The next three chapters take up in depth topics introduced in Enderton's chapter. Martin Davis's chapter pursues the uses of the theory of recursive functions for showing that certain classes of problems cannot be effectively decided — the word problem for groups being one of the best known. The related problem of decidable versus undecidable theories of first-order logic is discussed in Rabin's chapter.

Among undecidable problems, some are more undecidable than others. The definition of "degree" of unsolvability is introduced in Section 8 of Enderton's paper and a survey of important results on these degrees is given in Simpson's chapter.

Moving to the broader definition of recursion theory we come to Shore's chapter on the generalization of recursion theory to admissible ordinals. Shore presents a fine introduction to the basic notions and, as a case study, shows what new considerations arise when the Splitting Theorem is generalized to admissible ordinals. The chapter also contains a very useful annotated bibliography to the study of α-recursion theory.

The study of Kleene recursion in higher types (recursive functions of functions of functions, say, rather than recursive functions of natural numbers) has always been more or less inaccessible to all but the dedicated specialist — due to the difficulty of the basic papers in the subject. This situation should be remedied in the chapter by Kechris and Moschovakis, where a conceptually simple approach via inductive definability is taken.

The study of inductive definitions in general is taken up in Aczel's chapter. It should interest logicians of all persuasions since it combines the concerns of the recursion-theorist with the vantage points of the model-theorist and proof-theorist.

Martin's chapter discusses one of the major applications of recursion theory — to descriptive set theory. Here definability considerations over the continuum give rise to a beautiful theory which finds its origins in the French "constructivist" school of Borel, Baire and Lebesgue.

We had planned to have a chapter on the more "practical" aspects of recursion theory, those where running times of programs and computational complexity appear, but this chapter did not materialize. Among other chapters of the *Handbook* relevant to recursion theory we mention Statman's chapter on the equation calculus (in Part D), and Makkai's chapter on admissible sets (in Part A). The recursion-theorist might also be interested to see some proof-theoretic applications of recursion theory in Feferman's chapter in Part D.

C.1

Elements of Recursion Theory

HERBERT B. ENDERTON

Contents

HANDBOOK OF MATHEMATICAL LOGIC
Edited by J. Barwise

Introduction

This chapter presents an expository treatment of the elements of recursive function theory. It makes no claims of advancing to the frontiers of research in this field. It does attempt to indicate what background would be required of someone heading that way.

The proofs in this chapter are often merely sketched, with indication of the main ideas involved. There are several books that give more thorough treatment to these topics. The primary reference in this field is ROGERS [1967]. A more condensed treatment can be found in Chapters 6 and 7 of SHOENFIELD [1967]. Turing machines are discussed, among other places, in books by YASUHARA [1971] and by DAVIS [1958]. There is a fairly recent book on degrees of unsolvability by SHOENFIELD [1971] and an older one by SACKS [1963]. Finally, the classic book by KLEENE [1952] contains much recursion theory.

1. Informal computability

The simplest conception of recursive functions is as "effectively computable" functions. We will consider initially functions from natural numbers to natural numbers, postponing the matter of functions on other sets. Let $\mathbf{N}$ be the set $\{0, 1, 2, \ldots\}$ of natural numbers, and let $\mathbf{N}^k$ be the cartesian product $\mathbf{N} \times \mathbf{N} \times \cdots \times \mathbf{N}$ with k factors. Then the objects we want to consider will be functions f with $\operatorname{dom} f \subseteq \mathbf{N}^k$ for some positive k and $\operatorname{ran} f \subseteq \mathbf{N}$. Such an object will be called a k-place partial function. The word "partial" is a reminder that the domain is only a subset, possibly proper, of $\mathbf{N}^k$. (The partial function is said to be *total* if its domain is all of $\mathbf{N}^k$.)

It is clear from cardinality considerations that there are $2^{\aleph_0}$ k-place partial functions for each positive k. From this huge inventory we want to select the $\aleph_0$ functions that are recursive. We begin in this section with an intuitive description of the notions we seek to capture. And then in the next section we will turn to the methods for making the ideas precise.

Call a k-place partial function f *effectively computable* when there exists an effective procedure (i.e., an algorithm) that calculates f correctly. Now an effective procedure must meet the following criteria.

(i) There must be exact instructions (i.e., a program), finite in length, for the procedure. These instructions cannot demand any cleverness or even understanding on the part of the person or machine following them.

Executing the instructions must be a matter of merely following directions carefully.

(ii) If the procedure is given a k-tuple $\boldsymbol{x}$ in $\operatorname{dom} f$, then after a finite number of discrete steps the calculation must terminate and produce $f(\boldsymbol{x})$.

(iii) If the procedure is given a k-tuple $\boldsymbol{x}$ that does not belong to $\operatorname{dom} f$, then the procedure might go on forever, never halting. Or it might get stuck at some point, but it must not pretend to produce a value for f at $\boldsymbol{x}$.

One can picture an industrious and diligent clerk, well supplied with scratch paper, tirelessly following his instructions. Alternatively, one can picture an automated version, a digital computer executing a program.

Despite the fact that we have given only a suggestive description and not a mathematical definition, it is possible to develop nearly all of the theory of recursive functions on just this informal basis. (The recursive functions are the effectively computable functions, but we reserve the term "recursive" for the mathematically defined concept.) For evidence of this possibility, we refer the reader to the book ROGERS [1967].

As examples of effectively computable functions we can cite addition and multiplication of natural numbers. Effective procedures for these functions (using decimal representation) are taught in the elementary schools. Any function with a finite domain is effectively computable. The instructions for computing such a function can contain a table listing all of its values.

There are several sorts of restrictions of a practical nature that we do *not* impose on effective procedures.

(i) Although each argument given the procedure as input must be a (finite) natural number, there is no bound imposed in advance on the size of the arguments. We do not rule out arguments that exceed the number of electrons in the universe, for example.

(ii) Although the procedure must produce $f(\boldsymbol{x})$, when $\boldsymbol{x} \in \operatorname{dom} f$, after a finite number of steps, there is no bound imposed in advance on this number.

(iii) Similarly, there is no bound imposed in advance on the amount of scratch paper (memory space) the procedure might require. Even multiplication of very large numbers can require large amounts of scratch paper.

These considerations are relevant to the comparison of effective computability to "practical computability". A person with a digital computing machine may regard a function f as being computable only when $f(\boldsymbol{x})$ is computable on his machine in a reasonable length of time. Of course, the matter of what is reasonable may change from day to day. And next year he hopes to get a faster machine with more memory space and tape drives.

At that time, his idea of what is computable in a practical sense will be extended considerably.

The class of effectively computable functions is obtained in the ideal case where all of the practical restrictions on running time and memory space are removed. Thus the class is a theoretical upper bound on what can ever in any century be considered computable.

It should be clear that if f and g are functions that agree at all but finitely many arguments, then f is effectively computable iff g is also effectively computable. Thus the question whether a function is effectively computable hinges solely on the behavior of that function in neighborhoods of infinity.

2. Turing machines

There are many equivalent ways of formulating the definition of recursiveness. A version phrased in terms of imaginary computing machines was given by the English mathematician Alan Turing in a fundamental paper (TURING [1936]). (Related work was done simultaneously but independently by Emil Post in New York; see POST [1936].) Turing had the disadvantage of formulating this definition prior to the development of actual digital computers. In fact the flow of information was from the abstract to the concrete: von Neumann was familiar with Turing's work, and Turing himself later played an enthusiastic role in the development of computers.

On an informal level, we can begin by picturing a Turing machine as a black box together with a tape. The tape is marked off into squares, and each square can contain either the blank symbol 0 or the non-blank symbol 1. The tape is potentially infinite in both directions, in that we never come to the end of it, but at any time only finitely many squares can be non-blank. Initially the tape contains the input numbers, and ultimately it contains the output number. At intermediate times it serves as memory space for the calculation.

If we open up the black box, we find that it is a very simple device. It is capable of examining only one square of the tape at a time. The device contains a finite list of instructions (or *states*) $q_0, q_1, \ldots, q_n$. Each instruction can indicate two possible courses of action, one to be followed if the tape square under scrutiny contains a 0, the other to be followed if it contains a 1. In either event, a course of action can only consist of the following three steps:

(i) A symbol (possibly the same as the old symbol) is written on the tape square being scanned, thereby erasing the previous symbol.

(ii) The tape is moved one square right or left.

(iii) The next instruction is specified.

Thus the list of instructions determines a transition function that, given the number of the present instruction and the symbol being scanned, produces the three-part course of action. We can formalize these ideas by taking the Turing machine simply to be this transition function.

