Computability and Decidability 13

Key Topics

Computability
Completeness
Decidability
Formalism
Logicism

13.1 Introduction

It is impossible for a human or machine to write out all of the members of an infinite countable set, such as the set of natural numbers ℕ. However, humans can do something quite useful in the case of certain enumerable infinite sets: they can give explicit instructions (that may be followed by a machine or another human) to produce the *n*th member of the set for an arbitrary finite *n*. The problem remains that for all but a finite number of values of *n* it will be physically impossible for any human or machine to actually carry out the computation, due to the limitations on the time available for computation, the speed at which the individual steps in the computation may be carried out, and due to finite materials.

The intuitive meaning of computability is in terms of an algorithm (or effective procedure) that specifies a set of instructions to be followed to complete the task. Another words, a function *f* is *computable* if there exists an algorithm that produces the value of *f* correctly for each possible argument of *f*. The computation of *f* for a particular argument *x* just involves following the instructions in the algorithm, and

G. O'Regan, *Guide to Discrete Mathematics*, Texts in Computer Science,
DOI 10.1007/978-3-319-44561-8_13

it produces the result $f(x)$ in a finite number of steps if x is in the domain of f. If x is not in the domain of f then the algorithm may produce an answer saying so or it might run forever never halting. A computer program implements an algorithm.

The concept of computability may be made precise in several equivalent ways such as Church's *lambda calculus*, *recursive function theory* or by the theoretical *Turing machines*.[1] These are all equivalent and perhaps the most well known is the Turing machine (discussed in Chap. 7). This is a mathematical machine with a potentially infinite tape divided into frames (or cells) in which very basic operations can be carried out. The set of functions that are computable are those that are computable by a Turing machine.

Decidability is an important topic in contemporary mathematics. Church and Turing independently showed in 1936 that mathematics is not decidable. In other words, there is no mechanical procedure (i.e., algorithm) to determine whether an arbitrary mathematical proposition is true or false, and so the only way is to determine the truth or falsity of a statement is try to solve the problem. The fact that there is no a general method to solve all instances of a specific problem, as well as the impossibility of proving or disproving certain statements within a formal system may suggest limitations to human and machine knowledge.

13.2 Logicism and Formalism

Gottlob Frege (Fig. 14.8) was a nineteenth century German mathematician and logician who invented a formal system which is the basis of modern predicate logic. It included axioms, definitions, universal and existential quantification and formalization of proof. His objective was to show that mathematics was reducible to logic (logicism) but his project failed as one of the axioms that he had added to his system proved to be inconsistent.

This inconsistency was pointed out by Bertrand Russell, and it is known as *Russell's paradox*.[2] Russell later introduced the theory of types to deal with the paradox, and he jointly published *Principia Mathematica* with Whitehead as an attempt to derive the truths of arithmetic from a set of logical axioms and rules of inference.

The sentences of Frege's logical system denote the truth values of true or false. The sentences may include expressions such as equality ($x = y$), and this returns true if x is the same as y, and false otherwise. Similarly, a more complex expression such as $f(x, y, z) = w$ is true if $f(x, y, z)$ is identical with w, and false otherwise. Frege represented statements such as "5 is a prime" by "$P(5)$" where $P()$ is termed a concept. The statement $P(x)$ returns true if x is prime and false otherwise. His

[1]The Church-Turing Thesis states that anything that is computable is computable by a Turing Machine.

[2]Russell's paradox (discussed in Chap. 2) considers the question as to whether the set of all sets that contain themselves as members is a set. In either case there is a contradiction.

Fig. 13.1 David Hilbert

approach was to represent a predicate as a function of one variable which returns a Boolean value of true or false.

Formalism was proposed by Hilbert (Fig. 13.1) as a foundation for mathematics in the early twentieth century. The motivation for the programme was to provide secure foundations for mathematics, and to resolve the contradictions in the formalization of set theory identified by Russell's paradox. The presence of a contradiction in a theory means the collapse of the whole theory, and so it was seen as essential that there be a proof of the consistency of the formal system. The methods of proof in mathematics are formalized with axioms and rules of inference.

Formalism is a formal system that contains meaningless symbols together with rules for manipulating them. The individual formulas are certain finite sequences of symbols obeying the syntactic rules of the formal language. A formal system consists of

- A formal language
- A set of axioms
- Rules of inference.

