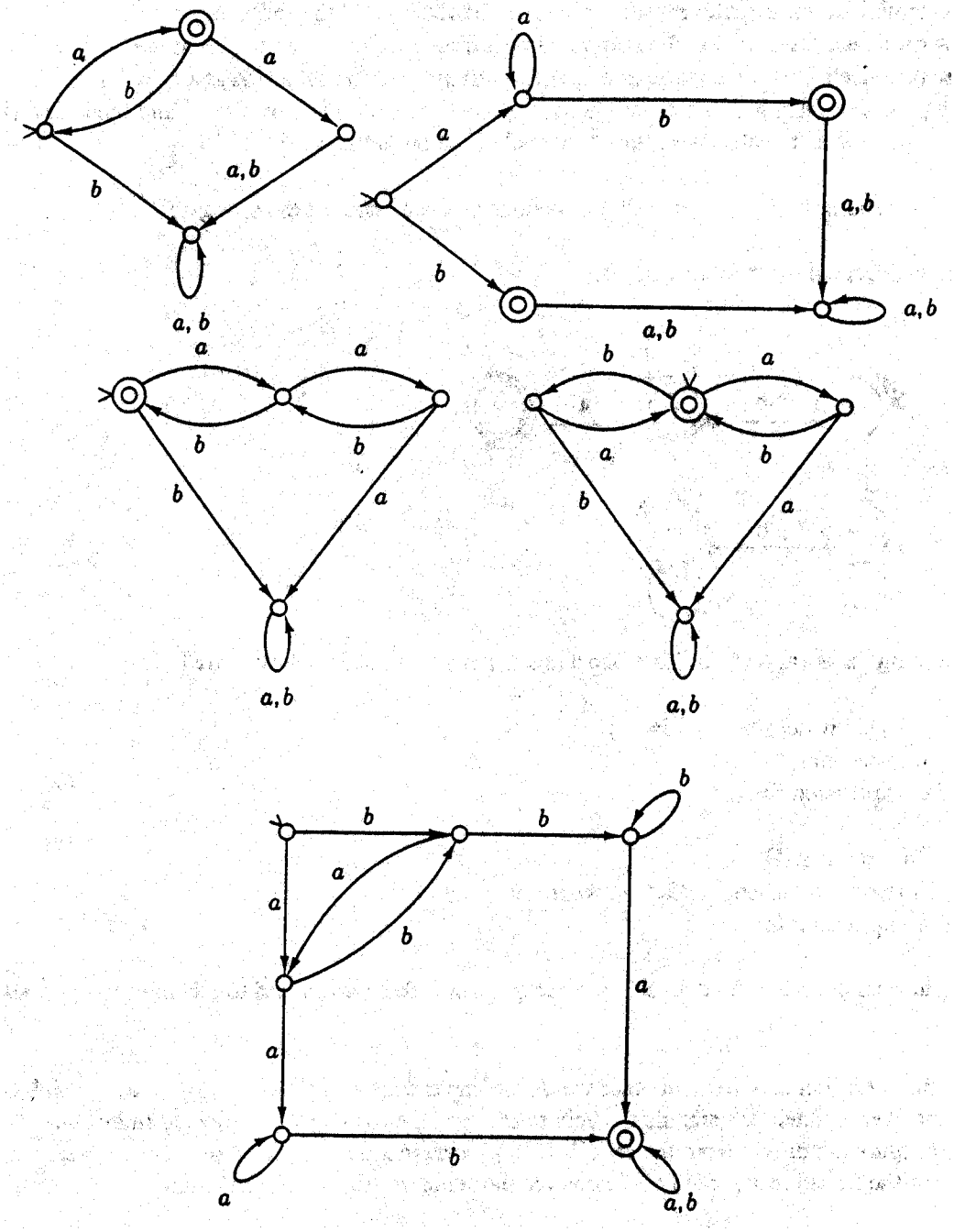
CS 341 Homework 4
Deterministic Finite Automata