2.1. DEFINITION. A *Turing machine* is a function M such that for some natural number n,

$$\operatorname{dom} M \subseteq \{0, 1, \ldots, n\} \times \{0, 1\},$$

$$\operatorname{ran} M \subseteq \{0, 1\} \times \{L, R\} \times \{0, 1, \ldots, n\}.$$

For example we might have $M(3, 1) = (0, L, 2)$. The intended meaning of this is that whenever the machine comes to instruction q_3 while scanning a square in which 1 is written, it is to erase the 1 (leaving a 0 in the square), move the tape so as to examine the square just to the left of the present square, and proceed next to instruction q_2. If $M(3, 1)$ is undefined, then whenever the machine comes to instruction q_3 while scanning a square in which 1 is written, it halts. (This is the only way of stopping a calculation.)

This intended interpretation is not embodied in the formal definition of a Turing machine. But it does motivate and guide the formulation of all subsequent definitions. In particular, we can define what it means for a machine M to move (in one step) from one configuration to another. We do not need to present the formal definitions here, since they are only translations of our informal ideas.

The input/output format consists of strings of 1's, separated by 0's. Let $\ulcorner x \urcorner$ be a string of 1's of length $x + 1$. Thus

$$\ulcorner x_1 \urcorner 0 \ulcorner x_2 \urcorner 0 \cdots 0 \ulcorner x_k \urcorner$$

is the result of combining k strings of 1's, each separated from the next by a 0.

At last we can define recursiveness. A k-place partial function f is said to be *recursive* if there exists a Turing machine M such that whenever we start M at instruction q_0 scanning the leftmost symbol of

$$\ulcorner x_1 \urcorner 0 \ulcorner x_2 \urcorner 0 \cdots 0 \ulcorner x_k \urcorner$$

(with the rest of the tape blank), then:

(i) If $f(x_1, \ldots, x_k)$ is defined, then M eventually halts scanning the leftmost symbol of

$$\ulcorner f(x_1, \ldots, x_k) \urcorner$$

and with the tape blank to the right of this string.

(ii) If $f(x_1, \ldots, x_k)$ is undefined, then M never halts.

If R is a k-ary relation on the natural numbers, then R is said to be *recursive* if its characteristic function $\chi_R : \mathbb{N}^k \to \{0, 1\}$ is recursive. (Caution: If a k-place partial function is recursive, then it does not follow that its graph is a recursive $(k+1)$-ary relation.)

For example the identity function $f(x) = x$ is recursive, being computed by the empty machine. A less trivial case is addition $x + y$, which is computed by the machine whose values are listed in Table 1. The comments to the right are to help the reader, not the machine. (Turing defined M to be a set of quintuples instead of a function from pairs to triples. The table, being the graph of M, is essentially a set of quintuples.)

Table 1

0 1	1 *R* 0	pass over *x*
0 0	1 *R* 1	fill gap
1 1	1 *R* 1	pass over y
1 0	0 *L* 2	end of y
2 1	0 *L* 3	erase a 1
3 1	0 *L* 4	erase another 1
4 1	1 *L* 4	back up
4 0	0 *R* 5	halt

It is an exercise in programming to produce Turing machines for multiplication and exponentiation.

We should remark that many of the details of our definition of a Turing machine are somewhat arbitrary. If there were more than one tape, the class of computable functions would remain unchanged (although some functions could be computed more rapidly). Similarly we could allow more than the symbols 0 and 1. Or we could have the tape extend in only one direction from a starting point, instead of both directions. None of this affects the class of computable functions. What *is* essential in the definition is the provision for arbitrarily large amounts "scratch-pad" storage space and arbitrarily long calculations.

We can give a specific example of a non-recursive function by using the "busy beaver competition" of RADO [1962]. An n-state entry in this competition is a Turing machine M with $n+1$ instructions $q_0, \ldots, q_n$, the last of which is used only for halting (both $M(n, 0)$ and $M(n, 1)$ are undefined), and such that when started on a blank tape, M eventually halts. When M does halt, its score in the competition is the number of 1's on the tape. Thus the machine tries to write as many 1's on the tape as it possibly can, but it must halt. Let $\Sigma(n)$ be the maximum possible score for an n-state entry.

2.2. THEOREM (RADO [1962]). *The function Σ is not recursive. In fact for any total recursive f on $\mathbb{N}$, we have $f(x) < \Sigma(x)$ for all sufficiently large x.*

PROOF. The function whose value at x is

$$\max[f(2x+2), f(2x+3)]$$

is recursive, and hence is computed by some machine M having, say, k instructions. For each x, consider a machine N_x that writes $\ulcorner x \urcorner$ on a blank tape and then behaves like M. Then N_x is a $(x+k+2)$-state entry in the busy beaver competition. So its score (the number displayed above plus 1) is bounded by $\Sigma(x+k+2)$, which for all $x \geq k$ is bounded by $\Sigma(2x+2)$. □

We will construct other non-recursive functions later, but the Σ function has a striking simplicity. The first few values of Σ are known: $\Sigma(1) = 1$, $\Sigma(2) = 4$, and $\Sigma(3) = 6$ (LIN and RADO [1965]). Next $\Sigma(4) = 13$ (BRADY [1966, 1975]). Beyond this point, only lower bounds are known. $\Sigma(5) \geq 17$ $\Sigma(6) \geq 35$, $\Sigma(7) \geq 22961$, and $\Sigma(8) > 8 \times 10^{44}$ (GREEN [1964]).

3. Church's thesis

In Section 1 we discussed an informal concept of computability. In Section 2 we defined the mathematical concept of recursiveness. Do these two match? That is, is the concept of recursiveness the correct formalization of our intuitive concept of effective computability? The claim that it is indeed correct is known as Church's thesis. This claim was advanced and defended by CHURCH [1935, 1936], and has been almost universally accepted.

There are two arguments supporting the view that the class of recursive functions is broad enough to contain all effectively computable functions.

The first has to do with specific procedures: the procedures that have been felt to be effective have, when examined, been found to be executable by Turing machines. The second argument has to do with the class of effective procedures as a whole: the several attempts that were made to formalize the concept of computability have all yielded concepts equivalent to recursiveness. In particular, the natural ways of liberalizing the definition of recursiveness (such as allowing several tapes) in the end yield notions equivalent to recursiveness. (The proofs of these results are, for the most part, not difficult once the techniques of the following section are known).

Historically, the first appearance of a definition of recursiveness was in Kurt Gödel's original paper (GÖDEL [1931]) on the incompleteness of formal systems. He defined a relation to be *entscheidungsdefinit* if it was binumerable in a certain formal system of number theory. (The concept of binumeration may be found in Chapter D.1.) This is equivalent to our definition of recursive relation. But Gödel was not at this time attempting to formalize the concept of effective decidability or computability, and attention was not focussed on this definition. In the same paper he defined a class of functions called "recursive" (*rekursiv*); this is now called the class of primitive recursive functions. The name "recursive" was appropriate, since the central feature of the definition was a provision for finding $f(n+1)$ from $f(n)$.

Gödel visited Princeton several times in the 1930's, before moving there permanently in 1940. In 1934, during one of these visits, he gave a talk for which mimeographed notes were circulated. The notes were taken by Kleene and Rosser, who about this time completed dissertations at Princeton under Church. (The notes were eventually published as GÖDEL [1965].) In this talk he raised the issue of effective computability. He noted that more general forms of recursion would have to be admitted before his previous recursive functions could include all computable functions. He then defined a class he called "general recursive functions", using ideas that had been suggested to him in a letter from Herbrand. (This definition, which involved formal rules for deriving equations from others, is also equivalent to our definition of recursiveness.)

Church had been at Princeton since 1929 and together with his student Kleene had developed the concept of λ-definable functions. The question of the relationship between λ-definability and effective computability was studied by Church. CHURCH [1936] not only contained the proposal now bearing his name, but also provided the first example of an unsolvable decision problem. This is the problem whether a formula in the λ-calculus has a normal form, which can be regarded as a decision problem in

elementary number theory. The proof of the equivalence of the concept of λ-definable functions and the concept of general recursive functions was due primarily to KLEENE [1936b].

TURING [1936] referred to Church's paper, and presented yet another definition of recursiveness (essentially the definition of Section 2). Turing had independently had the idea of formalizing the concept of effective computability, but was led to publish only when Church's paper appeared. In an appendix, Turing proved the equivalence of his definition to λ-definability.

POST [1936] described Church's thesis as being not a definition or an axiom but a natural law, a "fundamental discovery" concerning "the mathematicizing power of Homo Sapiens", in need of "continual verification".

