Beyond Busy Beaver

Over the next few rounds, the duel became a search for more and more powerful generalizations of the notion of a Turing Machine.

Imagine equipping a Turing Machine with a "halting oracle": a primitive operation that allows it to instantaneously determine whether an ordinary Turing Machine would halt on an empty input.

Call this new kind of machine a Super Turing-Machine.

We know that the Busy Beaver function is not Turing-computable. But, as you'll be asked to verify in an exercise below, it can be computed using a Super Turing-Machine. As you'll also be asked to verify below, this means that for any ordinary Turing Machine with sufficiently many states, there is a super Turing Machine that has fewer states but is much more productive. This means that the function $BB_1(n)$—i.e. the Busy Beaver function for super Turing Machines—can be used to express numbers which are much bigger than $BB(10^{100})$(for instance:$BB_1(10^{100})$).

As one might have expected, no super Turing Machine can compute $BB_1(n)$.

But we could compute this function using a super-duper Turing Machine: a Turing Machine with a halting oracle for super Turing Machines. And $BB_2(10^{100})$—the Busy Beaver function for super-duper Turing Machines—can be used to express numbers which are much bigger than $BB_1(10^{100})$.

It goes without saying that by considering more and more powerful oracles, one can extend the hierarchy of Busy Beaver functions further still. After $BB(n)$ and $BB_1(n)$ come $BB_2(n)$,$BB_3(n)$, and so forth. Then come $BB_\omega(n)$, $BB_{\omega+1}(n)$, $BB_{\omega+2}(n), \ldots, BB_{\omega+\omega}(n), \ldots, BB_{\omega+\omega+\omega}(n), \ldots, BB_{\omega\times\omega}(n), \ldots BB_{\omega\times\omega\times\omega}(n) \ldots$. And do forth. (It is worth noting that even if α is an infinite ordinal, $BB_\alpha(10^{100})$ is a finite number and therefore a valid entry to the competition.)

The most powerful Busy Beaver function that Adam and I considered was $BB_\theta(n)$, where θ is the first non-recursive ordinal---a relatively small infinite ordinal. So the next entry to the competition was $BB_\theta(10^{100})$. And although it's not generally true that $BB_\alpha(10^{100})$ is strictly larger than $BB_\beta(10^{100})$ when $\alpha > \beta$, it's certainly true that $BB_\theta(10^{100})$ is much, much bigger than $BB_1(10^{100})$, which had been our previous entry.

The last entry to the competition, the winning entry, was a bigger number still:

> The smallest number with the property of being larger than any number that can be named in the language of set theory using 10^{100} symbols or less.

This particular way of describing the number wouldn't have been a valid entry to the competition because it includes the expression "named", which counts as semantic vocabulary and is therefore ruled out. But the description that was actually used in the competition didn't rely on forbidden vocabulary. It instead relied on a second order language: a language that is capable of expressing not just singular quantification ("there is a number such that it is so and so") but also plural quantification ("there are some numbers such that they are so and so"). Second-order languages are so powerful that they allow us to characterize a non-semantic substitute for the notion of being named in the language of set theory using 10^{100} symbols or less.

And what if we had a language that was even more expressive than a second-order language? A third-order language -- a language capable of expressing "super plural" quantification -- would be so powerful that it would allow us to characterize a non-semantic substitute for the notion of being named in the language of *second-order* set theory using 10^{100} symbols or less. And that would allow one to name a number even bigger than the winning entry of our competition. Our quest to find larger and larger numbers has now morphed into a quest to find more and more powerful languages!

Video Review: Beyond the Busy Beaver

0:00 / 0:00 Speed 1.50x

Once we've gone through every Turing Machine,
we'll have the largest output that we can get.
That's exactly right.
So the whole problem for these things
is that you don't know whether the machines are going to halt.
Because we know that we can simulate machines.
So it's in principle possible to say,
well you just go one by one.
But of course, if you get trapped
into one that never halts, then your procedure
is never going to terminate.
But, if, as you suggested, you first
use your magic box to check and see
whether they're going to halt, then this procedure
is well defined.
It's always going to terminate, and you can calculate the Busy
Beaver Function.

Problem 1

1/1 point (ungraded)
Give an informal description of how a super-Turing Machine might compute $BB(n)$.

☑ Done

✔

Explanation
Here is an informal a description of how such a super Turing Machine M could work:

Step 1
M enumerates all possible Turing Machine programs of n symbols or less. (There are finitely many such programs.)

Step 2
For each such program, M uses its oracle to check whether the program halts when given a blank input tape.

Step 3
For each program that does halt, M acts like a universal Turing Machine to simulate its program and print out its output.

Step 4
M then compares all such outputs, and chooses the biggest as its own output.

Submit

Answers are displayed within the problem

Problem 2

1/1 point (ungraded)
True or false?

There is a number k such that for any ordinary Turing Machine with more than k states, there is a super Turing Machine that has fewer states but is much more productive.

- (●) True
- ( ) False

✔

Explanation
Suppose that you have an ordinary Turing Machine machine M, and with $\overline{M}$ states. We can build a super-Turing Machine M_1, that is much, much more productive than M, as follows:

Step 1
M_1 writes $\overline{M}$ (or more) ones on the tape.

Step 2
M_1 applies the busy beaver function to the contents of the tape a couple of times.

The first of these steps can be carried out using approximately $log\,(\overline{M})$ steps (since M_1 can use $log\,(\overline{M})$ states to write out $log\,(\overline{M})$ ones, and then use a constant number of steps to apply the function 10^x to that number). The second of these steps requires a constant number of states, since M_1 can use a super Turing Machine computing the Busy Beaver Function as a subroutine. This means that M_1 requires only $log\,(\overline{M}) + c$ states, where c is a constant. So as long as $\overline{M}$ is bigger than some fixed k such that $k > log\,(k) + c$, M_1 will have fewer states than M.

Submit

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: Week 9 / Beyond Busy Beaver

Add a Post

Show all posts ⌄ by recent activity ⌄

There are no posts in this topic yet.

✖