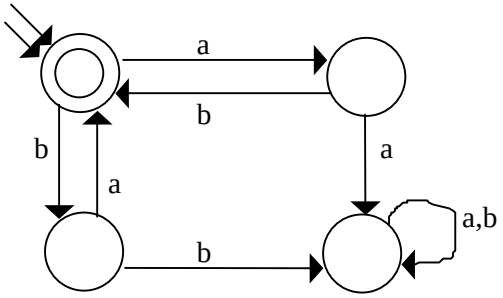
CS 341 Homework 8
Finite Automata, Regular Expressions, and Regular Grammars

1. We showed that the set of finite state machines is closed under complement. To do that, we presented a technique for converting a *deterministic* machine M into a machine M' such that L(M') is the complement of L(M). Why did we insist that M be deterministic? What happens if we interchange the final and nonfinal states of a nondeterministic finite automaton?

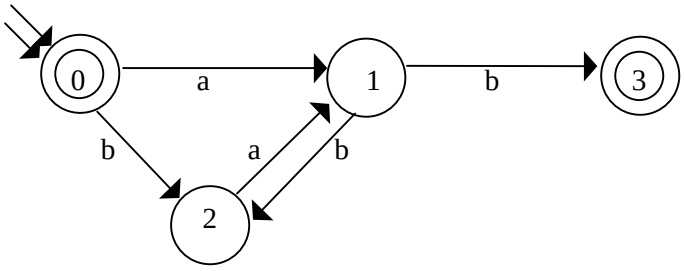
2. Give a direct construction for the closure under intersection of the languages accepted by finite automata. (Hint: Consider an automaton whose set of states is the Cartesian product of the sets of states of the two original automata.) Which of the two constructions, the one given in the text or the one suggested in this problem, is more efficient when the two languages are given in terms of nondeterministic finite automata?

3. Using the either of the construction techniques that we discussed, construct a finite automaton that accepts the language defined by the regular expression: $a^*(ab \cup ba \cup \varepsilon)b^*$.

4. Write a regular expression for the language recognized by the following FSM:



5. Consider the following FSM M:

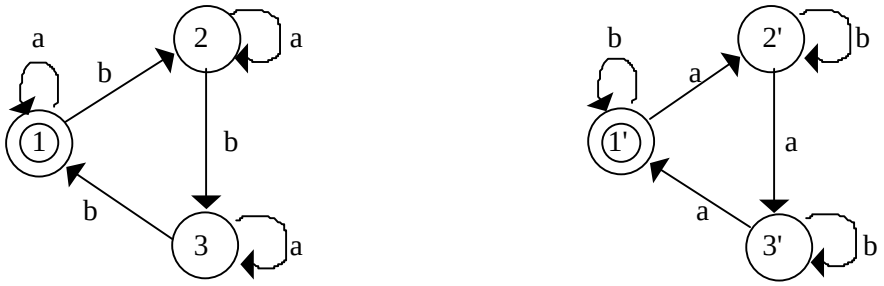


(a) Write a regular expression for the language accepted by M.
(b) Give a deterministic FSM that accepts the complement of the language accepted by M.

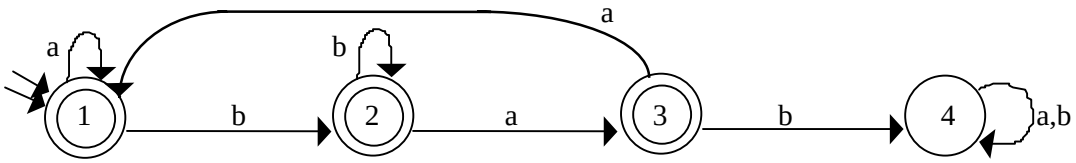
6. Construct a deterministic FSM to accept each of the following languages:
(a) $(aba \cup aabaa)^*$
(b) $(ab)^*(aab)^*$

7. Consider the language $L = \{w \in (a, b)^* : w \text{ has an odd number of a's}\}$
(a) Write a regular grammar for L.
(b) Use that grammar to derive a (possibly nondeterministic) FSA to accept L.

