Homework 2 solutions

1. (10 + 5 = 15 points)
(i) Define Rev(L) = $\{x^R \mid x \in L\}$. Note x^R is the reversal of the string x.
Thus, e.g., if L = {ab, aba, babb} then Rev(L) = {ba, aba, bbab}.

Let L_1 be a regular language that is recognized by a DFA $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$.
Provide a construction (be precise) to create a DFA or an NFA to accept Rev(L_1).

Intuitively, the idea is to 'reverse' the starting and final states as well as all the transitions in M_1 to generate an NFA N_2 which accepts Rev(L_1).

The reversal of the transitions causes $\Delta_2(p, a) = \{ q \mid \delta_1(q, a) = p \}$ for each state $p \in Q_1$ (note that $\Delta_2(p, a)$ can be a set of states).

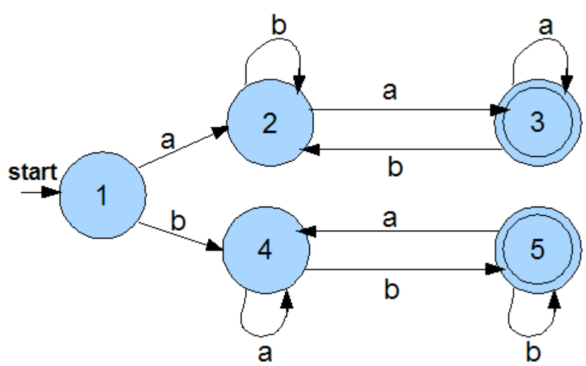
In order to have a single 'Start State' in N_2, a 'Start State' is added with ϵ-transitions to each state in F_1 such that Δ_2('Start State', ϵ) = F_1.

Ultimately, the construction of the NFA N_2 that will accept Rev(L_1) is as follows:
$N_2 = (Q_2, \Sigma, \Delta_2, s_2, F_2)$ where:

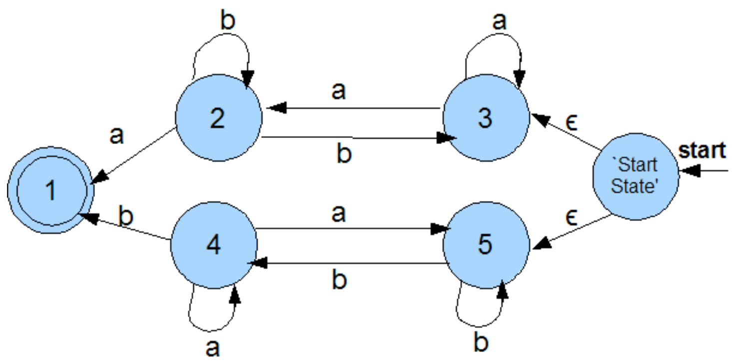
$Q_2 = Q_1$ U 'Start State'
$\Sigma = \Sigma$
Δ_2: as described above
s_2 = 'Start State' (which has ϵ transitions to each state in F_1)
$F_2 = \{s_1\}$

(ii) Apply your construction to the DFA given below.

Input DFA M_1 which accepts L:



NFA N_2 that accepts Rev(L) where L is accepted by the input DFA M_1:



2. (10 X 3 = 30 points)
Give regular expressions that denote the following sets:

a. The set of strings where the third last symbol is an ‘a’. Here, $\Sigma = \{a, b\}$.

Strings in this set start with any number ≥ 0 of a’s or b’s in any order; this is represented by (a | b)*.
Then, the third to last symbol in the string must be an ‘a’, represented by ‘a’.
Finally, the last two symbols can be either a or b, represented by (a | b)(a | b).

Resulting regular expression: (a | b)*a(a | b)(a | b)

b. The set of strings of the form w_1cw_2 where w_1 and w_2 belong to $\{a, b\}^*$ and w_2 contains a ‘b’ if and only if w_1 has at least 2 occurrences of ‘a’.

First, look at the case where w_1 has at least 2 occurrences of ‘a’, so w_2 contains a ‘b’:

w_1 with at least 2 occurrences of ‘a’ begins with any number ≥ 0 of a’s or b’s in any order;
this is represented in a regular expression by (a | b)*.
Then ‘must’ contain an ‘a’, which is represented as an ‘a’ in a regular expression.
Then there is another set of any number ≥ 0 of a’s or b’s in any order,
another ‘a’, and finally a third set of any number ≥ 0 of a’s or b’s in any order.

