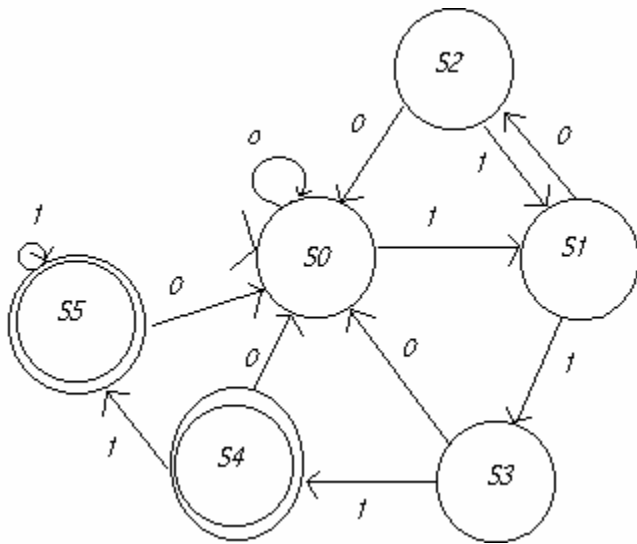


Homework 15

Note: for all of exercises 1-3 for Chapter 17, the alphabet must be specified as $\{0,1\}$.

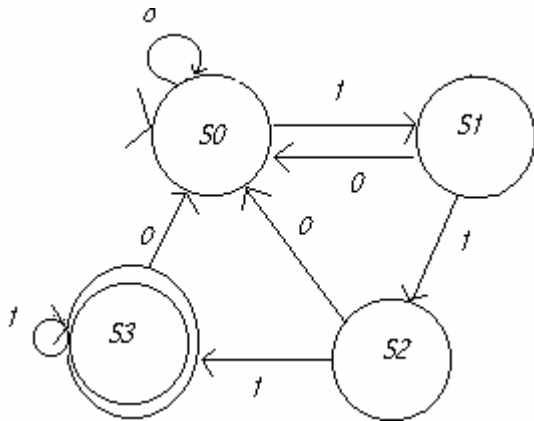
PtMW, Ch 17, pp 480-484. Exercises 1-11.

1. (a) (ii), (iii), and (iv) are accepted.
 - (b) It takes at least two 1's to get to the final state. Adding one 1 to that gets us back to the initial state, and another 1 to the non-final state S1. Once we have five 1's, however, we get to the final state again. The number of 0's in the string is irrelevant. Therefore, what is accepted is the set of all strings containing a total of n 1's, where n is congruent to 2 (modulo 3).
2. (a)



(b) It takes at least three consecutive occurrences of 1 to get us to one of the final states S4. Adding another 1 to that also gets us to the final state S5. And we stay in S5 as long as 1 is the only thing that is being added. Adding 0 to a string in either of those final states, however, sends us back to the initial state. Therefore what we have is the set of strings that end in at least three successive 1's.

(c) We can get to the final stage with three 1's, and only then, so S0 cannot be an initial state, and neither can the subsequent two states. S3 is therefore the only final state. In the initial state, 0 doesn't take us anywhere and simply loops back. In the subsequent stages, one instance of 0 is enough to send us back to the initial state, so that we can begin from scratch (in our attempt to build three subsequent 1's). Once we have three subsequent 1's, adding more 1's leaves us in the final stage, while adding 0 loops us back to the initial state (since we want the string to end in 1 as well).

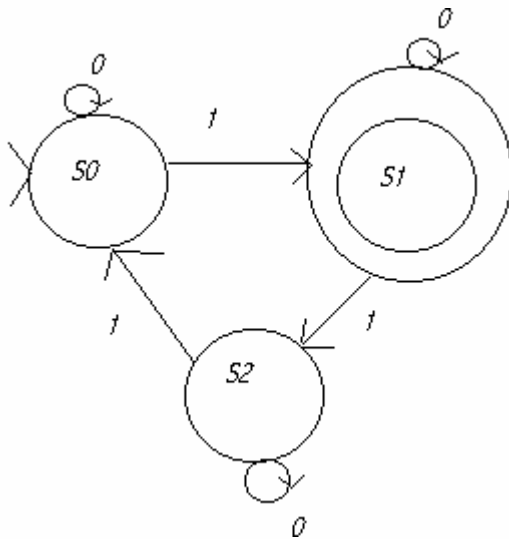


3. State diagrams, using as few states as possible:

(a) This automaton accepts any string with one 1, four 1's, seven 1's, etc. S1, where one 1 has been added to the string, is the final state. The addition of a second 1 simply takes it to another non-final state. The addition of a 3rd 1 is equivalent to the string where no 1 was present—our initial state.