1. If M is a deterministic finite automaton. Under exactly what circumstances is $\varepsilon \in L(M)$?

2. Describe informally the languages accepted by each of the following deterministic FSMs:



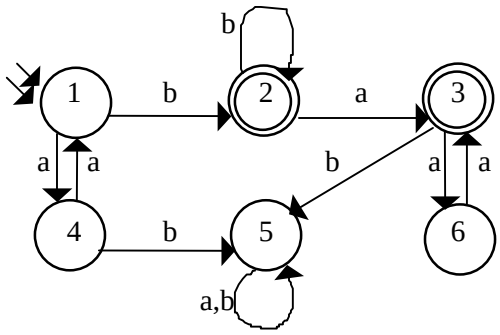
(from Elements of the Theory of Computation, H. R. Lewis and C. H. Papdimitriou, Prentice-Hall, 1998.)

3. Construct a deterministic FSM to accept each of the following languages:
(a) {w ∈ {a, b}* : each 'a' in w is immediately preceded and followed by a 'b'}
(b) {w ∈ {a, b}* : w has abab as a substring}
(c) {w ∈ {a, b}* : w has neither aa nor bb as a substring}
(d) {w ∈ {a, b}* : w has an odd number of a's and an even number of b's}
(e) {w ∈ {a, b}* : w has both ab and ba as substrings}

4. Construct a deterministic finite state transducer over {a, b} for each of the following tasks:
(a) On input w produce a^n, where n is the number of occurrences of the substring ab in w.
(b) On input w produce a^n, where n is the number of occurrences of the substring aba in w.
(c) On input w produce a string of length w whose i^{th} symbol is an a if i = 1 or if i > 1 and the i^{th} and $(i-1)^{st}$ symbols of w are different; otherwise, the i^{th} symbol of the output is b.

5. Construct a dfa accepting L = {w ∈ {a, b}* : w contains no occurrence of the string ab}.

6. What language is accepted by the following fsa?



7. Give a dfa accepting {x ∈ {a, b}* : at least one a in x is not immediately followed by b}.

8. Let L = {w ∈ {a, b}* : w does not end in ba}.
(a) Construct a dfa accepting L.
(b) Give a regular expression for L.

9. Consider L = $\{a^n b^n : 0 \le n \le 4\}$
(a) Show that L is regular by giving a dfa that accepts it.
(b) Give a regular expression for L.

10. Construct a deterministic finite state machine to accept strings that correspond to odd integers *without* leading zeros.

11. Imagine a traffic light. Let Σ = {a}. In other words, the input consists just of a string of a's. Think of each a as the output from a timer that signals the light to change. Construct a deterministic finite state transducer whose outputs are drawn from the set {Y, G, R} (corresponding to the colors yellow, green, and red). The outputs of the transducer should correspond to the standard traffic light behavior.

12. Recall the finite state machine that we constructed in class to accept $1.00 in change or bills. Modify the soda machine so that it actually does something (i.e., some soda comes out) by converting our finite state acceptor to a finite state transducer. Let there be two buttons, one for Coke at $.50 and one for Water at $.75 (yes, it's strange that water costs more than Coke. The world is a strange place). In any case, there will now be two new symbols in the input alphabet, C and W. The machine should behave as follows:

- The machine should keep track of how much money has been inserted. If it ever gets more than $1.50, it should spit back enough to get it under $1.00 but keep it above $.75.
- If the Coke or Water button is pushed and enough money has been inserted, the product and the change should be output.
- If a button is pushed and there is not enough money, the machine should remember the button push and wait until there is enough money, at which point it should output the product and the change.

13. Consider the problem of designing an annoying buzzer that goes off whenever you try to drive your car and you're not wearing a seat belt. (For simplicity, we'll just worry about the driver's possible death wish. If you want to make this harder, you can worry about the other seats as well.) Design a finite state transducer whose inputs are drawn from the alphabet {KI, KR, SO, SU, BF, BU}, representing the following events, respectively: "key just inserted into ignition", "key just removed from ignition", "seat just became occupied", "seat just became unoccupied", "belt has just been fastened", and "belt has just been unfastened". The output alphabet is {ON, OFF}. The buzzer should go on when ON is output and stay off until OFF is output.

