CS 341 Homework 10
State Minimization

1. (a) Give the equivalence classes under $\approx_L$ for these languages:

(i) $L = (aab \cup ab)^*$
(ii) $L = \{x : x$ contains an occurrence of aababa$\}$
(iii) $L = \{xx^R : x \in \{a, b\}^*\}$
(iv) $L = \{xx : x \in \{a, b\}^*\}$
(v) $L_n = \{a, b\}a\{a, b\}^n$, where $n > 0$ is a fixed integer
(vi) The language of balanced parentheses

(b) For those languages in (a) for which the answer is finite, give a deterministic finite automaton with the smallest number of states that accepts the corresponding language.

2. Let $L = \{x \in \{a, b\}^* : x$ contains at least one a and ends in at least two b's$\}$.
(a) Write a regular expression for L.
(b) Construct a deterministic FSM that accepts L.
(c) Let R_L be the equivalence relation of the Myhill-Nerode Theorem. What partition does R_L induce on the set
$\{a, bb, bab, abb, bba, aab, abba, bbaa, baaba\}$?
(d) How many equivalence classes are there in the partition induced on Σ^* by R_L?

3. Let $L = \{x \in \{a, b\}^* : x$ begins with a and ends with b$\}$.
(a) What is the nature of the partition induced on Σ^* by R_L, the equivalence relation of the Myhill-Nerode Theorem? That is, how many classes are there in the partition and give a description of the strings in each.
(b) Using these equivalence classes, construct the minimum state deterministic FSM that accepts L.

4. Suppose that we are given a language L and a deterministic FSM M that accepts L. Assume L is a subset of $\{a, b, c\}^*$. Let R_L and R_M be the equivalence relations defined in the proof of the Myhill-Nerode Theorem. True or False:
(a) If we know that x and y are two strings in the same equivalence class of R_L, we can be sure that they are in the same equivalence class of R_M.
(b) If we know that x and y are two strings in the same equivalence class of R_M, we can be sure that they are in the same equivalence class of R_L.
(c) There must be at least one equivalence class of R_L that has contains an infinite number of strings.
(d) R_M induces a partition on $\{a, b, c\}^*$ that has a finite number of classes.
(e) If $\varepsilon \in L$, then $[\varepsilon]$ (the equivalence class containing ε) of R_L cannot be an infinite set.

5. Use the Myhill-Nerode Theorem to prove that $\{a^n b^m c^{\max(m,n)} b^p d^p : m, n, p \geq 0\}$ is not regular.

6. (a) In class we argued that the intersection of two regular languages was regular on the basis of closure properties of regular languages. We did not show a construction for the FSM that recognizes the intersection of two regular languages. Such a construction does exist, however, and it is suggested by the fact that $L_1 \cap L_2 = \Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$.

Given two deterministic FSMs, M_1 and M_2, that recognize two regular languages L_1 and L_2, we can construct an FSM that recognizes $L = L_1 \cap L_2$ (in other words strings that have all the required properties of both L_1 and L_2), as follows:

1. Construct machines M_1' and M_2', as deterministic versions of M_1 and M_2. This step is necessary because complementation only works on deterministic machines.
2. Construct machines M_1'' and M_2'', from M_1' and M_2', using the construction for complementation, that recognize $\Sigma^* - L_1$ and $\Sigma^* - L_2$, respectively.
3. Construct M_3, using the construction for union and the machines M_1'' and M_2'', that recognizes $((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$. This will be a nondeterministic FSM.
4. Construct M_4, the deterministic equivalent of M_3.
5. Construct M_L, using the construction for complementation, that recognizes $\Sigma^* - ((\Sigma^* - L_1) \cup (\Sigma^* - L_2))$.

Now consider: $\Sigma = \{a, b\}$
$L_1 = \{w \in \Sigma^* : \text{all a's occur in pairs}\}$ e.g., aa, aaaa, aabaa, aabbaabbb $\in L_1$
aaa, baaab, ab $\notin L_1$

$L_2 = \{w \in \Sigma^* : w \text{ contains the string bbb}\}$

Use the procedure outlined above to construct an FSM M_L that recognizes $L = L_1 \cap L_2$.

