Creativistic Philosophy: Exploring the Limits of Formalization, #6[1]—Changing the Vantage Point

Andreas Keller

(Travis212, 2011)[2]

We have seen that the computable predicates on natural numbers are not algorithmically enumerable. One can write an algorithm for each of them (that is what we mean when we say that they are computable), but a program that produces all of these algorithms and only those algorithms is impossible.

The last part of the previous sentence is important. We will see in a later installment that for a given programming language, it is possible to algorithmically enumerate all programs that can be written in it. We could line all of them up and number them in ascending order. The programs for all the computable predicates must be included somewhere in this sequence.

---

[2] The picture shows the vantage point called "Guano Point" in the Grand Canyon area. While somebody at that point has a great view of the surrounding landscape, they cannot see all of Guano Point itself. From a higher point, perhaps a drone or helicopter, you can have a view to Guano Point, but you don't see the drone that picture was taken from, and so on. The vantage point from which you can see the previous vantage point is invisible from itself, and so on.

What is impossible, however, is to write a program that can tell for each of the programs in this sequence whether it calculates a computable predicate or not. If we could do so, we could go through the sequence of all programs and identify those that are computing computable predicates, and thus produce a complete enumeration of them. But we have seen that this is impossible.

As we are going to see, the crucial point is that it is impossible to write a program that can decide, for every program given as its input, whether that program will eventually halt for all its inputs. We will look into this matter, known as the "halting problem," in a later installment.

But, while we cannot write a program that produces all computable predicates, we always can, for any given known set of such predicates, produce a new predicate, and another one, and another one, and so on. We can always extend the set of known algorithms. We have seen that we can do so by producing the negated diagonal of the table of known predicates.[3]

An enumeration of all the programs of a programming language yields a sequence of programs P_1, P_2, P_3 etc. We can define a predicate X that has the value "true" if the nth program, P_n, computes a computable predicate, and the value "false" if it does not. As explained above, this predicate is not computable because if it were, we could use it to produce a complete enumeration of all computable predicates, which, as we have seen, is impossible. So, we have here our first example of a predicate (and a function) that is not computable.

However, we can always get new "data points" for this non-computable predicate. Each time we find a new program P_m calculating a computable predicate, we know that X(m) = "true." So, although we cannot have a "complete" algorithm to calculate X for arbitrary cases, we can always extend our knowledge of X and produce additional "data points" for it, extending the partial algorithm for X. In this sense, we are able to "compute" non-computable functions.

The point we have reached enables us to start with some more philosophical reflections, so before I plunge back into the more mathematical line of thought of the previous three installments, I want to step back a little bit and look at what we are doing here from a little further away.

[3] An operation like producing the "negated diagonal" is what is called a "productive function." The computable one-argument predicates on the natural numbers form what is known as a "productive set." This means they are not algorithmically enumerable, but for each algorithmically enumerable subset of them, the productive function produces a new element of the productive set. The term "productive set" was introduced in (Dekker, 1955).

An algorithm can calculate a non-computable function for a true subset of that function's domain, i.e., the set of possible input values. It can also always be extended, so that the value of the function becomes known for additional elements of the function's domain. However, the algorithm cannot do that extension by itself. What a given algorithm can calculate is fixed. What can be derived inside a formal theory (a notion equivalent to the concept of "algorithm") is fixed. Algorithms/formal theories are static in that respect. It is not possible that "from inside" a formal system (algorithm or formal theory), by means of that algorithm itself, something *becomes* derivable or computable that was not derivable before. Formal systems cannot and do not develop.

However, they can *be developed*—from the outside. What is needed to change an algorithm is an external reference to it. As noted in (Ammon, 2016), what algorithms are unable to do is to produce a reference to themselves as a whole.

If they could produce a complete self-reference, they would be able to apply the productive function of generating the negated diagonal and then extend themselves by the resulting new algorithm, changing what they do; but this is something an algorithm cannot do. The calculation processes of algorithms are immersed in a closed domain of knowledge, so to speak, and they cannot emerge from it on their own.[4]

We can compare this limitation of algorithms with the inability of tourists to see the vantage point where they are standing. They can view a large part of a landscape from there, but to see the vantage point itself, they would have to move to a higher vantage point. The external reference to the algorithm can be compared to that new "meta-vantage point." From there, the landscape and the original vantage point is visible, but not the new vantage point itself, and so on.

By producing a complete external reference to an algorithm/formal theory, we can inspect, extend, or modify it (which includes the application of "productive functions"), merge it with other algorithms, and so on. All of this is done from the outside. But the operation of creating an external reference is an operation that cannot be done from inside an algorithm. Algorithms are restricted to operations that are formalizable. So, the ability of creating a complete self-reference is, in this sense, not formalizable.

But this is something mathematicians and computer programmers are doing all the time. I was tempted to say "routinely do all the time," but since this operation is not formalizable, it is not done "by routine." Nevertheless, it looks like somehow

[4] This restriction, of course, also applies to algorithmic AI systems as well. They are always limited and incapable of changing and developing themselves.

human beings are capable of doing it. We seem to be capable of stepping outside[5] of any formal description of what we are doing and moving to a new vantage point, so to speak.

This ability we seem to have which goes beyond what algorithms can do has been called "creativity." This is what gives this series its title: "Creativistic Philosophy." In the next installment, I am going to give this topic a closer look.

[5] We could also say "transcend" here, but this word and the words derived from it have become so overloaded with different meanings and connotations during the history of philosophy that I want to avoid that term.

REFERENCES

(Ammon, 2016) Ammon, K. "Informal Physical Reasoning Processes." Available online at URL = <https://arxiv.org/abs/1608.04672>.

(Dekker, 1955). Dekker, J.C.E. "Productive Sets." *Transactions of the American Mathematical Society* 78: 129–149.

(Travis212, 2011). Travis212. "Guano Point." *Wikimedia Commons*. Available online at URL = <https://commons.wikimedia.org/wiki/File:GuanoPoint.jpg>.