The regular expression for w_1 here is (a | b)*a(a | b)*a(a | b)*.

Next, w_2 with a ‘b’ begins with any number ≥ 0 of a’s or b’s in any order,
then it ‘must’ contain an ‘b’, then concludes another set of any number ≥ 0 of a’s or b’s.

The regular expression for w_2 here is (a | b)*b(a | b)*.

The regular expression for the ‘c’ between w_1 and w_2 is simply ‘c’.

Therefore, the complete regular expression for this ‘first’ case is
(a | b)*a(a | b)*a(a | b)*c(a | b)*b(a | b)*).

Next, look at the case where w_1 has ≤ 2 occurrences of ‘a’, so w_2 does not contain any b’s:

w_1 with ≤ 2 occurrences of ‘a’ begins with any number ≥ 0 of b’s;
this is represented in a regular expression by b*. Next, it can contain an ‘a’
but doesn’t have to; this is represented by (a | ϵ) in a regular
expression, with ϵ corresponding to an empty string. w_1 in this case
concludes with ≥ 0 b’s, represented by b*.

The regular expression for w_1 here is b*(a | ϵ)b*.

w_2 without ‘b’ simply contains any number ≥ 0 of a’s, represented by a* in a regular expression.

The regular expression for the ‘c’ between w_1 and w_2 is simply ‘c’.

Therefore, the complete regular expression for this ‘second’ case is
b*(a | ϵ)b*ca*.

The final regular expression is simply a disjunction of the two cases, resulting in:
((a | b)*a(a | b)*a(a | b)*c(a | b)*b(a | b)*) | (b*(a | ϵ)b*ca*)

c. The set of strings that do not contain the substring ‘ab’. Here, $\Sigma = \{a, b\}$.

A string without the substring ’ab’ cannot contain any ‘b’ once there has been an ‘a’, resulting in the regular expression ‘b*a*’ where the string can contain any number of b’s followed by any number of a’s.

Resulting regular expression: b*a*

3. (10 X 3 = 30 points) Use the pumping lemma to show that the following sets are not regular.

a. Set A = $\{a^l b^m c^n \mid l{=}100,\ m > l,\ n > m\}$

Given the ‘demon’s’ k value...(see p. 71 of book)
Let xyz = $a^{100}b^{k+100}c^{k+100+1} \in$ set A
Let x = a^{100}, y = b^{k+100}, z = $c^{k+100+1}$
Now v in y = uvw must contain at least 1 ‘b’ since y only contains b’s and v cannot be empty
Note that m = k + 100 and n = k + 100 + 1, so n = m + 1 when i=1.
Now let i = 2.
The presence of i=2 forces the addition if at least 1 ‘b’ to y,

so now m $\geq$ (k + 100 + 1) $\rightarrow$ m $\geq$ n.
Therefore, it is no longer the case that n > m.
And therefore, $xuv^iwz \notin$ set A when i = 2
Therefore, the set is not regular.

b. The set A of strings of the form ww where w $\in$ {a, b}*.
Thus abbaabba is included in this set but abba is not.

Given the 'demon's' k value...(see p. 71 of book)
Let xyz = $a^kb^ka^kb^k \in$ set A
Let x = $a^kb^ka^k$, y = b^k, z = ϵ
Now v in y = uvw must contain at least 1 'b' since y only contains b's and v cannot be empty.
Now when i = 2, $uv^2w = b^j$ where j > k
Since $a^kb^ka^kb^j \notin$ language when j > k, $xuv^iwz \notin$ set A when i = 2.
Therefore the set is not regular.

c. The set A of strings of the form $a^{n1}b^{n2}c^{n3}d^{n4}$, where n1 = n3 or n2 = n4. (n1, n2, n3, n4 $\geq$ 1)

Given the 'demon's' k value...(see p. 71 of book)
Let xyz = $a^kb^{k+1}c^kd^{k+2} \in$ set A
Let x = ϵ, y = a^k, z = $b^{k+1}c^kd^{k+2}$
Now v in y = uvw must contain at least 1 'a' since y only contains a's and v cannot be empty...
Now when i = 2, y = $uv^2w = a^j$ where j > k
Since $a^jb^{k+1}c^kd^{k+2} \notin$ set A when j > k, $xuv^iwz \notin$ set A when i = 2.
Therefore, the set A is not regular.

