2 Turing Machines – Busy Beavers, Coding, and the Halting Problem

2.1 Coding TMs; Gödel numbering

Suppose the *symbols* for our TM are from

$$\Sigma = \{b, 1, s_2, s_3, \cdots\} \qquad \text{(identify } b = s_0, 1 = s_1\text{)}$$

and the *states* are from

$$Q = \{q_0, q_1, q_2, \cdots\}$$

(Of course, any particular TM has only finitely many symbols and finitely many states.)

The the set of *actions* is

$$A = \{L, R, b, 1, s_2, s_3, \cdots\} = \{a_0, a_1, a_2, \cdots\} \qquad \text{(say)}$$

We can then view any TM as a finite string (or word) from $A \cup Q$ of the form

$$\mathbf{w} = q_{i_1} s_{j_1} a_{k_1} q_{\ell_1} q_{i_2} s_{j_2} a_{k_2} q_{\ell_2} \dots q_{i_n} s_{j_n} a_{k_n} q_{\ell_n}$$

Note:

- The same TM can be represented by different words, for example by reordering the quadruples or by permuting the states.
- Two *completely different* TMs (ie, where the action of the machines are entirely different) might have the same behavior, and in particular might compute the same function.

We can then *code* all TMs built from these states and symbols as a natural number in a natural way; given a TM M represented by the word $\mathbf{w}$ above, put

$$\ulcorner M \urcorner = 2^{i_1+1} 3^{j_1+1} 5^{k_1+1} 7^{\ell_1+1} 11^{i_2+1} 13^{j_2+1} 17^{k_2+1} 19^{\ell_2+1} \dots \pi_{4n-3}^{i_n+1} \pi_{4n-2}^{j_n+1} \pi_{4n-1}^{k_n+1} \pi_{4n}^{\ell_n+1}$$

where π_n is the n^{th} prime number.

What is important about this coding is (i) that it is one-to-one; (ii) that it is *effective* or *computable* in the sense that there is an algorithm or machine that takes M to $\ulcorner M \urcorner$; and (iii) that it is effectively invertible in the sense that given a natural number m, we can determine whether there is a machine M such that $m = \ulcorner M \urcorner$, and – if so – algorithmically recreate M from m.

Other than that, we could have used any other reasonable way to code machines. For example, we could have started by coding just quadruples $\mathbf{z}$ by

$$\ulcorner q_i s_j a_k q_\ell \urcorner = 2^{i+1}3^{j+1}5^{k+1}7^{\ell+1}$$

then coding a sequence of such quadruples by

$$\ulcorner \mathbf{z}_1\mathbf{z}_1\mathbf{z}_2\ldots\mathbf{z}_n\urcorner = 2^{\ulcorner \mathbf{z}_1\urcorner}3^{\ulcorner \mathbf{z}_2\urcorner}\ldots\pi_n^{\ulcorner \mathbf{z}_n\urcorner}$$

This is a *different* coding, but just as good.

2.2 The Busy Beaver problem

We now address the following question:

Are all functions from $\mathbb{N}$ to $\mathbb{N}$ computable?

The answer is obviously NO, since there are uncountably many such functions, but the set of all Turing Machines must be countable: after all, we can encode them with natural numbers!

In this section we consider an interesting example of an uncomputable function.

In 1962, Tibor Radó posed the following question: For any n let $B(n)$ be the largest integer constant that can be computed by a TM with at most n states. In other words: if you start a machine with n states on a blank tape, it might not halt; it might halt on something that isn't an integer in monadic notation; or it might halt on a block of $m+1$ 1s, in which case it has computed the constant m. Looking at *all* machines with at most n states, $B(n)$ is the largest value of m that can be computed by such a machine.

Radó's question was: Is B a computable function?

This is called the *Busy Beaver Problem*, because a machine that generates a large m with a small number of states would have to be dashing back and forth like a busy beaver. Or so it was said.

B is sometimes called a *productivity* function, as it gives a measure of how productive a machine with only n states can be.

The answer is: No, B is *not* computable.

