On a Means of Extensive Condensation of Formal Representations of Algorithms

[1]Pitambar Sai Goyal
[1]Student
[1]Department of Information Technology
[1]Loyola-ICAM College of Engineering and Technology(LICET)
Chennai, India

Abstract

There is a provision in the rules of set theory and the set theoretic definition of 'algorithm' which allows for condensation of the existing representations of algorithms, drawing on certain classical results of Cantor and Godel. This condensation was also completely effective on formalizations of programs running on a Turing Machine. An equivalence is demonstrated between an algorithmic process and a dynamic physical process involving the motion of a particle in a certain impulse field. This does not condense the problem in the context of modern digital computers, but greatly condenses it for the human brain and certain analog systems, as will be demonstrated. Two theorems will be presented, regarding formalism of 'algorithm', one by which we will see that all computationally solvable problems, with finite sized inputs, are equivalent to the solving of an equation, f(x) =0 for an integer valued function f on the integers. The second theorem will demonstrate that a large subclass of these problems are reduced to the factorization of an integer. It is understood that the invention of alternate formal representations of algorithms may be critical to several problems of computation, including the P vs NP problem.

Keyword- Algorithm, formalisation, theoretical computing, Church-Turing, Godel

I. INTRODUCTION

Kurt Godel demonstrated the incompleteness of mathematical logic in the discovery of mathematical relations, in the 1st half of the last century, by constructing injective maps on the set of all logical possibilities into the set of mathematical possibilities and applying Cantor's discoveries with regard the so-called 'Transfinite numbers'. These results from the basics of the major preoccupations of mathematicians in the 21st century, and have the potential to unlock some of the deepest secrets of theoretical computing. As it happens, since the fundamental notion of an algorithm is expressible as a set theoretic construct, we are at liberty to apply Cantor's classical results to this in order to condense these constructs and uncover certain truths about the feasibility and complexity of an algorithm. The way in which the following enables us to condense these constructs with the application of a transform is by translating the method into one that is a highly simplified process on analog machines. One of the theorems below manages to reduce the time complexity of a subclass of problems, but it is not determined how large this class is.

II. REPRESENTATION 'A' OF AN ALGORITHM

An algorithm is representable by a tuple (Q, I, Ω, *f)* where Q is a set containing I, Ω, as subsets. *f* is a function from Q to Q which leaves Ω fixed point wise. I is called the set of all legal inputs and Ω is the set of all outputs. This combination represents an algorithm if and only if there exists a finite n ∈ N associated with each x ∈ I such that:

$$f^{(n)}(x) = (f \circ f \circ f \circ \ldots \circ f)\,(x) \in \Omega$$

We call this representation Representation 'A' in the remainder of this document. Two features of algorithms in general are, however, neglected in this representation. Firstly, it is generally expected, due to considerations of feasibility, that 'n' is not just finite, but sufficiently small. Secondly, it is expected that the operation of f is executable by anyone with a pencil and paper[1].

'A' simulates all of the features of finite-time computational methods and every such method simulated on a computer with a discrete sequence of successive states can be mapped into this tuple- each state of computation is an element q of Q and corresponds to a value of f (i) (x) in a problem of input x.

III. REPRESENTATION 'R' OF AN ALGORITHM

It is the object of this document to transform representation A into an alternative 'R' with suitable objective maps, such that the resulting representation condenses the principal domain Q as well as the complexity of the operation of f uniformly for all problems

in the most general class of problems satisfying the assumptions of the Church- Turing thesis. We may say that an algorithm can be executed on a Turing machine if its representation in the 'A' form can be normalised so that:

a. Q is the set of all $\aleph_0$-length strings over a finite alphabet Σ

b. The operation of '*f* ' can be normalized to consist entirely in changing characters in an $\aleph_0$-length string according to fixed rules.

We shall therefore proceed to condense the A- representation by transforming the normalized Q and normalized '*f* ' as defined above.

