COMP 330 Autumn 2014
Mid-term Examination Solutions and Comments

Prakash Panangaden
McGill University

October 14, 2014

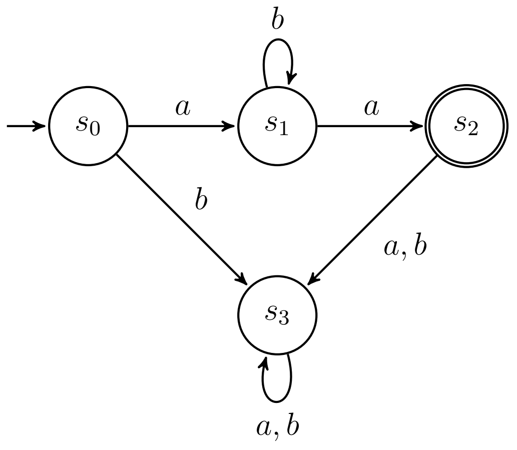
Question 1 [30 points] Let L be the following language defined over the alphabet $\Sigma = \{a, b\}$: L consists of words that start with an a, end with an a, have no other a's and have length *at least* two.

1 Give a regular expression for L. You need not prove that it is correct.

- **Solution**: ab^*a, there cannot be any other a's.

2 Construct a DFA recognizing L; show all states, including dead states explicitly. Your DFA should be designed to have as few states as possible. You do not have to prove that your machine is minimal.

- **Solution**



3 Show that any DFA to recognize L must have at least 4 states. This includes the dead states if any.

- **Solution** Consider the following 4 strings: ε, a, b, aa. First note that aa is accepted and the other three are rejected so aa must take the machine to a state different from any of the other three. Now consider a and b: they cannot take the initial state to the same state because aa is accepted but ab is not. Now ε of course leaves the machine in the start state. This state cannot be the state reached by a since $\varepsilon \cdot a = a$ is rejected but $a \cdot a = aa$ is accepted. Finally b cannot leave you in the start state because $b \cdot aa = baa$ is rejected but $\varepsilon \cdot aa = aa$ is accepted. Thus we must have at leave 4 distinct states. Comment: Minimizing the automaton above is a long-winded way of stating what I have just said. I was OK with people carrying out the minimization, even though it is a rather clumsy way to get to the solution. I was **not** happy with people just stating that they had minimized the machine without showing me that they had done it.

4 Write down a regular expression for the complement of L.

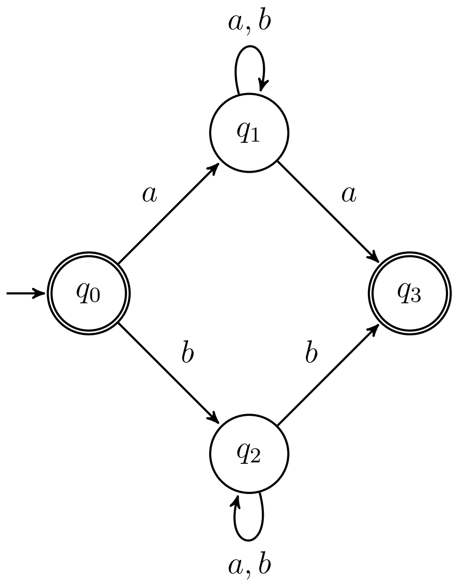
- **Solution** Since the machine is a DFA one can just flip the accept and reject states. Then one can read off the regular expression. There are many possible ways to write it, here is one possible answer: $\varepsilon + ab^* + b(a+b)^* + ab^*a(a+b)(a+b)^*$. Comment some of you wrote answers that were too complicated for us to check. We gave you zero, but if you come and convince us that your answer is correct we will give you your 5 points.

Question 2 [25 points] Is the language

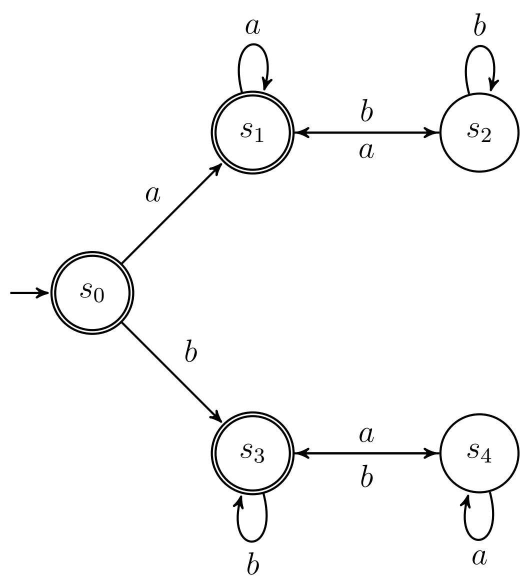
$$L = \{w \in \Sigma^* \mid w \text{ has an equal number of occurrences of } ab \text{ and } ba\}$$

regular? The alphabet is $\Sigma = \{a, b\}$. If you think it is regular, give a finite-state recognizer for it (DFA, NFA or NFA $+\varepsilon$ as you prefer) to recognize L. If you think that it is not regular, give a proof using the pumping lemma.

Solution This language *is* regular. You need to keep track of how many times you switch between a block of a's and a block of b's. Alternatively, you can observe that the word must start and end with the same letter. Here is an NFA for recognizing the language.



Here is a DFA based on keeping track of the transitions between blocks of a's and blocks of b's.



Question 3 [25 points] We fix the alphabet to be $\Sigma = \{a, b\}$. The language L is defined as follows: it consists of words where the number of occurrences of aa and bb are equal. We count overlapping occurrences as separate. For example, the word $aaabbaaba$ has three occurrences of aa and one of bb: this word is not in L. An example of a word that is in L is $aabbaabbbaa$. Show, using the pumping lemma that this language is not regular.

Solution Let the demon choose a number p. I choose the word $a^p b^p$. This is in the language since it has $(p-1)$ occurrences of aa and of bb. Now the demon is forced to choose $y = a^k$ where $0 < k \leq p$. I choose $i = 2$ and now there are more aa's than bb's so the string is not in the language. Comment Some of you were penalized heavily for not choosing your starting string in the language.

Question 4 [20 points] I am given two regular expressions R_1 and R_2; let the languages they describe be L_1 and L_2 respectively. Give an algorithm to decide whether $L_1 \bigcup L_2 = \Sigma^*$. You may use standard algorithms and constructions as building blocks. For example, you can just say "test for reachability" without describing an explicit algorithm to do so or "minimize the DFA" or "determinize the NFA" without explaining further.

Solution. From the regular expressions we can directly construct NFAs N_1 and N_2 respectively as in the proof of Kleene's theorem. Then we can construct an NFA to recognize the union of the two languages. Now we can determinize the NFA to get a DFA. Finally we check if any non-accept state is reachable: if there is such a reachable state the union does not equal Σ^*. Comment There is no reason to determinize the two initial NFAs before forming the union. It is not OK to just say "construct a DFA from the regular expressions" that it a two-step process. You cannot just test the NFA for reachability nor can you get the machine for the complement language by flipping the reject and accept states in an NFA.