Until now we have dealt with functions as the basic objects of study; we have made scant reference to k-ary relations on $\mathbf{N}$. Actually recursion theory can be developed in terms of either functions or relations, and with interchangeable results. We can, informally, call a relation R *decidable* if there is an effective procedure that, given any x, replies "yes" if $x \in R$ and replies "no" if $x \notin R$. (In discussing relations, we will write $x \in R$ and $R(x)$ synonymously.) Then R is decidable iff the characteristic function of R is effectively computable. Thus a consequence of Church's thesis (equivalent in fact to the original form) is that the concept of a recursive relation is the correct formalization of the informal concept of a decidable relation.

4. Universal machines and normal form

The initial application of recursive functions was to prove incompleteness theorems of logic. For that purpose, no deep results on the internal structure of the class of recursive functions are required. And in fact a more restricted class, such as the primitive recursive functions mentioned in the preceding section, would suffice.

But as will be seen, there are other applications for recursive functions. And if for no other reason, the recursive functions would be studied for their own interest as the effectively computable functions. And the basic fact that gets such a study off the ground is the possibility of encoding machines into integers that can then be supplied as input to other (or the same) machines.

There is a direct analogy here with actual digital computers. The earliest

such computers were programmed by setting switches and inserting wires into plugboards. It was then realized (by von Neumann) that for a suitably constructed computer, the program could be coded into machine words (i.e., integers) and stored in the machine in the same manner as data — the stored-program computer. The first practical benefit of this approach to programs is the speed at which new programs can be loaded into the computer to replace old programs. But a more significant benefit (for our purposes) is the possibility of executive programs, e.g. operating systems. An executive program accepts another program as incoming data. The executive program might then study the incoming object program and see that its instructions are carried out.

The ideas behind stored-program computers can be carried over to Turing machines. (Historically it was the other way around.) A Turing machine M might be given two numbers as input, one of them a suitable encoding of Turing machine N, and the other a number x. Machine M might then serve as an executive program, and the output might be just the result of applying N to x. M can then be called a *universal* Turing machine.

Carrying out these ideas and constructing a universal Turing machine turns out to be a straightforward (if somewhat lengthy) procedure. We will outline how it goes. First of all, each Turing machine is a finite object, and so can be encoded as a natural number under some fixed encoding. We can, for example, define the encoding

$$\langle x_0, x_1, \ldots, x_n \rangle = 2^{x_0+1}3^{x_1+1} \cdots p_n^{x_n+1}$$

in powers of primes as a way of condensing a finite string of numbers to a single number. We then need a decoding function $(x)_y$ with the property that for $i \leq n$,

$$(\langle x_0, x_1, \ldots, x_n \rangle)_i = x_i.$$

Turing machines can be found to effect the above encoding, and inversely to do the decoding.

At any point in the history of a Turing machine calculation, the entire configuration of the machine (the tape contents, the instruction number, and the square being scanned) can be described by a finite amount of information, and so can again be encoded into a number, called an *instantaneous description*. Then a *computation record* for machine M is a number encoding a finite sequence of instantaneous descriptions meeting the following conditions.

(i) The first instantaneous description specifies instruction q_0 (the "initial state").

(ii) The last one has the machine at an instruction q_i and scanning a symbol s for which $M(i, s)$ is undefined (a "halting configuration").

(iii) Each one is related to the next in that M, when in the configuration given by one instantaneous description, moves in one step to the configuration given by the next.

Thus a computation record is a natural number that encodes the entire history of one calculation by M, from its initial state q_0 (presumably with some input of interest on the tape) until it halts.

All this encoding would be pointless were it not for the fact: the results of the encoding are *recursive* functions and relations. That is, in going through the details of this encoding, one can verify at every step that Turing machines exist to handle the concepts involved. In the end one has the following two results.

(i) There is a recursive ternary relation T that holds of e, $\langle x_1, \ldots, x_k \rangle$, and y iff e encodes a Turing machine and y is a computation record for that machine, starting with $\ulcorner x_1 \urcorner 0 \ulcorner x_2 \urcorner 0 \cdots 0 \ulcorner x_k \urcorner$ on the tape.

(ii) There is a recursive function U such that whenever $T(e, \langle x_1, \ldots, x_k \rangle, y)$ holds, then $U(y)$ — the upshot of y — is the output value of the calculation (provided the halting configuration is such that this makes sense).

Even without the details, it should appear that T is intuitively decidable and U is computable. And so one would expect them to be recursive; that expectation is correct. Next we define, for each k, the k-place partial function

$$\{e\}^k(x_1, \ldots, x_k) = U[\text{the least } y \text{ such that } T(e, \langle x_1, \ldots, x_k \rangle, y)].$$

Here we write "the least y" although it is quite possible that no such y exists; if there is no such y then the function is undefined at that point. We abbreviate all this by the letter μ:

$$\{e\}^k(x_1, \ldots, x_k) = U(\mu y\, T(e, \langle x_1, \ldots, x_k \rangle, y)).$$

(The notation $\{e\}^k$ is Kleene's; the notation $\varphi_e^{(k)}$ is used by Rogers. The superscript k is omitted whenever possible.)

We can now conclude the following fundamental theorem. The theorem in this form is due to KLEENE [1936a, 1943], but universal Turing machines appeared in the original paper by TURING [1936].

4.1. Normal Form Theorem

(i) *The $(k+1)$-place partial function whose value at $(e, x_1, \ldots, x_k)$ is $\{e\}^k(x_1, \ldots, x_k)$ is recursive.*

(ii) *For each e, the k-place partial function* $\{e\}^k$ *is recursive.*
(iii) *Every k-place recursive partial function equals* $\{e\}^k$ *for some e.*

The number e will be called an *index* of $\{e\}^k$. Thus a partial function is recursive iff it has an index.

We want next to use this work to prove the unsolvability of the halting problem. Suppose we give input x to machine M and start it running. After the first million steps, we might become suspicious that it will never halt. On the other hand, maybe if we have just a little more patience, it will halt after a few more steps. Is there any way to test which of these two situations we are in? No, there is not. There is no effective procedure that, given M and x, will decide whether or not this calculation ever terminates. This is the content of the theorem below. For a partial function f, we write $f(x)<\infty$ to mean that $f(x)$ is defined.

4.2. THEOREM (unsolvability of the halting problem). *Neither* $\{(x, y): \{x\}(y)<\infty\}$ *nor* $\{x: \{x\}(x)<\infty\}$ *is recursive.*

PROOF. Our description of this proof (and others) relies on the reader's informal ideas of effective computability, but it can be translated into a rigorous description involving Turing machines.

Let $K=\{x: \{x\}(x)<\infty\}$. It suffices to show that K, the diagonal of the halting problem, is not recursive. We use a classical diagonal argument. Consider the function

$$g(x)=\begin{cases}\{x\}(x)+1 & \text{if } x\in K,\\ 0 & \text{if } x\notin K.\end{cases}$$

The function g is total, but it cannot be recursive (because $g(e)\neq\{e\}(e)$ for each e). But if K were recursive, then g would be. So K is not recursive. □

In the foregoing sections, we have been totally indifferent to questions regarding just how long it took a Turing machine to compute a function value. But now suppose we examine $\Phi_e(x)$, the number of steps the machine with index e uses in computing $\{e\}(x)$. For any recursive function, we have the choice of infinitely many different machines to compute it. Some recursive functions are so stubborn that any available machine takes almost forever:

4.3. THEOREM (RABIN [1960]). *For any total recursive H on* $\mathbf{N}$, *we can find a total recursive* $F:\mathbf{N}\to\{0,1\}$ *such that for any index e of F,*

$$\Phi_e(x)>H(x)$$

for all sufficiently large x.

The proof involves gradually detecting indices of fast machines, and defining F so as to disagree somewhere with the result of such machines.

A stronger result is the speed-up theorem, which indicates that a function need not have any fastest index, or even any almost fastest index for any reasonable meaning of "almost".

4.4. SPEED-UP THEOREM (BLUM [1967]). *For any total recursive function G on* $\mathbf{N}\times\mathbf{N}$, *we can find a total recursive* $F:\mathbf{N}\to\{0,1\}$ *such for each index i of F there exists another index j of F such that*

$$G(x,\Phi_j(x))<\Phi_i(x)$$

for all sufficiently large x.

For example, take $G(x, y)=2^y$. Then for any machine computing F, there exists another machine exponentially faster for almost all inputs. The theorems of Rabin and Blum are actually more general than we have indicated. In these theorems $\Phi_e(x)$ can be any reasonable measure of the complexity of computing $\{e\}(x)$, subject only to some very modest assumptions.

For any total recursive function L on $\mathbf{N}$ we can define the complexity class C_L of functions almost always computable in a number of steps bounded by L:

$$C_L=\{F: \text{for some index } f \text{ of } F,\ \Phi_f(x)\leq L(x) \text{ for all sufficiently large } x\}.$$

These complexity classes organize the recursive functions according to computational difficulty. In particular, we can say that F is no harder to compute than G if F belongs to every complexity class to which G belongs. This happens iff for every index of G there exists an index of F that is almost always just as fast.

