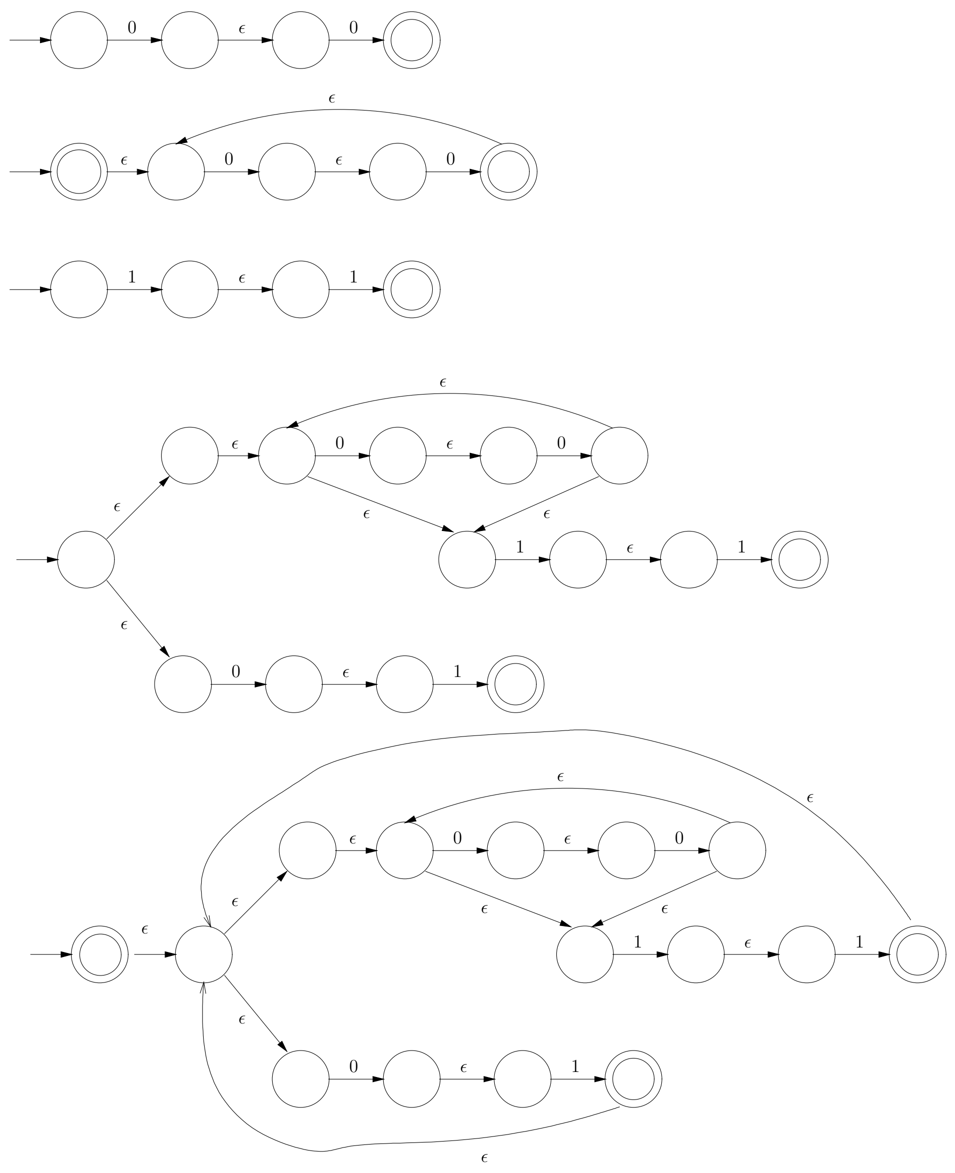


Figure 5: Problem 1.14 (b)