The expressions in a formal system are terms, and a term may be simple or complex. A simple term may be an object such as a number, and a complex term may be an arithmetic expression such as $4^3 + 1$. A complex term is formed via functions, and the expression above uses two functions namely the cube function with argument 4 and the plus function with two arguments.

A formal system is generally intended to represent some aspect of the real world. A rule of inference relates a set of formulas (P_1, P_2, …, P_k) called the premises to the consequence formula P called the conclusion. For each rule of inference there is a finite procedure for determining whether a given formula Q is an immediate consequence of the rule from the given formulas (P_1, P_2, ..., P_k). A *proof* in a formal system consists of a finite sequence of formulae, where each formula is either an axiom or derived from one or more preceding formulae in the sequence by one of the rules of inference.

Hilbert's programme was concerned with the formalization of mathematics (i.e. the axiomatization of mathematics) together with a proof that the axiomatization was consistent (i.e., there is no formula A such that both A and $\neg A$ are deducible in the calculus). The specific objectives of Hilbert's programme were to

- Provide a formalism of mathematics.
- Show that the formalization of mathematics was *complete*: i.e. all mathematical truths can be proved in the formal system.
- Provide a proof that the formal system is *consistent* (i.e. that no contradictions may be derived).
- Show that mathematics is *decidable*: i.e. there is an algorithm to determine the truth of falsity of any mathematical statement.

The formalist movement in mathematics led to the formalization of large parts of mathematics, where theorems could be proved using just a few mechanical rules. The two most comprehensive formal systems developed were *Principia Mathematica* by Russell and Whitehead, and the axiomatization of set theory by Zermelo–Fraenkel (subsequently developed further by von Neumann).

Principia Mathematica is a comprehensive three volume work on the logical foundations of mathematics written by Bertrand Russell and Alfred Whitehead between 1910 and 1913. Its goal was to show that all of the concepts of mathematics can be expressed in logic, and that all of the theorems of mathematics can be proved using only the logical axioms and rules of inference of logic. It covered set theory, ordinal numbers and real numbers, and it showed that in principle that large parts of mathematics could be developed using *logicism*.

It avoided the problems with contradictions that arose with Frege's system by introducing the theory of types in the system. The theory of types meant that one could no longer speak of the set of all sets, as a set of elements is of a different type from that of each of its elements, and so Russell's paradox was avoided. It remained an open question at the time as to whether the *Principia* were consistent and complete. That is, is it possible to derive all the truths of arithmetic in the system and is it possible to derive a contradiction from the Principia's axioms? However, it was clear from the three volume work that the development of mathematics using the approach of the Principia was extremely lengthy and time consuming.

13.3 Decidability

The question remained whether these axioms and rules of inference are sufficient to decide any mathematical question that can be expressed in these systems. Hilbert believed that every mathematical problem could be solved, and that the truth or falsity of any mathematical proposition could be determined in a finite number of steps. He outlined twenty-three key problems in 1900 that needed to be solved by mathematicians in the twentieth century.

He believed that the formalism of mathematics would allow a mechanical procedure (or algorithm) to determine whether a particular statement was true or false. The problem of the decidability of mathematics is known as the decision problem (*Entscheidungsproblem*).

The question of the decidability of mathematics had been considered by Leibnitz in the seventeenth century. He had constructed a mechanical calculating machine, and wondered if a machine could be built that could determine whether particular mathematical statements are true or false.

Definition 13.1 (*Decidability*) Mathematics is decidable if the truth or falsity of any mathematical proposition may be determined by an algorithm.

Church and Turing independently showed this to be impossible in 1936. Church developed the lambda calculus in the 1930s as a tool to study computability,[3] and he showed that anything that is computable by the lambda calculus. Turing showed that decidability was related to the halting problem for Turing machines, and that therefore if first-order logic was decidable then the halting problem for Turing machines could be solved. However, he had already proved that there was no general algorithm to determine whether a given Turing machine halts. Therefore, first-order logic is undecidable.