5. Oracles and functionals

Three years after his original paper (TURING [1936]), TURING [1939] introduced an extension of his concept of computability. Imagine a

computing agent (an industrious clerk or a machine) provided, as usual, with explicit instructions and plenty of scratch paper. But in addition we now provide a new feature: an oracle for a particular function α from $\mathbf{N}$ into $\mathbf{N}$. (Here dom α is required to be all of $\mathbf{N}$; let $\mathbf{N}^{\mathbf{N}}$ be the set of all such functions.) An oracle for α is a device that, given a number x, responds by producing the value $\alpha(x)$. For a recursive α, we can make an oracle for α from a Turing machine. But more generally we can imagine an oracle for an arbitrary function α. Our computing agent supplied with this oracle now can calculate not only the effectively computable partial functions, but can further calculate (when given the right instructions) any partial function that is "computable in α".

At first glance, the concept of computability in α seems quite odd. It combines the most constructive approach to functions (that of computability) with the least constructive approach (that of an oracle). But despite this paradoxical appearance, the concept has proved to be valuable. And it led eventually (in the 1950's) to the concept of a recursive functional, i.e., a recursive function accepting members of $\mathbf{N}^{\mathbf{N}}$ as arguments.

In general we will consider (k, m)-place partial functions; the domain of such a function is a subset of

$$\mathcal{N} = \mathbf{N} \times \mathbf{N} \times \cdots \times \mathbf{N} \times \mathbf{N}^{\mathbf{N}} \times \mathbf{N}^{\mathbf{N}} \times \cdots \times \mathbf{N}^{\mathbf{N}}$$

(with k factors of $\mathbf{N}$ and m factors of $\mathbf{N}^{\mathbf{N}}$) and its range is a subset of $\mathbf{N}$. Until now, we have discussed only the case where the space $\mathcal{N}$ was countable (i.e., $m = 0$). Henceforth we will often treat the case $k = m = 1$ for notational simplicity, with the understanding that the remarks generalize. We use $\mathfrak{x}, \mathfrak{y}, \ldots$ as variables over the space $\mathcal{N}$.

We now extend our notion of Turing machines to allow for m oracles. A machine can now write a number x on the tape and the oracle will, in one step, replace it with $\alpha(x)$. A partial function f on $\mathcal{N}$ is defined to be *recursive* if there is a Turing machine that computes f as before, where now the function arguments of f are supplied in the form of oracles.

As in Section 4, it is possible to encode the entire history of a single calculation into one number y, the computation record. Among other things, y encodes all information supplied by the oracle. Of course in any one terminating calculation, only a finite amount of the potentially infinite wisdom of the oracles can be utilized. Under any reasonable encoding of calculations, if y is a computation record then any value $\alpha(i)$ supplied to the calculation by the oracle will have $i < y$. Define for α in $\mathbf{N}^{\mathbf{N}}$ the "course-of-values" function $\bar{\alpha}$ by

$$\bar{\alpha}(y) = \langle \alpha(0), \alpha(1), \ldots, \alpha(y-1) \rangle.$$

Thus $\bar{\alpha}$ is again in $\mathbf{N}^{\mathbf{N}}$ and $\bar{\alpha}(y)$ encodes the first y values of α. (In particular $\bar{\alpha}(0) = \langle\ \rangle = 1$, no matter what α is.) For a computation record y, the number $\bar{\alpha}(y)$ enodes all values of α that were used in the calculation, since for $i < y$ the value $\alpha(i)$ can be decoded from $\bar{\alpha}(y)$, in fact $\alpha(i) = (\bar{\alpha}(y))_i$. This phenomenon leads to the following two results.

(i) There is a recursive 4-ary relation T that holds of the natural numbers $e, \langle x_1, \ldots, x_k \rangle, \langle \bar{\alpha}_1(y), \ldots, \bar{\alpha}_m(y) \rangle$, and y iff e encodes a Turing machine and y is a computation record for that machine, started with $\ulcorner x_1 \urcorner 0 \ulcorner x_2 \urcorner 0 \cdots 0 \ulcorner x_k \urcorner$ on the tape and supplied with oracles for $\alpha_1, \ldots, \alpha_m$.

(ii) There is a recursive function U such that whenever T holds of the above-mentioned four numbers, then $U(y)$ is the output value of the calculation.

Thus we can extend the normal form results of the preceding section by defining the (k, m)-place partial function $\{e\}^{k,m}$ where

$$\{e\}^{1,1}(x, \alpha) = U(\mu y\, T(e, \langle x \rangle, \langle \bar{\alpha}(y) \rangle, y)).$$

As usual, we omit the superscripts whenever possible. The Normal Form Theorem 4.1 then holds, mutatis mutandis. It is interesting to note that since U and T have only natural numbers as arguments, recursiveness on $\mathcal{N}$ can be characterized in terms of recursiveness on $\mathbf{N}$.

6. Recursive enumerability

We have defined a subset of $\mathcal{N}$ to be a recursive (k, m)-ary relation if its characteristic function was recursive. By Church's thesis, this is the correct formalization of the informal notion of a decidable set.

Now we want to consider sets that are only half recursive. Call a set R *semi-decidable* if there is an effective procedure that, given $\mathfrak{x}$, replies "yes" iff $\mathfrak{x} \in R$. The procedure is no longer required to be a decision procedure; now it can be thought of as an accepting procedure. If $\mathfrak{x} \in R$, then the procedure eventually says "yes", thereby accepting $\mathfrak{x}$. But if $\mathfrak{x} \notin R$, then in general the procedure will never terminate. But one never knows in advance whether the procedure will go on forever or will eventually halt and accept $\mathfrak{x}$.

When $\mathcal{N}$ is a countable space, we can give another characterization of the semi-decidable sets. Call R *effectively enumerable* if there is an effective procedure that lists, in some order, the members of R. (Of course if R is infinite then the list will never be completed. But for any particular member of R, it appears on the listing after some finite length of time.) To prove that effective enumerability is equivalent to semi-decidability, first

assume that R is effectively enumerable. Then given any $\boldsymbol{x}$, we can scan the listing of R as it appears, and say "yes" if and when we see $\boldsymbol{x}$. This shows that R is semi-decidable. Conversely assume that R is semi-decidable. To generate a listing of R, we must budget our time sensibly. Order $\mathbb{N}^k$ first according to maximum component and then lexicographically; this orders $\mathbb{N}^k$ in type ω. Then go through all k-tuples in order: $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$. At stage n of the listing procedure, spend n minutes on each of $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, testing them for acceptance into R. If any of these tests results in a "yes", then put that k-tuple on the output list. In this way, any member of R is eventually discovered and placed on the list. (Obviously this argument relies on having a countable space $\mathbb{N}^k$; an uncountable semi-decidable set cannot be listed in this sense.)

Next we want to give a precise counterpart of the informal concept of a semi-decidable set. One possibility would be to go back to Turing machines, regarding them not as transducers (with both input and output) but as acceptors. But there is a simpler alternative open to us. Any semi-decidable set is the domain of the computable partial function taking the value 0 on the set and undefined outside the set. Conversely, the domain of any computable partial function is semi-decidable; one says "yes" if and when the computation terminates. Hence we can formulate semi-decidability as follows.

6.1. DEFINITION. A subset of $\mathcal{N}$ is *semi-recursive* if it is the domain of some recursive partial function on $\mathcal{N}$. If $\mathcal{N}$ is countable, then semi-recursive sets are called *recursively enumerable* (abbreviated r.e.).

If R is semi-recursive by virtue of being the domain of f, then we can think of the Turing machine that computes f as being the accepting device for R, where acceptance amounts to halting. The phrase "recursively enumerable" is sometimes used as a synonym for "semi-recursive" regardless of the size of $\mathcal{N}$, but we will confine the phrase to countable $\mathcal{N}$.

If we have an accepting device for R and another for its complement (with respect to $\mathcal{N}$), then the two devices together decide membership in R. Hence we have the following result.

6.2. THEOREM. *A relation is recursive iff both it and its complement are semi-recursive.*

We can exploit our indexing of recursive partial functions to obtain an indexing of the semi-recursive sets. Simply define $W_e^{k,m}$ to be the domain of

$\{e\}^{k,m}$. (We omit the superscripts whenever possible.) Then a relation R is semi-recursive iff it is W_e for some e. Furthermore the $(k+1, m)$-ary relation

$$Q = \{(e, \mathfrak{x}) : \mathfrak{x} \in W_e\}$$

is semi-recursive (being the domain of the function computed by a universal Turing machine). The semi-recursive relation Q is "universal" for (k, m)-ary semi-recursive relations in the sense that a relation R is semi-recursive iff it is obtainable from Q by holding e fixed as a parameter.

