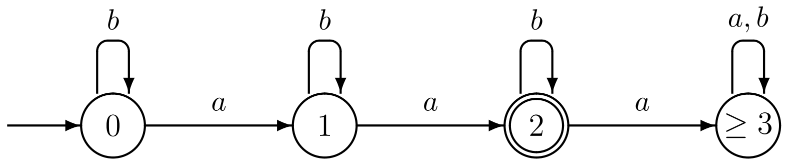
Some more solutions to exercises in

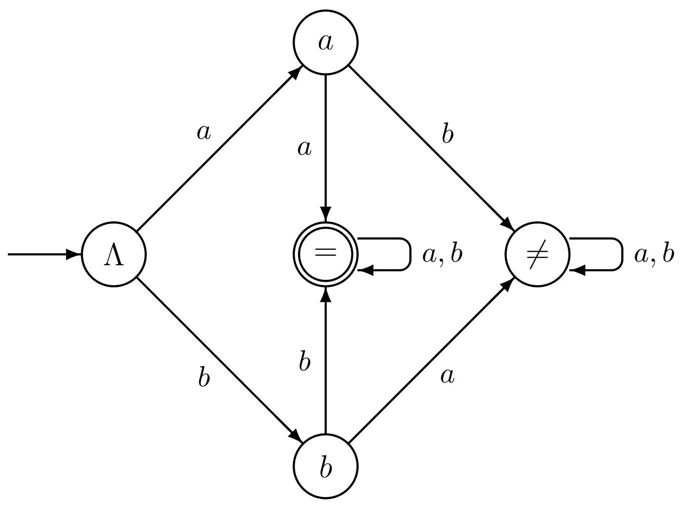
John C. Martin: Introduction to Languages and The Theory of Computation

fourth edition

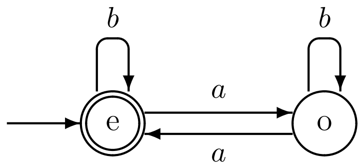
2.1(a) States count the number of a's read so far.



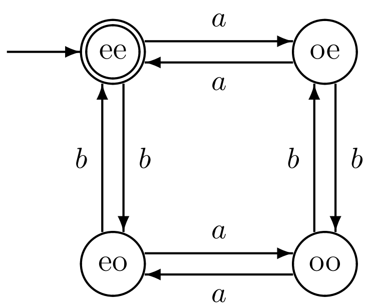
2.1(d) States keep track of the first two characters read.



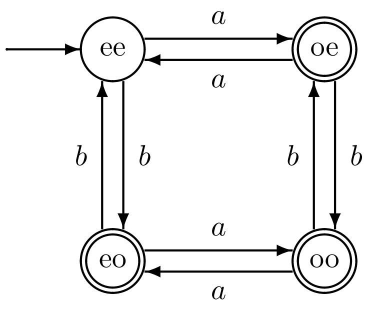
2.1(f) States represent whether the number of a's read so far is even or odd



2.1(g) States represent whether the number of a's read so far is even or odd and whether the number of b's read so far is even or odd.



2.1(g2) When you consider the language of strings over $\{a, b\}$ in which either the number of a's or the number of b's is odd (or both), you can take almost the same finite automaton. Only the accepting states and the nonaccepting states must be exchanged, because the language is the complement of the language from 2.1(g).



2.21(e) Let a^l and a^m be two different elements of $\{a^n \mid n \geq 0\}$. Without loss of generality, assume that $l < m$. We now give two different solutions, which are both correct:

- The idea is to let z consist of only b's: just enough to make $a^l z$ an element of L. This number of b's should, on the other hand, be too small to make $a^m z$ an element of L. Extension z must contain $\lfloor \frac{l}{2} \rfloor + 1$ b's to make $a^l z$ an element of L. One can verify that this works well both if l is even and if l is odd. Can we be certain that with this z, the string $a^m z$ is not an element of L? To answer this question we compare $n_b(a^m z)$ to $n_a(a^m z)$.

  Let us first assume that l is odd. Then

$$\begin{aligned} 2n_b(a^m z) &= 2(\lfloor \frac{l}{2} \rfloor + 1) \\ &= 2(\frac{l-1}{2} + 1) \\ &= l - 1 + 2 = l + 1 \leq m = n_a(a^m z) \end{aligned}$$

  The last inequality holds because $l < m$. Indeed, in this case, $a^m z$ is not an element of L.

