Chapter 2

Recursivity and Decidability

1. Introduction

The concept of decidability and the related concepts of semi-decidability and recursivity are fundamental to information technology as a whole, for they specify and characterize the objects dealt with by the subject. Thus:

1. the only problems that a [computer] program is able to solve are the decidable problems, that is, problems that can be expressed as a decidable predicate; and this term has a strict mathematical interpretation that is independent of the programming language;
2. the only functions that a [computer] program is able to calculate are the recursive functions, and again the term can be defined mathematically without ambiguity.

In this chapter we develop these ideas, not only because of their general importance, but also because they are essential in defining, formulating and explaining certain parts of mathematical logic. The theory of formal systems (see Chapter 3) is based on these ideas, and certain results that are essential in the predicate calculus and for methods for automatic theorem proving can only be expressed in terms of decidability results. A formal rigorous presentation, as would be given by Turing machines, would be too long and dense here, so we have based this book on the assumption that the reader is familiar with computer programming. Therefore, we are able to make good progress in developing the concepts and arriving at the key results. The reader who would like to study the subject more deeply should find the annotated bibliography for this chapter helpful.

Section 2 gives a review of the topics of finite, denumerable and non-denumerable sets and may be skipped by a reader who is already familiar with these ideas; we do not define 'set' or 'integers,' but take these as primitives. Here and throughout this book we use the notation:

$\mathbb{N}$ is the set of non-negative integers 0, 1, 2, ...;
$\mathbb{Z}$ is the set of integers, positive, negative and zero ... $-2, -1, 0, 1, 2, \ldots$;
$\mathbb{Q}$ is the set of rational numbers a/b where a, b are integers and $b \neq 0$;
$\mathbb{R}$ is the set of real numbers, which can be thought of as the set of infinite decimals (after equating e.g. 1.000 ... and 0.999 ...).

2. Finite, denumerable and non-denumerable sets

If A, B are two sets we define a *bijection*, or bijective mapping, between A and B as a mapping $f: A \rightarrow B$ which is such that:

(a) distinct elements of A are mapped on to distinct elements of B (*injection*);
(b) each element of B is the image of an element of A (*surjection*);

These conditions are expressed formally:

(a) $\forall\, x,y \in A(x \neq y \Rightarrow f(x) \neq f(y))$
(b) $\forall\, y \in B\ \exists\, x \in A(f(x) = y)$

For example, the relations:

$f(0) = a, f(1) = b, f(2) = c, \ldots, f(25) = z$

define a bijection between the sets $\{0, 1, 2, \ldots, 25\}$ and $\{a, b, c, \ldots, z\}$.

If there is a bijection between A and B we say that the two sets are *equipotent*, or that they have the same *cardinality*; intuitively, this means that they have 'the same number of elements.'

Sets that are equipotent to a set of the form $\{0, 1, 2, \ldots, n - 1\}$, where $n \in \mathbb{N} \geqslant 1$ are said to be *finite*; sets that are not finite are *infinite*. Infinite sets that are equipotent to $\mathbb{N}$ are said to be *countable* or *denumerable*; sets not equipotent to $\mathbb{N}$ are *uncountable* or *non-denumerable*. If a set A is denumerable (and we shall take it as understood that denumerable implies infinite), any bijective mapping of A on to $\mathbb{N}$ is called an *enumeration* of A.

The set of even integers is denumerable because it is enumerated by the mapping $a: n \rightarrow 2n$. This is a functional form of notation; frequently instead we use a form such as $a_0 = 0, a_1 = 2, \ldots, a_n = 2n, \ldots$.

The set of primes is denumerable. A possible enumeration is:

$p_0 = 2$
$p_n =$ 'the smallest integer not divisible by $p_0, p_1, \ldots, p_{n-1}$'

A definition of this form is said to be 'by recurrence.'

Proposition 1: any infinite subset of a denumerable set is itself denumerable
Proof

Let $A = \{a_0, a_1, \ldots, a_n, \ldots\}$ so that a is an enumeration of A.

Let $B \subset A$ be an infinite subset of A. Given any set S of integers we can define the minimum element as that element m such that $\forall\, n \in S: m \leq n$. With this, we can define an enumeration of B as follows.

$b_0 = a_{i_0}$ where $i_0 = \min\,[i \in \mathbb{N} \mid a_i \in B]$
$b_j = a_{i_j}$ where $i_j = \min\,[i \in \mathbb{N} \mid i > i_{j-1}, a_i \in B]$

(which is the same procedure as that used for enumerating the primes).

Thus B is denumerable. An open triangle denotes the end of a proof or proposition.

$\triangle$

Proposition 2: if B is a denumerable subset of a non-denumerable set A, then there are elements of A that are not in B

Proof

For if every element of A were also in B, any enumeration of B (which is possible, B being denumerable) would be an enumeration of A also, contradicting the statement that A is non-denumerable.