14. Is it possible to construct a finite state transducer that can output the following sequence:

101001000100001000010000001…

If it is possible, design one. If it's not possible, why not?

Solutions

1. $\varepsilon \in$ L(M) iff the initial state is a final state. Proof: M will halt in its initial state given ε as input. So: (IF) If the initial state is a final state, then when M halts in the initial state, it will be in a final state and will accept ε as an element of L(M). (ONLY IF) If the initial state is not a final state, then when M halts in the initial state, it will reject its input, namely ε. So the only way to accept ε is for the initial state to be a final state.

2.

(a) You must read a to reach the unique final state. Once there, you may read ba and still accept. So the language is $a(ba)^*$. (Or $(ab)^*a$.) This problem is fairly easy to analyze. (Informally, you could describe this as all strings that begin and end with a, and the symbols alternate a and b, or something of this nature; giving the regular expression is much clearer and easier.)

(b) There are two final states that are reachable. This one is quite easy because once you reach the final states you cannot go further. The obvious answer is $aa^*b \cup b$. This can be simplified to a^*b. The machine is distinguishing between whether the number of a's is positive or 0, but there is no need to.

(c) This one is trickier. How can we reach the final state here? By going to the middle state with a and then returning with b. This can be iterated. But while in the middle state we may iterate ab. So the answer is $(a(ab)^*b)^*$.

(d) This one is similar to (c) but easier. We can reach the final state by reading ab or ba, and in either case we may iterate again. So $(ab \cup ba)^*$ is the solution.

(e) Number the states 1,2,3,4,5,6 going right to left, top to bottom. The following properties characterize each state:

1: e has been read.

2: xb has been read, for some $x \in (a \cup b)^*$ not ending in b.

3: xbb has been read, for some $x \in (a \cup b)^*$.

4: xa has been read, for some $x \in (a \cup b)^*$ not ending in a.

5: xaa has been read, for some $x \in (a \cup b)^*$.

6: $xbbay$ or $xaaby$ has been read, for some $x, y \in (a \cup b)^*$.

Therefore the language is all strings containing bba or aab as a substring, i.e., $(a \cup b)^*(bba \cup aab)(a \cup b)^*$.

3.

(a) $L = \{w \in \{a,b\}^* :$ each a in w is immediately preceded and immediately followed by a $b\}$.

(A regular expression for L is $(b^*ba)(b^*ba)^*bb^* \cup b^*$, or, using $^+$, $(b^+a)^+b^+ \cup b^*$. Notice the necessary distinction between strings with no a's and those with a's. Why doesn't the simpler $b^*(b^+ab^+)^*$ work?)

This will need a machine with a deadstate because as soon as we see an a not preceded or followed by a b, the string should be rejected and no matter what comes later, the string is bad. I.e., we will assume the string is ok until a specific occurence which tells us to reject the string.

Clearly $e \in L$ since every a in e has the property. Now for any longer string, the machine only needs to remember what the last symbol was to determine if the string should be rejected.

So we could make states with the properties:

1: $e \in L$ has been read.

2: xa has been read, for some $x \in L$ not ending in a (the string so far is ok, but we'd better see a b next since $xa \notin L$.)

3: $xb \in L$ has been read (the string so far is ok.)

4: x has been read, such that for no y is $xy \in L$. (we know the string is bad – no matter what comes later.)

You should be able to draw the machine now. Notice that $s = 1$, $F = \{1,3\}$.

(b) $L = \{w \in \{a,b\}^* : w$ has $abab$ as a substring$\}$.

(A regular expression for L is easy: $(a \cup b)^*abab(a \cup b)^*$.)

Again we need to keep track only of the last part of the string, in this case the last 3 symbols. In this one we are looking for an occurence in the string which *will* make us accept the string (compare to Problem (a).) Once there has been an occurence of $abab$, whatever follows is irrelevant.

