Surprising Aspects of the Mod-3 Acceptor

Robert M. Keller
10 November 2010

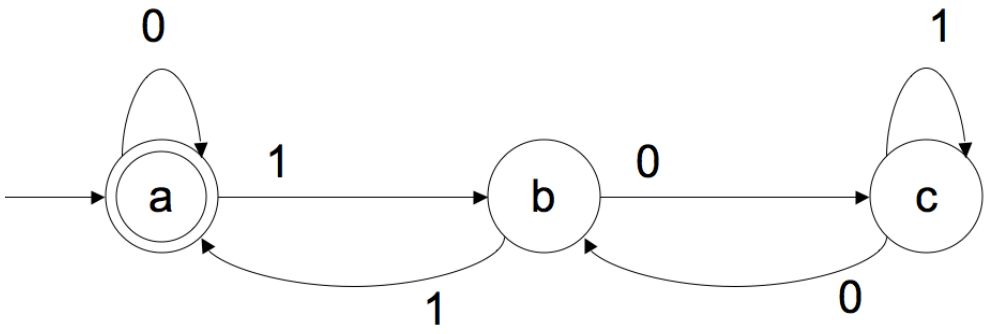
The problem is to design a finite-state acceptor that accepts exactly the strings over alphabet {0, 1} that represent a number divisible by 3 in binary, e.g. the set {0, 00, 11, 000, 011, 110, 0000, 0011, 0110, 1001, 1100, 1111, …}, where we have listed the strings in "lexicographic" order.

For simplicity, we adopt the convention that the empty string is considered to be divisible by 3.

Before proceeding with a solution, we should answer the question of whether the string is being read left-to-right or right-to-left by the acceptor.

The surprising thing is that exactly the same acceptor works for both directions of reading, but for different reasons.

Left-to-Right Analysis: For example, a step in reading would be from 110 to 1101 (i.e. 6 to 13 in decimal). Let r be the mod-3 residue of the number being read. It is easy to see that reading a 0 changes r to 2r, where arithmetic is done mod 3. That is, 0 → 0, 1 → 2, and 2 → 1 when a 0 is read. Also reading a 1 changes r to 2r+1, i.e. 0 → 1, 1 → 0, and 2 → 2. This gives us the following acceptor for Left-to-Right:



Here the doubly circled state is the accepting state, and the arrow with no source points to the initial state. The following table indicates what each state represents.

State	Represents strings
a	having residue 0 mod 3
b	having residue 1 mod 3
c	having residue 2 mod 3

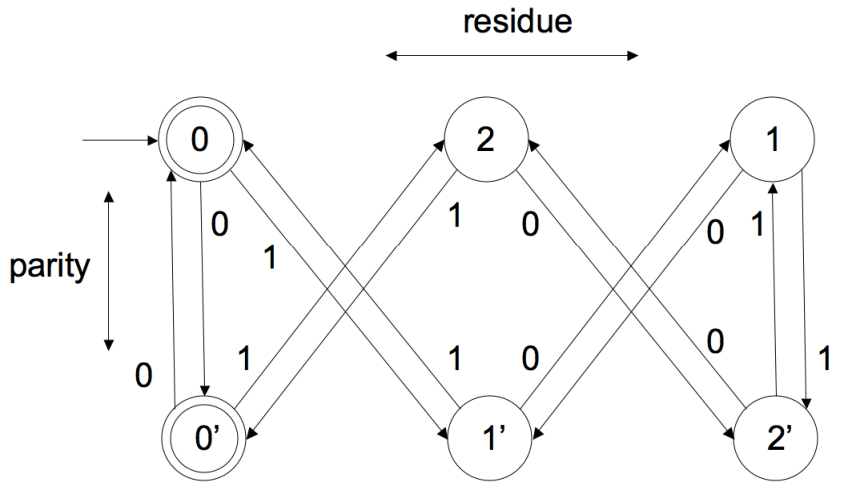
Right-to-Left Analysis: For example, a step in reading would be from 110 to 1110 (i.e. 6 to 14 in decimal). Here the analysis is trickier. Each new bit right-to-left has the effect of multiplying what comes after on the left. A 0, for example, multiplies what comes after (on the left) by 2, but the effect on the residue depends on the *parity* of the length the string on the right that is lead by the bit in question. A 1 in an odd-parity position (e.g. 1, 100, 10000, …) adds 1 to the residue, whereas a 1 in an even-parity position (e.g. 10, 1000, 100000, …) adds 2 to the residue. Fortunately, these are the only two possibilities. So as the machine reads right-to-left, it has to keep track of both the parity and the residue. However, this can be done by the same acceptor as before, with a revised meaning for the states:

State	Represents strings
a	having residue 0 mod 3
b	odd parity with residue 1 OR even parity with residue 2
c	even parity with residue 1 OR odd parity with residue 2

For example, 1110 (14 decimal) takes us to state b right-to-left (starting with 0) and is even parity with residue 2.

There are two other pathways to arrive at the same acceptor.

Pathway 1: Create an acceptor that keeps track of both parity and residue. Such an acceptor is shown below:



However, analyzing the states of this machine for equivalence shows that 0 ≡ 0′, 1 ≡ 2′, and 2 ≡ 1′, so this machine can be reduced to the first machine that we showed (with a ≡ 0 ≡ 0′, b ≡ 1′ ≡ 2, and c ≡ 2′ ≡ 1).

Pathway 2: We know that finite-state languages are closed under reversal. That is, for any finite-state acceptable language L, the language L^R consisting of all strings in L reversed, is also finite-state acceptable. This can be shown by starting with an acceptor M for L, reversing all the arrows, making the starting state accepting, and making each accepting state a starting state. The result is a "non-deterministic" finite-state machine, which, it can be shown, can be converted into a deterministic one.

The surprising thing about our original mod-3 machine is that applying this transformation, we get back exactly the same machine and the result is deterministic. This tells us that the same machine accepts both the left-to-right language and the right-to-left language, and in fact, the two languages are the same. We can get a hint about this by looking at the enumeration of strings and noticing the prevalent symmetries.

Questions to ponder:

1. For which numbers b is there a finite-state acceptor that accepts the multiples of b in binary?

2. Characterize the numbers b for which the same acceptor usable for both left-to-right and right-to-left?