First, we see that $Q = \Sigma^{\aleph_0}$. If

$$\overline{\overline{\Sigma}} = k,$$

Then the cardinality of this set is:

$$\overline{\overline{Q}} = \overline{\overline{\Sigma^{\aleph_0}}} = = (\overline{\overline{\Sigma}})^{\aleph_0} = k^{\aleph_0}$$

From Cantor's theorems [2] on the transfinite numbers we know that this is the same as the cardinality of the set of all real numbers, R, since k is an integer > 1. Because this theorem is in some neglect regarding the general discourse, we prove it here below.

A. Theorem 1. (Due to Cantor)
The set of all infinite-length strings over a finite alphabet Σ has the same cardinality as the set of real numbers.

1) Proof
We use here Cantor's notation '~' to relate two sets with the same cardinal number. Then

$$\mathbf{R} \sim (0, 1). \qquad [I]$$

Since the function $g(x) = 1/x - 1$ maps (0,1) to the set of all positive reals objectively, so that $\mathbf{R}^+ \sim (0,1)$. Then, the function exp(x) objectively maps **R** to $\mathbf{R}^+$. From the transitive nature of '~', we have (I) [N1].
Now we represent all the elements of (0, 1) numerically, to the base k (=cardinality of Σ). They will take the form

$$0.\ a_1 a_2 a_3 \ldots\ldots$$

Where $a_i \in \{0,1, \ldots., k-1\}$. Therefore, a_i is a character from the finite alphabet {0, 1, ... , k-1}, and the numerical representation is an $\aleph_0$-length string over this alphabet. This in itself demonstrates that

$$\mathbf{R} \sim \Sigma^{\aleph_0}$$

It remains to be demonstrated that this result holds even though numbers having a terminating sequence after the radix point have 2 different representations each. This was easily achieved by Cantor and will be left out here.
It is thus demonstrated that Q can be replaced by R for the purposes of a Turing Machine. This implies that f becomes a real valued function on the reals:

$$f: \mathbf{R} \rightarrow \mathbf{R}$$

This reveals the 1st of the new theorems:

B. Theorem 2. (Condensation of Q)
Any problem's solution by a Turing Machine is equivalent to the solution of an equation

$$f(x)-x = 0$$

In the reals by the method of fixed-point iteration that converges in finite time for a well-defined set of initial solutions.

2) Proof
From the modification of Q above, it can be seen that we can reframe representation A as below:
An algorithm is representable by a tuple (I, Ω, *f*) where I, Ω, are sets of real numbers. *f* is a function from **R** to **R** which leaves Ω fixed point wise. I is called the set of all legal inputs and Ω is the set of all outputs. This combination represents an algorithm if and only if there exists a finite n ∈ N associated with each x ∈ I such that:

$$f^{(n)}(x) = (f \circ f \circ f \circ \ldots. \circ f)\ (x) \in \Omega$$

Quick inspection shows us that this requires repeated application of *f* until the equation:

$$f(t)=t$$

is satisfied by some result 't' of the repeated application. This fully defines fixed point iteration and thus establishes the equivalence. Since the set of all initially assumed solutions, in this case, is I, which is well defined as the set of all legal inputs, the process always converges for a certain well defined set of starting points.

Since fixed point iteration has been studied extensively, it appears that this equivalence is somewhat significant, however its immediate significance becomes apparent when we consider the form *f* must take in order for this condensation to be effective.

f must be an easily computable real valued function so that the iteration takes up the principle labour of computation. If indeed all problems solvable by a Turing Machine can be mapped to such a process as above where *f* is a simple real function, then our concern will experience a great shift from the operations on strings to which *f* was formerly restricted to functions of common occurrence in real analysis and pure mathematics.
Next we shall consider the large sub-class of problems wherein Q is a finite- length string. Then $Q = \Sigma^*$.

C. Theorem 3
Any problem's solution by an algorithm wherein $Q = \Sigma^*$ is completely equivalent to the solution of an equation $f(x) - x = 0$ in the integers by fixed point iteration that converges in finite time for a well-defined set of starting points.