Here are the relevant properties of the string as it is read in:

1: x has been read, for some $x \in (a \cup b)^*$ such that $x \notin L$ and x does not end in a.

2: xa has been read, for some $x \in (a \cup b)^*$ such that $x \notin L$ and x does not end in ab.

3: xab has been read, for some $x \in (a \cup b)^*$ such that $x \notin L$ and x does not end in ab.

4: $xaba$ has been read, for some $x \in (a \cup b)^*$ such that $x \notin L$ and x does not end in ab.

5: $xababy$ has been read, for some $x, y \in (a \cup b)^*$ such that $x \notin L$ and x does not end in ab.

So a 5 state machine can do the trick. The start state is 1, because that's the property e has ($e \in (a \cup b)^*$ and e does not end in a.) Any string with property 1 which is then followed by b continues to have property 1, so $\delta(1, b) = 1$. Any string with property 1 which is then followed by a now has property 2, so $\delta(1, a) = 2$. And so on. Clearly $\delta(5, \sigma) = 5$ since once $abab$ has been seen, that fact cannot be changed – $abab$ continues to have been seen. Clearly a string has $abab$ as a substring iff it has property 5, so $F = \{5\}$. Now you draw the DFA.

(c) $L = \{w \in \{a, b\}^* : w$ has neither aa nor bb as a substring$\}$.

(A regular expression for L is $e \cup a(ba)^*(b \cup e) \cup b(ab)^*(a \cup e)$. This distinguishes between whether the string starts with a or b or is empty. Another one is $(a \cup e)(ba)^*(b \cup e)$, though this is perhaps less obvious.)

Like Problem (a), we should assume the string is ok until we see a bad occurence (aa or bb). To test this, we clearly only need to keep track of the last symbol read. So the relevant properties are:

1: e has been read (and so a or b may follow.)
2: xa has been read, for some $xa \in L$ (so only b may follow.)
3: xb has been read, for some $xb \in L$ (so only a may follow.)
4: x has been read, for some $x \notin L$.

Clearly any string with property 1, 2 or 3 is in L, so $F = \{1, 2, 3\}$. The start state is 1. Now you draw it.

(d) $L = \{w \in \{a, b\}^* : \#(a, w)$ is odd and $\#(b, w)$ is even$\}$.

I use the function $\#(\sigma, x)$ to mean "the number of occurences of symbol σ in string x." E.g., $\#(a, aba) = 2$ and $\#(b, aaa) = 0$.

Unlike the previous problems, there is no specific occurence we are looking for, either to reject or accept the string. Instead, we need to continually monitor it. When the string is all read in, its status will then determine whether it is accepted or rejected.

Clearly what we need to monitor is the parity (even or odd) of the number of a's and the number of b's. These are independent data, so there are $2 \times 2 = 4$ possible states or properties:

(0,0): x has been read, where $\#(a, x)$ and $\#(b, x)$ both even.
(0,1): x has been read, where $\#(a, x)$ even and $\#(b, x)$ odd.
(1,0): x has been read, where $\#(a, x)$ odd and $\#(b, x)$ even.
(1,1): x has been read, where $\#(a, x)$ and $\#(b, x)$ both odd.

Since $\#(\sigma, e) = 0$, and 0 is even, the start state is (0,0). (A fair number of people unnecessarily distinguish between 0 and other even numbers, producing machines with more states than necessary.) The only final state is (1,0). δ can be defined by $\delta((m, n), a) = (m + 1 \bmod 2, n)$ and $\delta((m, n), b) = (m, n + 1 \bmod 2)$.

