Analysis

Edwin Park

2023

Contents

1. Recursive Functions

1.1. The Informal Notion of Algorithm

Rogers sets out by listing what exactly we are trying to capture with our notion of recursiveness. He lists informally a few criteria for which we characterise algorithms, with criteria 6 to 10 being contentious, leading to different notions. Criteria 1 to 5 are universally agreed upon.

1. An algorithm is a set of instructions of finite size.
2. An agent who acts on those instructions exists.
3. There are facilities for storing intermediate computations.
4. The computation process carried about the agent is done in a discrete fashion, without the use of continuous/analogue devices.
5. The computation process is deterministic; there is no randomness involved.
6. Can the size of inputs become arbitrarily large?
7. Can the size of instructions become arbitrarily large?
8. Can the memory (RAM/storage) available become arbitrarily large?
9. Can the computing agent's abilities become arbitrarily large?
10. Can the time available/number of steps carried out in computation become arbitrarily large?

An anaylsis into criteria 6 to 10 is appropriate. It seems absurd that an algorithm should not be applicable to due to a size of its input, so "most" mathematicians answer positively to 6. Similarly, arbitrarily large instructions may still be carried out in a consistent matter, so most answer positively to 7. Rogers states that "most mathematicians... would assert that a general theory of algorithms should concern computations which are possible *in principle*, wihout regard to practical limitations" to justfiy the positive answers to 6 and 7.

Criteria 8 has a caveat, with memory meaning the "storage space which is necessary over and above the space neeeded to store instructions, input, and output." For example, $\lambda x[2x]$ can be calculated for arbitrary x with bounded memory, but not for $\lambda x[x^2]$. Mathematicians consider this restriction too narrowing and so reject this bound, but it's interesting what sort of functions may arise from this restriction. Apprently, the classes of such functions are called *functions computable by a finite-state machine*, but is dependent on the choice of symbolism.

We answer positively to 9 as an agent may transfer the arbitrary complexity of an input by use of the arbitrary memory available.

10 is attempting to answer whether we should "have some idea, ahead of time, of how long the computation will take". We leave no answer at this stage.

1.2. Primitive Recursive Functions

The comment on the footnote of §1.2, that intuitionists reject the claim that the function

$$h(x) = \begin{cases} 1, & \text{if the Goldbach conjecture is true;} \\ 0, & \text{otherwise;} \end{cases}$$

is primitive recursive from the reasoning that it must either be true or false, is actually trivial. One just needs to understand that intuitionists have a different notion of "true" and "false". If one asks me in the midst of a chess game whether I've won, I would answer no, because the game is still going, but that does not mean I have lost; similarly, a statement to an intuitionist is true, false, or contingent/neither, with the latter being due to a non-existence of an explicit construction. The idea behind intuitionism is to define truth/existence/reality to hinge on (somewhat circularly) existence/reality/explicit construction. (Perhaps this is saying even consistent constructions are not "real" if not realised in the universe? Or, that "consistent" is should be defined as such?)

1.5. Formal Characterisation

Rogers raises the interesting point how, from the diagonal argument it seems impossible to formally capture the notion of algorithms, in an algorithmic way. Indeed for any formalism relying solely on total functions it can be applied "to any case where the sets of insutrction in the P-symbolism can be effectively (i.e., algorithmically) listed." The solution to this is by allowing non-totality, by which $\varphi_x(x) = \varphi_x(x) + 1$ need not be false as $\varphi_x(x)$ may not be defined. The acceptance of non-totality here is analogous (actually equal) to introducing minimalisation to the generators in the primitive recursive scheme.

1.8. Gödel Numbers

In the previous section Rogers discusses universality/Church's Thesis as "the Basic Result", noting the equivalence of different formalisms, giving Turing's and Kleene's as examples. Rogers now lists without proof a few theorems regarding codings of recursive functions. We fix some enumeration P of algorithms with P_x denoting the x^{th} algorithm (personally I prefer the PRIM+minimalisation scheme), and $\varphi_x^{(k)}$ refers to the partial function of k-variables determined by P_x.

Theorem 1.1. There are exactly $\aleph_0$ recursive functions.

Proof. The constant functions $\lambda x[k]$ are recursive, and there are $\aleph_0$ of them. But by enumeration, there are at most $\aleph_0$ recrusive functions, so $|R| = \aleph_0$. □

Theorem 1.2. There exist non-recursive functions.

Proof. The set of all functions from $\mathbb{N} \to \mathbb{N}$ has cardinality $\aleph_0$. The theorem follows. □

Theorem 1.3. Each recursive function has $\aleph_0$ distinct indices.

Proof. See problem 2.8. □

Theorem 1.4.

$$\exists\, z, \quad [z](x, y) = [x](y).$$

Equality with undefined values is to be interpreted as follows: if $[x](y)$ is undefined, $[z](x, y)$ is undefined.

Proof. I would like to give a cool explicit construction but I might get lost in the details in this stage. Instead I show a python implementation (which is more for demonstrative purposes).

```python
#We may use a simple coding such as translating a python program into an
    integer.
def universal(index, x):
    eval(index_to_function_definition(index)) #index_to_function_definition
        returns the corresponding function definition from the given index
    return f(x)
```

Listing 1: Example Python code

□

2. Unsolvable Problems

2.1. Exercises

Problem 2.1. Show that the function

$$g(x) = \begin{cases} 1, & \text{if a consecutive run of at least } x \text{ 5's occurs in the decimal expansion of } \pi; \\ 0, & \text{otherwise;} \end{cases}$$

is primitive recursive.

Solution. We just need to find an algorithm for π that is primitive recursive; the BBP-formula trivially solves this. (Actually, I think the question wants you to explicitly develop a primitve parsing algorithm, supposing that π is already given. But whatever.) □

Problem 2.2. Define f by

$$f(x) = \begin{cases} 1, & \text{if } \varphi_x(x) = 1; \\ 0, & \text{otherwise.} \end{cases}$$

Is f recursive?

Solution. Immediately this reeks of being unrecursive. Suppose that f is recursive. Then $g = \neg f$ is also recursive. Let $g = \varphi_y$. Is $g(y) = 0$? Then $\varphi_y(y) = 0$ and $\varphi_y(y) = 1$; if $g(y) = 1$ we get a similar contradiction. □

Problem 2.3. Consider the list of primitive recursive derivations decribed in §1.4. Let f_x be the primitive recursive function determined by the $(x+1)$st derivation in this list $x = 0, 1, 2, \ldots$. Define $g = \lambda xy[f_x(y)]$. Is g recursive? Is g primitive recursive?

Solution. This question has already been answered in §1.4. Again like in 2.2 we abuse the self-referring capability that g possesses. Let $h(x) = f_x(x) + 1$. Suppose h is recursive, i.e. $h = \varphi_y$. Then,

$$\begin{aligned} h(y) &= g(y, y) + 1 \\ &= \varphi_y(y) + 1 \\ &= h(y) + 1. \end{aligned}$$

□

Problem 2.7. Give a more formal proof for Theorem I-VIII analogous to the more formal proof for Theorem I-VII.

Solution. The problem wants us to show that determining whether an arbitrary Turing machine is total is recursively unsolvable. I CBF doing this formally, I will outline a solution.
For any φ_x, one can construct a new Turing machine that computes $\lambda xy[\varphi_x(y)]$. Construct a one-variable function $f_{xy}(z) = \varphi_x(y)$, and the totality question becomes equivalent to the halting problem. □