

# Preface

The study of the class of computable partial functions (i.e., recursive partial functions) stands at the intersection of three fields: mathematics, theoretical computer science, and philosophy.

- Mathematically, this study, computability theory, originates from the concept of an *algorithm*. It leads to a classification of functions according to their inherent *complexity*.
- For the computer scientist, computability theory shows that quite apart from practical matters of running time and memory space, there is a purely theoretical limit to what computer programs can do. This is an important fact, and leads to the questions: Where is the limit? What is on this side of limit, and what lies beyond it?
- Computability is relevant to the philosophy of mathematics, and in particular to the questions: What *is* a proof? Does every true sentence have a proof?

Computability theory is not an ancient branch of mathematics; it started in 1936. In that year, Alonzo Church, Alan Turing, and Emil Post each published fundamental papers that characterized the class of computable partial functions. Church's paper introduced what is now called "Church's Thesis" (or the Church–Turing Thesis), to be discussed in Chapter 1. Turing's paper introduced what are now called Turing machines. (1936 was also the year in which *The Journal of Symbolic Logic* began publication, under the leadership of Alonzo Church and others. Finally, it was also the year in which I was born.)

These notes are intended to serve as a textbook for a one-term course on computability theory (i.e., recursion theory), for upper-division mathematics and computer science students. And the notes are *focused* on this one topic, to the exclusion of such computer-science topics as automata theory, context-free languages, and the like. This makes it possible to get fairly quickly to core results in computability theory, such as the unsolvability of the halting problem.

The only prerequisite for reading these notes is a willingness to tolerate a certain level of abstraction and rigor. The goal here is to prove theorems, not to calculate numbers or write computer programs.