1 Introduction

Up to now I've been taking it as a given that 'recognising a language' is a reasonable proxy for 'doing computation'. This is not really a technical point – after all we're free to make what definitions we like, but it's probably sensible to make some sort of a reasonableness argument for it, and to explore the consequences of it.

2 Computing the square root of 2

The problem: *compute the square root of 2* is certainly a computational problem. How does it correspond to a problem of language recognition?

A natural response to being asked to compute $\sqrt{2}$ is to ask: how accurately? That is, we recognise that the answer is in the form of a string of some length. So, it's easy enough to associate a language to this: the language consisting of all finite strings that represent $\sqrt{2}$ to k digits of accuracy, for any positive integer k, i.e., the language consisting of:

$$1, 1.4, 1.414, 1.4142, ...$$

I don't care whether you want them truncated or rounded - that's just a detail, it's still a language.

But that's a real straw man since we'd normally think of 'computing square roots' as the computational problem, rather than 'computing a specific square root'.

3 Computing square roots

So now the computational problem is: *Given a number, compute its square root*. Again, we have to add the caveat to any desired degree of accuracy.

But this is just as easy as a language recognition problem – we take our inputs to be like the buttons on a calculator, i.e., digits (possibly a single decimal point), followed by a square root button. But then, rather than seeing the output on the machine, we just continue with the string representing the answer. So, using S for the 'square root button'

the language we're interested in contains strings like this:

$$\begin{aligned} 4S2 \\ 4.0S2.000 \\ 2S1.4142 \end{aligned}$$

but presumably not strings like this

$$\begin{aligned} 4S2S \\ 4S3 \\ 2S1.4143 \end{aligned}$$

I think it's easy to see how this argument extends to cover numerical computation more generally. Indeed almost anything of the *what is the answer to this question* variety fits this model as the language we consider is just a language consisting of strings representing a valid question, followed by some separator, followed by strings consisting of the answer to the preceding question.

4 How are things represented?

One thing we do insist on is that our model of computation should be finitistic. That is, we consider only computational devices that can be described in a finite amount of space over some finite alphabet, and languages consisting of finite strings over some finite alphabet. These alphabets need not be the same. We might well find it convenient to work with languages over a binary alphabet, and allow much greater latitude in describing our machines.

5 From integers to strings

Consider the two-character alphabet $\{0, 1\}$. Any finite string over this alphabet can be interpreted as a non-negative integer by just reading it as a binary number. However, there's a minor technical issue with this in that all of 0, 00, 000, 0000 would read as 0. And, what would the empty string read as? So, just to keep things tidy, we could index

the finite strings over this alphabet by positive integers where the string associated to a positive integer k is the string obtained by writing k in binary and then adding a leading 1. So now 0 is represented by 10_2, i.e., 2, while 000 is represented by 1000_2 i.e., 8, and the empty string is represented as 1_2 i.e., 1.

We can do the same thing for any alphabet and will implicitly assume that we have done so. To be precise, we associate to any alphabet Σ an indexing method such that each sting $w \in \Sigma^*$ is associated to a positive integer w_Σ in such a way that distinct strings are associated with distinct integers. In reverse, for each positive integer n there may be (or may not be) an associated string $\Sigma(n) \in \Sigma^*$ such that $\Sigma(n)_\Sigma = n$.

6 Not everything is computable

Now we aim to show that in our finitistic model (whatever it might be) that there exist languages L that are not recognised by any computational device. Let Σ be the alphabet of our languages, and Γ be the alphabet that we use to represent our machines.

The trick is the same one used by Cantor to establish that there are different types of infinity (in the context of arguing that there exist infinite sets A and B such that no map from A to B can cover every element of B), and akin to that of Russell's paradox which shows that one cannot legitimately consider things like 'the set of all sets'.

Our computational devices are represented as strings over Γ. We could also think of them as being represented by positive integers n – where n represents the device $\Gamma(n)$ (for convenience, assume this is always defined so that we don't have to keep saying 'whenever this makes sense').

Each element $w \in \Gamma^*$ recognises some language that we will denote $L(w)$ which is a subset of Σ^*. But we could also think of this language as a subset of the positive integers namely:

$$N(w) := \{n \in \mathbb{N} : \Sigma(n) \in L(w)\}$$

Now, for a positive integer n consider the question does n belong to $N(\Gamma(n))$? This is a yes/no question about positive integers and so defines a set of positive integers consist-

ing of those to which the answer is 'no'. That is, define

$$L = \{n \in \mathbb{N} : n \notin N(\Gamma(n))\}$$

I claim this language is not computable, i.e., is not of the form $N(w)$ for any w. For, suppose that $L = N(w)$. Consider $n = w_\Gamma$ and ask the question: *is n in L*?

If the answer were yes, then by the definition it would be the case that $n \notin N(\Gamma(n))$. But, $\Gamma(n) = w$ so $N(\Gamma(n)) = L$. That is, the answer would be no! That's impossible.

On the other hand, if the answer were no, then by the definition it would be the case that $n \in N(\Gamma(n))$. But, $\Gamma(n) = w$ so $N(\Gamma(n)) = L$. That is, the answer would be yes! That's impossible too.

So we have a contradiction - and the only thing we've assumed is that L is a computable language. Therefore, it isn't, and uncomputable languages exist.

7 But that's not fair!

Our description of an uncomputable language essentially required us to consider all computing devices simultaneously. For each n we used a different computing device to determine whether to include n in L or not. That seems unfair.

We'll need to find a rather more convincing and detailed model of computation to be able to demonstrate languages that are reasonable to talk about, but which we can show cause similar paradoxes.

In particular, we'll need to be able to show that our model includes the possibility of a universal machine: a single machine/device that can simulate all the others.

8 Exercises

Some soft questions.

1. Why is the alphabet not really a significant issue?
2. Think about other computational problems and whether or not the language recognition model can be stretched to accommodate them.
3. Why should we be comfortable with the notion of a universal machine?