6.3. THEOREM. *The following conditions on a (k, m)-ary relation are equivalent.*

(i) *R is semi-recursive.*

(ii) *For some recursive $(k+1, m)$-ary relation Q,*

$$R = \{\mathfrak{x} : \exists w\, Q(w, \mathfrak{x})\}.$$

(iii) *For some recursive $(k+l, m)$-ary relation P,*

$$R = \{\mathfrak{x} : \exists w_1 \cdots \exists w_l\, P(w_1, \ldots, w_l, \mathfrak{x})\}.$$

PROOF. We have (i) $\Rightarrow$ (ii) because $x \in W_e \Leftrightarrow \exists y\, T(e, \langle x \rangle, y)$. Trivially (ii) $\Rightarrow$ (iii). To prove (iii) $\Rightarrow$ (i) we use sequence encoding: $R = \operatorname{dom} f$ where $f(\mathfrak{x}) = \mu w P((w)_1, (w)_2, \ldots, (w)_l, \mathfrak{x})$. For recursive P, the partial function f is also recursive. □

In Section 4 we showed that the set

$$K = \{x : \{x\}(x) < \infty\}$$

was not recursive. But K is a recursively enumerable subset of $\mathbb{N}$, since

$$x \in K \Leftrightarrow \exists y\, T(x, \langle x \rangle, y)$$

for a recursive relation T. So we may conclude that the complement $\bar{K}$ is not r.e.

Although K is undecidable, there is a sense in which questions about membership in any r.e. subset of $\mathbb{N}$ are reducible to questions about K. Consider any r.e. subset W_e of $\mathbb{N}$ and define for each x the function

$$f(t) = \{e\}(x).$$

Then $f(t)$ is independent of t, and in fact f is the empty function if $x \notin W_e$. But f is total if $x \in W_e$. Now f is a recursive partial function, but more to the point is that we can recursively find an index $\pi(e, x)$ for f. On an

informal level, this is clear: the equation displayed above tells how to calculate f, and we can arrive at this equation effectively when given e and x. Formally, we use the parameter theorem below.

If g is a two-place recursive partial function, then $g(8, y)$ is, as a function of y, recursive. A more significant fact is that we can recursively find an index for this function from an index for g. This fact, stated more generally, is the following theorem.

6.4. PARAMETER THEOREM. *For each k and m there is a one-to-one total recursive function ρ such that*

$$\{e\}(x_1, \ldots, x_n, y_1, \ldots, y_k, \alpha_1, \ldots, \alpha_m) = \\ = \{\rho(e, \langle x_1, \ldots, x_n\rangle)\}(y_1, \ldots, y_k, \alpha_1, \ldots, \alpha_m)$$

always holds.

Here $x_1, \ldots, x_n$ are parameters being held fixed. The idea is to have $\rho(e, v)$ encode instructions for writing v to the left of the other input on the tape, and then following instruction encoded by e. (The parameter theorem is also known as the "S–m–n theorem", for historical reasons.)

We can now apply the parameter theorem to the function

$$g(e, x, t) = \{e\}(x)$$

to get a one-to-one total recursive π such that $g(e, x, t) = \{\pi(e, x)\}(t)$ and hence

$$x \in W_e \Rightarrow \{\pi(e, x)\} \quad \text{is total,}$$

$$x \notin W_e \Rightarrow \{\pi(e, x)\} \quad \text{is empty,}$$

$$x \in W_e \Leftrightarrow \pi(e, x) \in K.$$

This reduces questions about W_e to questions about K. There are other r.e. sets besides K for which such reductions exist. The most obvious example is $\{\langle x, y\rangle : x \in W_y\}$.

For subsets A and B of $\mathbf{N}$, define A to be *many–one reducible* to B ($A \leq_m B$) if for some total recursive f,

$$x \in A \Leftrightarrow f(x) \in B.$$

Call A *one–one reducible* to B ($A \leq_1 B$) if in addition f can be required to be one-to-one. Then any r.e. subset of $\mathbf{N}$ is one–one reducible to K. It is clear (at least on the informal level) that if either $A \leq_m B$ or $A \leq_1 B$ and B is recursive, then A is also recursive. The same is true with "recursive" replaced by "recursively enumerable".

The parameter theorem is a standard tool in formalizing reductions of one decision problem to another. Such a reduction may prove that a decision problem is unsolvable, as in the following result.

6.5. THEOREM (RICE [1953]). *Let $\mathscr{C}$ be a set of one-place recursive partial functions. Then the set $\{e : \{e\} \in \mathscr{C}\}$ of indices of members of $\mathscr{C}$ is recursive iff either $\mathscr{C}$ is empty or $\mathscr{C}$ contains all one-place recursive partial functions.*

PROOF. The "$\Leftarrow$" half is trivial. So assume that the set of indices

$$I = \{e : \{e\} \in \mathscr{C}\}$$

is recursive. Since both the hypothesis and the conclusion of the theorem are symmetric with respect to $\mathscr{C}$ and its complement, we may suppose that the empty function $\emptyset$ is not in $\mathscr{C}$. We will show that $\mathscr{C}$ is empty by showing that we could otherwise reduce membership questions about K to the recursive set I.

So assume that, contrary to our hopes, some function ψ is in $\mathscr{C}$. The idea is to end up with $x \in K \Leftrightarrow g(x) \in I$ by arranging to have $\{g(x)\}$ be ψ or $\emptyset$ as x either is or is not in K. Informally, $g(x)$ encodes instructions for: given y, compute first $\{x\}(x)$ and then $\psi(y)$. Formally, $g(x) = \rho(e, \langle x\rangle)$ where

$$\{e\}(x, y) = U(\mu z[T(x, \langle x\rangle, (z)_0) \;\&\; T(q, \langle y\rangle, (z)_1)])_1$$

and q is an index for ψ. This gives us $K \leq_1 I$, contradicting the fact that I is recursive and K is not. □

As immediate consequences of Rice's theorem, we have the following negative statements. The set of indices of total recursive functions is not recursive. For any fixed recursive partial function f on $\mathbf{N}$, the set of indices of f is not recursive (and hence is infinite). The set $\{e : W_e \text{ is finite}\}$ is not recursive. And so forth.

A more subtle consequence of the parameter theorem is the recursion theorem, due to KLEENE [1938].

6.6. RECURSION THEOREM. (i) *For any total recursive $f : \mathbf{N} \to \mathbf{N}$ we can find a number e for which $\{e\} = \{f(e)\}$.*

(ii) *For any recursive partial function g we can find a number e such that $\{e\}(\mathfrak{x}) = g(e, \mathfrak{x})$ for all $\mathfrak{x}$.*

The proof is very like the proof that gives us self-referential sentences in number theory (e.g. Theorem 2.2.1 in Chapter D.1).

PROOF. Parts (i) and (ii) are equivalent. To prove (i), we obtain from the parameter theorem a total recursive γ such that $\{\gamma(x, y)\} = \{\{x\}(y)\}$ for any x and y. Let r be an index for the function whose value at x is $f(\gamma(x, x))$ and let $e = \gamma(r, r)$. This number e works:

$$\{e\} = \{\gamma(r, r)\} = \{\{r\}(r)\} = \{f(\gamma(r, r))\} = \{f(e)\}.$$

To prove part (ii), we first get from the parameter theorem a total recursive f such that $\{f(t)\}(\mathfrak{x}) = g(t, \mathfrak{x})$. Then by part (i) there is a number e such that $\{e\}(\mathfrak{x}) = \{f(e)\}(\mathfrak{x}) = g(e, \mathfrak{x})$. □

We can use the recursion theorem to give a short proof of Rice's theorem. Suppose that $\{a\} \in \mathscr{C}$ and $\{b\} \notin \mathscr{C}$, and define $f(x)$ to be b or a as x is or is not in I. There can be no e such that $\{e\} = \{f(e)\}$, and hence f cannot be recursive. So I is not recursive.

7. Logic and recursion theory

Why is recursive function theory part of mathematical logic? If logicians had not invented recursive functions, computer scientists would have developed the subject later. But it was not a mere historical accident that recursive funtions were invented by logicians. There are certain aspects of logic that inevitably involve the notions of constructiveness and effectiveness.