Is M_L guaranteed to be deterministic?

(b) What are the equivalence classes under $\approx_L$ for the language $L = L_1 \cap L_2$?

(c) What are the equivalence classes under $\sim_M$ for M_L in (a) above?

(d) Show how $\sim_M$ is a refinement of $\approx_L$.

(e) Use the minimization algorithm that we have discussed to construct from M_L in (a) above a minimal state machine that accepts L.

7. If you had trouble with this last one, make up another pair of L_1 and L_2 and try again.

Solutions

1. (a)

(i) $L = (aab \cup ab)^*$

1. [ε, aab, ab, and all other elements of L]
2. [a or wa : w ∈ L]
3. [aa or waa : w ∈ L]
4. [everything else, i.e., strings that can never become elements of L because they contain illegal substrings such as bb or aaa]

(ii) L = {x : s contains an occurrence of aababa}

1. [$(a \cup b)^*aababa(a \cup b)^*$, i.e., all elements of L]
2. [ε or any string not in L and ending in b but not in aab or aabab, i.e., no progress yet on "aababa"]
3. [wa for any w ∈ [2]; alternatively, any string not in L and ending in a but not in aa, aaba, or aababa]
4. [any string not in L and ending in aa]
5. [any string not in L and ending in aab]
6. [any string not in L and ending in aaba]
7. [any string not in L and ending in aabab]

Note that this time there is no "everything else". Strings never become hopeless in this language. They simply fail if we get to the end without finding "aababa".

(iii)$L = \{xx^R : x \in \{a, b\}^*\}$

1. [a, which is the only string for which the continuations that lead to acceptance are all strings of the form wa : where w ∈ L]
2. [b, which is the only string for which the continuations that lead to acceptance are all strings of the form wb : where w ∈ L]
3. [ab, which is the only string for which the continuations that lead to acceptance are all strings of the form wba : where w ∈ L]
4. [aa, which is the only string for which the continuations that lead to acceptance are all strings of the form waa : where w ∈ L]

And so forth. Every string is in a distinct equivalence class.

(iv) $L = \{xx : x \in \{a, b\}^*\}$

1. [a, which is the only string for which the continuations that lead to acceptance are all strings that would be in L except that they are missing a leading a]
2. [b, which is the only string for which the continuations that lead to acceptance are all strings that would be in L except that they are missing a leading b]
3. [ab, which is the only string for which the continuations that lead to acceptance are all strings that would be in L except that they are missing a leading ab]

4. [aa, which is the only string for which the continuations that lead to acceptance are all strings that would be in L except that they are missing a leading aa]

And so forth. Every string is in a distinct equivalence class.

(v) $L_n = \{a, b\}a\{a, b\}^n$

0. [ε]
1. [a, b]
2. [aa, ba]
3. [aaa, aab, baa, bab]

.
.
.

n+2. [$(a \cup b)a(a \cup b)^n$]

n+3. [strings that can never become elements of L_n]

There is a finite number of strings in any specific language L_n. So there is a finite number of equivalence classes of $\approx_L$. Every string in L_n must be of length n+2. So there are n+3 equivalence classes (numbered 0 to n+2, to indicate the length of the strings in the class) of strings that may become elements of L_n, plus one for strings that are already hopeless, either because they don't start with ab or aa, or because they are already too long.

(vi) L = The language of balanced parentheses

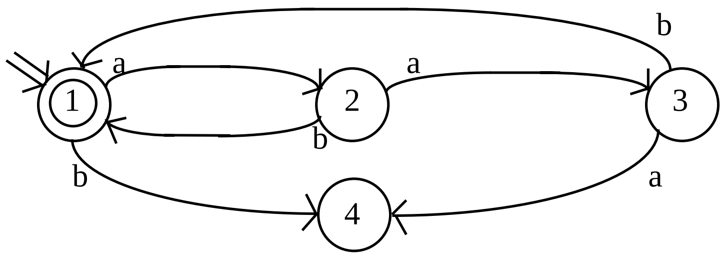
1. [w*(w*: w ∈ L] /* i.e., one extra left parenthesis somewhere in the string
2. [w*((w* : w ∈ L] /* two “
3. [w*(((w* : w ∈ L]
4. [w*((((w* : w ∈ L]
5. [w*(((((w* : w ∈ L]

… and so on. There is an infinite number of equivalence classes.