  Now, let us assume that l is even. Then

$$\begin{aligned} 2n_b(a^m z) &= 2(\lfloor \frac{l}{2} \rfloor + 1) \\ &= 2(\frac{l}{2} + 1) \\ &= l + 2 \end{aligned}$$

  If $m = l + 1$, then this final number $l + 2$ is larger than $m = n_a(a^m z)$. In that case, $a^m z$ is also an element of L, and z does not distinguish a^l and a^m with respect to L. For example, if $l = 8$ and $m = 9$, then $z = b^{4+1} = b^5$ does not distinguish a^l and a^m. Hence, if l is even, the proposed string z does not suit our needs. A small adjustment is, however, suffcient. If l is even, then we may simply prepend an a to z, thus making the number of a's in $a^l z$ odd, after all: $z = a \cdot b^{\lfloor \frac{l}{2} \rfloor + 1}$. One can verify that this string z does distinguish a^l and a^m if l is even.

- A simpler solution, but maybe harder to think of. The string $z = a^m b^m$, containing equally many a's and b's, works for all l and m with $l < m$. Indeed,

$$\begin{aligned} n_a(a^l z) &= l + m < 2m = 2n_b(a^l z) \\ n_a(a^m z) &= m + m = 2m = 2n_b(a^= z) \end{aligned}$$

Hence, $a^l z \in L$, whereas $a^m z \notin L$.

2.22(d) Hint: note that $j = i$ is also a multiple of i.

2.22(e) Suppose that L can be accepted by a finite automaton M. Let n be the number of states of M.

We choose $x = a^{2n-1}b^n$. Indeed $|x| = 3n - 1 \geq n$ and $n_a(x) = 2n - 1 < 2n = 2n_b(x)$, so $x \in L$ (note that there are many other strings x that could be used to find a contradiction).

Let u, v, w be arbitrary strings in $\{a, b\}^*$ such that $x = a^{2n-1}b^n = uvw$, $|uv| \leq n$ and $|v| \geq 1$. Then obviously, uv consists of only a's. In particular, so does v, say $v = a^k$ for some k with $1 \leq k \leq n$.

Now consider the string $x' = uv^2w$ (hence, we choose $m = 2$). We have $x' = uv^2w = uvvw = a^{2n-1+k}b^n$. As $n_a(x') = 2n - 1 + k \geq 2n - 1 + 1 = 2n = 2n_b(x')$, x' does not satisfy $n_a(x') < 2n_b(x')$, and so $x' \notin L$, regardless of what u, v and w look like exactly.

This contradicts the pumping lemma for regular languages. Therefore, the assumption that L can be accepted by a finite automaton must be wrong. We conclude that L cannot be accepted by a finite automaton.

2.24 The proof of this generalization of the pumping lemma is very similar to that of the pumping lemma itself.

Assume that language L is accepted by an FA M. Let n be the number of states of M.

Now, let x be an arbitrary element of L satisfying $|x| \geq n$, and let $x_1x_2x_3$ be an arbitrary decomposition of x (hence, $x = x_1x_2x_3$) such that $|x_2| = n$.

In processing input x, M first reads the letters of x_1, then those of x_2, and then those of x_3. While reading the n letters of x_2, M moves from state to state. Doing so, it 'sees' $n+1$ states, from just before the first letter of x_2 until just after the last of x_2. As M only has n different states, it must see at least one state q at least two times while reading the letters of x_2.

Let u be the prefix of x_2 after which M is in state q for the first time (after having read x_1), and let u' be the prefix of x_2 after which M is in state q for the second time. Obviously, u is a prefix of u'. Hence, $u' = uv$ for some string v satisfying $|v| \geq 1$. Let w be the final part of x_2, after prefix $u' = uv$: $x_2 = uvw$.

When M processes input $x = x_1x_2x_3 = x_1uvwx_3$, it traverses a cycle from state q just before substring v to state q just after substring v. M might skip this cycle (corresponding to input $x_1uv^0wx_3$) or traverse this cycle multiple times (input $x_1uv^mwx_3$ for some $m \geq 2$). In all cases M ends up in the same state as it does for input $x = x_1uv^1wx_3$, which is an accepting state.

We conclude that for every $m \geq 0$, M accepts $x_1uv^mwx_3$, i.e., $x_1uv^mwx_3 \in L$. $\square$