Exercises on Regular Expressions and Finite Automata

With solutions

Throughout this exercise sheet, assume that the alphabet is $\Sigma = \{a, b\}$.

1. Recall that (basic) regular expressions are formed from 0 (matches nothing), 1 (matches just the empty word), alternation ($r + s$), concatenation (rs), and iteration (r^*). Write regular expressions that accept exactly the following languages:

   (a) words that consist of an even number of a's followed by an odd number of b's,
   Solution: $(aa)^*b(bb)^*$.

   (b) words of an even length,
   Solution: $(aa + ab + ba + bb)^*$.

   (c) words that contain exactly three a's,
   Solution: $b^*ab^*ab^*ab^*$.

   (d) words that contain no more than three a's,
   Solution: $b^*(1 + a)b^*(1 + a)b^*(1 + a)b^*$.

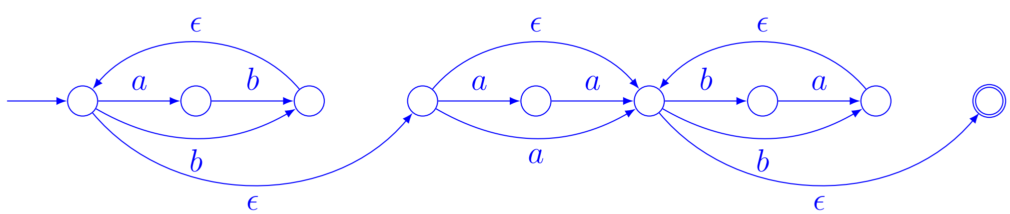
   (e) words that do not contain two consecutive a's,
   Solution: $((a + 1)b)^*(a + 1)$.

   (f) words where every a is immediately followed by a b, and
   Solution: $((a + 1)b)^*$.

   (g) words that do not end with ba.
   Solution: $(a + b)^*(b + aa) + a + 1$.

2. Using Thompson's algorithm or otherwise, convert the regular expression $(b + ab)^*(1 + a + aa)(b + ba)^*$ into an NFA.

   Solution:

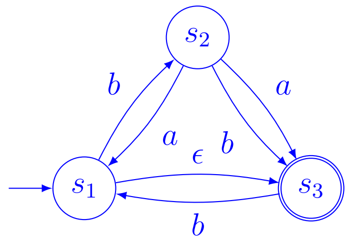


   NB: I've removed a few of the clearly-unnecessary ϵ-transitions that Thompson's construction introduces. But mostly I've left the ϵ-transitions in, because it is easy to introduce mistakes when removing them!

3. Consider an NFA with three states, named s_1, s_2, and s_3. The start state is s_1 and the accepting state is s_3. The a-transitions are $s_2 \to s_1$ and $s_2 \to s_3$. The b-transitions are $s_1 \to s_2$, $s_2 \to s_3$, and $s_3 \to s_1$. There is an ϵ-transition from s_1 to s_3.

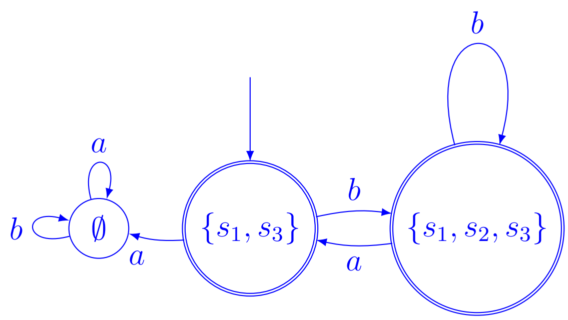
   (a) Draw this NFA.

   Solution:



   (b) Using the subset construction or otherwise, convert this NFA into a DFA.

   Solution:



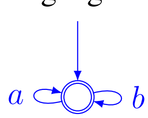
   (c) DFAs with fewer states can be more efficiently simulated. Can you simplify your DFA into a DFA that accepts the same language but requires fewer states?

   Solution: No, this DFA can't be simplified.

   (d) Describe (in simple English) the language that your DFA accepts.

   Solution: As a regex: $(b(1 + a))^*$. Or in English: all words where every two a's have a b between them.

4. Let us write Σ^* for the language that contains every word made up from characters in the alphabet Σ. This is sometimes called the *universal language over* Σ. Draw a DFA that accepts the language Σ^*.

   Solution: 

   (NB: This DFA corresponds to the regular expression $(a + b)^*$.)