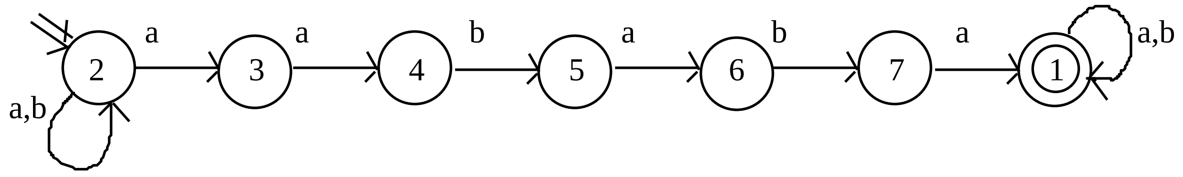
Each of these classes is distinct, since) is an acceptable continuation for 1, but none of the others;)) is acceptable for 2, but none of the others,))) is acceptable for 3, but none of the others, and so forth.

1. (b)

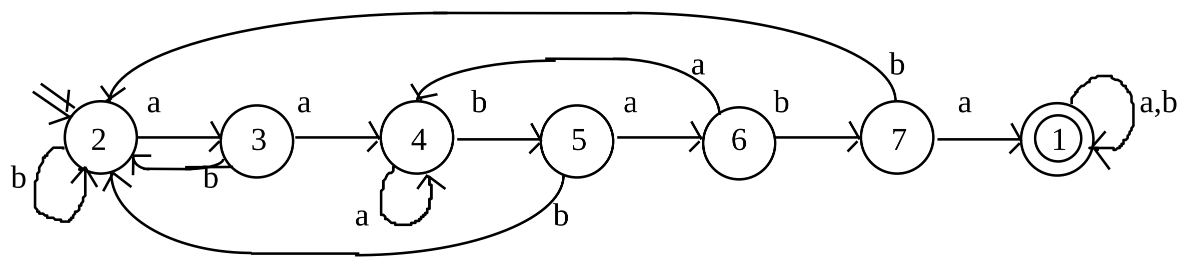
(i)



(ii) There's always a very simple nondeterministic FSM to recognize all strings that contain a specific substring. It's just a chain of states for the desired substring, with loops on all letters of Σ on the start state and the final state. In this case, the machine is:



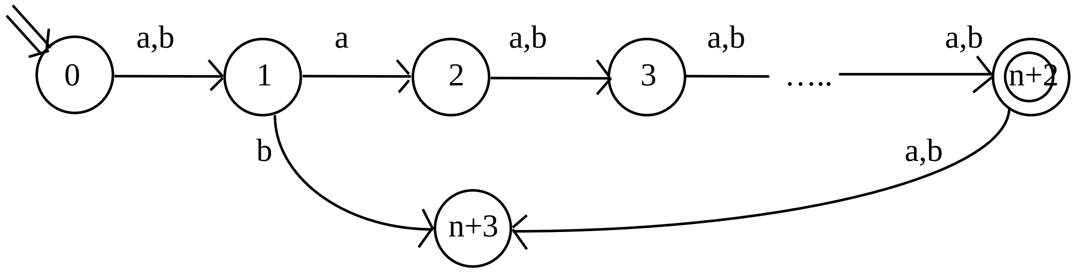
To construct a minimal, deterministic FSM, you have two choices. You can use our algorithm to convert the NDFSM to a deterministic one and then use the minimization algorithm to get the minimal machine. Or you can construct the minimal FSM directly. In any case, it is:



(iii) There is no FSM for this language, since it is not regular.

(iv) There is no FSM for this language, since it is not regular.

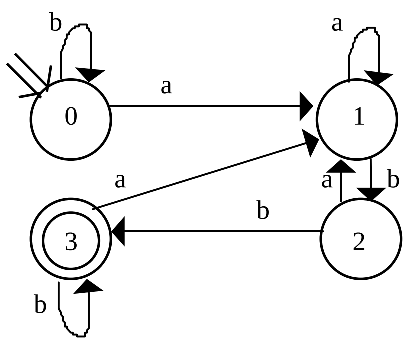
(v)



(vi) There is no FSM for this language, since it is not regular.