8. Construct a deterministic FSM to accept the intersection of the languages accepted by the following FSMs:



9. Consider the following FSM M:



(a) Give a regular expression forL(M).
(b) Describe L(M) in English.

Solutions

1. We define acceptance for a NDFSA corresponding to the language L as there existing at least one path that gets us to a final state. There can be many other paths that don't, but we ignore them. So, for example, we might accept a string S that gets us to three different states, one of which accepts (which is why we accept the string) and two of which don't (but we don't care). If we simply flip accepting and nonaccepting states to get a machine that represents the complement of L, then we still have to follow all possible paths, so that same string S will get us to one nonaccepting state (the old accepting state), and two accepting states (the two states that previously were nonaccepting but we ignored). Unfortunately, we could ignore the superfluous nonaccepting paths in the machine for L, but now that those same paths have gotten us to accepting states, we can't ignore them, and we'll have to accept S. In other words, we'll accept S as being in the complement of L, even though we also accepted it as being in L. The key is that in a deterministic FSA, a rejecting path actually means reject. Thus it makes sense to flip it and accept if we want the complement of L. In a NDFSA, a rejecting path doesn't actually mean reject. So it doesn't make sense to flip it to an accepting state to accept the complement of L.

2.

Given two DFA's $M_1 = (K_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \delta_2, s_2, F_2)$, we wish to construct a new machine $M = (K, \Sigma, \delta, s, F)$ such that $L(M) = L(M_1) \cap L(M_2)$. (Notice that of course the alphabets of the 3 DFA's will be equal.)

Since the regular languages are closed under union and complementation, and since $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$, closure under intersection is already proved. This direct construction will avoid using the earlier constructions and illustrates a different proof technique.

The hint is to let $K = K_1 \times K_2$. Thus each state of M is really a pair (q_1, q_2) of states from M_1 and M_2. The intuition will be that M simultaneously simulates M_1 and M_2 on a given input string. M will keep track of what states M_1 and M_2 would be in if they were reading the string. These are two independent pieces of data; hence the use of a pair for M's state.

Initially, M_1 and M_2 start in their start states, s_1 and s_2. Therefore we should let $s = (s_1, s_2)$.

Now suppose that M_1 is in some state $q_1 \in K_1$ and reads symbol σ. What state does M_1 enter? $\delta_1(q_1, \sigma)$. Similarly for M_2. So we would like M, when in state (q_1, q_2) and reading σ, to enter state $(\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$; otherwise M would not be correctly keeping track of what M_1 and M_2 would do. So we define, for all $(q_1, q_2) \in K$ and all $\sigma \in \Sigma$,

$$\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)).$$

Notice that $\delta : K \times \Sigma \to K$. so everything is consistent and correct. Since $K = K_1 \times K_2$, this means δ is actually a function taking a pair of states (from M_1 and M_2) and a symbol from Σ.

We've now got the transitions defined, and M correctly simulates M_1 and M_2. I.e.,

$$\delta(s, x) = (q_1, q_2)$$

iff

$$\delta_1(s_1, x) = q_1 \text{ and } \delta_2(s_2, x) = q_2.^1$$

So we only need to define F. When should M accept x? Exactly when both M_1 and M_2 do, since $x \in L(M_1) \cap L(M_2)$ iff $x \in L(M_1)$ and $x \in L(M_2)$. Therefore F should consist of all those states $(q_1, q_2) \in K$ such that $q_1 \in F_1$ and $q_2 \in F_2$. This can be written as

$$F = \{(q_1, q_2) : q_1 \in F_1 \text{ and } q_2 \in F_2\},$$

or more succinctly as $F = F_1 \times F_2$.

Thus the complete answer is

$$M = (K_1 \times K_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$

where

$$\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)).$$

Notice that this assumes M_1 and M_2 are deterministic. What if M_1 and M_2 are not deterministic? We can assume that they are deterministic without loss of generality, because if they were not, the subset construction can be applied to them to produce equivalent DFA's. However, this construction can be modified to work directly on NFA's if desired. Unfortunately, it gets rather messy because of the following problem:

We are given two NFA's $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$, and we wish to construct a new machine $M = (K, \Sigma, \Delta, s, F)$ such that $L(M) = L(M_1) \cap L(M_2)$.

---

[1] Technically, δ is a function of symbols not strings; however we can easily extend it to strings by the recursive generalization:

$$\begin{aligned} \delta(q, e) &= q \\ \delta(q, \sigma x) &= \delta(\delta(q, \sigma), x) \end{aligned}$$

I.e., if it is determined what δ does with a single symbol, then it is determined what δ does with a string simply by tracing through symbol by symbol.

If we do the obvious thing and define

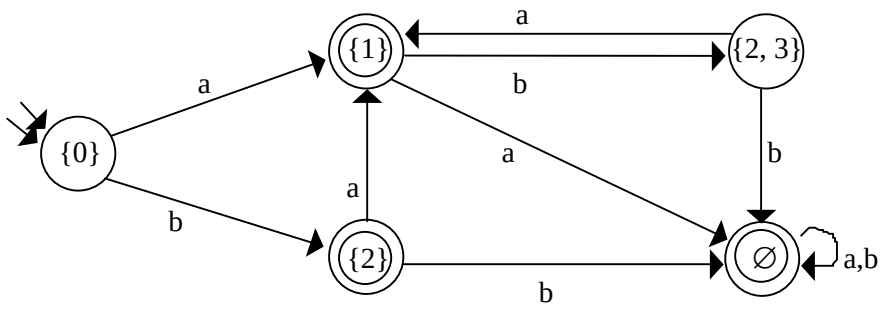
$$\Delta = \{((q_1, q_2), x, (q_1', q_2')) : (q_1, x, q_1') \in \Delta_1 \text{ and } (q_2, x, q_2') \in \Delta_2\},$$

i.e., we make a transition $(q_1, q_2) \xrightarrow{x} (q_1', q_2')$ in M exactly when there are transitions $q_1 \xrightarrow{x} q_1'$ in M_1 and $q_2 \xrightarrow{x} q_2'$ in M_2, then there is trouble. The trouble is that the transitions in an NFA need not read exactly 1 symbol, so M defined this way will be unable to simulate many of moves of M_1 and M_2. E.g., if M_1 has the transition (s_1, aa, q_1) and M_2 has (s_2, a, q_2), you can see that M will have difficulty keeping in synch. So Δ will have to be defined much more cleverly (and complexly). So it's much easier to just assume M_1 and M_2 are deterministic.

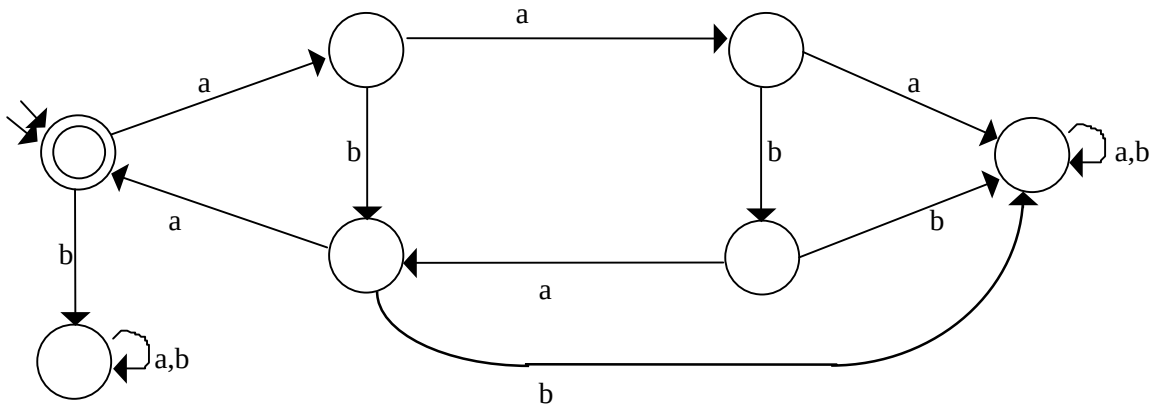
4. Without using the algorithm for finding a regular expression from an FSM, we can note in this case that the lower right state is a dead state, i.e., an absorbing, non-accepting state. We can leave and return to the initial state, the only accepting state, by reading ab along the upper path or by reading ba along the lower path. These can be read any number of times, in any order, so the regular expression is (ab ∪ ba)*. Note that ε is included, as it should be.