A basic concept of logic is that of a *proof*. Now a proof, viewed abstractly, is a series of statements that "establishes" without doubt the truth of its conclusion, given the truth of its assumptions. But to establish convincingly the truth of the conclusion, the proof must be verifiable by others. There must be some procedure by which an outsider can verify the correctness of the proof, without having to supply brilliant insight. That is, it must be possible to verify the correctness of proofs by an effective procedure. The set of proofs must be recursive. It would not do, for example, to consider just any series of true sentences of arithmetic to be a proof of its last line. We cannot, given a sentence of arithmetic, tell effectively whether or not it is true, because the set of true sentences of arithmetic is not recursive nor even r.e. (Section 10).

Now consider the set of all theorems, i.e., the provable sentences. A sentence σ is provable iff

$$\exists d\,[d \text{ is a proof of } \sigma].$$

The part in square brackets must be recursive. And so the set of theorems must be recursively enumerable. Thus as long as we can effectively recognize correct proofs, the set of theorems will be recursively enumerable! The Gödel incompleteness theorem discussed in Chapter D.1 stems from the fact that provability is r.e. whereas truth is not.

To be more specific, consider a first-order language L, such as the language for set theory, having finitely many non-logical symbols. (Actually there could be $\aleph_0$ non-logical symbols as long as they are arranged tidily.) We first assign numbers (called *Gödel numbers*) to the expressions of the language in a straightforward way. This permits us to apply notions of recursion theory to the expressions. (Alternatively we could have Turing machines work directly on the symbols of the language.) In fact we will not bother to distinguish between an expression and its Gödel number. One can verify that the set of formulas is recursive, as is the set of sentences. Now add a set A of axioms, such as the Zermelo–Fraenkel (ZF) axioms of set theory. We naturally expect A to be recursive, so that in verifying the correctness of a proof we will be able effectively to tell the axioms from the non-axioms. (For example, the set of ZF axioms *is* recursive.) For a recursive set A, the binary relation

$$\{(\sigma, d)\colon d \text{ is a proof of } \sigma \text{ from } A\}$$

is recursive, where "proof" is defined as in Section 4 of Chapter A.1. (In fact we could use either formal system from that chapter.) This is not a deep result; we intuitively expect proofhood to be decidable, so when the concepts involved are made precise it should not be surprising to find that it is indeed decidable. Call a theory *recursively axiomatizable* if it is given by a recursive set A of axioms in a language L as above.

7.1. THEOREM. *A recursively axiomatizable theory has a recursively enumerable set of theorems.*

PROOF. τ is a theorem iff

$$\exists d\,[d \text{ is a proof of } \tau \text{ from } A]$$

where A is the recursive set of axioms. The part in brackets is recursive. □

Take again the example of ZF set theory. By Theorem 7.1, the set of theorems of ZF is r.e. It follows that the sentences of arithmetic provable in ZF (this can be made precise) form a r.e. set. So they cannot coincide with

the true sentences of arithmetic. Either too much is provable (and ZF is lying to us) or too little is provable.

Now let us go one step further and suppose that we have a recursively axiomatizable theory that is complete (i.e., for any sentence σ, either σ or $(\neg\sigma)$ is a theorem). Then we can strengthen Theorem 7.1; the theory is actually decidable.

7.2. THEOREM. *A complete recursively axiomatizable theory has a recursive set of theorems.*

PROOF. The conclusion certainly holds if the theory is inconsistent, so assume the theory is consistent. Suppose we are given a sentence σ and we want to decide whether it is a theorem. We generate a listing of all the theorems; by Theorem 7.1 this is possible. Eventually either σ or $(\neg\sigma)$ appears in the listing. When this happens, we can stop and give the correct answer. □

Theorem 7.2 is the basis for a number of decidability results; see Chapter C.3. Of course it suffers from the limitation of being applicable only to theories that are complete.

Properly viewed, proofs and calculations are objects of the same sort. A calculation (written down with all the steps) is a proof that the value of a function of a given argument is a certain number. And a proof is a calculation of one value of the function whose domain is the set of theorems. It is a calculation in the sense of being a finite and verifiable record that correct procedures have been followed.

For example, it turns out that a set R of numbers is recursive iff it is binumerable in first-order Peano arithmetic (cf. Section 3). Here the role of Turing machine computations is played by the formal deductions, modus ponens and all, establishing that a given number is indeed in the set.

Turing machines themselves have proved to be convenient tools in a variety of undecidability problems in logic. Take for example the result of KAHR, MOORE and WANG [1962] that for any formula φ we can effectively find an $\forall\exists\forall$ formula that is satisfiable iff φ is satisfiable. This is *not* proved by syntactical manipulations on φ, but instead by finding for each Turing machine M an $\forall\exists\forall$ formula that is satisfiable iff M never halts. This, together with results of the previous section, yields the existence of the desired reduction.

Or suppose we want to prove that the set of sentences having models of every non-zero cardinality is undecidable. We can do this by showing how, given a Turing machine M, to find effectively a sentence that has models of every non-zero cardinality iff M never halts.

8. Degrees of unsolvability

All of the non-recursive sets have unsolvable decision problems. But some are more unsolvable than others. In this section we will see how some amount of order can be imposed on the unsolvable problems.

Consider a partial function f on $\mathcal{N}$, and let $\mathcal{B}$ be a subset of $\mathbf{N}^{\mathbf{N}}$. It may be possible to compute f if we are given oracles for each function in $\mathcal{B}$. Define f to be *recursive in* $\mathcal{B}$ if there exists some recursive partial function g and some $\beta_1, \ldots, \beta_n$ in $\mathcal{B}$ such that

$$f(\mathfrak{x}) = g(\mathfrak{x}, \beta_1, \ldots, \beta_n)$$

for all $\mathfrak{x}$. Other definitions can then be "relativized to $\mathcal{B}$". For example a subset of $\mathcal{N}$ is *recursive in* $\mathcal{B}$ if its characteristic function is recursive in $\mathcal{B}$, and it is *semi-recursive in* $\mathcal{B}$ if it is the domain of some partial function recursive in $\mathcal{B}$. Usually $\mathcal{B}$ will be a singleton $\{\beta\}$, so that we speak of recursiveness in β and so forth.

The extreme case of $\mathcal{B} = \mathbf{N}^{\mathbf{N}}$ deserves special mention. When we give ourselves oracles for all functions in $\mathbf{N}^{\mathbf{N}}$, matters of recursiveness are washed out. On a countable space $\mathcal{N}$, every function is recursive in $\mathbf{N}^{\mathbf{N}}$. But for uncountable $\mathcal{N}$ this does not happen. If f is recursive in $\mathbf{N}^{\mathbf{N}}$ then in calculating $f(\alpha)$ there is still one restriction: We can use only a finite amount of information about α. That is, there must be some y (depending on α) such that $f(\alpha) = f(\gamma)$ for any γ agreeing with α at the first y values. This is exactly the condition for f to be continuous, when we put the discrete topology on $\mathbf{N}$ and the product topology on $\mathbf{N}^{\mathbf{N}}$. The space $\mathcal{N}$ is then given the product topology.

8.1. THEOREM. (a) *A function on $\mathcal{N}$ is recursive in $\mathbf{N}^{\mathbf{N}}$ iff it is continuous.*
(b) *A subset of $\mathcal{N}$ is semi-recursive in $\mathbf{N}^{\mathbf{N}}$ iff it is open.*

All the uncountable spaces $\mathcal{N}$ are homeomorphic to $\mathbf{N}^{\mathbf{N}}$. For example a homeomorphism from $\mathbf{N}^{\mathbf{N}} \times \mathbf{N}^{\mathbf{N}}$ onto $\mathbf{N}^{\mathbf{N}}$ can map (α, β) to the function taking $2x \mapsto \alpha(x)$ and $2x+1 \mapsto \beta(x)$. And $\mathbf{N}^{\mathbf{N}}$ is homeomorphic to the irrationals; the mapping here uses continued fractions. Thus it is possible to give a uniform treatment of topological set theory of the irrationals on the one hand and recursion theory of $\mathcal{N}$ on the other. Both sides gain from this

connection. For further discussion on this vein, see Chapter C.8 on descriptive set theory.

But we have strayed from our main topic. We will be concerned with a special case of relative recursiveness. Let α and β be members of $\mathbb{N}^{\mathbb{N}}$. Then, by our previous definition, α is recursive in β iff there is a recursive partial function g for which

$$\alpha(x) = g(x, \beta).$$

(Although both α and β are total, we cannot demand that g be total.) If α is recursive in β, then we write $\alpha \leq_T \beta$ and say that α is *Turing reducible* to β. We can also (and in fact equivalently) work with subsets of $\mathbb{N}$: say that A is recursive in B (written $A \leq_T B$) if the characteristic function of A is recursive in the characteristic function of B.

8.2. THEOREM. *The binary relation $\leq_T$ (either on $\mathbb{N}^{\mathbb{N}}$ or on $\mathcal{P}(\mathbb{N})$) is reflexive and transitive.*

On an informal level, transitivity of $\leq_T$ corresponds to connecting machines in series.

