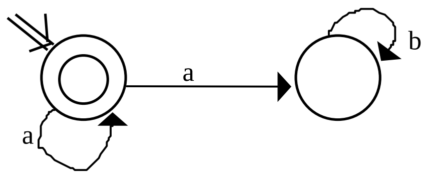
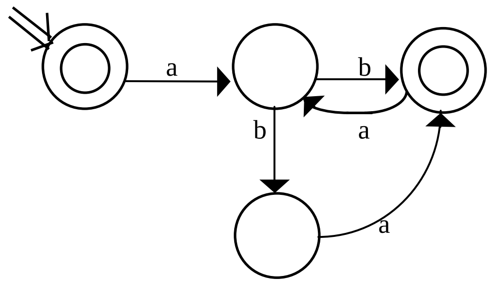
CS 341 Homework 6
Nondeterministic Finite Automata

1. (a) Which of the following strings are accepted by the nondeterministic finite automaton shown on the left below?
(i) a
(ii) aa
(iii) aab
(iv) ε





(b) Which of the following strings are accepted by the nondeterministic finite automaton on the right above?
(i) ε
(ii) ab
(iii) abab
(iv) aba
(v) abaa

2. Write regular expressions for the languages accepted by the nondeterministic finite automata of problem 1.

3. For any FSM F, let |F| be the number of states in F. Let R be the machine shown on the right in problem 1.
Let L = {w ∈ {0, 1}* : ∃M such that M is an FSM, L(M) = L(R), |M| ≥ |R|, and w is the binary encoding of |M|}. Write a regular expression for L.

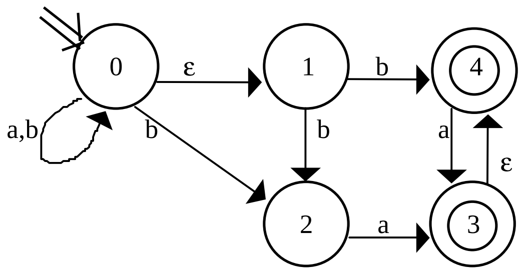
4. Draw state diagrams for nondeterministic finite automata that accept these languages:
(a) (ab)*(ba)* ∪ aa*
(b) ((ab ∪ aab)*a*)*
(c) ((a*b*a*)*b)*
(d) (ba ∪ b)* ∪ (bb ∪ a)*

5. Some authors define a nondeterministic finite automaton to be a quintuple (K, Σ, Δ, S, F), where K, Σ, Δ, and F are as we have defined them and S is a finite set of initial states, in the same way that F is a finite set of final states. The automaton may nondeterministically begin operating in any of these initial states. Explain why this definition is not more general than ours in any significant way.

6. (a) Find a simple nondeterministic finite automaton accepting ((a ∪ b)*aabab).
(b) Convert the nondeterministic finite automaton of Part (a) into a deterministic finite automaton by the method described in class and in the notes.
(c) Try to understand how the machine constructed in Part (b) operates. Can you find an equivalent deterministic machine with fewer states?

7. Construct a NDFSA that accepts the language (ba ∪ ((a ∪ bb) a*b)).

8. Construct a deterministic finite automaton equivalent to the following nondeterministic automaton:



9. L = { w ∈ {a, b}* : every a is followed by at least one b }

(a) Write a regular expression that describes L.

(b) Write a regular grammar that describes L.

(c) Construct an FSM that accepts precisely L.

10. Consider the following regular grammar, which defines a language L:

S -> bF
S -> aS
F -> ε
F -> bF
F -> aF

(a) Construct an FSM that accepts precisely L.

(b) Write a regular expression that describes L.

(c) Describe L in English.

Solutions

1. (a) [i.] yes [ii.] yes [iii.] no [iv.] yes
(b) [i.] yes [ii.] yes [iii.] yes [iv.] yes [v.] no

2. (a) a* Note that the second state could be eliminated, since there's no path from it to a final state.
(b) (ab ∪ aba)* Notice that we could eliminate the start state and make the remaining final state the start state and we'd still get the same result.

3. To determine L, we need first to consider the set of machines that accept the same language as R. It turns out that we don't actually need to know what all such machines look like because we can immediately see that there's at least one with four states (R), one with 5, one with 6, and so forth, and all that we need to establish L is the sizes of the machines, not their structures.. From R, we can construct an infinite number of equivalent machines by adding any number of redundant states. For example, we could add a new, nonfinal state 1 that is reachable from the start state via an ε transition. Since 1 is not final and it doesn't go anywhere, it cannot lead to an accepting path, so adding it to R has no affect on R's behavior. Now we have an equivalent machine with 5 states. We can do it again to yield 6, and so forth. Thus the set of numbers we need to represent is simply 4 ≤ n. Now all we have to do is to describe the binary encodings of these numbers. If we want to disallow leading zeros, we get 1(0 ∪ 1) (0 ∪ 1) (0 ∪ 1)*. There must be at least three digits of which the first must be 1.

4. (a) The easiest way to do this is to make a 2 state FSA for aa* and a 4 state one for (ab)*(ba)*, then make a seventh state, the start state, that nondeterministically guesses which class an input string will fall into.
(b) First we simplify. ((ab ∪ aab)*a*)*

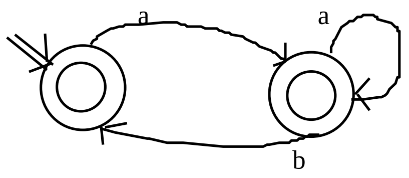
/ $(L_1{}^*L_2{}^*)^* = (L_1 \cup L_2)^*$ /

((ab ∪ aab) ∪ a)*

/ union is associative /

(ab ∪ aab ∪ a)*, which can be rewritten as

$(ab \cup a)^*$. This is so because aab can be formed by one application a, followed by one of ab. So it is redundant inside a Kleene star. Now we can write a two state machine:



If you put the loop on a on the start state, either in place of where we have it, or in addition to it, it's also right.