4. (15 points) Exercise 47 (Parts a and b only) on Page 326 of the textbook.

Minimize the following DFAs. Indicate clearly which equivslence class corresponds to each state of the new automaton.

Part a:

	a	b
1	6	3
2	5	6
3F	4	5
4F	3	2
5	2	1
6	1	4

Using algorithm in book (see p. 84) for computing the collapsing relation $\approx$

for a given DFA M with no inaccessible states. The algorithms marks (unordered) pairs of states {p, q}, where a pair {p, q} is marked as a reason is discovered as to why p and q are NOT equivalent.

1					
-	2				
-	-	3			
-	-	-	4		
-	-	-	-	5	
-	-	-	-	-	6

Pass 1: mark all pairs consisting of once accept state and one nonaccept state.

1					
-	2				
X	X	3			
X	X	-	4		
-	-	X	X	5	
-	-	X	X	-	6

Repeat until no changes: if there is an unmarked pair {p, q} such that $\{\delta(p, a), \delta(q, a)\}$ is marked for some $a \in \Sigma$, then mark {p, q}.

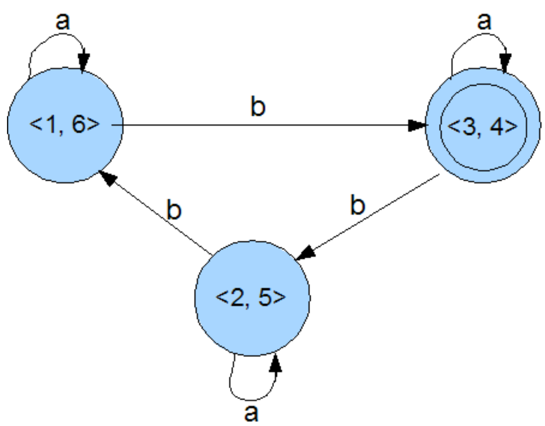
Mark {1, 2} since it goes to marked pair {3, 6} on input 'b'
Mark {5, 1} since it goes to marked pair {1, 3} on input 'b'
Mark {6, 2} since it goes to marked pair {5, 1} on input 'a'
Mark {6, 5} since it goes to marked pair {1, 2} on input 'a'

Table now looks as follows; no more values can be marked...

1					
X	2				
X	X	3			
X	X	-	4		
X	-	X	X	5	
-	X	X	X	X	6

Therefore, it is the case that state 1 $\approx$ state 6, state 2 $\approx$ state 5, and state 3 $\approx$ state 4.

Here is the resulting minimized DFA:



Part b:

	a	b
1	2	3
2	5	6
3F	1	4
4F	6	3
5	2	1
6	5	4

Again using algorithm in book (see p. 84) for computing the collapsing relation $\approx$ for a given DFA M with no inaccessible states. The algorithms marks (unordered) pairs of states {p, q}, where a pair {p, q} is marked as a reason is discovered as to why p and q are NOT equivalent.

1					
-	2				
-	-	3			
-	-	-	4		
-	-	-	-	5	
-	-	-	-	-	6

Pass 1: mark all pairs consisting of once accept state and one nonaccept state.

1					
-	2				
X	X	3			
X	X	-	4		
-	-	X	X	5	
-	-	X	X	-	6

Repeat until no changes: if there is an unmarked pair $\{p, q\}$ such that $\{\delta(p, a), \delta(q, a)\}$ is marked for some $a \in \Sigma$, then mark $\{p, q\}$.

Mark $\{1, 2\}$ since it goes to marked pair $\{3, 6\}$ on input 'b'
Mark $\{5, 1\}$ since it goes to marked pair $\{1, 3\}$ on input 'b'
Mark $\{6, 2\}$ since it goes to marked pair $\{5, 1\}$ on input 'a'
Mark $\{6, 5\}$ since it goes to marked pair $\{1, 4\}$ on input 'a'

Table now looks as follows; no more values can be marked...

1					
X	2				
X	X	3			
X	X	-	4		
X	-	X	X	5	
-	X	X	X	X	6

Therefore, it is the case that state 1 $\approx$ state 6, state 2 $\approx$ state 5, and state 3 $\approx$ state 4.

Here is the resulting minimized DFA: