CHAPTER 1

Background

The twentieth century has been filled with the most incredible shocks and surprises: the theory of relativity, the rise and fall of communism, psychoanalysis, nuclear war, television, moon walks, genetic engineering, and so on. As astounding as any of these is the advent of the computer and its development from a mere calculating device into what seems like a "thinking machine."

The birth of the computer was not wholly independent of the other events of this century. Its inception was certainly impelled if not provoked by war and its development was facilitated by the evolution of psycho-linguistics, and it has interacted symbiotically with all the aforementioned upheavals. The history of the computer is a fascinating story; however, it is not the subject of this course. We are concerned instead with the **theory of computers,** which means that we shall form several mathematical models that will describe with varying degrees of accuracy parts of computers, types of computers, and similar machines. The concept of a "mathematical model" is itself a very modern construct. It is, in the broadest sense, a game that describes some important real-world behavior. Unlike games that are simulations and used for practice or simply for fun, mathematical models abstract, simplify, and codify to the point that the subtle observations and conclusions that can be made about the game relate back in a meaningful way to the physical world, shedding light on that which was not obvious before. We may assert that chess is a mathematical model for war, but it is a very poor model because wars are not really won by the simple assassination of the leader of the opposing country.

The adjective "mathematical" in this phrase does not necessarily mean that classical mathematical tools such as Euclidean geometry or calculus will be employed. Indeed, these areas are completely absent from the present volume. What *is* mathematical about the models we shall be creating and analyzing is that the only conclusions that we shall be allowed to draw are claims that can be supported by pure deductive reasoning; in other words, we are obliged to *prove* the truth about whatever we discover. Most professions, even the sciences, are composed of an accumulation of wisdom in the form of general principles and rules that usually work well in practice, such as "on such and such a wood we recommend this undercoat," or "these symptoms typically respond to a course of medication X." This is completely opposite from the type of thing we are going to be doing. While most of the world is (correctly) preoccupied by the question of how best to do something, we shall be completely absorbed with the question of whether certain tasks can be done at all. Our main conclusions will be of the form, "this can be done" or "this can never be done." When we reach conclusions of the second type, we shall mean not just that techniques for performing these tasks are unknown at the present time, but that such techniques will never exist in the future no matter how many clever people spend millennia attempting to discover them.

The nature of our discussion will be the frontiers of capability in an absolute and timeless sense. This is the excitement of mathematics. The fact that the mathematical models that we create serve a practical purpose through their application to computer science, both in the development of structures and techniques necessary and useful to computer programming and in the engineering of computer architecture, means that we are privileged to be playing a game that is both fun and important to civilization at the same time.

The term computer is practically never encountered in this book—we do not even define the term until the final pages. The way we shall be studying about computers is to build mathematical models, which we shall call machines, and then to study their limitations by analyzing the types of inputs on which they operate successfully. The collection of these successful inputs we shall call the **language** of the machine, by analogy to humans who can understand instructions given to them in one language but not another. Every time we introduce a new machine we will learn its language, and every time we develop a new language we shall try to find a machine that corresponds to it. This interplay between languages and machines will be our way of investigating problems and their potential solution by automatic procedures, often called algorithms, which we shall describe in a little more detail shortly.

The history of the subject of computer theory is interesting. It was formed by fortunate coincidences, involving several seemingly unrelated branches of intellectual endeavor. A small series of contemporaneous discoveries, by very dissimilar people, separately motivated, flowed together to become our subject. Until we have established more of a foundation, we can only describe in general terms the different schools of thought that have melded into this field.

The most fundamental component of computer theory is the theory of mathematical logic. As the twentieth century started, mathematics was facing a dilemma. Georg Cantor had recently invented the theory of sets (unions, intersections, inclusion, cardinality, etc.). But at the same time he had discovered some very uncomfortable paradoxes—he created things that looked like contradictions in what seemed to be rigorously proven mathematical theorems. Some of his unusual findings could be tolerated (such as the idea that infinity comes in different sizes), but some could not (such as the notion that some set is bigger than the universal set). This left a cloud over mathematics that needed to be resolved.

To some the obvious solution was to ignore the existence of set theory. Some others thought that set theory had a disease that needed to be cured, but they were not quite sure where the trouble was. The naive notion of a general "set" seemed quite reasonable and innocent. When Cantor provided sets with a mathematical notation, they should have become mathematical objects capable of having theorems about them proven. All the theorems that dealt with finite sets appeared to be unchallengeable, yet there were definite problems with the acceptability of infinite sets. In other branches of mathematics the leap from the finite to the infinite can be made without violating intuitive notions. Calculus is full of infinite sums that act much the way finite sums do; for example, if we have an infinite sum of infinitesimals that add up to 3, when we double each term, the total will be 6. The Euclidean notion that the whole is the sum of its parts seems to carry over to infinite sets as well; for example, when the even integers are united with the odd integers, the result is the set of all integers. Yet, there was definitely an unsettling problem in that some of Cantor's "theorems" gave contradictory results.