As a consequence of the above theorem, the symmetric relation of Turing equivalence

$$\alpha \equiv_T \beta \quad \text{iff} \quad \alpha \leq_T \beta \text{ and } \beta \leq_T \alpha$$

is an equivalence relation on $\mathbb{N}^{\mathbb{N}}$. Let $[\alpha]$ be the equivalence class of α. The equivalence classes are called *degrees of unsolvability*. The degrees are partially ordered by the relation

$$[\alpha] \leq [\beta] \quad \text{iff} \quad \alpha \leq_T \beta.$$

(Clearly this is well defined on equivalence classes.) We get the same degree structure on $\mathcal{P}(\mathbb{N})$ as on $\mathbb{N}^{\mathbb{N}}$, since for any α we can find a set B with α Turing equivalent to the characteristic function of B.

Thus the degrees of unsolvability are partially ordered according to just how unsolvable they are. There is obviously a least degree $\mathbf{0}$, consisting of the recursive functions. Because for any fixed function β the set $\{\alpha: \alpha \leq_T \beta\}$ is countable, it follows that each degree is a countable set of functions. Consequently there are $2^{\aleph_0}$ degrees. Another consequence is that any chain of degrees — any linearly ordered subset — has cardinality at most $\aleph_1$. This strongly suggests that incomparable degrees exist. This suspicion can be proved to be correct (without having to deny the

continuum hypothesis). We can simultaneously construct α and β so as to sabotage each machine that might reduce one function to the other.

In fact much more is true. There is an antichain — a set of degrees no two of which are comparable — of cardinality $2^{\aleph_0}$. This result is just a piece of the extensive information known about the degrees. See Chapter C.4 for much more on this topic.

Call a degree *recursively enumerable* (r.e.) if it is the degree of some r.e. set. Recall that the set

$$K = \{x: \{x\}(x) < \infty\}$$

is a r.e. subset of $\mathbb{N}$ that is not recursive. Hence the degree of K, denoted $\mathbf{0}'$, is a r.e. degree greater than $\mathbf{0}$. In Section 9 we will consider the question whether there are other r.e. degrees (Post's problem).

The passage from $\emptyset$ to K can be relativized to give us a function $A \mapsto A'$ on $\mathcal{P}(\mathbb{N})$. Define the *jump* of A (denoted A') to be the set

$$\{x: \{x\}^A(x) < \infty\}$$

where $\{x\}^A$ is the partial function recursive in A with index x, i.e.,

$$\{x\}^A(y) = \{x\}(y, \chi_A)$$

where χ_A is the characteristic function of A. Then A' is r.e. in A, but is not recursive in A; the proof is the same as for K. In fact by relativizing the proof of the corresponding result for K we have the following.

8.3. THEOREM. (i) *A' is r.e. in A but is not recursive in A.*
(ii) *A set B is r.e. in A iff $B \leq_1 A'$.*

Thus among the sets that are r.e. in A, its jump A' ranks highest with respect to one–one reducibility. It follows from the foregoing theorem that the jump operation is well defined on degrees.

8.4. THEOREM. *$A \leq_T B$ iff $A' \leq_1 B'$.*

This lets us define for each degree $\mathbf{a}$ its jump $\mathbf{a}'$. By Theorem 8.3 we have $\mathbf{a} < \mathbf{a}'$. And so we can continue: $\mathbf{a} < \mathbf{a}' < \mathbf{a}'' < \cdots$. In particular there is no largest degree. And above any degree we can find a chain of order type ω. In fact we can find one of the order type of the first uncountable ordinal; at limit ordinal steps we gather together all the previous sets in some systematic way.

9. Creative and lesser sets

As observed in Section 7, any recursively axiomatizable theory has a r.e. set of theorems. Thus r.e. sets are of particular significance for logic. For example one could hope that by classifying r.e. sets one would obtain an interesting classification of axiomatizable theories. The binary classification of r.e. sets into the recursive and non-recursive sets partitions theories into decidable and undecidable theories. But one could hope for a refinement of this binary split.

The first classification to examine comes from the degrees of unsolvability. And next one could examine the refinement obtained from other reducibilities. Clearly

$$A \leq_1 B \Rightarrow A \leq_m B \Rightarrow A \leq_T B,$$

so when we define

$$A \equiv_1 B \quad \text{iff} \quad A \leq_1 B \ \& \ B \leq_1 A,$$
$$A \equiv_m B \quad \text{iff} \quad A \leq_m B \ \& \ B \leq_m A$$

the equivalence relation $\equiv_T$ is refined by $\equiv_m$ and further refined by $\equiv_1$. Section 6 shows that there is a largest r.e. degree (the degree of K) with respect to each of these reducibilities.

But what about the other r.e. degrees? Each such degree contains axiomatizable theories:

9.1. THEOREM (FEFERMAN [1957]). *Every r.e. degree of unsolvability contains a recursively axiomatizable theory.*

PROOF. For any set A of numbers we can form a theory in the language of equality by taking as axioms the set

$$\{\neg \varepsilon_n : n \in A\}$$

where ε_n is the sentence "there are exactly n things in the universe". Then the set of theorems is Turing equivalent to A. If A is r.e. then we certainly have a r.e. set of axioms. But any theory whose axioms can be recursively enumerated $\{\sigma_0, \sigma_1, \sigma_2, \ldots\}$ has a recursive set of axioms $\{\sigma_0, \sigma_0 \wedge \sigma_1, \sigma_1 \wedge \sigma_1 \wedge \sigma_2, \ldots\}$. □

HANF [1965] has shown that this theorem can be strengthened by requiring that the theory be finitely axiomatizable. But the effect of the theorem is offset by the empirical observation that all "natural" r.e. theories are either of degree $\mathbf{0}$ or of degree $\mathbf{0}'$. Proofs that a theory T is undecidable generally show, in effect, that the halting problem for Turing machines is reducible to T. And the reducibility here is (or can become) $\leq_1$, so that the proof shows that $K \leq_1 T$. If T is recursively axiomatizable, then we have $T \equiv_1 K$.

For the cases of $\equiv_1$ and $\equiv_m$ we can give an intrinsic characterization of the sets in the highest r.e. degree. Theorem 6.2 states that a r.e. set A is non-recursive iff each r.e. subset of its complement $\bar{A}$ fails to fill $\bar{A}$. By uniformizing this last condition, we obtain the following definition, due to POST [1944].

9.2. DEFINITION. A set A of numbers is *creative* if A is r.e. and there exists a recursive partial function f such that whenever $W_x \subseteq \bar{A}$ then $f(x) \in \bar{A} - W_x$.

The function f in this definition is called a *productive* function for $\bar{A}$. For example, our set K is creative; we can take $f(x) = x$.

9.3. THEOREM (MYHILL [1955]). *The following conditions on a set of numbers are equivalent.*

(i) *A is creative.*
(ii) *A is a r.e. set to which all r.e. sets are $\leq_1$-reducible.*
(iii) *A is a r.e. set to which all r.e. sets are $\leq_m$-reducible.*

Trivially (ii) ⇒ (iii) and it is not hard to show that (iii) ⇒ (i). The main part of the proof is establishing (i) ⇒ (ii). This breaks down into two steps, first showing that a creative set has a one-to-one total productive function, and secondly using a version of the recursion theorem.

The above theorem implies that the usual undecidable axiomatizable theories such as first-order Peano arithmetic have creative sets of theorems.

There is still the question of what (if anything) lies between the recursive sets and the creative sets. For $\leq_m$ and $\leq_1$ reducibilities, the existence of intermediate sets was established by POST [1944]. He noted that a creative set must have a very rich complement, and proceeded to construct sets with sparse complements. Call a r.e. set S *simple* if $\bar{S}$ is infinite but includes no infinite r.e. subset.

9.4. THEOREM (POST [1944]). *Simple sets exist.*

PROOF. Imagine a fixed method of enumerating all r.e. sets. For each n, put into S the first discovered (if any) member of W_n that is greater than $2n$.

Then S is r.e., S intersects every infinite r.e. set, but S contains at most n of the first $2n+1$ numbers. □

A simple set is obviously non-recursive. And it is easy to generate an infinite r.e. subset of the complement of any creative set, so a simple set cannot be creative. Thus the above theorem establishes the existence of r.e. sets of intermediate $\equiv_m$ and $\equiv_1$ degree. "Post's problem" is the question whether there are r.e. sets of Turing degree intermediate between $\mathbf{0}$ and $\mathbf{0}'$. This question is not answered by the simple sets, which can be of degree $\mathbf{0}'$. Post had hoped that by placing even more stringent requirements on $\bar{S}$, he could force S to be of intermediate degree of unsolvability. This approach turned out to be unsuccessful. Finally in 1956 (two years after Post's death) the problem was solved simultaneously and independently by Friedberg (in his senior thesis at Harvard and in FRIEDBERG [1957]) and MUČNIK [1956] in the Soviet Union. They showed that intermediate r.e. degrees do exist, and in great profusion. Their method of proof has been termed the "priority method". In broad terms, this method involves a construction in which there are infinitely many requirements to be satisfied. But some of these requirements conflict with one another, so at various stages of the construction one must satisfy requirements of high priority, allowing those of lower priority to be injured. If all goes well, at the end each requirement has received enough attention for the construction to succeed.

