Chapter 1

Introduction

The theory of computability concerns what it is possible for a human or a computing machine to accomplish by following an explicit, step-by-step procedure, irrespective of the (finite) amount of scratch paper or time required to complete the computation. As such, the theory is as much about what cannot be computed as it is about what can be computed. In fact, it is more about what cannot be computed because positive computability results can be obtained by simply providing a correct program (i.e., by "hacking"). To prove that something cannot be done requires some theory. A series of failed attempts to find a positive solution doesn't suffice to prove that something can't be done, for there may be a much more complicated or ingenious solution nobody has thought of. One needs to mathematize the notion of possible solutions and to prove that no possible solution is an actual solution.

1.1 Cartoon History

Where did the theory of computability come from? Ironically, not from computers, which were barely on the scene at the time. It arose from a philosophical critique of mathematical method, through a series of intellectual twists and turns. Here is a cartoon version of the standard history.

1.1.1 Foundational concerns

By the second half of the nineteenth century, Newton's physics, founded on his calculus, was widely regarded as approaching a successful completion. Unfortunately, free-wheeling algebraic manipulations could lead to contradictions (e.g., Leibniz applied algebraic operations to non-convergent series). Since nobody was in the mood to throw away physics, a clearer understanding of the calculus was required.

1.1.2 Analysis and more foundational concerns

The foundations of the calculus are known as analysis. In standard analysis, real numbers are defined as infinite sequences of rational numbers. Set theory was developed, in part, to understand these infinite sequences and calculus was put on a much more solid footing than it had been on previously. Formal proofs of the fundamental theorem of calculus, etc. could be given. As David Hilbert put it, the "infinitely small" of naive calculus had been replaced with "infinitely large" sets. However, the infinitely large sets raised difficulties of their own. First, there were new and recondite questions whose relevance to mainstream mathematics was unclear (e.g., the continuum hypothesis). More importantly, the original versions of set theory were all inconsistent as Russell showed. And it was the main idea of set theory, the "comprehension" notation that allows one to move from properties to sets, that led to the immediate contradiction:

$$\{x : x \notin x\} \in \{x : x \notin x\} \Leftrightarrow \{x : x \notin x\} \notin \{x : x \notin x\}.$$

The immediate problem could be patched in various ways. The way we still use (Zermelo-Frankel set theory) is to restrict the comprehension schema to members of some pre-existing set U, so the Russell construction becomes:

$$\{x \in U : x \notin x\}.$$

Then Russell's paradox is "harnessed" as a reductio proof that there exists no set U such that every set that is not a member of itself is in U. For if there were, then comprehension allows one to "carve out" of U the set of all sets that don't contain themselves and Russell's argument yields a contradiction (try it, in full detail). But blocking a one-liner doesn't rule out the possibilitiy of other, more subtle arguments to a contradiction. The fact that set theory employs some bold, unfamiliar postulates with remarkable consequences (e.g., the axiom of choice) didn't inspire confidence that further contradictions would not be forthcoming (e.g., can one construct U by other means available in the theory?).

1.1.3 Finitism:

One response to the consistency problem was a kind of "back-to-basics" fundamentalism in which actual infinity was to be purged from the mathematical corpus. The trouble here was not to throw the baby (calculus) out with the bathwater (actual infinity).

1.1.4 Formalism and David Hilbert's "Program"

David Hilbert agreed that the primary topic of mathematics is the natural numbers. But that doesn't mean that we should throw infinity away. It's too useful. And it doesn't matter whether it's true or false as long as it helps us to obtain elegant proofs of truths about the natural numbers and doesn't lead to contradiction (in which case it would prove too much about the natural numbers). But how can we check that it doesn't lead to contradiction? By assigning numbers to sentences and proofs and then showing in number theory that no contradiction is ever provable. After all, proofs are finite, discrete objects that can be checked according to determinate rules. Surely, number theory should be capable of showing that no contradiction can be produced by a proof. Then there is no need to restrict mathematics as the finitists recommend and the easy proofs invoking infinite objects can be retained. The idea that the mathematics of the infinite is a useful instrument for studying the natural numbers and that truth is irrelevant where infinity is concerned is often called "formalism" although that was not Hilbert's word.

1.1.5 Platonism and Gödel's incompleteness theorems

Kurt Gödel thought that infinite sets could be as real as atoms and molecules: the latter are invoked to explain observable features of nature and the former are invoked to account for complicated number theoretical facts. The view that mathematical objects are real is known as Platonism, after the 4th c. B.C. Athenian philosopher who thought that mathematics is a literal theory of unchanging, indestructable objects, whose immutability accounts for mathematical necessity. Gödel demolished Hilbert's program by showing that no extension of a very weak system of arithmetic can show its own consistency. Since Hilbert wanted a weak theory of arithmetic to show the consistency of set theory, Gödel's theorem crushed the foundational ambitions of the program (more restricted versions survive, for it may be possible to get "more than you would expect" out of a weak system of arithmetic).

1.1.6 Computability and the Church-Turing Thesis

Gödel's incompleteness phenomenon concerns what a "system" of arithmetic can accomplish, but what is a "system" of arithmetic? Clearly, asking God whether we are consistent should work, so that is not a "system". Gödel proposed a sort of ad hoc definition of "system" but one could surely wonder whether his account of "system" somehow overlooked some feature of mathematical practice that might make real mathematics powerful enough to prove its own consistency. To fully assess the significance of Gödel's incompleteness theorems one would like to seek the broadest conception of "system" for which the theorem would still hold. A touchstone of mathematics is that whereas considerable ingenuity is required to find a proof, there is a rote procedure for checking whether each line of the proof follows from previous lines. So it was natural to try to introduce an explicit mathematical model of computability that could be argued to embrace everything a mathematician following an explicit procedure could accomplish. Several different formalizations of computability were proposed, all of which turned out to be equivalent. More significantly, Alan Turing argued that a mathematician following a procedure may always be represented by a Turing machine. Turing's argument (it can't be a proof in set theory because "algorithm" is not a set theoretical concept) provided strong evidence that Gödel did not overlook some crucial aspect of mathematical practice. The Church-Turing thesis states that everything computable by an explicit algorithm is computable by a Turing machine and is based upon these two arguments (primarily the latter).

1.1.7 Freedom

The theory of computability soon took on a life of its own. Real digital computers were slowly arriving on the scene (Turing worked with one when he was cracking the Nazi's "Enigma" coding machine during the second world war). What such machines can do is an interesting question in its own right. Computability spread from departments of mathematics and philosophy to engineering and finally to new programs in computer science. It has been applied to questions of concrete problem solvability and to general questions about what computers can learn from experience. It also gave birth to the more practically relevant theory of NP completeness, which was explicitly developed along the lines of classical computability theory. Finally, the theory has been pursued for its own sake, just the way set theory is.

1.2 Computability and Learnability

My primary interest is not the philosophy or foundations of mathematics, but the philsosophy of scientific discovery and learning. So why should I care about computability? After all, Hilbert's program, Gödel's incompleteness theorems, and proof systems are all primarily mathematical concerns. Moreover, proofs and algorithms yield certainty, which empirical science usually cannot do. It seems I am barking up quite the wrong tree.

Indeed, traditional empiricist philosophers held that all truths may be divided into "relations of ideas" (i.e., vacuous logical truths) and "matters of fact" (hypotheses that have non-trivial empirical consequences). The former are certain and relatively unproblematic (the world needn't be consulted). The latter are uncertain and messy (no matter how much evidence we gather about the world, we could be surprised tomorrow). The philosophy of science is supposed to focus on the special difficulties concerning knowledge of matters of fact. In the philosophy of mathematics one speaks of "proofs" and "algorithms". In the philosophy of science one speaks of "empirical justification" and "confirmation" in the face of uncertainty. The articulation of the principles of confirmation is, therefore, the special province of the philosophy of science.

