Foundational Analyses of Computation

Yuri Gurevich

Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399,
United States of America

Abstract. How can one possibly analyze computation in general? The task seems daunting if not impossible. There are too many different kinds of computation, and the notion of general computation seems too amorphous. As in quicksand, one needs a rescue point, a fulcrum. In computation analysis, a fulcrum is a particular viewpoint on computation that clarifies and simplifies things to the point that analysis becomes possible.

We review from that point of view the few foundational analyses of general computation in the literature: Turing’s analysis of human computations, Gandy’s analysis of mechanical computations, Kolmogorov’s analysis of bit-level computation, and our own analysis of computation on the arbitrary abstraction level.

1 Introduction

Algorithms and computations are closely related concepts. Syntactically algorithms are programs (or recipes) but semantically they specify computations. And the only computations that we consider here are algorithmic (also known as mechanical). In this paper, we abstract from the syntax of algorithms, so that analysis of algorithms and analysis of computation are one and the same.

Turing’s analysis of algorithms was provoked by the Entscheidungsproblem, the problem whether the validity of first-order formulas is computable. Logicians have been interested in what functions are computable, and Turing’s analysis is often seen from that point of view. But there may be much more to an algorithm than its input-output behavior. In general algorithms perform tasks, and computing functions is a rather special class of tasks.

Here we concentrate on foundational analyses of algorithms/computations, not on what functions are computable.

2 Turing

Alan Turing analyzed computation in his 1936 paper “On Computable Numbers, with an Application to the Entscheidungsproblem” [21]. The validity relation on first-order formulas can be naturally represented as a real number,
and the Entscheidungsproblem becomes whether this particular real number is computable. “Although the subject of this paper is ostensibly the computable numbers, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique” [21, p. 230].

How could Turing analyze computation in such generality? The world of algorithms is large and diverse. Explicitly or implicitly, he imposed some constraints on the computations in consideration. And he found a fulcrum. We start with the fulcrum. There were no computers in Turing’s time but that does not seem to make Turing’s task much simpler. Humans are hard to analyze. Amazingly Turing found a way to do just that: Ignore how the algorithm is given, ignore what human computers have in their minds, and concentrate on what the computers do, what their observable behavior is. That is his fulcrum.

One may argue that Turing did not ignore the mind. He speaks about the state of mind of the human computer explicitly and repeatedly. For example, he says that “[t]he behaviour of the computer at any moment is determined by the symbols which he is observing, and his ‘state of mind’ at that moment” [21, p. 250]. But Turing postulates that “the number of states of mind which need be taken into account is finite.” The computer just remembers the current state of mind, and even that is not necessary: “we avoid introducing the ‘state of mind’ by considering a more physical and definite counterpart of it. It is always possible for the computer to break off from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of the ‘state of mind’.”

Turing introduced abstract computing machines that became known as Turing machines (and constructed a universal Turing machine). He defined a real number to be computable “if its decimal can be written down by a [Turing] machine” [21, p. 230]. His thesis was that Turing computable numbers “include all numbers which could naturally be regarded as computable” (Turing [21, p. 230]). He used the thesis to prove the undecidability of the Entscheidungsproblem. To convince the reader of his thesis, Turing used three arguments.

Reasonableness: He gave examples of large classes of real numbers which are [Turing] computable.

Robustness: He gave another explicit definition of computability and proved it is equivalent to the original one “in case the new definition has a greater intuitive appeal.” The robustness argument was strengthened in the appendix where, after learning about Church’s explicit definition of computability [6], he proved the equivalence of their definitions.

Appeal to Intuition: He analyzed computation appealing directly to intuition.

---

1 “Numerical calculation in 1936 was carried out by human beings; they used mechanical aids for performing standard arithmetical operations, but these aids were not programmable” (Gandy [8, p. 12]).