(c) First we simplify: $((a^*b^*a^*)^*b)^*$

/ $(L_1{}^*L_2{}^*L_3{}^*)^* = (L_1 \cup L_2 \cup L_3)^*$ /

$((a \cup b \cup a)^*b)^*$

/ union is idempotent /

$((a \cup b)^*b)^*$

There is a simple 2 state NDFSM accepting this, which is the empty string and all strings ending with b.

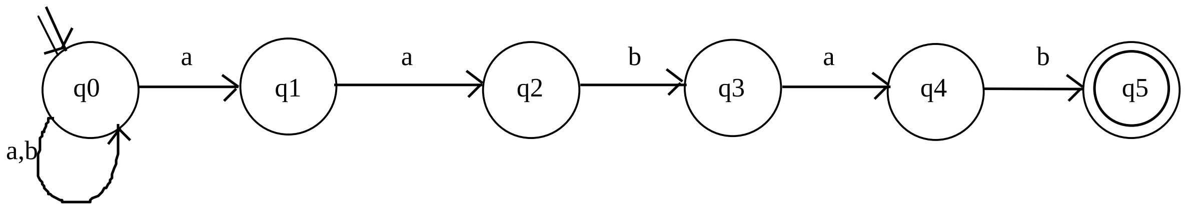
(d) This is the set of strings where either: (1) every a is preceded by a b,
or (2) all b's occur in pairs.

So we can make a five state nondeterministic machine by making separate machines (each with two states) for the two languages and then introducing ε transitions from the start state to both of them.

5. To explain that any construct A is not more general or powerful than some other construct B, it suffices to show that any instance of A can be simulated by a corresponding instance of B. So in this case, we have to show how to take a multiple start state NDFSA, A, and convert it to a NDFSA, B, with a single start state. We do this by initially making B equal to A. Then add to B a new state we'll call S0. Make it the only start state in B. Now add ε transitions from S0 to each of the states that was a start state in A. So B has a single start state (thus it satisfies our original definition of a NDFSA), but it simulates the behavior of A since the first thing it does is to move, nondeterministically, to all of A's start states and then it exactly mimics the behavior of A.

6. If you take the state machine as it is given, add a new start state and make ε transitions from it to the given start states, you have an equivalent machine in the form that we've been using.

7. (a)



(b) (1) Compute the E(q)s. Since there are no ε transitions, E(q), for all states q is just {q}.

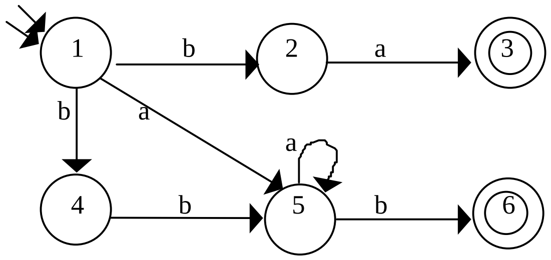
(2) S' = {q0}

(3) δ' = { ({q0}, a, {q0, q1}),
({q0}, b, {{q0}),
({q0, q1}, a, {q0, q1, q2}),
({q0, q1}, b, {q0}),
({q0, q1, q2}, a, {q0, q1, q2}),
({q0, q1, q2}, b, {q0, q3}),
({q0, q3}, a, {q0, q1, q4}),
({q0, q3}, b, {q0}),
({q0, q1, q4}, a, {q0, q1, q2}),
({q0, q1, q4}, b, {q0, q5}),
({q0, q5}, a, {q0, q1}),
({q0, q5}, b, {q0}) }

(4) K' = {{q0}, {q0, q1}, {q0, q1, q2}, {q0, q3}, {q0, q1, q4}, {q0, q5}}
(5) F' = {{q0, q5}}

(c) There isn't a simpler machine since we need a minimum of six states in order to keep track of how many characters (between 0 and 5) of the required trailing string we have seen so far.

8. We can build the following machine really easily. We make the path from 1 to 2 to 3 for the ba option. The rest is for the second choice. We get a nondeterministic machine, as we generally do when we use the simple approach.

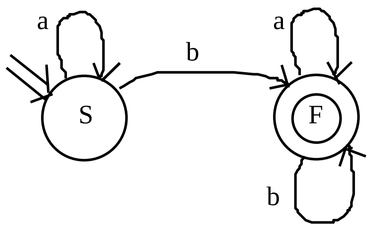


In this case, we could simplify our machine if we wanted to and get rid of state 4 by adding a transition on b from 2 to 5.

9. (1) E(q0) = {q0, q1}, E(q1) = {q1}, E(q2) = {q2}, E(q3) = {q3, q4}, E(q4) = {q4}
(2) s' = {q0, q1}
(3) δ' = { ({q0, q1}, a, {q0, q1}),
({q0, q1}, b, {q0, q1, q2, q4}),
({q0, q1, q2, q4}, a, {q0, q1, q3, q4}),
({q0, q1, q2, q4), b, {q0, q1, q2, q4}) }
({q0, q1, q3, q4}, a, {q0, q1, q3, q4}),
({q0, q1, q3, q4}, b, {q0, q1, q2, q4}),
(4) K' = { {q0, q1}, {q0, q1, q3, q4}, {q0, q1, q2, q4} }
(5) F' = { {q0, q1, q3, q4}, {q0, q1, q2, q4} }

This machine corresponds to the regular expression a*b(a ∪ b)*

10. (a)



(b) (a ∪ b)*ba* OR a*b(a ∪ b)*

(c) L = { w ∈ {a, b}* : there is at least one b }