This is a technique easily generalized. Finite automata cannot count to arbitrarily high natural numbers, but they can count modulo a number (so-called "clock arithmetic"). The DFA just given counts the number of a's and the number of b's modulo 2. (A number m is even iff m is congruent to 0 mod 2, written $x \equiv 0 \bmod 2$, e.g., $x = \ldots, -2, 0, 2, 4, \ldots$.) You could design a DFA to accept all strings x with $\#(a, x) \equiv 7 \bmod 12$ and $\#(b, x) \equiv 0 \bmod 5$ and and $\#(c, x) \equiv 2 \bmod 3$, i.e., $\#(a, x) = 7, 19, 26, \ldots$ and $\#(b, x)$ is a multiple of 5 and

$\#(c, x) = 2, 5, 8, \ldots$. A minimum state DFA to accept this language uses $12 \times 5 \times 3 = 180$ states. For notational convenience, I would call the states (i, j, k), where $0 \leq i \leq 12$, $0 \leq j \leq 5$ and $0 \leq k \leq 3$. Then the final state would be $(7, 0, 2)$. The start state is of course $(0, 0, 0)$.

What would you do if you wanted all strings x with $\#(a, x) \equiv 2$ or 3 mod 4, or $\#(b, x) \equiv 1$ mod 3? (Hint: the states are constructed in the same manner; only the final conditions are different.)

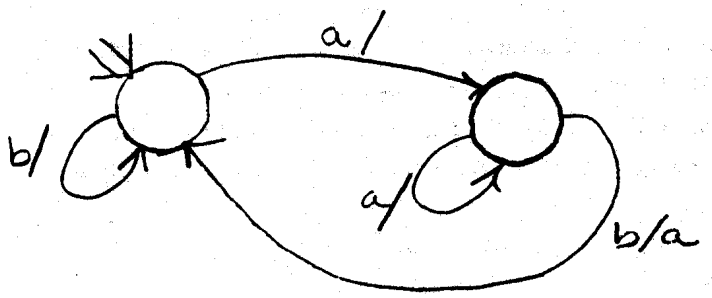
(e) $L = \{w \in \{a, b\}^* : w$ has both ab and ba as substrings$\}$.

Here we are looking for not one occurence but two in the string. There are two subtleties. Either event might occur first, so we must be prepared for the ab or the ba to be read first. Also, the definition of L does not require the two substrings of ab anb ba to be nonoverlapping: e.g., $aba \in L$.

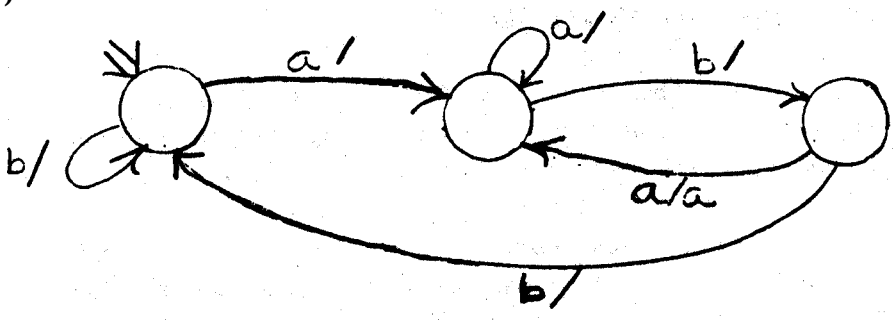
1: e has been read (so we have seen neither substring.)
2: a^m has been read, for some $m \geq 1$ (so we have seen neither substring.)
3: $a^m b^n$ has been read, for some $m, n \geq 1$ (so we have seen ab.)
4: b^m has been read, for some $m \geq 1$ (so we have seen neither substring.)
5: $b^m a^n$ has been read, for some $m, n \geq 1$ (so we have seen ba.)
6: $a^m b^n ax$ or $b^m a^n bx$ has been read, for some $m, n \geq 1$ and $x \in (a \cup b)^*$ (so we have seen ab and ba.)

Clearly, 1 is the start state and $\{6\}$ is the set of final states. You should be able to draw the DFA now.

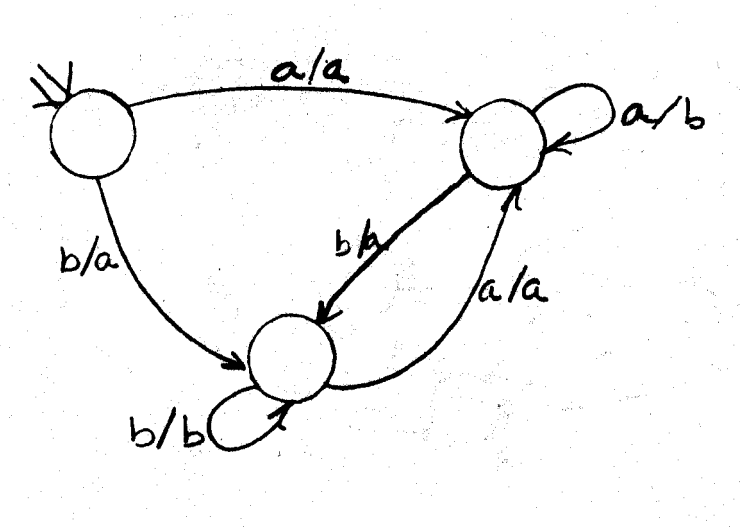
4. (a)



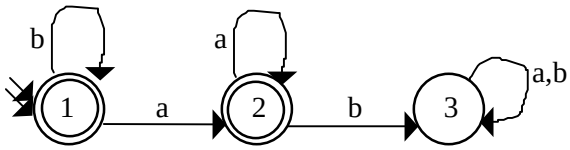
(b)



(c)

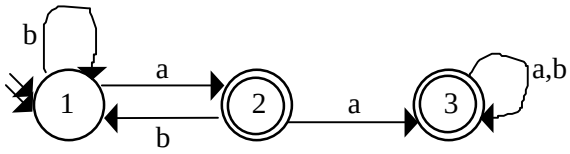


5.

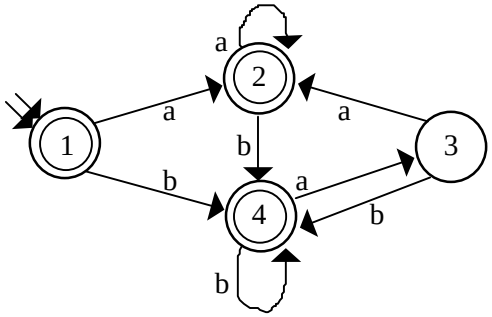


6. (aa)* (bb* ∪ bb*a(aa)*) = (aa)*b^+(ε ∪ a(aa)*) = all strings of a's and b's consisting of an even number of a's, followed by at least one b, followed by zero or an odd number of a's.

7.

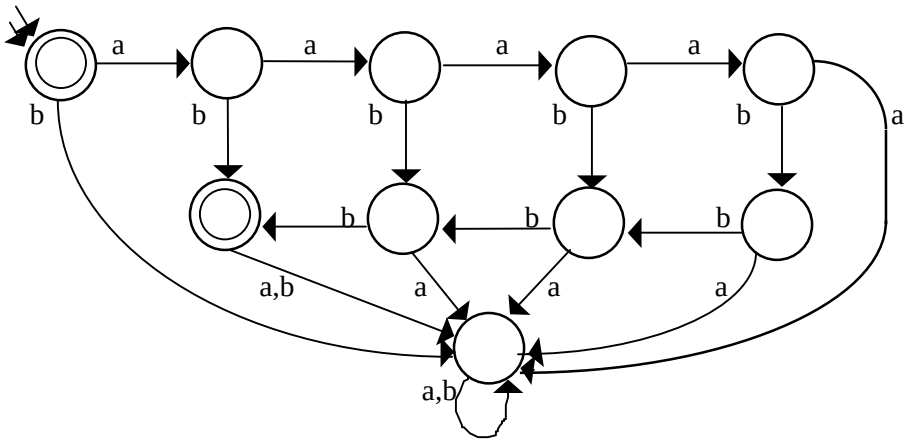


8. (a)



(b) ε ∪ a ∪ (a ∪ b)* (b ∪ aa)

9. (a)



(b) (ε ∪ ab ∪ aabb ∪aaabbb ∪aaaabbbb)