5. (a) ε ∪ ((a ∪ ba)(ba)*b)

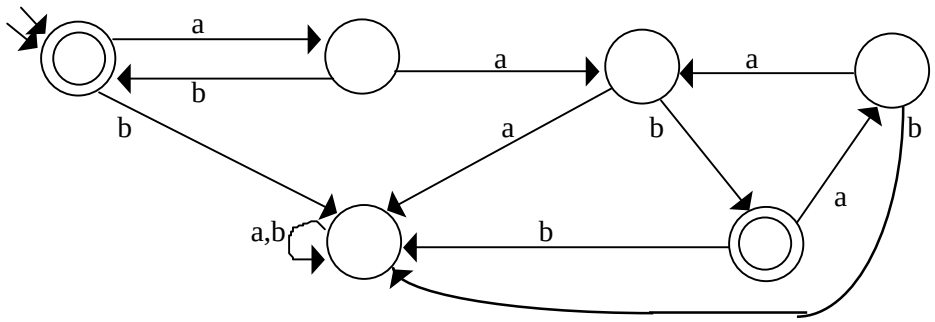
(b)



6. (a)



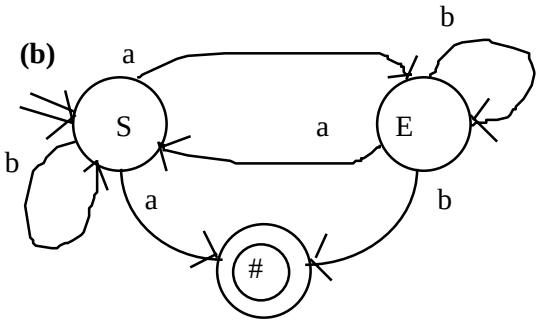
(b)



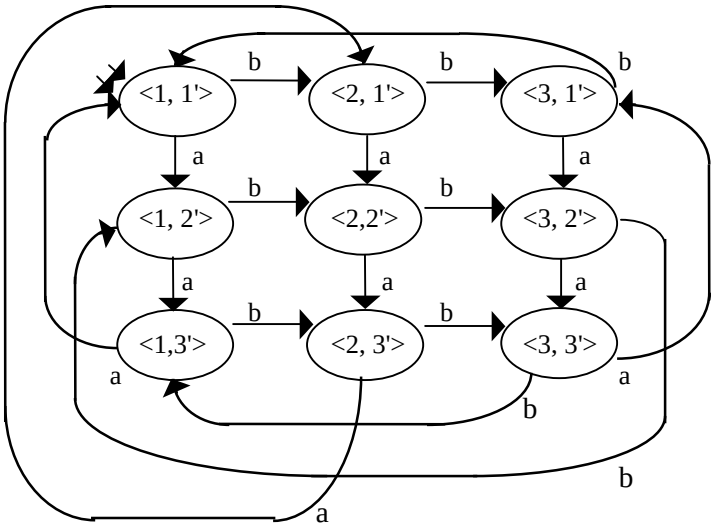
7. (a) Nonterminal S is the starting symbol. We'll use it to generate an odd number of a's. We'll also use the nonterminal E, and it will always generate an even number of a's. So, whenever we generate an a, we must either stop then, or we must generate the nonterminal E to reflect the fact that if we generate any more a's, we must generate an even number of them.

$S \rightarrow a$
$S \rightarrow aE$
$S \rightarrow bS$
$E \rightarrow b$
$E \rightarrow bE$
$E \rightarrow aS$

(b)



8.



9. (a) $(a \cup bb^*aa)^* (\varepsilon \cup bb^*(a \cup \varepsilon))$

(b) All strings in {a, b}* that contain no occurrence of bab.