The question as to whether a given Turing machine halts or not can be formulated as a first-order statement. If a general decision procedure exists for first-order logic, then the statement of whether a given Turing machine halts or not is within the scope of the decision algorithm. However, Turing had already proved that the halting problem for Turing machines is not computable: i.e. it is not possible algorithmically to decide whether or not any given Turing machine will halt or not. Therefore, since there is no general algorithm that can decide whether any given Turing machine halts, there is no general decision procedure for first-order logic. The only way to determine whether a statement is true or false is to try to solve it. However, if one tries but does not succeed this does not prove that an answer does not exist

There are first-order theories that are decidable. However, first-order logic that includes Peano's axioms of arithmetic (or any formal system that includes addition and multiplication) cannot be decided by an algorithm. That is, there is no algorithm to determine whether an arbitrary mathematical proposition is true or false. Propositional logic is decidable as there is a procedure (e.g. using a truth table) to determine whether an arbitrary formula is valid[4] in the calculus.

Gödel (Fig. 13.2) proved that first-order predicate calculus is *complete*. i.e. all truths in the predicate calculus can be proved in the language of the calculus.

[3]The Church Turing Thesis states that anytime that is computable is computable by Lambda Calculus or equivalently by a Turing Machine.

[4]A well-formed formula is valid if it follows from the axioms of first-order logic. A formula is valid if and only if it is true in every interpretation of the formula in the model.

Fig. 13.2 Kurt Gödel

Definition 13.2 (*Completeness*) A formal system is complete if all the truths in the system can be derived from the axioms and rules of inference.

Gödel's *first incompleteness theorem* showed that first-order arithmetic is incomplete; i.e. there are truths in first-order arithmetic that cannot be proved in the language of the axiomatization of first-order arithmetic. Gödel's *second incompleteness theorem* showed that any formal system extending basic arithmetic cannot prove its own consistency within the formal system.

Definition 13.3 (*Consistency*) A formal system is consistent if there is no formula A such that A and $\neg A$ are provable in the system (i.e. there are no contradictions in the system).

13.4 Computability

Alonzo Church (Fig. 13.3) developed the lambda calculus in the mid 1930s, as part of his work into the foundations of mathematics. Turing published a key paper on computability in 1936, which introduced the theoretical machine known as the Turing machine. This machine is computationally equivalent to the lambda

Fig. 13.3 Alonzo Church

calculus, and is capable of performing any conceivable mathematical problem that has an algorithm.

Definition 13.4 (*Algorithm*)An algorithm(or effective procedure) is a finite set of unambiguous instructions to perform a specific task.

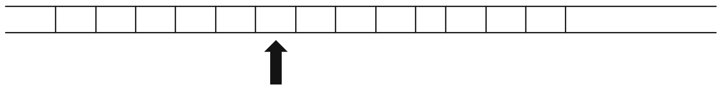
A function is *computable* if there is an effective procedure or algorithm to compute *f* for each value of its domain. The algorithm is finite in length and sufficiently detailed so that a person can execute the instructions in the algorithm. The execution of the algorithm will halt in a finite number of steps to produce the value of $f(x)$ for all x in the domain of f. However, if x is not in the domain of f then the algorithm may produce an answer saying so, or it may get stuck, or it may run forever never halting.

The *Church–Turing Thesis* that states that *any computable function may be computed by a Turing machine*. There is overwhelming evidence in support of this thesis, including the fact that alternative formalizations of computability in terms of lambda calculus, recursive function theory, and post-systems have all been shown to be equivalent to Turing machines.

A Turing machine (discussed previously in Chap. 7) consists of a head and a potentially infinite tape that is divided into cells. Each cell on the tape may be either blank or printed with a symbol from a finite alphabet of symbols. The input tape may initially be blank or have a finite number of cells containing symbols.

At any step, the head can read the contents of a frame. The head may erase a symbol on the tape, leave it unchanged, or replace it with another symbol. It may then move one position to the right, one position to the left, or not at all. If the frame is blank, the head can either leave the frame blank or print one of the symbols.

Turing believed that a human with finite equipment and with an unlimited supply of paper could do every calculation. The unlimited supply of paper is formalized in the Turing machine by a tape marked off in cells.



We gave a formal definition of a Turing machine as *a* 7-tuple M = $(Q, \Gamma, b, \Sigma, \delta, q_0, F)$ in Chap. 7. We noted that the Turing machine is a simple theoretical machine, but it is equivalent to an actual physical computer in the sense that they both compute exactly the same set of functions. A Turing machine is easier to analyse and prove things about than a real computer.

A Turing machine is essentially a finite state machine (FSM) with an unbounded tape. The machine may read from and write to the tape and the tape provides memory and acts as the store. The finite state machine is essentially the control unit of the machine, whereas the tape is a potentially infinite and unbounded store. A real computer has a large but finite store whereas the store in a Turing machine is potentially infinite. However, the store in a real computer may be extended with backing tapes and disks, and in a sense may be regarded as unbounded. The maximum amount of tape that may be read or written within n steps is n.

A Turing machine has an associated set of rules that defines its behaviour. These rules are defined by the transition function that specifies the actions that a machine will perform with respect to a particular input. The behaviour will depend on the current state of the machine and the contents of the tape.

A Turing machine may be programmed to solve any problem for which there is an algorithm. However, if the problem is unsolvable then the machine will either stop in a non-accepting state or compute forever. The solvability of a problem may not be determined beforehand, but, there is, of course, some answer (i.e. either the machine either halts or it computes forever).

Turing showed that there was no solution to the decision problem (*Entscheidungsproblem*) posed by Hilbert. Hilbert believed that the truth or falsity of a mathematical problem may always be determined by a mechanical procedure, and he believed that first-order logic is decidable: i.e. there is a decision procedure to determine if an arbitrary formula is a theorem of the logical system.

Turing was skeptical on the decidability of first-order logic, and the Turing machine played a key role in refuting Hilbert's claim of the decidability of first-order logic.

The question as to whether a given Turing machine halts or not can be formulated as a first-order statement. If a general decision procedure exists for first-order logic, then the statement of whether a given Turing machine halts or not is within the scope of the decision algorithm. However, Turing had already proved that the halting problem for Turing machines is not computable: i.e. it is not possible algorithmically to decide whether a give Turing machine will halt or not. Therefore, there is no general algorithm that can decide whether a given Turing machine halts. In other words, there is no general decision procedure for first-order logic. The only way to determine whether a statement is true or false is to try to solve it.

Turing also introduced the concept of a Universal Turing Machine and this machine is able to simulate any other Turing machine. Turing's results on computability were proved independently of Church's lambda calculus equivalent results in computability. Turing's studied at Princeton University in 1937 and 1938 and was awarded a PhD from the university in 1938. His research supervisor was Alonzo Church.[5]

Question 13.1 (Halting Problem)

Given an arbitrary program is there an algorithm to decide whether the program will finish running or will continue running forever? Another words, given a

[5]Alonzo Church was a famous American mathematician and logician who developed the lambda calculus. He also showed that Peano arithmetic and first order logic were undecidable. Lambda calculus is equivalent to Turing machines and whatever may be computed is computable by Lambda calculus or a Turing machine.

program and an input will the program eventually halt and produce an output or will it run forever?

Note (Halting Problem)
The halting problem was one of the first problems that was shown to be undecidable: i.e. there is no general decision procedure or algorithm that may be applied to an arbitrary program and input to decide whether the program halts or not when run with that input.

Proof We assume that there is an algorithm (i.e. a computable function (i, j)) that takes any program i (program i refers to the ith program in the enumeration of all the programs) and arbitrary input j to the program such that

$$H(i,j) = \begin{cases} 1 & \text{If program } i \text{ halts on input } j. \\ 0 & \text{otherwise} \end{cases}$$

We then employ a diagonalization argument[6] to show that every computable total function f with two arguments differs from the desired function H. First, we construct a partial function g from any computable function f with two arguments such that g is computable by some program e.

$$g(i) = \begin{cases} 0 & \text{if} f(i,i) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

There is a program e that computes g and this program is one of the programs in which the halting problem is defined. One of the following two cases must hold

$$g(e) = f(e,e) = 0 \tag{13.1}$$

In this case $H(e, e) = 1$ because e halts on input e.

$$g(e) \text{ is undefined and} f(e,e) \neq 0. \tag{13.2}$$

In this case $H(e, e) = 0$ because the program e does not halt on input e.

In either case, f is not the same function as H. Further, since f was an arbitrary total computable function all such functions must differ from H. Hence, the function H is not computable and there is no such algorithm to determine whether an arbitrary Turing machine halts for an input x. Therefore, the halting problem is not decidable.

[6]This is similar to Cantor's diagonalization argument that shows that the Real numbers are uncountable. This argument assumes that it is possible to enumerate all real numbers between 0 and 1, and it then constructs a number whose nth decimal differs from the nth decimal position in the nth number in the enumeration. If this holds for all n then the newly defined number is not among the enumerated numbers.

13.5 Computational Complexity

An algorithm is of little practical use if it takes millions of years to compute particular instances. There is a need to consider the efficiency of the algorithm due to practical considerations. Chapter 10 discussed cryptography and the RSA algorithm, and the security of the RSA encryption algorithm is due to the fact that there is no known efficient algorithm to determine the prime factors of a large number.

There are often slow and fast algorithms for the same problem, and a measure of complexity is the number of steps in a computation. An algorithm is of *time complexity f*(n) if for all n and all inputs of length n the execution of the algorithm takes at most $f(n)$ steps.

An algorithm is said to be *polynomially bounded* if there is a polynomial $p(n)$ such that for all n and all inputs of length n the execution of the algorithm takes at most $p(n)$ steps. The notation P is used for all problems that can be solved in polynomial time.

A problem is said to be *computationally intractable* if it may not be solved in polynomial time—there is no known algorithm to solve the problem in polynomial time.

A problem L is said to be in the set *NP* (non-deterministic polynomial time problems) if any given solution to L can be verified quickly in polynomial time. A non-deterministic Turing machine may have several possibilities for its behaviour, and an input may give rise to several computations.

A problem is *NP complete* if it is in the set NP of non-deterministic polynomial time problems and it is also in the class of *NP hard* problems. A key characteristic to NP complete problems is that there is no known fast solution to them, and the time required to solve the problem using known algorithms increases quickly as the size of the problem grows. Often, the time required to solve the problem is in billions or trillions of years. Although any given solution can be verified quickly there is no known efficient way to find a solution.

13.6 Review Questions

1. Explain computability and decidability.
2. What were the goals of logicism and formalism and how successful were these movement in mathematics?
3. What is a formal system?
4. Explain the difference between consistency, completeness and decidability.

5. Describe a Turing machine and explain its significance in computability.
6. Describe the halting problem and show that it is undecidable.
7. Discuss the complexity of an algorithm and explain terms such as 'polynomial bounded', 'computationally intractable' and 'NP complete'.

13.7 Summary

This chapter provided an introduction to computability and decidability. The intuitive meaning of computability is that in terms of an algorithm (or effective procedure) that specifies a set of instructions to be followed to solve the problem. Another words, a function *f* is computable if there exists an algorithm that produces the value of *f* correctly for each possible argument of *f*. The computation of *f* for a particular argument *x* just involves following the instructions in the algorithm, and it produces the result *f*(*x*) in a finite number of steps if *x* is in the domain of *f*.

The concept of computability may be made precise in several equivalent ways such as Church's lambda calculus, recursive function theory or by the theoretical Turing machines. The Turing machine is a mathematical machine with a potentially infinite tape divided into frames (or cells) in which very basic operations can be carried out. The set of functions that are computable are those that are computable by a Turing machine.

A formal system contains meaningless symbols together with rules for manipulating them, and is generally intended to represent some aspect of the real world. The individual formulas are certain finite sequences of symbols obeying the syntactic rules of the formal language. A formal system consists of a formal language, a set of axioms and rules of inference

Church and Turing independently showed in 1936 that mathematics is not decidable. In other words it is not possible to determine the truth or falsity of any mathematical proposition by an algorithm.

Turing had already proved that the halting problem for Turing machines is not computable: i.e. it is not possible algorithmically to decide whether a given Turing machine will halt or not. He then applied this result to first-order logic to show that it is undecidable. That is, the only way to determine whether a statement is true or false is to try to solve it.

The complexity of an algorithm was discussed, and it was noted that an algorithm is of little practical use if it takes millions of years to compute the solution. There is a need to consider the efficiency of the algorithm due to practical considerations. The class of polynomial time bound problems and non-deterministic polynomial time problems were considered, and it was noted that the security of various cryptographic algorithms is due to the fact that there are no time efficient algorithms to determine the prime factors of large integers.

The reader is referred to [1] for a more detailed account of decidability and computability.

Reference

1. Cornerstones of Undecideability. Grzegorz Rozenberg and Arto Salomaa. Prentice Hall. 1994.