3) Proof
It is sufficient to prove that

$$Q \sim \mathbf{Z}$$

This is achieved by an idea that is derived from Godel numbering [3]. First we impose an arbitrary ordering on the alphabet Σ. Now, using the position of each character in the ordering we map Σ to the natural numbers from 1 to k. Then Q is equivalent to the set of all vectors of arbitrary dimensionality whose components are integers in the range of 1 to k. We now perform the mapping:

$$g(s_1, s_2, \ldots, s_r) = 2^{s_1} \cdot 3^{s_2} \cdots P_r^{s_r}$$

Where $s_i \in \{1, 2, \ldots, k\}$, and where P_r is the r^{th} prime. By the Fundamental Theorem of Arithmetic, this is a objective function which assigns a unique positive integer to every string in Q, thus completing the proof.
At the risk of leaving a large number of potential advantages furnished by this theorem unexplored, we will focus on the equivalences which exist between this representation and certain familiar systems.
Our final effort in this regard will be towards establishing of another surprising equivalence that exists between a large sub-class of these problems and familiar algorithms in arithmetic.

D. Theorem 4
Any problem's solution which can be solved on a finite state machine, or whose final solution can be restricted to a finite set of possible outputs by some simple method of determination, can be reduced to the factorization of an integer, if the restricted range of outputs is sure to contain only one possible output.

4) Proof
Let the range of outputs be $\sim \{(x_i, y_i)\}$. Since this range is finite, it can be fitted onto a polynomial's graph whose explicit expression is fully deterministic. Let this polynomial function be $y = p(x)$. At least one of the points in the determined range satisfies $y = x$, which implies that the equation $p(x) = x$ has a solution in the integers. The problem is then reduced to:
1) Factorise the constant term
2) Substitute each of the factors one-by-one for x in the equation $p(x) = x$. The zero of the equation is the solution
Here we see that the factorization of the integer can be taken as the principal operation. If the sum of multiplicities of the constant's prime factors is less than the input size, then the problem is reduced to a polynomial time problem. In fact, if the number of points in the range is less than 5, there is a deterministic formula to determine the zeroes of this equation [N2].

Due to the abundance of results involving curves in the Cartesian plane, these theorems pave the way for condensation of algorithms using continuous analysis, as in analog systems, etc. It is easily seen from graphical representations of fixed point iteration that it simulates the motion of a particle in an impulse field. It is hoped that the equivalences so demonstrated will revive the study of real analysis and explicit functions in the general discourse of this subject.

IV. CONCLUSIONS

It can therefore be seen that all algorithmic processes can be modelled after fixed-point iteration, an age-old method of solving numerical equations. It is clear that of the 3 new theorems presented above, the 3rd and 4th are the most promising as regards further relevant discoveries, since all real computing problems are solved on finite state machines and with each state being in the form of a finite length string. It is not at present clear, however, whether this equivalence demonstrates the existence of a class of problems insoluble by a Turing machine, since the complement of the set of finite-time, convergent fixed point iteration runs may or may not have a pre-image in the set of algorithms of representation A. This appears to be best deserving of further exploration. There has been no attempt to apply existing knowledge of the reduced methods, since there is a vast abundance of these. The actual condensation consists mainly in the solution of problems using analog systems such as a particle in an impulse field, or by modelling a signal f(t), and resolving close inputs, to obtain a clear solution.

NOTES

1) There exist many functions that may be used in lieu of these. For example, $\ln(1/x)$ maps (0,1) to the set of all positive reals.
2) There exists a deterministic formula for the solution of all one dimensional polynomial equations with degree<5.

REFERENCES

[1] Donald E. Knuth, The Art of Computer Programming, Vol I
[2] Georg Cantor, Contributions to the founding of the theory of transfinite numbers
[3] Kurt Godel, On formally undecidable propositions of Principia Mathematica and related systems