9.4 Optional: The Big Number Duel[19]

Imagine the following competition: two contestants, one blackboard, biggest number wins. My friend Adam Elga, a philosophy professor at Princeton University, challenged me to such a duel after reading Scott Aaronson's "Who Can Name the Bigger Number?". This optional final section is a blow-by-blow account of our mighty confrontation.

9.4.1 The rules of the game

We began by fixing some rules. The first rule is that we would take turns writing numbers on the board, and that the last person to write down a valid entry would be the winner. The second rule is that only finite numbers would be allowed. The number 17, for example, would count as a valid entry, as would the number 1010101010. But $\omega + \omega$ would not. The third rule is that semantic vocabulary—expressions like "names" or "refers to" or "is true of"—would be disallowed. This restriction is crucial. Without it, Adam would have been in a position to write "the biggest number ever named by Agustín, plus one" on the board, and win on the first round. (Worse still: I would have been able to retort with "the biggest number ever named by Adam, plus one".)

Finally, we agreed not to engage in unsporting behavior. Each time one of us wrote a number on the board, the other would either recognize defeat or respond with a much bigger number. How much bigger? Big enough that it would be impossible to reach it in practice (e.g. before the audience got bored and left) using only methods that had been introduced in previous stages of the competition. That means, for example, that if Adam's last entry was "101010", it would be unsporting for me to respond with "10101010". We wanted the competition to be a war of originality, not a war of patience!

9.4.2 The first few rounds

The competition took place at MIT. As the hometown hero, I got to go first. Without thinking too much about it I wrote down a sequence of thirty or forty ones

$$111111111111111111111111111111$$

"We're still warming up", I thought. But my first effort proved disastrous. Adam approached the board with an eraser, and erased a line across all but the first two of my ones, leaving

$$11\,!$$

"Eleven factorial, factorial, factorial, ...", he declared triumphantly. To get a sense of just how big this number is, note that 11! is 39,916,800, and that 11!! is approximately $6 \times 10^{286,078,170}$ (which is much more than the number of particles in the universe). 11!!! is

[19] An earlier version of this material appeared in "El duelo de los números grandes", A. Rayo, *Investigación y Ciencia*, August 2008.

so big that it cannot be written using an expression of the form "10^n" in a practical amount of time, where "n" is a numeral in base 10. So 11!!! cannot be reached, in practice, by writing a long sequence of ones on the blackboard. And Adam's entry had many more factorials than that. There was no question that Adam had named number much bigger than mine.

Fortunately, I was able to remember the Busy Beaver function, $BB(n)$, and made my next entry $BB(10^{100})$: the productivity of the most productive Turing Machine with a googol states or less. How does this number compare to Adam's last entry? Well, it is possible to write a relatively short Turing Machine program that computes the factorial function, and outputs the result of applying the function 30 or 40 times, starting with 11—or, indeed, the result of applying the function $10^{10^{10^{10}}}$ times starting with 11. I don't know how many states it takes to do so, but the number is significantly smaller than 10^{100}. So $BB(10^{100})$ must be bigger than Adam's last entry.

In fact, it is much, much bigger. $BB(10^{100})$ is a truly gigantic number. Every Turing Machine program we can write in practice will have fewer than 10^{100} states. So no matter how big a number is, it will be smaller than $BB(10^{100})$, as long as it is possible in practice to program a Turing Machine to output it.

9.4.3 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. And, 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: $BB_1(10^{100})$).

No Super Turing Machine can compute $BB_1(n)$. But we could compute this function using a Super-Duper Turing Machine: a Turing Machine equipped with a halting oracle for Super Turing Machines. $BB_2(10^{100})$—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 $BB_\omega(n)$, $BB_{\omega+1}(n)$, $BB_{\omega+2}(n)$,… $BB_{\omega+\omega}(n)$,…, $BB_{\omega+\omega+\omega}(n)$,…,$BB_{\omega\times\omega}(n)$,… $BB_{\omega\times\omega\times\omega}(n)$,…. 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.

9.4.4 The winning entry

The last entry to the competition 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 would have been disallowed in 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 did not 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 standard language of set theory.

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!

Exercises

1. Give an informal description of how a Super Turing Machine might compute $BB(n)$.
2. Show 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.

9.5 Conclusion

We have discussed the notion of a Turing Machine and identified two functions that fail to be Turing-computable: the Halting Function and the Busy Beaver Function.

Since a function is computable by an ordinary computer if and only if it is Turing-computable, it follows that neither the Halting Function nor the Busy Beaver Function can be computed by an ordinary computer, no matter how powerful. On the assumption