Giuseppe Primiero. *On the Foundations of Computing*. Oxford University Press 2020. 320 pp. $105.00 USD (Hardcover ISBN 9780198835646); $55.00 USD (Paperback ISBN 9780198835653).

First of all, readers should not be scared by the title of this book. It is written by a philosopher of science, and it discusses the foundations of computing and computer science from a philosophical point of view. The foundations are divided into topics, which are associated with mathematics, engineering and experimentation. A unique feature of this book is that it elevates the experimental aspects of computing into a self-contained theme even if not quite equal in importance (or length) to the two other areas mentioned. Tests and trials, and the empirical side of computer science, are often lumped together with the engineering (or more broadly, practical) aspects of computing, which does not do justice to them.

The book is divided into three main parts. To lay the historical ground for the mathematical foundations Primiero reaches all the way back to Leibniz, which is in line with the idea that calculation (with numbers or concepts) and computation are intertwined activities. On this path, we encounter approaches to the foundations of mathematics (such as logicism, intuitionism and Hilbert's program), as well as notions and methods from set theory (infinities and diagonalization) (7–22). Axiomatic theories of mathematics come with a natural notion of a proof, which ensures that proofs can be viewed as computations on sequences of symbols. In other words, the theorems of the theory can be mechanically (/effectively/algorithmically) generated. Of course, mathematicians typically do not proceed this way when proving theorems. However, the ideal, once enunciated by Hilbert, would be to deduce theorems from the axioms and to be able to determine when such a deduction is impossible (19). Nowadays, these ideal theories are called decidable, and the problem of delineating permissible methods for such determination (as well as for deduction) was an impetus for some of the definitions of the concepts of computability. Primiero opts to introduce several notions of computability such as recursive functions, λ-definability and Turing computability. Considering dissimilar formalizations of the intuitive notion of computability is helpful for the discussion of the Church–Turing thesis, which was formulated initially using the two former notions, and then using Turing Machines (TM's). Indeed, Primiero first discusses the Church Thesis (38–40), and then the Turing Thesis (56–63). Having introduced a range of models of computation, he can present the best supporting argument for the plausibility of the Church–Turing thesis, namely, the equivalence of these formal notions. We might add that some newer models (e.g., register machines, Markov algorithms and 0-type grammars) also turned out to be as powerful as the earlier models. Although TM's are quite remote from contemporary computers in their structure and steps of computation, TM's continue to be theoretically noteworthy; hence, their detailed treatment allows readers to develop an idea of how one kind of abstract computing devices work (43–56). A further advantage of focusing on TM's is that a concrete non-computable function can be constructed, and the concept of a universal computer (the universal TM) can be presented. Primiero's examples of TM's provide a fun and entertaining way to do some `real computations.'

The chapters on the engineering foundations start with Shannon's circuits which date back to the mid-1930s, and touch upon some of the most important computers and their inventors around the middle of the 20th century. Historical facts will help the reader to situate the development of modern electronic computers within the general history of the last century. Given the ubiquity of computers nowadays, together with their small size (such as a tablet or a phone), it may be difficult to imagine how rare and cumbersome computers were just half a century ago. The description of a range of early

ideas as to how to equip computers with memory, and then the implementations of those ideas, is very instructive. The `(paper) tape' of a TM can be naturally thought of as the (main) memory of the machine; indeed, some of the first computers in the 1940s had paper tapes attached to them. Tapes work fine in an abstract machine such as a TM, but in an electronic machine that performed tens or even hundreds of operations per second (by 1950 or so), paper tapes were inadequate. In the design that became known as the von Neumann architecture, memory is an essential component (131). Compared to a TM, in which part of the memory is in the `head' of the machine (as its state), and where the tape also serves as an input–output device, von Neumann's design is clean-cut, because it separates the functionally distinct components of a computer from each other. The leading memory devices of the time were valves (cathode ray tubes) and mercury delay lines. The pictures of these devices (124, 136) should impress on a reader how different the current static memory devices (such as EPROM and flash drives) are in every possible way save that they too function as memory.

Another area where technology produced phenomenal progress is the central processing unit. Some of the early computers such as ENIAC did not need such a component, but since the 1950s, when the EDVAC design became the general blueprint for modern computers, the central processing unit became part of every computer. Primiero describes the invention of the transistor, and includes several patent diagrams to make the underlying concepts palpable (145–53). It would be easy to get lost in the numerical characterizations of the various devices, but Primiero turns to a series of laws that were formulated by practicing computer scientists. Perhaps the best-known is Moore's law, which is often mentioned in different forms. The explanation behind the variations is that Moore revised his own predictions as technology and fabrication changed. Some other laws (e.g., Amdahl's and Gustafson's) are less widely known, however, their appearance in the discussion signals Primiero's view that stresses a certain similarity between the natural sciences and computer science. Of course, this analogy cannot go all the way, because computing devices are man-made, which impacts the laws that are based on observations while intended to provide predictions.

The special status of computer science reveals itself, for instance, in our understanding of the relationship of the models of computation to the implemented or physical computation. Primiero introduces five different interpretations (causal, syntactic, semantic, linguistic and mechanistic) of physical computations, and creates ontological and matching epistemological ladders based on levels of abstraction (174). The epistemological structure can be captured in a sentence (with the bold words indicating levels of abstraction): A **problem** is expressed in a **task**, which is interpreted in **instructions**, satisfied by **operations**, and finally, executed in terms of **actions**. Similarly, careful distinctions are made in the discussions of functionality, usability and efficiency, which are concepts that are necessary for the assessment of qualitative features of computing. Primiero has an interest in the classification of what could be called errors (of any kind). On the one hand, `What is an error (in a computation)?' has a philosophical origin, because the question stems from the relation of abstract idealized computation to the computation carried out on some physical device; on the other hand, the interest in errors has a very practical source. Ever since the introduction of high-level programming languages, but especially, since the boom in personal computing in the last 40 years or so, users had to contend with software `bugs.' The debate about the nature of computer science which occupied some computer scientists for many years had two sides, which are very clearly manifested in their approach to ensuring that complex software systems function as intended. To put it simply, `the mathematics camp' advocated the use of formalization, formal methods and proofs of correctness, whereas `the engineering camp' stressed the need for a dynamic process in which the correctness (perhaps better called suitability or acceptability) of software is the result of testing (maybe even

feedback from the users). Primiero uses levels of abstraction to analyze the debate on program correctness (197).

The first decades of electronic computers involved a lot of experimentation in physics and the emerging area of quantum computing falls squarely into that tradition. However, a philosopher of science cannot overlook that computer experimentation became available not only to those who can access supercomputers. Primiero considers the use of computers in experiments, in which the careful distinction of mathematical and computational models and their epistemic roles is crucial. This part of the book is rich in philosophical analysis of the types of models (237–8), their relationships and interpretations. However, some readers might wish to complement the content of the last four chapters with examples at the level of concreteness seen in the treatment of the mathematical and engineering foundations.

The book could be used as a textbook in a course that deals with philosophical aspects of computer science and computing. Every chapter ends with a series of exercises, many of which are topics for discussion or for short essays.

Katalin Bimbo, University of Alberta