As plausible as it sounds, the picture is wrong. The mistaken assumption is that "relations of ideas" are certain because they are "internal". In the theory of computability, clearly definable formal questions turn out to be unsolvable for reasons quite analogous to the skeptical problem of induction. Consider the "halting problem", which asks us to determine whether an arbitrary, given program will halt (produce an output after finite time) on an arbitrary, given input. The intuitive difficulty is that no matter how long we have watched a computer grind away on a given input, as soon as we conclude that it is in an infinite loop it may fool us by halting at the very next moment. One may think that by looking at the actual code of the program in question one could tell much more than by simply waiting for it to halt, and in special cases that is true (else debugging code would be impossible). But there are some beautiful results in computability to the effect that eventually the programs become so hard to understand that a computable agent can do no better by looking at the code than by behavioristically running the program on different inputs to see what will happen. All of this sounds a lot like Hume's problem of induction (will the bread that has nourished in the past continue to do so in the future?).

So here is an alternative view: uncomputability and incompleteness are simply "internal" manifestations of the philosophical problem of induction. If this is right, then instead of basing one's theory of knowledge on a fundamental distinction between formal and empirical reasoning, one ought to seek a unified account of both from the ground up. That is what computational learning theory is about. Computational learning theory is a computability-based study of what computational agents can learn from experience, in the sense of arriving at a correct answer eventually on the basis of an unending stream of input data.

1.3 Hierarchies

What do hierarchies have to do with computability?

1.3.1 Closure

We can think of lots of operations for building new stuff from old stuff. A set of objects (e.g., problems) is closed under a set of techniques if applying the techniques over and over doesnt generate anything new. Closure is nice because then we don't have to worry about operations being undefined over a given collection of arguments. But what do we do when a collection isn't closed under the operations in question?

1.3.2 Hierarchy = Failed Closure Law

Then closing under all the operations generates "new" objects. If the operation still isn't closed, then applying the operations again results in a larger collection, and so forth. Objects that require more applications of the operation in question may be thought of as more complex. The successively larger collections so generated are called complexity classes and the entire system of nested complexity classes is called a complexity hierarchy. Once we have a complexity hierarchy, we understand an object by placing it in its appropriate complexity class.

1.3.3 Hierarchies of Unsolvability

Suppose that the objects under consideration are problems and that the solvable problems are not preserved under some operations that build new problems out of old problems (e.g., combining questions by logical connectives). Then we obtain a complexity hierarchy of unsolvable problems. If we have a concept of "no harder than" we can define "harder than" and it may turn out that higher complexity classes always contain problems harder than lower ones. Then complexity matches intrinsic computational difficulty. In light of the analogy between uncomputability and unlearnability discussed above, it is also possible to define parallel hierarchies of unlearnability that reflect degrees of what philosophers of science call "underdetermination" of theory by fact.

1.3.4 Some Hierarchies

Number of recursions: Grzegorczyk hierarchy.

Number of alternations of "and" and "or":] trial-and-error hierarchy.

Number of alternations between discrete quantifiers: arithmetical hierarchy.

Number of alternations between continuous quantifiers: analytical hierarchy.

Levels of discontinuity: Baire hierarchy.

Number of alternations between finite intersection and union: difference hierarchy.

Number of alternations of countable union and intersection: Borel hierarchy.

Number of projections onto a subspace: projective hierarchy.

1.3.5 Mathematical Analogies

A lot of interesting logical questions arise in a natural way when one compares one complexity hierarchy to another. One such analogy concerns computability and topological continuity. I will argue that this is no accident: both concepts have to do with a kind of empirical verification.

Computability theory	:	topology ::
Computable	:	continuous ::
Semi-decidable	:	*open* ::
Arithmetically definable	:	Borel ::
Analytic	:	projective ::
Formal	:	empirical.

1.3.6 Vagueness of Abilities

Plato convinced philosophers from the very beginning that they have to find a precise, univocal explication of each vague concept they use before they can use it. Solvability concepts like tractable, learnable, discoverable and definable are vague. (How easy is tractable? How learnable is learnable?) Instead of defining univocal senses for such terms, we can define a precise complexity hierarchy and study the precise degrees of the concept under consideration. This way, sharp-edged logic can be used to study fuzzy concepts of success, short-circuiting interminable qualitative debates about what really counts as success. The potential for interesting philosophical applications is apparent when one considers the following:

Computability	=	the computable.
Epistemology	=	the knowable.
The problem of induction	=	the learnable.
Philosophy of mind	=	the physiologically definable.
Positivism	=	the observationally definable.
Philosophy of mathematics	=	the provable.
Cryptography	=	the unbreakable.