Incidentally, the actual function $B(n)$ obviously depends on what symbols we allow, and whether we are talking about quadruple TMs or quintuple TMs. In Radó's case - and ours - we work with quadruple machines and only the symbols b and 1.

Begin the proof with some observations:

- B is a nondecreasing function. Increasing the number of states available doesn't *eliminate* any machines you can construct with the smaller number of states, so you can produce at least as large an output.
- In fact, B is strictly increasing. If $B(n) = m$, let M be a machine that produces the output m. If q_k is the final state of that machine, there can't be a quadruple in M that looks like $q_k 1xx$ since otherwise M would not have halted on the first 1 in the block of $m+1$ 1s. Create a new machine M' by taking the description of M and adding the two quadruples

  $$q_k 1 L q_r q_r b 1 q_r$$

  where q_r is a new state that hasn't appeared in the description of M. Then M' has $n+1$ states and outputs $m+1$, so $B(n+1) \geq B(n)+1$

Now, suppose (for a contradiction) that B is computable by a machine M. Let:

- γ =the number of states in the machine M
- δ = the number of states in a "doubler" machine, one which starts on a block of n 1s and ends on a block of $2n$ 1s (ie, a machine that computes $(n-1) \mapsto 2n-1$). We already know that we can build such a machines with a fairly small, constant number of states, something like $\delta = 13$.
- M_n be a machine with $n+2$ states that starts with a blank tape and puts down n 1s, ie computes the constant $n-1$. We've already seen how to do this in the early examples of TMs.

Now, for any n on a blank tape we run the machine M_n, then the doubler machine, then the machine M that computes B.

This combination machine uses a total of at most $n+2+\gamma+\delta$ many states, and leaves $B(2n-1)$ as the output on the tape. In other words,

$$B(n+2+\gamma+\delta) \geq B(2n-1)$$

But for large enough n,

$$n+2+\gamma+\delta < 2n-1,$$

so

$$B(n+2+\gamma+\delta) < B(2n-1)$$

a contradiction.

Incidentally, $B(n)$ has been computed for several values of n. It is known that $B(3) = 6$, and that

$$B(12) \geq 6 \cdot \underbrace{4096^{4096^{4096^{4096^{4096^{\cdots^{4096^4}}}}}}}_{166 \text{ times}}$$

2.3 Halting problem

Another way to build a non-computable function is to emulate Russell's proof that there is no set of all sets, or Cantor's proofs that the set of real numbers is uncountable. This kind of argument is called a *diagonal argument*.

The informal question that forms the basis of this construction is, given a machine M and an integer n, can we tell in advance whether M will halt on input n?

Define $h(m, n) = \begin{cases} 1, & \text{if there is a machine } M \text{ with } \ulcorner M \urcorner = m \text{ such that } M \\ & \quad \text{halts on input } n; \\ 0, & \text{otherwise.} \end{cases}$

The *Halting Problem* is to determine of h is computable.

Answer: No, h is not computable.

Proof Suppose (for a contradiction) that h is computable by a machine M_h.

Let $g(n) = \begin{cases} 1, & \text{if } h(n, n) = 0; \\ \text{undefined}, & \text{otherwise.} \end{cases}$

So show that g is representable by a TM, we need two helper machines:

- M_1 is a machine that, on input k, either halts on 1 (that is, 11 in monadic notation) if $k = 0$, otherwise it does not halt.
- M_2 is a machine that starts on input n, and duplicates it so that it halts with output (n, n) (that is, two blocks of $n + 1$ 1s separated by a space).

It is an exercise to show that such machines exist.

Now, on input n we run M_2, then M_h, then M_1. It is easy to see that this composite machine (call it M_g) computes g.

Now, let $e = \ulcorner M_g \urcorner$. (We could let e be the code of any machine computing g.)

If $g(e) = 1$ then $h(e, e) = 0$, so M_g does not halt on input e, so $g(e)$ is undefined.

If $g(e) \neq 1$ then $g(e)$ is undefined, so $h(e, e) \neq 0$, so $h(e, e) = 1$, so M_g halts on input e, $g(e)$ is not undefined.

Either way there is a contradiction.