From ternary to binary base and back

by N. MARTINS-FERREIRA AND T. VAN DER LINDEN

Abstract

We describe a two-state machine which converts a number given in ternary base to this number in binary base—or vice versa.

Ternary and binary base

Sometimes a number given in one base (say 2100012, base 3) has to be converted to another base (say base 2, then its expansion becomes 11010101010). We explain that—for use, for instance, in an electronic circuit—a two-state machine is all you need to efficiently convert a number between ternary and binary base. Moreover, this procedure is easily extended to other bases.

Expressing a natural number in binary base is a matter of repeatedly dividing the number by two and recording the remainders of those divisions. Indeed, the binary expansion of a natural number n is a finite string $x_k x_{k-1} \dots x_1 x_0$ of zeros and ones, so $x_i \in \{0,1\}$ for all $0 \le i \le k$, such that

$$n = \sum_{i=0}^{k} x_i 2^i.$$

When dividing this number by 2, we find $x_k \dots x_1$ as the binary expansion of the quotient, and x_0 as the remainder of the division, because

$$\begin{aligned} n &= \sum_{i=0}^{k} x_i 2^i = x_0 + \sum_{i=1}^{k} x_i 2^i = x_0 + \sum_{j=0}^{k-1} x_{j+1} 2^{j+1} \\ &= x_0 + 2 \cdot \sum_{j=0}^{k-1} x_{j+1} 2^j. \end{aligned}$$

Repeating this process k times on the number n, we obtain its entire binary expansion.

When the number is given in ternary base, the division itself becomes a particularly simple process which may be performed by a small two-state machine. Moreover, this process is reversible, so that converting a binary number to ternary base may be done by running the same machine backwards.

Finite state machines

A *finite state machine* is a machine with a finite number of inputs and outputs and a finite number of internal states. This is usually modeled as a five-tuple

$$M = (I, O, S, s_0, w)$$

where I and O are the respective sets of inputs and outputs, and S is the set of states. $s_0 \in S$ is a chosen initial state for M: the state to which it returns when the machine is reset. The "internal wiring" of the machine M is captured in the transition function $w\colon I \times S \to O \times S$. For a given input $i \in I$ and a given state $s \in S$ it returns a pair $(o,t) = w(i,s) \in O \times S$, where o is the machine's output and t is its new state. Since the sets I, O and S are all finite, the transition function w may be conveniently expressed in a table or in a diagram.

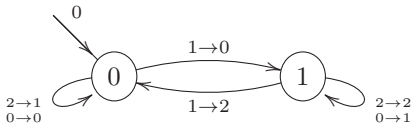
A two-state machine

As explained above, a conversion between ternary and binary base is done by dividing a given ternary number by two again and again, each time recording the remainder of this division. Dividing a ternary number by two can be done by repeatedly applying a small machine M_1 to it which knows how to divide a ternary digit (0, 1 or 2) by two. At each step, the output should be the quotient of the division, while the remainder is remembered as an internal state.

So we have $\{0,1,2\} = I = O$ as set of inputs and outputs and $S = \{0,1\}$ as set of internal states for our M_1: the remainder of a division by 2 is either 0 or 1. The state 0, which is also the machine's initial state s_0, means that in its last run the remainder of the division was 0. The transition function w is given by the table

w	0	1
0	$(0,0)$	$(1,1)$
1	$(0,1)$	$(2,0)$
2	$(1,0)$	$(2,1)$

while the entire machine may be pictured in one diagram as follows.



(1)

An arrow from state s to state t labeled with $i \to o$ is to be understood as the behavior of the machine when it reads input i with internal state s, producing output o and changing to internal state t.

Given $(i,s) \in I \times S$, the output o is the quotient, and the new internal state t is the remainder, of $s \cdot 3 + i$ divided by 2. For instance, 1 divided by 2 is 0 with remainder 1. If now the machine remembers a remainder 1 from the previous division and the input is again 1, then we have $1 \cdot 3 + 1 = 4$ (base 10) to divide by 2, which is 2 with remainder 0. Indeed 11 (base 3) divided by 2 is 2 (base 3).

The following example illustrates the entire procedure for the number 2100012 (base 3, which is 1706 in

base 10).

2	1	0	0	0	1	2	
1	0	1	1	1	2	1	0
0	1	2	0	2	1	0	1
0	0	2	1	2	2	0	0
0	0	1	0	2	2	1	1
0	0	0	1	2	2	2	0
0	0	0	0	2	2	2	1
0	0	0	0	1	1	1	0
0	0	0	0	0	2	0	1
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1

The result of the process is the binary word 11010101010, the right hand side column of remainders read from bottom to top. The machine passes over the first row from left to right, which produces the quotient 1011121 and remainder 0, the machine's final internal state. Then it is reset, after which it passes over the quotient in the second row to produce the number 0120210 and the remainder 1. This continues until the quotient is zero as in the last row.

Reversing the process

To obtain the ternary expansion of a number given in base 2, it suffices to reverse the process explained above. Running the machine M_1 backwards (from right to left) on the last row of zeros in the example, starting with initial internal state 1, gives the next-to-last row 0000001. Thus we move up in the table until we arrive at its first row and the given binary number is "used up".

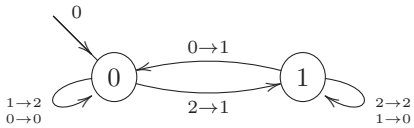
The machine M_1 can indeed be run backwards, as its transition function w is a bijection from $I \times S$ to $O \times S$. This corresponds to reversing the arrows in the diagram (1). But instead of one two-state machine we now need two, depending on the internal state we have to start in. We get

$$M_{20} = (O, I, S, 0, w^{-1})$$

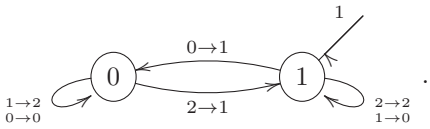
and

$$M_{21} = (O, I, S, 1, w^{-1}),$$

which in diagrammatic form become



and



.

The given binary number tells us which machine we have to run: on the two last rows we would use M_{21} and then we use M_{20} on the next row.

Other bases

It is easy to extend this procedure and construct finite state machines which convert numbers from any given (integer) base to any other base. Of course when one base divides the other (as in the case of hexadecimal-to-binary conversion, for instance) an immediate conversion is possible, but otherwise some kind of division needs to be done.

Suppose we want to convert from base p to base q with $p > q$. Then the machine M_1 can be extended to a q-state machine M_3 with p inputs and outputs, the digits $0, \ldots, p-1$, and q internal states $0, \ldots, q-1$. Given input $i \in \{0, \ldots, p-1\}$ and internal state $s \in \{0, \ldots, q-1\}$, the transition function w gives back as the output the quotient of $sp + i$ by q and changes the internal state to the remainder of this division.

Ternary computers

Nowadays computers are generally binary. There seems, however, to be evidence [1] that ternary computers would be more efficient, while still not being too different in their basic set-up. In fact, from an information theory point of view, the most efficient base is e. But since for many reasons integer bases (digital computers) are preferable, the optimal choice is 3, which is closer to $e \approx 2.718$ than 2. If ever ternary computers become a real option, then conversion between ternary and binary base will be essential in making communication between the two architectures possible.

References

[1] B. Hayes, Third base, American Scientist **89** (2001), no. 6, 490.

N. Martins-Ferreira, IPleiria
T. Van der Linden, UCLouvain