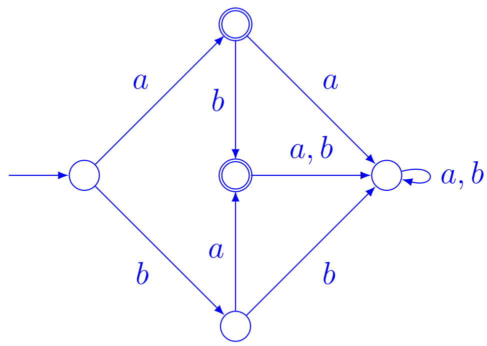
5. Suppose you are given an NFA that accepts a language L_1, and another NFA that accepts a language L_2. Explain how to construct a new NFA that accepts the language $L_1 \cup L_2$ (that is, all words that are in *either* language).

   Solution: First, write out both NFAs side by side. Then make a new state, make it the start-state, and place an ϵ-transitions from it to both of the old start-states. (NB: This is the same as Thompson's construction for the regular expression $r + s$.)

6. For any language L, we shall define its *complement* as $\overline{L} = \Sigma^* \setminus L$; that is, the set of all words that are not in L.

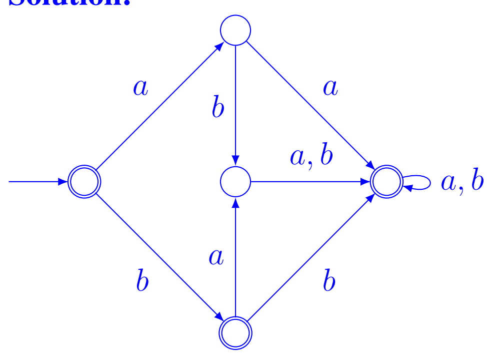
   (a) Draw a DFA that accepts the language $\{a, ba, ab\}$.

   Solution:



   (b) Now draw a DFA that accepts the complement of that language.

   Solution:



   (c) Suppose you are given a DFA that accepts a language L. Explain how to construct a new DFA that accepts the language $\overline{L}$.

   Solution: Just toggle accepting and non-accepting states.

7. Suppose you are given an NFA that accepts a language L_1, and another NFA that accepts a language L_2. Explain how to construct a new NFA that accepts the language $L_1 \cap L_2$ (that is, all words that are in *both* languages). *[Hint: bear De Morgan's law in mind!]*

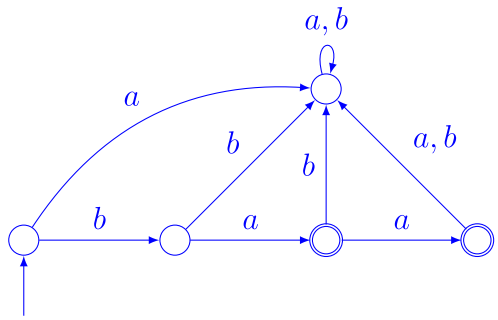
   Solution: Suppose the given NFAs are called A and B. Write $union$ for the union construction on NFAs from part 5. Write $comp$ for the complement construction on DFAs from part 6. Write det for the NFA-to-DFA construction. Then the construction we want is:

$$comp(det(union(comp(det(A)), comp(det(B)))))$$

8. For any language L, we shall write L^{-1} for the *reverse* language of L. L^{-1} accepts the same words as L, but each word has its sequence of characters reversed. For instance, ab is accepted by L^{-1} if and only if ba is accepted by L.

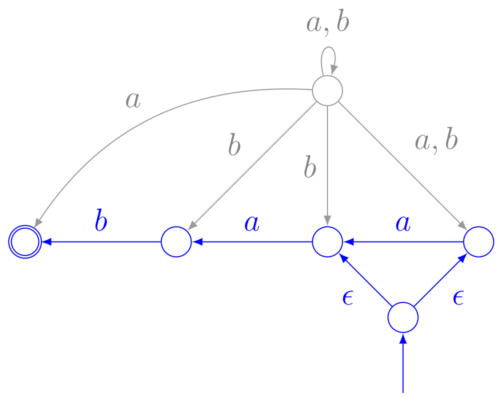
(a) Draw a DFA that accepts $\{ba, baa\}$.

Solution:



(b) Draw an NFA that accepts the reverse language of $\{ba, baa\}$.

Solution:



(NB: The dimmed states aren't reachable so can be removed. I've included them to make the construction more apparent.)

(c) Suppose you are given a DFA that accepts the language L. Explain how to construct an NFA that accepts the language L^{-1}.

Solution: Suppose that the DFA has start state (call it s) and zero or more accepting states (call them a_0, a_1, and so on). Build a new DFA that has the same set of states, but all transitions are reversed. Add a new state, make it the new start-state (instead of s), and place ϵ-transitions from it to each of the original accepting states (a_0, a_1, etc.). Finally, mark states a_0, a_1, etc. as non-accepting states, and mark s as an accepting state.