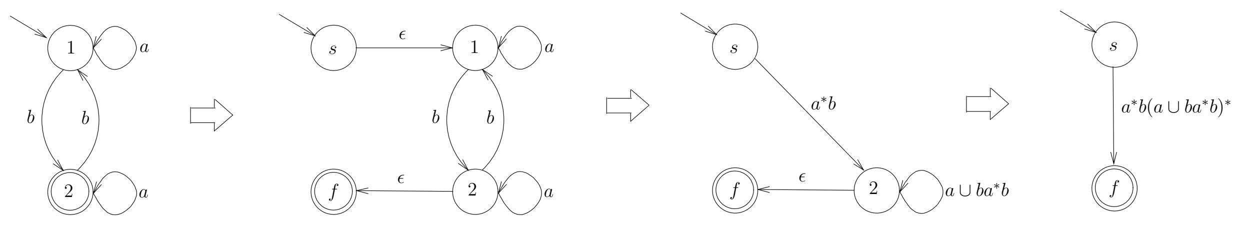
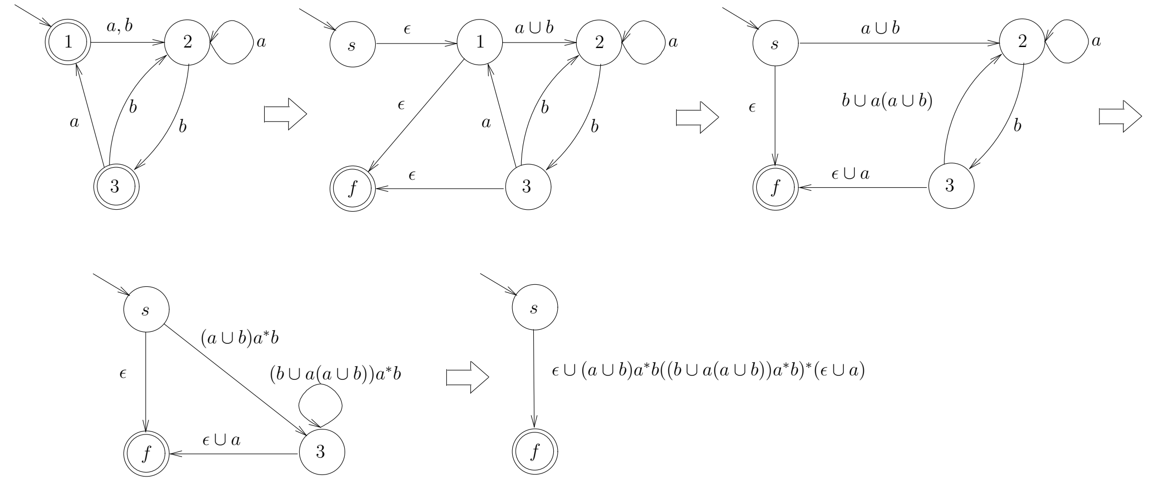


Figure 6: Problem 1.16 (a). Transforming a NFA into the equivalent RE. First, convert the original NFA (first graph) into a GNFA (second graph); then remove state 1 (third graph) and state 2 (last graph) until only states s and f are left.



1
2
3
a, b
a
b
s
f
ϵ
$a \cup b$
$b \cup a(a \cup b)$
$\epsilon \cup a$
$(a \cup b)a^*b$
$(b \cup a(a \cup b))a^*b$
$\epsilon \cup (a \cup b)a^*b((b \cup a(a \cup b))a^*b)^*(\epsilon \cup a)$

Figure 7: Problem 1.16 (b) Transforming a NFA into the equivalent RE. First, convert the given NFA (upper left) into a GNFA (upper middle); then in sequence remove state 1 (upper right), 2 (lower left), and 3 (lower right) until only states s and f are left.

Let the length of this part be q s.t. $0 \leq q \leq p$. By Clause 1 of the Pumping Lemma: $\forall k \in N, \exists n \in N$ s.t. $2^n = 2^p + kq$
Let us consider the case where $k = 1$. By the Pumping Lemma, there must be an n s.t. $2^n = 2^p + q$ where $0 < q \leq p$.
Now we know that for $\forall m \in N$, $2^m + m < 2^{m+1}$. Since $q < p$, it follows that $2^p + q < 2^{p+1}$. Hence it is not possible that the string we get by one round of pumping be a member of A_3. That is, there is a "long enough" string in A_3 that cannot be pumped. Hence A_3 is not regular.
In general, languages that involve more than "linear" growth are never regular. ▌

1.23 c). Consider $L = \{\, a^n b^m \,:\, n \neq m \,\}$. Prove this language is not regular by using the Pumping Lemma.

Proof: Assume by contradiction that L is regular. Then, by PL, we know that there exists a pumping length $p > 0$ such that any word $w \in L$, $|w| \geq p$ can be partitioned as $xyz = w$ (with $|xy| \leq p$ and $|y| > 0$) in such a way that, for any $i \geq 0$, the word xy^iz also belongs to L.

Notice that reaching a contradiction may be tricky in this case, since it's not enough to show that *there exists* a partition $xyz = w$ such that w cannot be "pumped" (recall that by "pumping a word $w = xyz$" we mean considering words of the form xy^iz for $i \geq 0$). Instead, we need to exhibit a word $w \in L$ such that, *any* possible partition $xyz = w$, w cannot be pumped without falling out language L.

