The Busy Beaver Problem

Deepak Sirone J Arpan Kapoor

June 7, 2016

A function is merely an abstraction used by mathematicians. It is defined by its input type called the domain, output type called the range and suitable definitions of the domain and the range. A suitable definition is one which does not cause ambiguity, that is one which does not render itself to multiple interpretations.

For example:

1. $f(x) = x \times x$, where $x \in \mathbb{R}$. Here we assume that the operation $\times$ is closed in $\mathbb{R}$
2. $f(x) = \{e \mid \forall k \in x, e \geq k\}$ where x is a finite subset of natural numbers

Many a time we are interested in computing the output of a function using an algorithm. The description of the first example in this case, implicitly gives us an algorithm for computing the output if the algorithm for performing the operation $\times$ is known. The second example intuitively asks us to iterate through the elements in the set and find the largest element in it.

However, the notion of a function does not define the algorithm to get from the input to the output. This means that using this definition of functions, you have the freedom to define your own functions, provided you can specify the domain and range precisely. So a function which outputs the number of sand grains in the Sahara desert at a particular instant of time is perfectly legal! This however does not ensure that there exists an algorithm to do it.

The example functions above, are definable by an algorithm. Such functions are called computable functions. There also exists functions which can be proved to not have an algorithm and they are called non-computable functions. One such function known as the Busy Beaver function was defined by Timos Rado in his 1961 paper "On Non-Computable Functions"[6].

The busy beaver function is concerned with a model of executing algorithms known as Turing machines. A Turing machine is a machine which has an infinite bidirectional input/output tape, a tape head and rules governing the movement of the tape head on the tape. The tape is divided into cells and the symbols written on the tape are from an alphabet Σ. Similar to a finite state machine, a Turing machine has a set of states, Q and a transition function δ. The tape head reads one symbol on the tape at a time and based on the transition function, overwrites the symbol with another symbol and shifts (steps either right or left on the tape [1]. The Turing machine also has a halt state whereby the machine stops running. The running time of a Turing machine on an input is measured as the number of shifts it performs before it enters its halting configuration. A binary Turing machine is one with $\Sigma = \{0, 1\}$.

Not all Turing machines stop running on its given input. The problem of determining whether a Turing machine M halts on input x is known as the halting problem. The halting problem has been shown to be undecidable, meaning that there is no Turing machine (or algorithm for that matter) to solve it. We could try simulating a given machine, to see if it stops after a while.

[1] The Turing machine which we define here does not have a center shift, meaning that the tape head must move left or right after each move

But the problem is that we don't know for how long the simulation should continue. Suppose we simulate a Turing machine M on input x using another Turing machine [2] for t steps and conclude that it does not halt, we can always find another Turing machine M' which halts on input x in $t+1$ steps. Hence simulation does not work.

Turns out that the Turing machine model can simulate circuits (both sequential and combinational) and hence we can safely conclude that it is capable of simulating all algorithms that can be run on a real computer.

We call a binary Turing Machine a k-state Busy Beaver if on an empty tape as input it halts and runs for at least as many steps as any other k-state binary Turing machine.

In order to find a k-state Busy Beaver, we would have to evaluate all k-state binary Turing machines. How many such Turing machines exist? Since we have fixed the cardinality of Q and the alphabet ($\Sigma = \{0, 1\}$), the number of possible Turing machines is equal to the number of possible transition function. The transition function's domain is the cross product of the set of states with the alphabet and codomain is the cross product of the set of states, the alphabet and the set $\{L, R\}$.

$$\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$$
$$|Q| = k$$
$$|\Sigma| = 2$$
$$|\{L, R\}| = 2$$

From the above, we see that there are $(2 \times 2 \times k)^{2k}$, i.e. $(4k)^{2k}$ number of k-state binary Turing machines.

The Busy Beaver function's input is a positive integer n and it outputs the number of steps taken by a n-state Busy Beaver Turing machine to halt. Assuming this function to be computable gives us the following algorithm to solve the halting problem, which we know is not possible. Hence our assumption must be wrong.

function HALTS?(M, x)
    k = |M.Q| ▷ Number of states of the Turing machine M
    bbk = BB(k) ▷ Number of steps taken by a k-state Busy Beaver
    if UTM(M, x, bbk) **then return** HALTS ▷ UTM simulates M on input x for bbk steps
    else return DOES NOT HALT
    end if
end function

Another interesting fact that can be observed using the same argument is that the Busy Beaver function grows faster than all computable functions, i.e. $f(n) = O(BB(n))$ for any computable function f. This is because if some computable function f grew faster than the Busy Beaver function, the function call to BB in the above algorithm can be replaced with a call to this function f, enabling us to solve the halting problem. Again our assumption would be wrong. Table (xx) shows what we know currently about the busy beaver series.

	2 State	3 State	4 State	5 State	6 State
BB	6	21	107	$\geq 47,176,870$	$> 7.4 \times 10^{36534}$

So far only we know the exact values of the busy beaver series upto 5 state Turing machines. The Busy Beaver series suggests a new method for solving theoretical problems in mathematics. For example, suppose we have a Turing machine M which verifies Goldbach's conjecture (which

[2] called the Universal Turing Machine or UTM

states that every even number n, $n \geq 4$ is the sum of two primes) and it has k states. If we were able to find $BB(k)$ exactly then we could run M for $BB(k)$ steps and conclude that the conjecture holds if it does not stop within that many steps, else we conclude that the conjecture does not hold.

References

[1] Scott Aaronson. *Who Can Name the Bigger Number?* 1999. URL: http://www.scottaaronson.com/writings/bignumbers.html.

[2] John Baez. *The Busy Beaver Game*. May 21, 2016. URL: https://johncarlosbaez.wordpress.com/2016/05/21/the-busy-beaver-game/.

[3] *Busy beaver*. In: *Wikipedia, the free encyclopedia*. May 11, 2016. URL: https://en.wikipedia.org/w/index.php?title=Busy_beaver&oldid=719770898.

[4] *Busy Beaver Turing Machines - Computerphile*. Sept. 2, 2014. URL: https://www.youtube.com/watch?v=CE8UhcyJS0I.

[5] Peteris Krumins. *The Busy Beaver Problem*. Oct. 29, 2009. URL: http://www.catonmat.net/blog/busy-beaver/.

[6] T. Rado. "On non-computable functions". In: *The Bell System Technical Journal* 41.3 (May 1962), pp. 877–884. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1962.tb00480.x.

[7] Michael Sipser. *Introduction to the theory of computation*. 3rd Ed. Boston, MA: Course Technology Cengage Learning, 2012. ISBN: 9781133187790.

[8] Adam Yedidia and Scott Aaronson. "A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory". In: (May 13, 2016). arXiv: 1605.04343 [cs]. URL: http://arxiv.org/abs/1605.04343.