In the year 1900, David Hilbert, as the greatest living mathematician, was invited to address an international congress to predict what problems would be important in the century to come. Either due to his influence alone, or as a result of his keen analysis, or as a tribute to his gift for prophecy, for the most part he was completely correct. The 23 areas he indicated in that speech have turned out to be the major thrust of mathematics for the twentieth century. Although the invention of the computer itself was not one of his predictions, several of his topics turn out to be of seminal importance to computer science.

First of all, he wanted the confusion in set theory resolved. He wanted a precise axiomatic system built for set theory that would parallel the one that Euclid had laid down for geometry. In Euclid's classic texts, each true proposition is provided with a rigorous proof in which every line is either an axiom or follows from the axioms and previously proven theorems by a specified small set of rules of inference. Hilbert thought that such an axiom system and set of rules of inference could be developed to avoid the paradoxes Cantor (and others) had found in set theory.

Second, Hilbert was not merely satisfied that every provable result should be true; he also presumed that every true result was provable. And even more significant, he wanted a methodology that would show mathematicians how to find this proof. He had in his mind a specific model of what he wanted.

In the nineteenth century, mathematicians had completely resolved the question of solving systems of linear equations. Given any algebraic problem having a specified number of linear equations, in a specified set of unknowns, with specified coefficients, a system had been developed (called linear algebra) that would guarantee one could decide weather the equations had any simultaneous solution at all, and find the solutions if they did exist.

This would have been an even more satisfactory situation than existed in Euclidean geometry at the time. If we are presented with a correct Euclidean proposition relating line segments and angles in a certain diagram, we have no guidance as to how to proceed to produce a mathematically rigorous proof of its truth. We have to be creative—we may make false starts, we may get completely lost, frustrated, or angry. We may never find the proof, even if many simple, short proofs exist. Linear algebra guarantees that none of this will ever happen with equations. As long as we are tireless and precise in following the rules, we must prevail, no matter how little imagination we ourselves possess. Notice how well this describes the nature of a computer. Today, we might rephrase Hilbert's request as a demand for a set of computer programs to solve mathematical problems. When we input the problem, the machine generates the proof.

It was not easy for mathematicians to figure out how to follow Hilbert's plan. Mathematicians are usually in the business of creating the proofs themselves, not the proof-generating techniques. What had to be invented was a whole field of mathematics that dealt with algorithms or procedures or programs (we use these words interchangeably). From this we see that even before the first computer was ever built, some people were asking the question of what programs can be written. It was necessary to codify the universal language in which algorithms could be stated. Addition and circumscribing circles were certainly allowable steps in an algorithm, but such activities as guessing and trying infinitely many possibilities at once were definitely prohibited. The language of algorithms that Hilbert required evolved in a natural way into the language of computer programs.

The road to studying algorithms was not a smooth one. The first bump occurred in 1931 when Kurt Gödel proved that there was no algorithm to provide proofs for all the true statements in mathematics. In fact, what he proved was even worse. He showed that either there were some true statements in mathematics that had no proofs, in which case there were certainly no algorithms that could provide these proofs, or else there were some false statements that did have proofs of their correctness, in which case the algorithm would be disastrous.

Mathematicians then had to retreat to the question of what statements do have proofs and how can we generate these proofs? The people who worked on this problem, Alonzo

Church, Stephen Kleene, Emil Post, Andrei Andreevich Markov, John von Neumann, and Alan Turing, worked mostly independently and came up with an extraordinarily simple set of building blocks that seemed to be the atoms from which all mathematical algorithms can be comprised. They each fashioned various (but similar) versions of a universal model for all algorithms—what, from our perspective, we would call a universal algorithm machine. Turing then went one step farther. He proved that there were mathematically definable fundamental questions about the machine itself that the machine could not answer.

On the one hand, this theorem completely destroyed all hope of ever achieving any part of Hilbert's program of mechanizing mathematics, or even of deciding which classes of problems had mechanical answers. On the other hand, Turing's theoretical model for an algorithm machine employing a very simple set of mathematical structures held out the possibility that a physical model of Turing's idea could actually be constructed. If some human could figure out an algorithm to solve a particular class of mathematical problem, then the machine could be told to follow the steps in the program and execute this exact sequence of instructions on any inserted set of data (tirelessly and with complete precision).