Most choices of words do not work since they can be pumped. *Magically*, (hopefully, by the end of the problem you'll see why) we use the word $0^p 1^{p+p!}$, where $p! = p \cdot (p-1) \cdots 2 \cdot 1$.

Clearly $|w| \geq p$. Moreover, any partition of w into xyz must be such that y comprises only $0's$ (since $|xy| \leq p$). Then, it must be the case that $y = 0^k$ for some k $0 < k \leq p$ (since $|y| > 0$). Now, we consider the word $w' = xy^iz$, for some $i \geq 0$ which we leave unspecified for now.

The word w' equals $xy^iz = 0^{p+(i-1)k}1^{p+p!}$. We want to prove that *for any value of* k (that is, any possible y and thus, any possible partition) *there exists a value of* $i \geq 0$ *which causes* w' *to have the same number of* $0's$ *and* $1's$: $n = p + (i-1)k = p + p! = m$. This contradicts the condition $n \neq m$ for words in L.

Indeed, by solving $p + (i-1)k = p + p!$ we get $i = \frac{p!}{k} + 1$. So, for any value of k (recall that $0 < k \leq p$) $\frac{p!}{k} + 1$ will be a positive integer and thus, there exists a value $i \geq 0$ (namely $i = \frac{p!}{k} + 1$) such that $w' \notin L$. Nevertheless, by PL, $w' \in L$. We've got a contradiction.

<u>Alternative solution:</u>

We can also show that this language is not regular by using closure properties of regular languages.

By contradiction, assume L is regular. Then $\overline{L}$ is also regular. Let's see how $\overline{L}$ looks like: $\overline{L}$ is the language of all the words that either (a) are of the form a^nb^m, where $n = m$, or (b) *contain* the substring ba (which is not allowed in L). Therefore, $\overline{L} = \{\, a^nb^n \,:\, n \geq 0 \,\} \cup \{\, w \,:\, w = (a \cup b)^*ba(a \cup b)^* \,\}$. And thus,

$$\overline{L} \setminus \{\, w \,:\, w = (a \cup b)^*ba(a \cup b)^* \,\} = \{\, a^nb^n \,:\, n \geq 0 \,\}.$$

The left hand side of the last expression is regular because the *set-minus* operation is regular (recall that $A \setminus B = A \cap \overline{B}$). However, the right hand side is not. We've got a contradiction. ▮

d). Show that the language $L = \{w \mid w \in \{0,1\}^* \text{ is not palindrome }\}$.

Proof: Recall that a word w is palindrome if $w = w^R$, where w^R is the word formed by reversing the symbols in w (eg. if $w = 010111$, then $w^R = 111010$). For example $w = 0100110010$ is palindrome.

As always, we prove it by contradiction. Assume L is regular. Then the language complement of L, say $L' = \overline{L} = \{w \mid w \in \{0,1\}^* \text{ is palindrome }\}$ is also regular.

If L' is regular, then by pumping lemma, there exits an integer $p > 0$ (the pumping length) such that any word $w \in L'$, $|w| \geq p$ can be partitioned as $xyz = w$ (with $|xy| \leq p$ and $|y| > 0$) in such a way that, for any $i \geq 0$, the word xy^iz also belongs to L'.

However, consider the word $w = 0^p10^p$. Clearly, $|w| = 2p + 1$ so it satisfies the condition $|w| \geq p$ of the theorem. However, any valid partition of w into words x, y, z (valid in the sense that $w = xyz$, $|y| > 0$, and $|xy| \leq p$) must be such that $y = 0^k$ for some integer $k > 0$, (since $|xy| \leq p$, string y can't contain the middle 1) and then $x = 0^{p-k}$, and $z = 10^p$. If we form the word $w' = xy^0z$ (that is, take $i = 0$). then $w' = 0^{p-k}\epsilon 10^p = 0^{p-k}10^p$ which cannot be palindrome since $p - k < p$. Thus, $w' \notin L'$, which contradicts the result of the pumping lemma. ▮

10. Give a DFA (if one exists) for the following languages:

1). $L = \{w \in \{0,1\}^* | w \text{ is a multiple of } 5\}$. The corresponding DFA M is shown in Figure 8. See also remark below.

2). $L = \{w \in \{0,1\}^* | w \text{ is a power of } 2\}$. The corresponding DFA M is shown in Figure 9.

3). $L = \{w \in \{0,1\}^* | w \text{ is a power of } 3\}$. It's likely that no DFA exists for this language. Can you prove L is not regular?