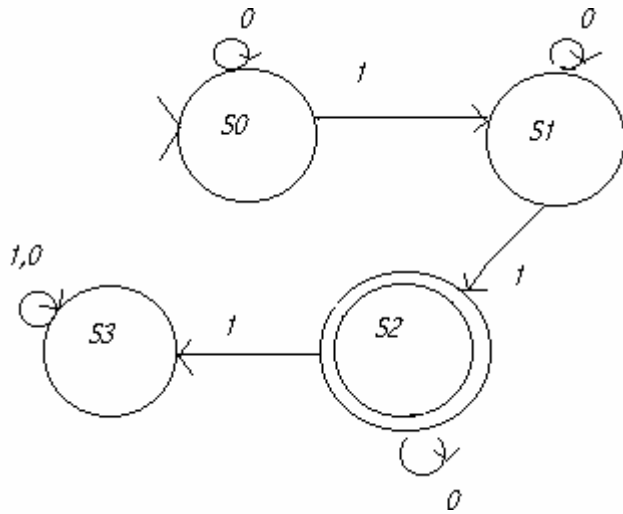
The number of 0's is irrelevant, so whichever state the string(-to-be) is in, 0 just loops back to that state.

The diagram meeting the above conditions is as follows:

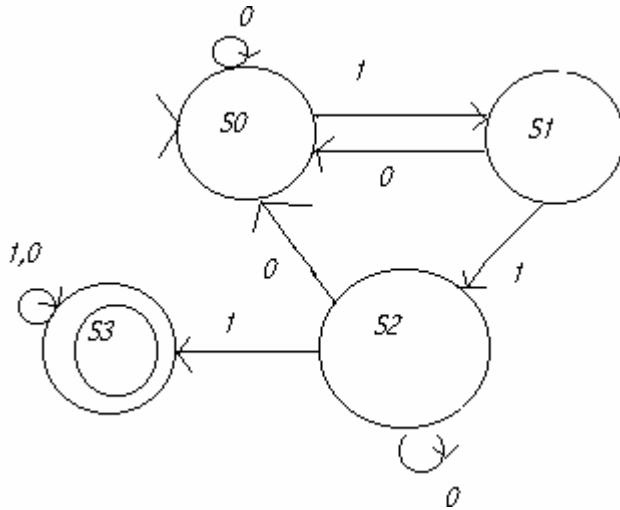


(b) Here, two 1's have to be added before we reach the final state. Adding more simply takes us to a non-final state. This non-final state is however distinct from the initial state, since we want exactly two 1's and no more.

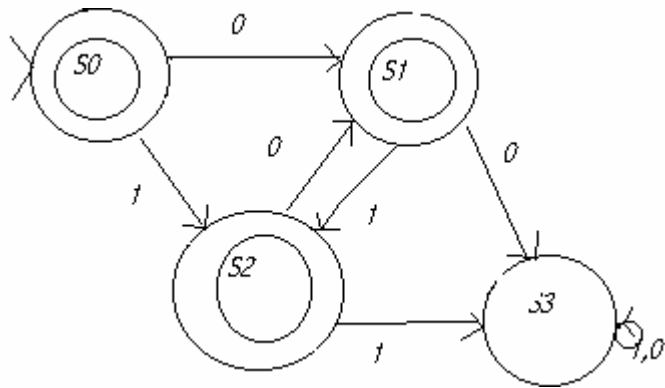
The number of 0's is irrelevant, as before.



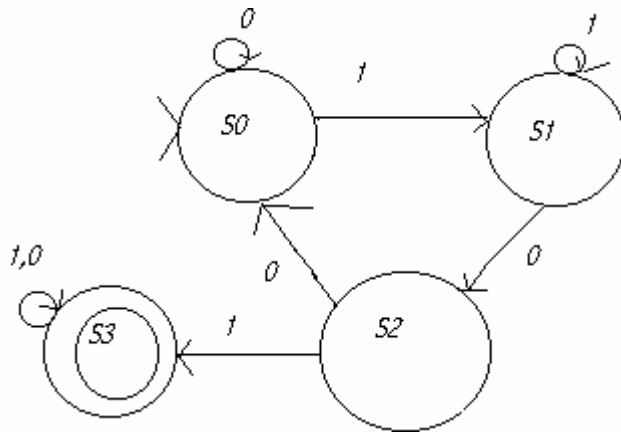
(c) We need at least three 1's, so we need at least three more states besides the initial state. Once we've made that block of three, however, we need not worry whether we encounter a 0 or a 1 (i.e. these loop back to that final state). It is crucial here that 0 should not intervene and break up the sequence of the 1's. Whenever we encounter a 0, we have to go back to the initial state and start afresh.



