Waiting for Gödel

Abstract
The concept of computation is familiar to most of us regardless of any mathematical or engineering background: computing could be intuitively described as doing something in an organised/programmed manner. However, it is only in the middle of the 20th century that computing became a subject of interrogation in itself. The question switched from "what is the result of this or that computation?" to one step behind: "what does it mean to compute?". Quite a number of scientists came up with formal answers to that question: Alonzo Church with lambda calculus in the 30s, Stephen Cole Kleene with general recursive functions in 1936, Alan Turing with Turing machines in 1936, Emil Post with tag systems in 1943, von Neumann with the RAM model in 1945 and many more after that.

Surprisingly, all these models of computation, despite being very different looking, turn out to be equivalent: they all can simulate one another. This realisation gives more sense to the informal sentence "the concept of computation is familiar to most of us" and is embodied in the philosophical Church-Turing thesis: "A function is computable if and only if it is computable by a Turing machine [or any other of the above models]".

Approximately at the same time of these discoveries, in 1931, Kurt Gödel published his incompleteness theorems which, roughly speaking, state that there exist arithmetical statements which are true in the natural numbers but that you cannot prove using simple tools such as inductive proofs and common knowledge on addition and multiplication. Worse, even if you try to use more complex tools, let's say that you 'level up' to Set Theory, there will always be such statements that are true in the natural numbers, but not provable with your gear [1].

Surprisingly, again, the fact that some statements are true but not provable is intimately related to the existence of uncomputable functions: functions that no Turing machines, or any other physical model of computation, can compute.

Even more surprisingly, this link between computability and proofs yields to the creation of machines of which behavior goes far beyond current Human mathematical knowledge [2], widening the gap between what is true and what we can know to be true. Meaning that you can look at such machines for your entire life with a very slim hope of ever figuring out (a) what the machine is doing (b) if the machine will ever stop. And worse, like a challenge to our mathematical pride, we suspect that very small machines are such incomprehensible demons [3].

The goal of this talk is first to give a very concrete presentation of what Turing machines are: an archetypal programming language running on an ideal computer. Then, we'll make the link with proof theory and use our knowledge of Turing machines to prove Gödel incompleteness theorems. From there, we'll exhibit concrete examples of Turing machines whose behavior is independent of common mathematical theories and we'll take a look at very small machines that no one has explained yet.

References
[1] *Christopher C. Leary. Lars Kristiansen. 2015. A Friendly Introduction to Mathematical Logic (2nd. ed.). Geneso, NY.*

[2] *Adam Yedidia. Scott Aaronson. 2016. A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory. (preprint)* https://arxiv.org/abs/1605.04343

[3] *Scott Aaronson. 2019. The Busy Beaver Frontier. (preprint)* https://www.scottaaronson.com/papers/bb.pdf