2. (a) (a ∪ b)*a(a ∪ b)*bbb*   or   (a ∪ b)*a(a ∪ b)* bb

(b)



(c) It's easiest to answer (d) first, and then to consider (c) as a special case.

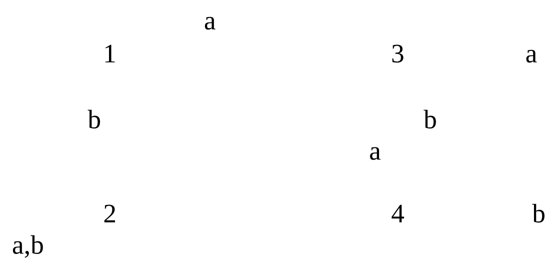
(d) [0] = all strings that contain no a
[1] = all strings that end with a
[2] = all strings that end with ab
[3] = all strings that contain at least one a and that end with bb, i.e., all strings in L.

It is clear that these classes are pairwise disjoint and that their union is {a, b}*. Thus they represent a partition of {a, b}*. It is also easy to see that they are the equivalence classes of R_L of the Myhill-Nerode Theore, since all the members of one equivalence class will, when suffixed by any string z, form strings all of which are in L or all of which are not in L. Further, for any x and y from different equivalence classes, it is easy to find a z such that one of xz, yz is in L and the other is not.

Letting the equivalence relation R_L be restricted to the set in part (c), gives the partition
{{bb}, {a, bba, abba, bbaa, baaba}, {bab, aab}, {abb}}.

3. **(a)** [1] = {ε}
[2] = b (a ∪ b)*
[3] = a ∪ a(a b)*a
[4] = a(a ∪ b)*b

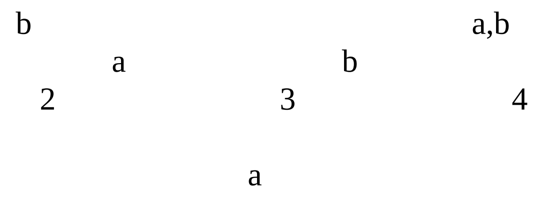
(b)



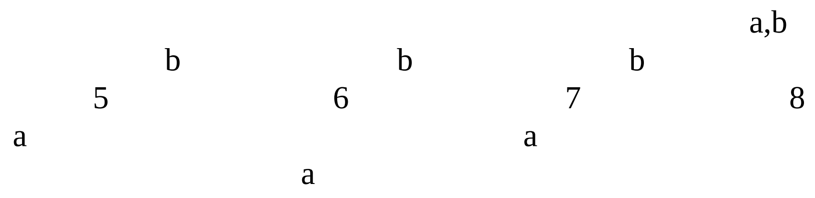
4. **(a)** F, **(b)** T, **(c)** T (Σ* is infinite and the number of equivalence classes is finite), **(d)** T, **(e)** F.

5. Choose any two distinct strings of a's: call them a^i and a^j ($i < j$). Then they must be in different equivalence classes of R_L since $a^ib^ic^i \in L$ but $a^jb^ic^i \notin L$. Therefore, there is an infinite number of equivalence classes and L is not regular.

6. **(a)** M_1, which recognizes L_1, is:



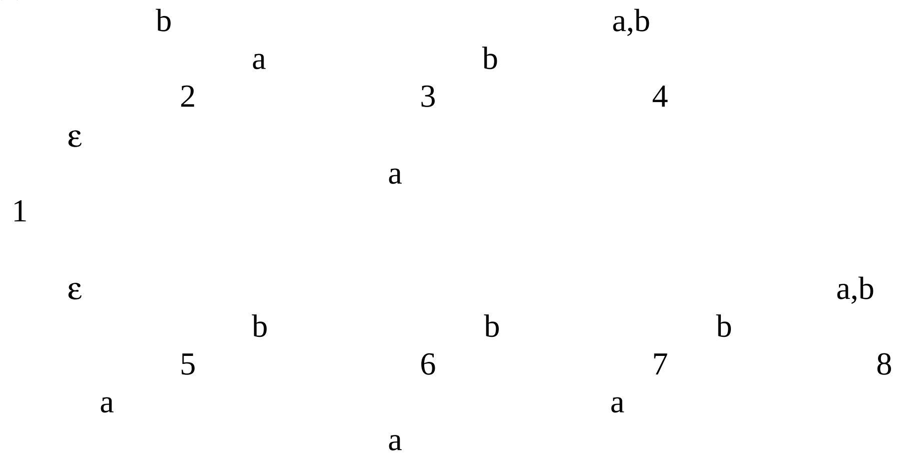
M_2, which recognizes L_2, is:



Step (1) $M_1' = M_1$ because M1 is deterministic.
$M_2' = M_2$ because M2 is deterministic.

Step (2) M_1'' is M_1' except that states 3 and 4 are the final states.
M_2'' is M_2' except that states 5, 6, and 7 are the final states.

Step (3) M_3 is:



Step (4) M_4 is:

{1, 2, 5}, a, {3, 5}	{2, 5}, a, {3, 5}	{4, 5}, a, {4, 5}	{4, 8}, a, {4, 8}
b, {2, 6}	b, {2, 6}	b, {4, 6}	b, {4, 8}
{3, 5}, a, {2, 5}	{4, 6}, a, {4, 5}	{4, 7}, a, {4, 5}	{3, 8}, a, {2, 8}
b, {4, 6}	b, {4, 7}	b, {4, 8}	b, {4, 8}
{2, 6}, a, {3, 5}	{2, 7}, a, {3, 5}	{2, 8}, a, {3, 8}	
b, {2, 7}	b, {2, 8}	b, {2, 8}	

s = {1, 2, 5}
F = K - {2, 8}, i.e., all states except {2, 8} are final states.

You may find it useful at this point to draw this out.

Step (5) $M_L = M_4$ except that now there is a single final state, {2, 8}.

M_L is deterministic. M_4 is deterministic by construction, and Step 5 can not introduce any nondeterminism since it doesn't introduce any transitions.

(b)
1. [strings without bbb and with any a's in pairs, including ε]
2. [strings without bbb but with a single a at the end]
3. [strings that cannot be made legal because they have a single a followed by a b]
4. [strings without bbb, with any a's in pairs, and ending in a single b]
5. [strings without bbb, with any a's in pairs, and ending with just two b's]
6. [strings with bbb and with any a's in pairs]
7. [strings with bbb but with a single a at the end]

(c)

{1, 2, 5}	[ε]
{3, 5}	[strings without bbb but with a single a at the end]
{2, 6}	[strings without bbb, with any a's in pairs, and ending in a single b]
{2, 5}	[strings without bbb and with at least one pair of a's and any a's in pairs]
{4, 6}	[strings that cannot be made legal because they have a single a followed by a b and where every b is preceded by an a and the last character is b]
{2, 7}	[strings without bbb, with any a's in pairs, and ending with just two b's]
{4, 5}	[strings that cannot be made legal because they have a single a followed by b and where there is no bbb and the last character is a]
{4, 7}	[strings that cannot be made legal because they have a single a followed by a b and where there is no bbb but there is at least one bb and the last character is b]
{2, 8}	[all strings in L]
{4, 8}	[strings that cannot be made legal because they have a single a followed by a b and where there is a bbb, but the ab violation came before the first bbb]
{3, 8}	[strings with bbb but with a single a at the end]

(d) {1, 2, 5} ∪ {2,5} = 1
{3, 5} = 2
{4, 6} ∪ {4, 5} ∪ {4, 7} ∪ {4, 8} = 3
{2, 6} = 4
{2, 7} = 5
{2, 8} = 6
{3, 8} = 7

(e) $\equiv_0$ = A: [{2, 8}],

B: [{1, 2, 5}, {2, 5}, {3, 5}, {4, 6}, {4, 5}, {4, 7}, {4, 8}, {2, 6}, {2, 7}, {3, 8}]