$\triangle$

It may seem counter-intuitive that there are such things as non-denumerable sets because this implies that there are infinities of essentially different kinds; and even, from Proposition 2, that some infinities are 'more infinite' than others. We therefore give three proofs of the following proposition.

Proposition 3: there are infinite sets that are non-denumerable

Proof 1

Let $\mathbb{R}^+$ denote the set of positive real numbers, and suppose that it is denumerable; let $r_0, r_1, \ldots, r_n, \ldots$ be an enumeration.

Any positive real number r_i can be expressed as an infinite decimal:

$$r_i = r_{i0} \cdot r_{i1}, r_{i2}, \ldots, r_{im}, \ldots \text{ where } r_{i1} \in \mathbb{N},\ r_{im} \in \{0, 1, 2, \ldots, 9\}$$

This means that:

$$r_i = r_{i0} + r_{i1} \cdot 10^{-1} + r_{i2} \cdot 10^{-2} + \ldots + r_{im} \cdot 10^{-m} + \ldots$$

If we impose the condition that the sequence does not terminate with an infinity of 9s, this representation is unique; and every such sequence defines a unique real number.

We can now list the set of real numbers in the order given by the enumeration, as follows:

$$\begin{array}{l} r_0 = r_{00} \cdot r_{01}, r_{02}, \ldots, r_{0m}, \ldots \\ r_1 = r_{10} \cdot r_{11}, r_{12}, \ldots, r_{1m}, \ldots \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ r_n = r_{n0} \cdot r_{n1}, r_{n2}, \ldots, r_{nm}, \ldots \\ \ldots \quad \ldots \quad \ldots \quad \ldots \end{array}$$

and the assumption of denumerability means that every real number appears somewhere in this list.

Consider now the real number s_i defined by the sequence:

$$\begin{aligned} s_i &= 0 \text{ if } r_{ii} \text{ is odd} \\ &= 1 \text{ if } r_{ii} \text{ is even} \end{aligned}$$

which does not end in an infinity of nines.
Clearly:

$s \neq r_0$ because $s_0 \neq r_{00}$
$s \neq r_1$ because $s_1 \neq r_{11}$

and so on, for every number in the list: so s does not appear in the list, which contradicts the assumption that $\mathbb{R}^+$ is denumerable.

This is Cantor's 'diagonal proof' of the non-denumerability of the set of real numbers.

△

Proof 2

Here we prove that the set $\mathscr{P}(S)$ of all subsets of a set S is never equipotent with S, from which it follows that $\mathscr{P}(\mathbb{N})$ is non-denumerable.

Suppose that for a given set S, $\mathscr{P}(S)$ is equipotent to S, so that there is a bijection $f: S \rightarrow \mathscr{P}(S)$. This means that to any element x of S there corresponds a single subset of S; and this subset may or may not contain x.

Consider now the set A defined by:

$A = \{x \in S \mid x \notin f(x)\}$

i.e. the set of all those elements x of S such that x is not contained in the subset corresponding to x. A is a subset of S and therefore, by the assumption of bijectivity, there is a unique element a of S corresponding to A, i.e. $f(a) = A$. Either $a \in f(a)$ or $a \notin f(a)$.

If $a \in f(a)$, then $a \notin A$ by definition of A; therefore $a \notin f(a)$.
If $a \notin f(a)$, then $a \in A$ and therefore $a \in f(a)$.

In either case we have a contradiction, so the assumption that $\mathscr{P}(S)$ and S are equipotent must be false.

△

Proof 3

We prove that if S has more than two elements, the set of mappings of S on to S, denoted by F(S, S), is never equipotent with S; from which it follows that $F(\mathbb{N}, \mathbb{N})$ is not denumerable.

Suppose the contrary, so that there is a bijection $E \rightarrow F(S, S)$ given by a function $f: x \rightarrow f_x$.

Choose and fix two elements of S, s_1 and s_2, such that $s_1 \neq s_2$. Consider the set:

$B = \{x \mid f_x(x) = s_1\}$

Now define g by:

$$\begin{aligned} g(x) &= s_2 \text{ if } x \in B \\ &= s_1 \text{ if } x \notin B \end{aligned}$$

Let s_3 be the element of S such that:

$f_{s_3} = g$

If $s_3 \in B$, then $f_{s_3}(s_3) = s_1$; but $f_{s_3} = g$ and therefore:
$f_{s_3}(s_3) = g(s_3) = s_2$, from the definition of g
If $s_3 \notin B$, then $f_{s_3}(s_3) \neq s_1$ and therefore, again because $f_{s_3} = g$:
$f_{s_3}(s_3) = g(s_3) = s_1$

In either case we arrive at a contradiction, so the assumption that S and F(S, S) are equipotent must be false. Δ

Note that it follows from Proof 2 that:

1. $\mathscr{P}(\mathbb{N})$ is non-denumerable, and therefore has 'more elements' than $\mathbb{N}$.
2. $\mathscr{P}(\mathscr{P}(\mathbb{N}))$, the set of all subsets of $\mathscr{P}(\mathbb{N})$, cannot be put in bijective relation with $\mathscr{P}(\mathbb{N})$ and therefore has 'more elements' than the latter. Loosely, we may say that $\mathscr{P}(\mathbb{N})$ is 'more infinite' than $\mathbb{N}$ and $\mathscr{P}(\mathscr{P}(\mathbb{N}))$ is 'more infinite' than $\mathscr{P}(\mathbb{N})$ and so on.

Thus we have an infinity of different kinds of infinite sets.

3. Programs with i integer inputs and j integer outputs

We assume that we have available a machine with unlimited memory capacity and a programming language such as Basic, Pascal, Fortran, etc. We assume also that with the machine and the language we can use integers of whatever size we wish, and that the machine is perfectly reliable.

By a program with a single integer input we mean any program whose first executable statement is an instruction of the form:

```
input integer n
```

and for which there is no other input command and no possibility of a return to this first instruction—not GO TO that instruction.

By a program with a single integer output we mean any program that ends:

```
print k
stop
```

which has no other output instruction and no other stop instruction.

We define programs with several integer inputs and outputs analogously, and denote by $P_{(i,j)}$ the set of programs with i integer inputs and j integer outputs.

Proposition 4: for all i, j $\in \mathbb{N}$ $P_{(i,j)}$ is denumerable

Let $S = \{a_1, a_2, \ldots, a_n\}$ be the set of symbols used in the programming language (necessarily a finite set). Every program written in this language is a finite sequence of these elements, so the set of all correct (i.e. free of syntax errors) programs is a subset of the set of all finite sequences of the elements of S. We shall sometimes use the term 'word' or 'finite word' for such a sequence, writing it either $(x_1, x_2, \ldots, x_m)$ or $x_1x_2 \ldots x_m$.

We give two proofs. The second is more direct.

Proof 1

It can be shown (and is given as an exercise at the end of this chapter) that the set of all finite sequences of S (written S*) is denumerable. $P_{(i,j)}$ is a subset of S* and is therefore denumerable, by Proposition 1.

△

Proof 2

The proposition is proved if we can exhibit an enumeration of the set of programs: a list in which every program appears and in which different programs are in different positions. The following is an explicit method for constructing such a list.

- Consider first the elements of S* of length 1 (i.e. consisting of 1 symbol). These are finite in number and some of them may be members of $P_{(i,j)}$; number these $P_0, P_1, \ldots, P_{n_1}$, using any suitable rule, for example alphabetical order.
- Consider the elements of S* of length k; these are finite in number and some of them may be members of $P_{(i,j)}$; number these $P_{n_{k-1}+1}, \ldots, P_{n_k}$ using the same rule as before.

Continuing this process gives the required listing and so proves the assertion.

△

We can now assume the existence, once and for all, of a definitive enumeration of the programs $(P_{(1,1)}$; this could be the one given by the above method, but others are possible. Let this listing be:

$$P_{(1,1)} = \{P_0, P_1, \ldots, P_n, \ldots\}$$

If for the input $k \in \mathbb{N}$ the program $P_n \in P_{(1,1)}$ stops after a finite time and prints the integer r, we write:

$$P_n(k) = r$$

and if for the same input it never stops we write:

$$P_n(k) = \perp$$

with analogous notation for programs of i inputs and j outputs.

Proposition 5 (existence of a universal program): there is a program $Q \in P_{(2,1)}$ with

$$Q(j,k) = r \text{ if } P_j(k) = r,$$
$$Q(j,k) = \perp \text{ if } P_j(k) = \perp.$$

Proof

An informal proof is as follows, in which it is assumed that the programming language available is sufficiently powerful to admit the required operations.

We write a program to:

(a) construct P_j from the integer j;
(b) execute P_j with input k.

More precisely, Q has the following structure:

```
input j
input k
[construct $P_j$]
[compute $P_j(k)$]
print $P_j(k)$
stop
```

If $p_j(k) = \perp$, Q with inputs j, k will loop in the part [compute $P_j(k)$], so:

$Q(j, k) = \perp$

Otherwise it will print the integer value $P_j(k)$ after a finite time and stop, so in this case:

$Q(j, k) = r$

△

As we said, for this to be possible the programming language must have a certain minimum power; for example, it must allow additions, multiplications, for loops, tests. Without going into detail we can say that languages such as Basic, Pascal and Fortran have this power.

Q may be called a 'universal' program because it mimics every program of $P_{(1,1)}$.

Proposition 6: there is no program $R \in P_{(2,1)}$ with

$R(j, k) = 0$ if $P_j(k) = \perp$
$R(j, k) = 1$ otherwise, i.e. if there is an $r \in \mathbb{N}$ such that $P_j(k) = r$

This definition of R may seem very similar to that for the Q in Proposition 5; however, the aim of R, if it existed, would be to detect whether or not the program P_j stopped after a finite time, for the input k. The proposition is that no such program can be written, however powerful the language may be and however idealized the machine on which it is run: recall that we have assumed that the machine has unlimited memory capacity and can handle integers of any desired size. This means that there are procedures which, though perfectly well defined and relatively simple to express, cannot be programmed: a result, although rather differently stated, was first established by Turing in 1936.

Proof

The proof is again by *reductio ad absurdum*. If R exists we can, with its aid, construct a program $S \in P_{(1,1)}$ with the properties:

$S(j) = 0$ if $P_j(j) = \perp$
$S(j) = 1$ otherwise

This is done as follows:

```
input the integer j
[execute R with inputs (j, j) and assign the result R(j, j) to a variable v]
print v
stop
```

Given S we can now construct a program $T \in P_{(1,1)}$ with the properties

$T(j) = 0$ if $P_j(j) = \perp$
$T(j) = \perp$ otherwise

as follows:

```
input integer j
[execute S with input j and assign the result S(j) to a variable w
# if w = 1 go to #
print w
stop
```

T is a program belonging to the set $P_{(1,1)}$ which is denumerable, so there is an integer h such that $T = P_h$; consider how this behaves.

If $P_h(h) = \perp$, then by definition of T, $T(h) = 0$; but $T = P_h$, so if $T_h(h) = 0$, $P_h(h) \neq \perp$, which is a contradiction.

If $P_h(h) \neq \perp$, then $T(h) = \perp$; but then $P_h(h) = \perp$, so again we have a contradiction.

It follows that no program such as R can exist. △

4. Recursive functions, Church's Thesis

First we recall some definitions concerning functions. A partial function $f:A \rightarrow B$ defines a relation between the elements a of A and the elements b of B, mapping the whole or some part of A on to the whole or some part of B. A is called the *domain* of f and B the *co-domain*. If f is defined over the entire region of A, it is called a *total function* (over A). The region of B over which the result of the mapping extends is called the *range* of f; it may or may not be the complete co-domain B.

Now let f: $\mathbb{N} \to \mathbb{N}$ be a partial function over $\mathbb{N}$. We say that f is *recursive* or *computable* if there is a program $P \in P_{(1,1)}$ such that:

$$\begin{aligned} P(n) &= \perp \text{ if } n \notin \text{domain } f \\ &= f(n) \text{ if } n \in \text{domain } f \end{aligned}$$

In other words, a recursive function is a function that can be programmed.

Proposition 7: there are functions f: $\mathbb{N} \to \mathbb{N}$ that are not recursive

Proof

By Proposition 3 the set F($\mathbb{N}$, $\mathbb{N}$) of mappings (i.e. total functions) of $\mathbb{N}$ on to $\mathbb{N}$ is not denumerable. The set of all partial functions $F\mathscr{P}(\mathbb{N}, \mathbb{N})$, is non-denumerable, for otherwise F, being a subset of $F\mathscr{P}$, would be denumerable by Proposition 1. Let Φ be the mapping that associates each recursive function f with the first-occurring program in the enumeration of $P_{(1,1)}$ that computes f. This mapping is a bijection between the set of recursive functions and an infinite subset S of $P_{(1,1)}$, and since $P_{(1,1)}$ is denumerable, by Proposition 4, S is also denumerable (Proposition 1): that is, the set of recursive functions is denumerable.

Thus we have the result that whilst the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is non-denumerable, the set of recursive functions is denumerable; it follows that there must be functions from $\mathbb{N}$ to $\mathbb{N}$ that are not recursive.

Δ

A non-denumerable set has many more elements than a denumerable set: for example, even a denumerable infinity of disjoint denumerable sets does not constitute a non-denumerable set. What the above proof shows is that recursive functions are of very rare occurrence in the set of all functions $\mathbb{N} \to \mathbb{N}$; however, all the functions that one is generally aware of, such as $n \to 2n$, $n \to n!$, $n \to$ 'nth prime' are recursive, as is obvious because programs are easily written that will compute these. Thus Proposition 7 seems paradoxical: it tells us that the great majority of functions are not recursive, yet at first sight it seems impossible for us to imagine a function that is not recursive.

Proposition 8 will resolve this apparent paradox by giving an explicit construction for a total function that is not recursive; from which we can derive an unlimited number of other non-recursive functions.

Proposition 8: there are total functions from $\mathbb{N}$ to $\mathbb{N}$ that are non-recursive

Proof

Define f: $\mathbb{N} \rightarrow \mathbb{N}$ as follows:

$$\begin{aligned} f(n) &= 1 \text{ if } P_n(k) \text{ is defined for all } k \\ &= 0 \text{ otherwise} \end{aligned}$$

Suppose f is recursive, in which case it is computable by a program P. We can then construct a program Q such that:

$$\begin{aligned} Q(n) &= P_n(n) + 1 \quad &\text{if } f(n) = 1 \\ &= 1 &\text{if } f(n) = 0 \end{aligned}$$

Thus Q(n) is always defined, and since $Q \in P_{(1,1)}$ there will be a value of k such that $Q = P_k$. So by definition of Q:

$$P_k(k) = Q(k) = P_k(k) + 1$$

and the contradiction proves that f is not recursive. △

We have seen that recursive functions, which are the functions that can be computed by means of a program, are not the only functions $\mathbb{N} \rightarrow \mathbb{N}$; and further, that we can give simple and unambiguous definitions of functions over the whole of $\mathbb{N}$ that prove to be non-recursive. What this means is that the power of programming languages is limited, meaning of course the theoretical power; every actual language has its own particular strengths which make it particularly valuable for programming this or that type of problem. What is remarkable is that the same limitation applies to every language. Provided that a language has sufficient basic resources it can compute any recursive function; but no matter what is added above these resources it cannot compute any but recursive functions (in fact, its power cannot exceed that of elementary devices called Turing machines, see Bibliography). All attempts made so far to extend the power of these languages have failed, and detailed study has always shown that the only functions computable by a proposed language are the recursive functions.

All those mathematicians, logicians and computer scientists who have studied these questions are agreed that:

> the only functions from $\mathbb{N}$ to $\mathbb{N}$ that are computable by means of algorithms are the recursive functions.

This statement is known as Church's Thesis. It cannot be proved (although its converse is fairly obvious) because the term 'computable by means of algorithms' cannot be defined formally. However, no extension so far envisaged of the concept of an algorithm has succeeded in computing other than recursive functions, so it is reasonable to accept Church's Thesis as 'experimentally established': in fact, the mathematical concept of a recursive function and the 'physical-philosophical' concept of a function computable by an algorithm coincide exactly.

5. Recursive and recursively enumerable sets

Let $A \subset \mathbb{N}$. A is said to be *recursive* iff:

(α) the function $f\colon \mathbb{N} \to \mathbb{N}$ defined by

$$f(n) = 1 \quad \text{if } n \in A$$
$$f(n) = 0 \quad \text{if } n \notin A$$

is recursive or alternatively

(α′) there is a program with a single input that, for every input $n \in N$ prints, after a finite time:

YES if $n \in A$
NO if $n \notin A$

A is said to be *recursively enumerable* (written r.e.) iff:

(β) there is a recursive function f such that:

$$A = \text{domain } f$$

or alternatively

(β′) there is a program with no inputs that prints the elements of A in succession.

The equivalence of (α) and (α′) and (β) and (β′) follows immediately from the definition of recursive functions in terms of programs; proof of the second involves a principle analogous to that used in the proof of Proposition 11(b) below.

Proposition 9: a recursive set is also recursively enumerable

Proof

Let P be a single-input program that for every input $n \in \mathbb{N}$ prints YES if $n \in A$ and NO if $n \notin A$. This program can be modified so that instead of YES it prints 1, or any other integer or symbol, and instead of printing NO it goes into a loop. The recursive function f associated with this modified program will verify that $A = \text{domain } f$; so A is r.e.

To be explicit, the modification of P to give the required program Q is as follows:

```
  input integer n
  [execute P with input n and assign the result to a variable v]
# if v = NO go to #
  print 1
  stop
```

△

Proposition 10: if A ⊂ ℕ and its complement ℕ − A are both recursively enumerable, then A is recursive

Proof

Let P_A be the program that prints the elements of A and $P_{\mathbb{N}-A}$ the program that prints the elements of ℕ − A. Let $P_A[m]$, $P_{\mathbb{N}-A}[m]$ be the mth integers printed by P_A, $P_{\mathbb{N}-A}$, respectively. Consider the following program:

```
input integer n
BOOL := TRUE
m := 0
While BOOL = TRUE do
    if P_A[m] = n then do
      s := 1
      BOOL := FALSE
      end if
    if P_{N-A}[m] = n then do
      s := 0
      BOOL := FALSE
      end if
    m := m + 1
    end while
print s
stop
```

This program can of course be written in whatever suitable language is available. Its action is to search simultaneously the lists of integers produced by P_A and $P_{\mathbb{N}-A}$ respectively until it finds one equal to the input n in either list. It then prints 1 if $n \in A$ or 0 if $n \notin A$. It will complete this action in a finite time because n, an integer, must appear at some point in one or other of the two lists.

It follows from the definition (α) that the recursive function associated with this program establishes that A is recursive.

△

Proposition 11: (a) Recursive sets exist; (b) Recursively enumerable sets exist that are not recursive; (c) Non-recursively enumerable sets exist

Proof

(a) $\mathbb{N}$ is obviously recursive, as also are the set of even integers and the set of primes.

(b) The set $A = \{m \mid P_m(m) \neq \perp\}$, where P_m is a program of the set $P_{(1,1)}$ as defined in Proposition 4, is recursively enumerable but not recursive.

Consider the following program:

```
  n := 1
# for m := 0 to n do
      if (P_m with input m produces a result in less than n units of time)
        and (m has not been printed already) then
        print m
      end if
  n := n + 1
  go to #
```

For all $m \in A$ there is an integer n such that 'P_m with input m produces a result in less than n units of time,' and therefore the above program can print any $m \in A$. It will print only those m that are elements of A. Thus A is recursively enumerable.

If A is recursive the function f defined by:

$$f(n) = 1 \text{ if } P_n(n) \neq \perp$$
$$f(n) = 0 \text{ if } P_n(n) = \perp$$

would be recursive and a program Q could be constructed such that:

$$Q(n) = P_n(n) + 1 \quad \text{if } f(n) = 1$$
$$Q(n) = 1 \qquad\qquad \text{if } f(n) = 0$$

which, by Proposition 8, is impossible. Therefore A, which is recursively enumerable, is not recursive.

(c) If A is recursively enumerable but not recursive (as has just been shown to be possible) then its complement $\mathbb{N} - A$ is neither recursive nor recursively enumerable.

- If $\mathbb{N} - A$ were recursively enumerable then, by Proposition 10, A would be recursive, which is not so.
- If $\mathbb{N} - A$ were recursive it would also be recursively enumerable, by Proposition 9; and this also is not so.

△

Let S be the finite set $S = \{s_1, s_2, \ldots, s_n\}$ where the elements s_i are the symbols of the programming language that we are using, arranged in some order. Let S* be the set of all finite sequences of these elements.

Consider the following enumeration of the elements of S*, s set in italic denotes sequence.

$s_0 = (\quad)$ the empty sequence
$s_1 = (s_1), s_2 = (s_2), \ldots, s_n = (s_n)$
$s_{n+1} = (s_1, s_1), s_{n+2} = (s_1, s_2), \ldots, s_{n+n^2} = (s_n, s_n)$
....
$s_{n(p-1)} = \underbrace{(s_1, s_1, \ldots, s_1)}_{\text{(p times)}}, \ldots, s_{n(p)} = \underbrace{(s_n, s_n, \ldots, s_n)}_{\text{(p times)}}$

where $n(p-1) = n + n^2 + \ldots + n^{p-1} + 1$, $n(p) = n + n^2 + \ldots + n^p$.

This is an enumeration that could be programmed, meaning that a program could be written such that, given any integer input m, it would print s_m. The following definitions are independent of the details of the enumeration, requiring only that it be programmable.

Let $A \subset S^*$. A is said to be recursive iff:

(γ) the set $\{m \mid s_m \in A\}$ is recursive
or alternatively
(γ') there is a program that, for any input $s \in S^*$, prints after a finite time
YES if $s \in A$
NO if $s \notin A$

A is said to be recursively enumerable iff:

(δ) the set $\{m \mid s_m \in A\}$ is recursively enumerable
or alternatively
(δ') there is a zero-input program that prints the elements of A in succession.

The equivalence of (γ) and (γ') and of (δ) and (δ') follows from the definition of recursive functions, together with the fact that the enumeration of S* is programmable.

Just as we have defined the concepts of recursive and recursively enumerable subsets of S*, where S is a finite set, we can give corresponding definitions for the subsets of $\mathbb{N}$, that is, for sets of finite sequences of integers; and also for the subsets of S^k where S is any finite set and k an integer; for those of $\mathbb{N}^k$ and more generally for subsets of X* and for X^k provided the concepts have been defined for subsets of X. Propositions 9, 10 and 11 remain true in all these cases (X being any set).

Note
S denotes set
$A_i (i \in I)$ is a *partition* of S

$$\leftrightarrow \begin{bmatrix} \bigcup_{i \in I} A_i = S, \forall i,j \in I \; i \neq j \; A_i \cap A_j = \Phi \\ \forall i \in I \quad A_i \neq \Phi \end{bmatrix}$$

Example

$S = \{a, b, c, d, e, f\}$
$A_1 = \{a, c, e\}$
$A_2 = \{b, d\}$ $(A_i)_{i\in\{1,2,3\}}$ is a partition of S
$A_3 = \{f\}$

$\triangle$

6. Decidable and semi-decidable predicates

An expression containing variables, as for example:

$n \geqslant m$, $\exists\, p: p^2 = q$, 'the sequence *s* begins 0101'

is called a predicate, and the above may be written as predicates;

$P(n, m)$, $R(q)$, $S(s)$

For any given values of n and m P(n, m) is either true or false, e.g. P(3,2) is true but P(2,3) is false. Similarly, if p, q are integers R(q) may be true or false, e.g. R(4) is true but R(5) is false; and correspondingly for S.

Let $T(x_1, x_2, \ldots, x_n)$ be a general predicate with n variables $x_1, x_2, \ldots, x_n$ having values in a set X for which the concepts of recursive and recursively enumerable subsets have been defined. Possibilities for X are $\mathbb{N}$, S^* with S finite, $\mathbb{N}^*$, $\mathbb{N}^k$. T is said to be *decidable* (or recursive) iff:

(θ) $\{x_1, x_2, \ldots, x_n \mid T(x_1, x_2, \ldots, x_n)$ is true$\}$ is recursive
or equivalently
(θ') there is a program P that, for any input $(x_1, x_2, \ldots, x_n)$ in X prints, after a finite time:
YES if $T(x_1, x_2, \ldots, x_n)$ is true
NO if $T(x_1, x_2, \ldots, x_n)$ is false

T is said to be *semi-decidable* iff:

(μ) $\{x_1, x_2, \ldots, x_n \mid T(x_1, x_2, \ldots, x_n)$ is true$\}$ is recursively enumerable
or equivalently
(μ') there is a program P that, for any input $(x_1, x_2, \ldots, x_n)$ in X
if $T(x_1, x_2, \ldots, x_n)$ prints YES after a finite time
if $T(x_1, x_2, \ldots, x_n)$ is false loops indefinitely

or again

(μ'') there is a zero-input program that prints all the n-tuples $(x_1, x_2, \ldots, x_n)$ for which T is true

A decidable predicate corresponds to an infinite family of questions for which there is an algorithm that will give the correct answer to any of the questions within a finite time, although the time required will not be known in advance. For example, the predicate:

T(n, m): 'n is a divisor of m'

is decidable.

A semi-decidable predicate corresponds to such a family and an algorithm that will give a positive answer within a finite time when that is correct but will give no answer in other cases. Thus the predicate:

T(n): 'n is the serial number of a single-input program that, given the input n, will stop after a finite time'

is semi-decidable but not decidable.

Semi-decidability can be established in this case by the following algorithm, which can be put in the form of a program:

```
given n, construct the single-input program with serial number n
run the program with input n
if the program stops then print YES
```

This predicate is not decidable, for if it were then the set $\{m \mid P_m(m) \neq \perp\}$ would be recursive, which by Proposition 11 is not the case.

Corresponding to Section 5 it can be shown that:

(a) if the predicate $T(x_1, x_2, \ldots, x_n)$ is decidable, it is also semi-decidable (b) if both T and its negation not-T are semi-decidable, then T is decidable (c) predicates exist that are not semi-decidable

an example for the last being 'n is the serial number of a program that does not stop.'

The examples given here of predicates that are not decidable or not semi-decidable may seem rather artificial; however, cases arise in logic, theory of languages, arithmetic and computer science in which the question of decidability is of clear importance. We see in Chapter 5 that the predicate:

'formula F of the first-order predicate calculus is a theorem'

is semi-decidable but not decidable; one consequence of this is that there can be no completely satisfactory inference engine for languages based on first-order predicate calculus, such as PROLOG.

Further, AI has from the start to face these results concerning decidability and undecidability. Our view is that awareness of these topics is essential if one is to avoid fruitless searches for methods that can be shown not to exist.

Exercises

Denumerable Sets

1. Show that $\mathbb{N} \times \mathbb{N}$ is denumerable.
2. Show that the set of finite sequences of the elements of a finite set $S = \{s_1, s_2, \ldots, s_n\}$ is denumerable.

3. Show that the set of finite sequences of the elements of a denumerable set is denumerable.
4. Show that the set of infinite sequences of the elements of a set having two or more elements is non-denumerable.

Finite and Infinite Sets

For each of the following state whether the set is finite, denumerable or non-denumerable:

1. $\mathbb{Z}$, the set of all integers positive, negative or zero.
2. $\mathbb{Q}$, the set of rational numbers.
3. $\{(x,y) \mid x \in \mathbb{Z}, y \in \mathbb{Z}, x^2+y^2 \leq 100\}$.
4. $\{(x,y) \mid x \in \mathbb{Q}, y \in \mathbb{Q}, x^2+y^2 \leq 100\}$.
5. $\{(x,y) \mid x \in \mathbb{R}, y \in \mathbb{R}, x^2+y^2 \leq 100\}$.
6. $\{\alpha \in \mathbb{N}^* \mid \Sigma\alpha_i = 5\}$
   where $\mathbb{N}^*$ is the set of finite words (sequences) formed with the elements of $\mathbb{N}$ and $a = a_1, a_2, \ldots, a_n$.
7. {increasing functions from $\mathbb{N}$ to $\mathbb{N}$}.

Programs with i Inputs and j Outputs

1. Is there a program $P \in P_{(2,1)}$ as follows?

$$\begin{aligned} P(k,l) &= P_k(P_k(l)) \text{ if } P_k(l) \neq \perp \text{ and } P_k(P_k(l)) \neq \perp \\ &= \perp \text{ otherwise} \end{aligned}$$

2. Is there a program $P \in P_{(3,2)}$ as follows?

$$\begin{aligned} P(k,m,n) &= P_k(m) + P_k(n) + 1 && \text{if } P_k(m) \neq \perp \text{ and } P_k(n) \neq \perp \\ &= P_k(m) && \text{if } P_k(m) \neq \perp \text{ and } P_k(n) = \perp \\ &= P_k(n) && \text{if } P_k(m) = \perp \text{ and } P_k(n) \neq \perp \\ &= 0 && \text{if } P_k(m) = \perp \text{ and } P_k(n) = \perp \end{aligned}$$

Detection of Looping in Programs with Zero Inputs and One Output

With the enumeration $P_{(0,1)} = \{Q_0, Q_1, Q_2, \ldots\}$ show that there is no program $U \in P_{(1,1)}$ such that:

$U(i) = 0$ if Q_i does not stop
$U(i) = 1$ if Q_i stops after a finite time

Recursive Functions

1. Show that every decreasing total function $f: \mathbb{N} \to \mathbb{N}$ is recursive.
2. If $g: \mathbb{N} \to \mathbb{N}$ is a non-injective total function, construct a total function $h: \mathbb{N} \to \mathbb{N}$ such that h is non-recursive and $f: x \to g(h(x))$ is recursive.

Non-recursive Functions

Define a total function $f: \mathbb{N} \to \mathbb{N}$ which is:

1. increasing and non-recursive.
2. non-recursive, but $f \circ f$ is recursive.

Recursive and Non-recursive Functions

State, with brief justification, which of the following functions are recursive and which non-recursive:

1. $f(n)$ = number of programs with fewer than n symbols;
2. $f(n) = 0$ if $P(0) = n$ for an infinite number of programs $P \in P_{(1,1)}$
   $f(n) = 1$ otherwise;
3. $f(n)$ = nth digit in the decimal development of π;
4. $f(n) = 1$ for all $n \in \mathbb{N}$ if there are $p, q, r \in \mathbb{N} - \{0\}$,
   $m \in \mathbb{N} - \{0, 1, 2\}$ such that
   $p^m + q^m = r^m$
   $f(n) = 0$ otherwise;
5. $f(n) = 1$ if $P_n(k) \neq 0$ for all $k \in \mathbb{N}$
   $f(n) = 0$ otherwise;
6. $f(n) = \perp$ if $P_n(k) \neq 0$ for all $k \in \mathbb{N}$
   $f(n) = 1$ otherwise.

Recursive and Recursively Enumerable Sets

1. State, with reasons, which of the following sets are:

   recursive
   recursively denumerable but non-recursive
   not recursively enumerable

   (a) $\{2^n \mid n \in \mathbb{N}\}$;
   (b) $\{p \mid p \text{ is the product of two primes}\}$;
   (c) $\{a \in \mathbb{N}^* \mid \text{the string a contains } 0\}$;
   (d) $\{a \in \mathbb{N}^* \mid \text{the string a does not contain } 0\}$;
   (e) $\{P \in P_{(1,1)} \mid P(0) = 0\}$;

(f) $\{P \in P_{(1,1)} \mid P(0) \neq 0\}$;
(g) $\{(P, Q) \mid P, Q \in P_{(1,1)}: \forall n: P(n) = Q(n)\}$.

2. If A is recursively enumerable, show that it is recursive if and only if there is a program that enumerates the elements of A in increasing order A.
3. Give in detail proofs of the equivalences (α) and (α'), (β) and (β'), (γ) and (γ'), (δ) and (δ'), (θ) and (θ') and (μ), (μ') and (μ'').

Decidable and Semi-decidable Predicates

1. State, with reasons, which of the following predicates are:

   decidable
   semi-decidable but not decidable
   not semi-decidable

   (a) $m = n + p$;
   (b) $\exists n: m = n + p$;
   (c) n is the sum of two primes;
   (d) n is the serial number of a program in $P_{(1,1)}$ which contains an instruction ‘if ... then ...’;
   (e) n is the serial number of a program in $P_{(1,1)}$ which executes an instruction ‘if ... then ...’ for input n;
   (f) n is the serial number of a program in $P_{(1,1)}$ which executes no instruction ‘if ... then ...’ for input n.