Introduction to computer theory

Technology, Computer

Background

The twentieth century has been filled with the most incredible shocks and surprises: the theory of relativity, Communist revolutions, 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.

The history of the computer is a fascinating story; however, it is not the subject of this course. We are concerned with the Theory of Computers, which means that we form several abstract mathematical models that will describe with varying degrees of accuracy parts of computers and types of computers and similar machines. Our models will not be used to discuss the practical engineering details of the hardware of computers, but the more abstract questions of the frontiers of capability of these mechanical devices.

There are separate courses that deal with circuits and switching theory (computer logic) and with instruction sets and register arrangements (computer ar-chitecture) and with data structures and algorithms and operating systems and compiler design andartificial intelligenceand so forth. All of these courses have a theoretical component, but they differ from our study in two basic ways. First, they deal only with computers that already exist; our models, on 3 4 Automate Theory the other hand, will encompass all computers that do exist, will exist, and that can ever be dreamed of.

Second, they are interested in how best to do things; we shall not be interested in optimality at all, but rather we shall be concerned with the question of possibility-what can and what cannot be done. We shall look at this from the perspective of what language structures the machines we describe can and cannot accept as input, and what possible meaning their output may have. This description of our intent is extremely general and perhaps a little misleading, but the mathematically precise definition of our study can be understood only by those who already know the concepts introduced in this course.

This is often a characteristic ofscholarship---after years of study one can just begin to define the subject. We are now embarking on a typical example of such a journey. In our last chapter (Chapter 31) we shall finally be able to define a computer. The history of Computer Theory is also 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 obvious component of Computer Theory is the theory of mathematical logic. As the twentieth century started, mathematicswas facing a dilemma. Georg Cantor (1845-1918) 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 that infinity comes in different sizes), but some could not (such as that some set is bigger than the universal set). This left a cloud over mathematics that needed to be resolved. David Hilbert (1862-1943) wanted all of mathematics put on the same sound footing as Euclidean Geometry, which is characterized by precise definitions, explicit axioms, and rigorous proofs. The format of a Euclidean proof is precisely specified. Every line is either an axiom, a previously proven theorem, or follows from the lines above it by one of a few simple rules of inference.

The mathematics that developed in the centuries since Euclid did not follow this standard of precision. Hilbert believed that if mathematics X'ere put back on the Euclidean standard the Cantor paradoxes would go away. He was actually concerned with two ambitious projects: first, to demonstrate that the new system was free of paradoxes; second, to find methods that would guarantee to enable humans to construct proofs of all the true statements in mathematics. Hilbert wanted something formulaic-a precise routine for producing results, like the directions in a cookbook.

First draw all these lines, then write all these equations, then solve for all these points, and so on and so on and the proof is done-some approach that is certain and sure-fire without any reliance Background on unpredictable and undependable brilliant mathematical insight. We simply follow the rules and the answer must come. This type of complete, guaranteed, easy-to-follow set of instructions is called an algorithm. He hoped that algorithms or procedures could be developed to solve whole classes of mathematical problems.

The collection of techniques called linear algebra provides just such an algorithm for solving all systems of linear equations. Hilbert wanted to develop algorithms for solving other mathematical problems, perhaps even an algorithm that could solve all mathematical problems of any kind in some finite number of steps. Before starting to look for such an algorithm, an exact notion of what is and what is not a mathematical statement had to be developed. After that, there was the problem of defining exactly what can and what cannot be a step in an algorithm.

The words we have used: " procedure," " formula," " cookbook method," " complete instructions," are not part of mathematics and are no more meaningful than the word " algorithm" itself. Mathematical logicians, while trying to follow the suggestions of Hilbert and straighten out the predicament left by Cantor, found that they were able to prove mathematically that some of the desired algorithms cannot exist-not only at this time, but they can never exist in the future, either. Their main I result was even more fantastic than that.

Kurt Godel (1906-1978) not only showed that there was no algorithm that could guarantee to provide proofs for all the true statements in mathematics, but he proved that not all the true statements even have a proof to be found. G6del's Incompleteness Theorem implies that in a specific mathematical system either there are some true statements without any possible proof or else there are some false statements that can be " proven.

" This earth-shaking result made the mess in thephilosophyof mathematics even worse, but very exciting.

If not every true statement has a proof, can we at least fulfill Hilbert's program by finding a proof-generating algorithm to provide proofs whenever they do exist? Logicians began to ask the question: Of what fundamental parts are all algorithms composed? The first general definition of an algorithm was proposed by Alonzo Church. Using his definition he and Stephen Cole Kleene and, independently, Emil Post were able to prove that there were problems that no algorithm could solve. While also solving this problem independently, Alan Mathison Turing (1912-1954) developed the concept of a theoretical " universal-algorithm machine. Studying what was possible and what was not possible for such a machine to do, he discovered that some tasks that we might have expected this abstract omnipotent machine to be able to perform are impossible, even for it. Turing's model for a universal-algorithm machine is directly connected to the invention of the computer. In fact, for completely different reasons (wartime code-breaking) Turing himself had an important part in the construction of the first computer, which he based on his work in abstract logic.

On a wildly different front, two researchers in neurophysiology, Warren Sturgis McCulloch and Walter Pitts (1923-1969), 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. Mathematical models of real and abstract machines took on more and more importance.

Along with mathematical models for biological processes, models were introduced to study psychological, economic, and social situations. Again, entirely independent of these considerations, the invention of the vacuum tube and the subsequent developments in electronics enabled engineers to build fully automatic electronic calculators. These developments fulfilled the age-old dream of Blaise Pascal (1623-1662), Gottfried Wilhelm von Leibniz (1646-1716), and Charles Babbage (1792-1871), all of whom built mechanical calculating devices as powerful as their respective technologies would allow.

In the 1940s, gifted engineers began building the first generation of computers: the computer Colossus at Bletchley, England (Turing's decoder), the ABC machine built by John Atanosoff in Iowa, theHarvardMark I built by Howard Aiken, and ENIAC built by John Presper Eckert, Jr. and John William Mauchly (1907-1980) at the University of Pennsylvania. Shortly after the invention of the vacuum tube, the incredible mathematician John von Neumann (1903-1957) developed the idea of a stored-program computer.

The idea of storing the program inside the computer and allowing the computer to operate on (and modify) the program as well as the data was a tremendous advance. It may have been conceived decades earlier by Babbage and his co-worker Ada Augusta, Countess of Lovelace (1815-1853), but theirtechnologywas not adequate to explore this possibility. The ramifications of this idea, as pursued by von Neumann and Turing were quite profound. The early calculators could perform only one predetermined set of tasks at a time.

To make changes in their procedures, the calculators had to be physically rebuilt either by rewiring, resetting, or reconnecting various parts. Von Neumann permanently wired certain operations into the machine and then designed a central control section that, after reading input data, could select which operation to perform based on a program or algorithm encoded in the input and stored in the computer along with the raw data to be processed. In this way, the inputs determined which operations were to be performed on themselves.

Interestingly, current technology has progressed to the point where the ability to manufacture dedicated chips cheaply and easily has made the prospect of rebuilding a computer for each program feasible again. However, by the last chapters of this book we will appreciate the significance of the difference between these two approaches. Von Neumann's goal was to convert the electronic calculator into a reallife model of one of the logicians' ideal universal-algorithm machines, such as those Turing had described.

Thus we have an unusual situation where the advanced theoretical work on the potential of the machine preceded the demonstration that the machine could really exist. The people who first discussed these machines only dreamed they might ever be built. Many were very surprised to find them actually working in their own lifetimes. Along with the concept of programming a computer came the question: What is the " best" language in which to write programs?

Many languages were invented, owing their distinction to the differences in the specific machines they were to be used on and to the differences in the types of problems for which they were designed. However, as more languages emerged, it became clear that they had many elements in common. They seemed to share the same possibilities and limitations. Thisobservationwas at first only intuitive, although Turing had already worked on much the same problem but from a different angle. At the time that a general theory of computer languages was being developed, another surprise occurred.

Modem linguists, some influenced by the prevalent trends in mathematical logic and some by the emerging theories of developmentalpsychology, 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, since 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 also misleading, since it popularly connotes arithmetical operations that are 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.