To compute $\equiv_1$: Consider B (since clearly A cannot be split). We need to look at all the single character transitions out of each of these states. We've already done that in Step (3) of part (a) above, so we can use that table to tell us which of our current states each state goes to. Now we just need to use that to determine which element of $\equiv_0$ they go to. We notice that all transitions are to elements of B except: ({2, 7}, b, A) and ({3, 8}, a, A). So we must split these two states from B. and they must be distinct from each other because their a and b behaviors are reversed. So we have:

$\equiv_1$ = A: [{2, 8}],
B: [{1, 2, 5}, {2, 5}, {3, 5}, {4, 6}, {4, 5}, {4, 7}, {4, 8}, {2, 6}]
C: [{2, 7}]
D: [{3, 8}]

To compute $\equiv_2$: Again we consider B:
On reading an a, all elements of B go to elements of B.
But on b: ({2, 6}, b, C), so we must split off {2, 6}. This gives us:

$\equiv_2$ = A: [{2, 8}],
B: [{1, 2, 5}, {2, 5}, {3, 5}, {4, 6}, {4, 5}, {4, 7}, {4, 8}]
C: [{2, 7}]
D: [{3, 8}]
E: [{2, 6}]

To compute $\equiv_3$: Again we consider B:
On reading an a, all elements of B go to elements of B.
But on b, {1, 2, 5} and {2, 5} go to E, while everyone else goes to B. So we have to split these two off. This gives us:

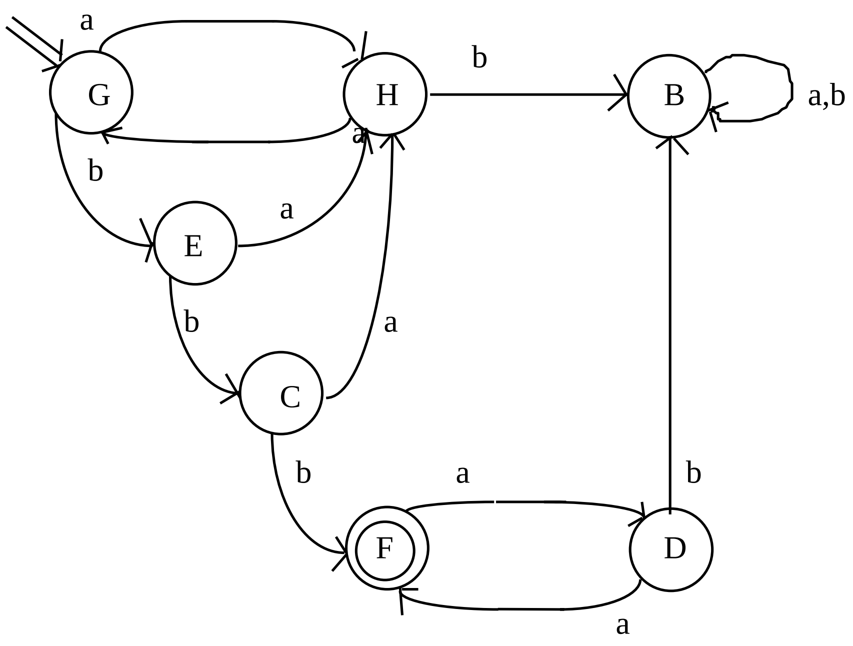
$\equiv_3$ = A: [{2, 8}],
B: [{3, 5}, {4, 6}, {4, 5}, {4, 7}, {4, 8}]
C: [{2, 7}]
D: [{3, 8}]
E: [{2, 6}]
F: [{1, 2, 5}, {2, 5}]

To compute $\equiv_4$: Again we consider B:
On reading b, all elements of B stay in B. But on reading a, {3, 5} goes to F, so we split it off. This gives us:

$\equiv_4$ = A: [{2, 8}],
B: [{4, 6}, {4, 5}, {4, 7}, {4, 8}]
C: [{2, 7}]
D: [{3, 8}]
E: [{2, 6}]
F: [{1, 2, 5}, {2, 5}]
G: [{3, 5}]

To compute $\equiv_5$: Again we consider B: On both inputs, all elements of B stay in B. So we do no further splitting, and we assert that $\equiv$ = $\equiv_4$. Notice that this is identical to what we expected from (d) above.

We can now show the minimal machine:


a
b
a,b
a
b
a
b
a
b
a
b
a
G
H
B
E
C
F
D