The electronic discoveries that were needed for the implementation of such a device included vacuum tubes, which just coincidentally had been developed recently for engineering purposes completely unrelated to the possibility of building a calculating machine. This was another fortuitous phenomenon of this period of history. All that was required was the impetus for someone with a vast source of money to be motivated to invest in this highly speculative project. It is practically sacrilegious to maintain that World War II had a serendipitous impact on civilization no matter how unintentional, yet it was exactly in this way that the first computer was born—sponsored by the Allied military to break the German secret code, with Turing himself taking part in the construction of the machine.

What started out as a mathematical theorem about mathematical theorems—an abstraction about an abstraction—became the single most practically applied invention since the wheel and axle. Not only was this an ironic twist of fate, but it all happened within the remarkable span of 10 years. It was as incredible as if a mathematical proof of the existence of intelligent creatures in outer space were to provoke them to land immediately on Earth.

Independently of all the work being done in mathematical logic, other fields of science and social science were beginning to develop mathematical models to describe and analyze difficult problems of their own. As we have noted before, there is a natural correspondence between the study of models of computation and the study of linguistics in an abstract and mathematical sense. It is also natural to assume that the study of thinking and learning—branches of psychology and neurology—play an important part in understanding and facilitating computer theory. What is again of singular novelty is the historical fact that, rather than turning their attention to mathematical models to computerize their own applications, their initial development of mathematical models for aspects of their own science directly aided the evolution of the computer itself. It seems that half the intellectual forces in the world were leading to the invention of the computer, while the other half were producing applications that were desperate for its arrival.

Two neurophysiologists, Warren McCulloch and Walter Pitts, constructed a mathematical model for the way in which sensory receptor organs in animals behave. The model they constructed for a "neural net" was a theoretical machine of the same nature as the one Turing invented, but with certain limitations.

Modern linguists, some influenced by the prevalent trends in mathematical logic and some by the emerging theories of developmental psychology, had been investigating a very similar subject: What is language in general? How could primitive humans have developed language? How do people understand it? How do they learn it as children? What ideas can be expressed, and in what ways? How do people construct sentences from the ideas in their minds?

Noam Chomsky created the subject of mathematical models for the description of languages to answer these questions. His theory grew to the point where it began to shed light on the study of computer languages. The languages humans invented to communicate with one another and the languages necessary for humans to communicate with machines shared many basic properties. Although we do not know exactly how humans understand language, we do know how machines digest what they are told. Thus, the formulations of mathematical logic became useful to linguistics, a previously nonmathematical subject. Metaphorically, we could say that the computer then took on linguistic abilities. It became a word processor, a translator, and an interpreter of simple grammar, as well as a compiler of computer languages. The software invented to interpret programming languages was applied to human languages as well. One point that will be made clear in our studies is why computer languages are easy for a computer to understand, whereas human languages are very difficult.

Because of the many influences on its development, the subject of this book goes by various names. It includes three major fundamental areas: the **theory of automata,** the **theory of formal languages,** and the **theory of Turing machines.** This book is divided into three parts corresponding to these topics.

Our subject is sometimes called **computation theory** rather than computer theory, because the items that are central to it are the types of tasks (algorithms or programs) that can be performed, not the mechanical nature of the physical computer itself. However, the name "computation" is misleading, since it popularly connotes arithmetical operations which comprise only a fraction of what computers can do. The term computation is inaccurate when describing word processing, sorting, and searching and awkward in discussions of program verification. Just as the term "number theory" is not limited to a description of calligraphic displays of number systems but focuses on the question of which equations can be solved in integers, and the term "graph theory" does not include bar graphs, pie charts, and histograms, so too "computer theory" need not be limited to a description of physical machines but can focus on the question of which tasks are possible for which machines.

We shall study different types of theoretical machines that are mathematical models for actual physical processes. By considering the possible inputs on which these machines can work, we can analyze their various strengths and weaknesses. We then arrive at what we may believe to be the most powerful machine possible. When we do, we shall be surprised to find tasks that even it cannot perform. This will be our ultimate result, that no matter what machine we build, there will always be questions that are simple to state that it cannot answer. Along the way, we shall begin to understand the concept of **computability,** which is the foundation of further research in this field. This is our goal. Computer theory extends further to such topics as complexity and verification, but these are beyond our intended scope. Even for the topics we do cover—automata, languages, Turing machines—much more is known than we present here. As intriguing and engaging as the field has proven so far, with any luck the most fascinating theorems are yet to be discovered.