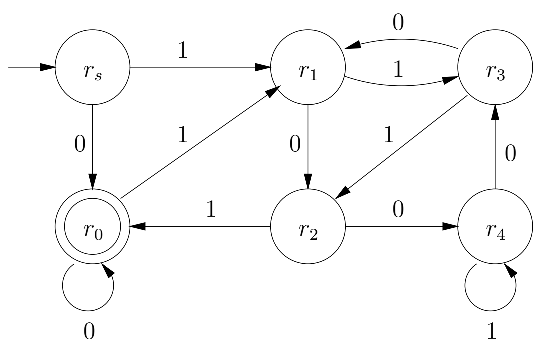


Figure 8: NFA M for problem 10.1

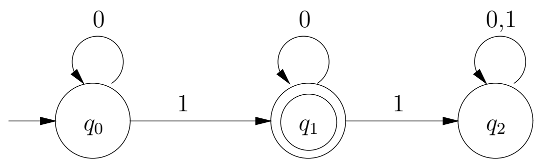


Figure 9: NFA M for problem 10.2

Remark 1 How was problem 10.1 **solved?**: The solution relies on the following fact: If $w \in \{0,1\}^*$ is a string that represent a number n in binary, then

1. "$w0$" represents the number $2n$ (in binary), and
2. "$w1$" represents the number $2n+1$ (in binary).

We want a DFA that recognizes all strings w that represent (in binary) a number n multiple of 5. That is, there is an integer $k > 0$ such that $n = 5k$.

Clearly, for ANY string w there are 5 possible cases, each one which can further divided into two cases:

1. **Case (0)**: w represents the number $n = 5k$.

   If k is even: there is an integer q such that $k = 2q$. Plugging this into the above relation, we get $n = 5(2q) = 2(5q)$. Notice this means w (which represent number n) is of the form $w'0$ for some other string w' which represents the number $5q$ (also multiple of 5).

   Similarly, if k is odd: $k = 2q+1$ means $n = 5(2a+1) = 2(5q+2)+1$ which means w is of the form $w'1$ for some string w' which represents the number $5q+2$.

2. **Case (1)**: w represents the number $n = 5k+1$.

   As before, if $k = 2q$, then $n = 5(2q)+1 = 2(5q)+1$, so $w = w'1$ for w' representing a number equal to $5q$. If $k = 2q+1$, then $n = 5(2q+1)+1 = 2(5q+3)$, so $w = w'0$ for w' representing $5q+3$.

3. **Case (2)**: w represents the number $n = 5k+2$.

   As before, if $k = 2q$, then $n = 5(2q)+2 = 2(5q+1)$, so $w = w'0$ for w' representing a number equal to $5q+1$. If $k = 2q+1$, then $n = 5(2q+1)+2 = 2(5q+3)+1$, so $w = w'1$ for w' representing $5q+3$.

4. **Case (3)**: w represents the number $n = 5k + 3$.

   As before, if $k = 2q$, then $n = 5(2q) + 3 = 2(5q + 1) + 1$, so $w = w'1$ for w' representing a number equal to $5q + 1$. If $k = 2q + 1$, then $n = 5(2q + 1) + 3 = 2(5q + 4)$, so $w = w'0$ for w' representing $5q + 4$.

5. **Case (4)**: w represents the number $n = 5k + 4$.

   As before, if $k = 2q$, then $n = 5(2q) + 4 = 2(5q + 2)$, so $w = w'0$ for w' representing a number equal to $5q+2$. If $k = 2q+1$, then $n = 5(2q+1)+4 = 2(5q+4)+1$, so $w = w'1$ for w' representing $5q + 4$.

Now, we will give a DFA that recognizes strings w multiples of 5 but when **the string is read from right to left** as follows.

Consider states r_0, r_1, r_2, r_3 and r_4. Each state r_k will represent the information "we are in case (**k**)". For example, start in state r_0 (meaning case (**0**)), so assume input w represents the number $5k$. If we read a 0[1] on the input, that case tells us that what follows (to the left) should be a string w' encoding a number equal to $5q$ for some q. Such number corresponds to the same case, so we "stay" in state r_0.

Similarly, if we are in state r_0 (case (**0**)) w represents number $5q$ and we read a 1, it means the remaining string (to the left) represents number $5q + 2$, so we move to state r_2 (which deals with the case (**2**)).

If we follow the same idea for each state (case) we obtain a DFA that will recognize multiples of 5 but when the input is read backwards (from right to left). To obtain the right DFA, we just reverse the directions of the arrows and we add a state r_s which on input 0 goes to state r_0 (since number 0 is a multiple of 5 anyway) and on input 1 goes to state r_1 (that's what we do when we read the first 1 on state r_0). You can check that the resulting DFA is the one shown in Figure 8.

[1]Remember that, since we are reading from right to left, this 0 is the last symbol in string w!