(d) This machine weeds out blocks of any kind and encourages alternation. On the other hand, it also accepts the empty string ϵ . This means that the initial state is also a final state. We should not, however, have 0 or 1 loop back to this initial state, if we are to achieve the alternation effect. Therefore, we map the addition of 0 to a second state S1, which is final, and likewise 1 to a third state S2, which is also final. Now from S1, we want to add 1, which will make it an acceptable output; this final state could be S2. If we add 0 instead, however, we will reach a fourth state S3, which is not a final state. Likewise for S2: if we add 0, we map onto S1, a final state; otherwise, we map onto S3, a non-final state. Any addition to S3 is not going to improve the situation; in other words, both 0 and 1 loop back onto S3.



(e) This is similar to (c), with the exception that what we have in between the two 1's is a zero. But this machine is somewhat more generous to the intervention of a foreign element than (c): instead of sending us back to the initial state when it hits a 1 after the first 1 (the first element of the 101 block), it simply loops us back to the same state, from which we can attempt again to form the 101 string.



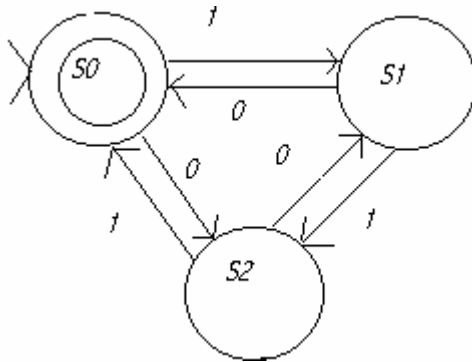
(f) The strings where the number of 0's and the number of 1's are congruent modulo 3 may contain:

zero/three/six... 0's	zero/three/six ... 1's
one/four/seven ... 0's	one/four/seven ... 1's
two/five/eight... 0's	two/five/eight ... 1's

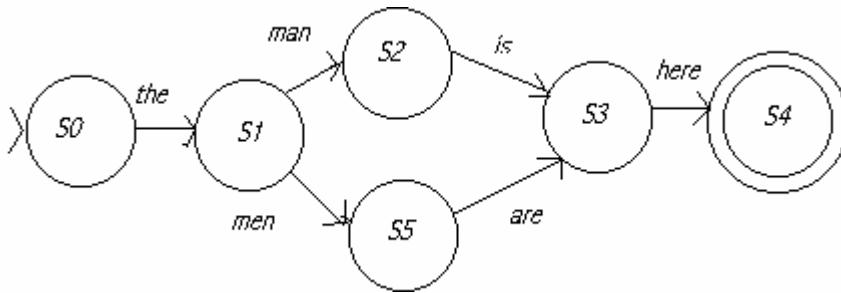
Let's first take the case where there is no 0 and the number of 1's is congruent to that of 0's.

- The empty string is accepted; therefore, the initial state is also a final state.
- Adding one or two 1's takes us to states that are non-final (therefore distinct from the initial state).
- Adding multiples of three 1's takes us to the final state. There are therefore three states strung together.
- With three states strung together, all we need to get the congruence between the number of 0's and the number of 1's is to have the functions involving them be the inverse of each other. This allows for the addition of either symbol at any state.

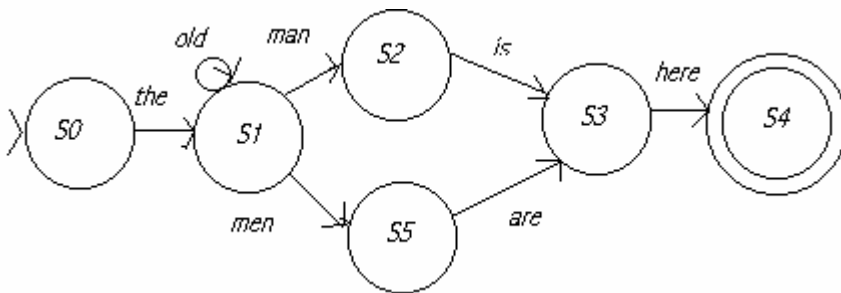
Our automaton satisfying all the above conditions can be represented as follows:



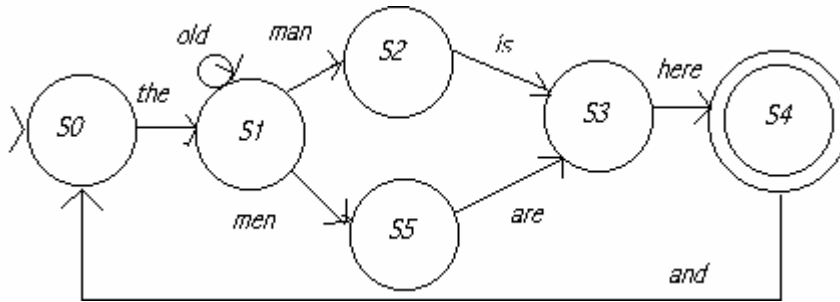
4. (a) All the elements in those two strings are required for those strings to be accepted (shorter strings are not, at least in this example). In fact, the two strings are identical, save for *man* and *men*, and *is* and *are*.



(b) The addition of *old* is between *the* and *man/men* only; infinitely many instances of it are permitted, suggesting looping:

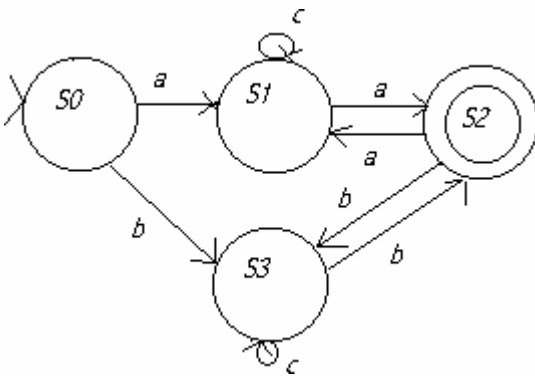


(c) The addition of *and* takes us back to the beginning of a string, the initial state:



5. (a) The strings described by these rewrite rules are of the following form: ac^*a , bc^*b , and concatenation of those strings.
- The empty string is not accepted; the initial state is not a final state.
 - It takes pair(s) of a 's (and pairs of b 's) to attain a final state. The addition of one a (or b) takes us to a non-final intermediate state. The addition of two a 's (or b 's) take us to a final state. From then on, we loop back to the intermediate state with another a (or b), and back to the final state with yet another a (or b).
 - Both odd and even number of recursions of c 's are allowed: c gives us a loop on that final state. Note that this final state, however, is distinct from the one that pairs of a 's or b 's take us to; were it to be the same, we would be generating strings starting with symbols that are not c 's but ending in c 's, which our rules do not allow.
 - c also loops on the non-final states where the addition of odd-numbered a 's and b 's takes us.
 - The remaining problem is to string two S 's together. Each S is a string containing even numbers of a 's and b 's. a and b do not alternate; pairs of a 's and pairs of b 's do. Therefore, just as in the case of the odd-number addition of a 's or b 's, we want the addition of a b after even numbers of a 's (and therefore in a final state) to take us to a non-final state, from which we can reach a final state by simply adding another b . (And likewise for pairs of a 's following b 's).

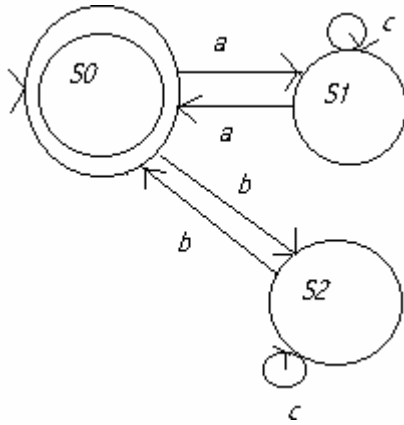
Our automaton satisfying the above condition is represented as follows:



Note that from S_2 , the addition of c would take us to a “dump” state.

- (b) Since the empty string is allowed, the initial state is also a final state. Otherwise, the automaton has to satisfy the same conditions as above:
- The final state reached by c and the one reached by pairs of a 's or b 's (or their concatenation) must be distinct.
 - The addition of odd numbers of a 's or b 's takes us to a non-final state.

Therefore:



6. Transitions to the initial state, as seen in the numerous examples above, achieve the following effects:

- In those cases where the initial state is not a final state, looping back to the initial state means a fresh start.
- In those cases where the initial state is also a final state, looping back to the initial state is attaining a final state. Also, a final initial state allows for the empty string.

Therefore, in the absence of transitions to the initial state, all we need is:

- In those cases where the initial state is not a final state, we merely need another state from which to make a fresh start: the same transitions we had from the initial state are now copied onto this new state.
- In those cases where the initial state is also a final state, we simply need another final state (and copy all the transitions from and to the initial state). Since permitting the empty string can take place without any transition, however, we need not make special reference to the empty string for this final state.