Since 1956 the r.e. degrees have been studied extensively. We will quote here two theorems, both due to SACKS [1964, 1963]. The r.e. degrees are dense, in that if $\mathbf{a} < \mathbf{c}$ then $\mathbf{a} < \mathbf{b} < \mathbf{c}$ for a third r.e. degree $\mathbf{b}$. And any countable partial ordering can be embedded in the partial ordering of r.e. degrees. (For this it suffices to embed some countable atomless Boolean algebra — as a partial ordering — in the r.e. degrees.)

10. Definability and recursion

Mathematics is, as it has always been, largely the science of measurement. But "measurement" must here be understood as referring to more than the meter stick. The genus of a topological figure measures one of its aspects; objects of genus zero are in a sense simpler than those of higher genus. There are many dimensions of measurement in mathematics, and they go by many names: characteristic, transcendence degree, cardinality, fundamental group, etc. Occasionally we are so successful in the science of measurement that we can completely characterize an object (at least to within some concept of isomorphism) by giving, as it were, its latitude and longitude, i.e., its measurements in the relevant dimensions.

Usually the measurement values in a particular dimension can be ordered or at least partially ordered. They then induce a ranking on the objects being measured, according to whether the measurement of the object yields a value that is small or large.

Now consider the case of measurement of sets of natural numbers. This case is not as specialized as it might seem, since it is applicable to anything that can be encoded into sets of numbers, such as formal languages (sets of strings of symbols). First of all we have a binary measurement: a set of numbers can be recursive or non-recursive. For example the set of primes is recursive, and the set of theorems of set theory or group theory, suitable encoded, is not recursive. There are countably many recursive sets, so almost all sets (in the usual measure on $\mathcal{P}(\mathbf{N})$) are non-recursive.

The degrees of unsolvability provide one scale of measurement, indicating how far a set is from being recursive. But the degrees themselves form an untidy array, with only a few reliable bench marks to help us get our bearings.

The scale of measurement to be treated in this section is definability. It is applicable, in the present context, to sets that are definable in the structure $\mathfrak{N} = (\mathbf{N}, 0, S, +, \cdot)$ by formulas of first or second order. For each natural number n, let $\mathbf{n}$ be the corresponding numeral in the formal language of $\mathfrak{N}$. A formula $\varphi(x)$ with just x free is said to define A in $\mathfrak{N}$ if for every n,

$$n \in A \Leftrightarrow \mathfrak{N} \vDash \varphi(\mathbf{n}).$$

(This notation is from Chapter A.1.) Thus A consists of exactly those numbers making φ true in $\mathfrak{N}$. In the case of a k-ary relation, we use a formula $\varphi(x_1, \ldots, x_k)$ with several free variables.

Obviously only countably many sets are definable. We can produce an artificial example of a non-definable set by diagonalization. Nevertheless, among the $\aleph_0$ sets that are definable in $\mathfrak{N}$ lie most of the interesting sets of numbers. After all, if we are interested in A, then we probably know what A is, and that knowledge can probably be formalized to yield a definition of A in $\mathfrak{N}$.

If a set is definable in $\mathfrak{N}$, then we want to have a measurement indicating *how* definable it is. For a start, call a set *arithmetical* if it is definable in $\mathfrak{N}$ by a first-order formula, and call it *analytical* if it is definable by a second-order formula. Then the arithmetical sets constitute a subclass of the analytical sets, and it is a subclass of sets that are more easily definable than are the sets in its complement.

Within the class of arithmetical (or analytical) sets, we can ask for finer measurements as to just how definable a given set is. For example, we could use as a measurement the length of the shortest defining formula, or the number of quantifiers. But to obtain intrinsically significant measurements, some care is needed.

For example, exponentiation (as a ternary relation) is definable in $\mathfrak{N}$. But it is not easy to find a defining formula, and any defining formula will be fairly long. Yet our decision to include addition and multiplication in $\mathfrak{N}$ and to exclude exponentiation was rather arbitrary. We want measurements that are free from such arbitrary choices. For this reason we decide to measure "definability modulo recursiveness". We have available the following theorem, which is essentially due to GÖDEL [1931].

10.1. THEOREM. *Every recursive relation on* $\mathbf{N}$ *is arithmetical.*

The proof uses the techniques of Section 4 to translate statements about Turing machines into statements about numbers. It is necessary to have an arithmetical way of encoding a sequence of numbers into a single number. The Chinese remainder theorem provides such a method.

From this theorem we can further conclude that r.e. relations on $\mathbf{N}$ are also arithmetical, since the defining formula for $\{x: \exists y R(x, y)\}$ requires only one quantifier more than the defining formula for R. And complements of r.e. relations are arithmetical, and so forth.

Before making that "and so forth" more systematic, we should note that we can now prove the undecidability of number theory. Let T be the set of sentences true in $\mathfrak{N}$, suitably encoded into numbers. The set K is arithmetical, and so for some formula φ, we have $x \in K$ iff $\varphi(\mathbf{x}) \in T$. But this yields $K \leq_1 T$, so T cannot be recursive. We will see presently that a modification of this argument shows that T is not even arithmetical.

To return to definability, note that another consequence of Theorem 10.1 is that the arithmetical relations on $\mathbf{N}$ are exactly those definable in the structure with universe $\mathbf{N}$ and with *all* recursive relations. This is the natural structure in which to measure definability modulo recursiveness. Any relation definable in this structure is of course definable by a formula in prenex form. We will use the number of alternations between universal and existential quantifiers in that prenex formula as a measure of definability.

We can formulate these ideas in the following way, and for an arbitrary space $\mathcal{N}$. Let Π_0 be the class of recursive relations. Define Σ_{n+1} to be the class of all projections of Π_n relations, where in the present context P is called a projection of R if

$$P = \{\mathfrak{x}: \exists y_1 \cdots \exists y_l R(y_1, \ldots, y_l, \mathfrak{x})\}.$$

Thus if R is a $(k + l, m)$-ary relation, then P is a (k, m)-ary relation. We allow projection here only along numerical axes, not function axes. To complete the definition, let Π_n be the class of complements of relations in Σ_n.

An equivalent definition of Σ_n and Π_n can, at least for countable $\mathcal{N}$, be formulated in terms of the structure $\mathfrak{R}$ with universe $\mathbf{N}$ and with all recursive relations. A relation is in Σ_{n+1} iff it is definable in $\mathfrak{R}$ by a prenex formula having n alternations between universal and existential quantifiers, and whose outermost quantifier is existential. Π_{n+1} can be given a similar characterization, wherein the outermost quantifier is universal. (For uncountable $\mathcal{N}$, analogous characterizations are possible, if provision for function variables is made. We must allow formulas expressing $\tilde{\alpha}(x) = y$.)

Although we have deliberately shifted from the structure $\mathfrak{N}$ to $\mathfrak{R}$, it was proved in 1970 that the above paragraph remains correct with $\mathfrak{N}$ in place of $\mathfrak{R}$. Matijacevič's theorem that solved (or rather unsolved) Hilbert's tenth problem shows that any r.e. set of numbers can be put into the form

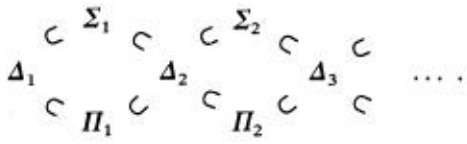
$$\{y: \exists \mathbf{x}\, (p(\mathbf{x}, y) = q(\mathbf{x}, y))\}$$

where $\mathbf{x} \in \mathbf{N}^k$ and p and q are polynomials over $\mathbf{N}$. For details on this theorem, see Chapter C.2.

To return to the arithmetical hierarchy, it is convenient to define also the class

$$\Delta_n = \Sigma_n \cap \Pi_n$$

when $n \geq 1$. For example, Theorem 6.2 implies that Δ_1 is the class of recursive relations, since Σ_1 is the class of semi-recursive relations and Π_1 is the class of complements of semi-recursive relations. By using vacuous quantifiers in defining formulas we can easily establish the following inclusions.



The class of arithmetical relations is the union of all these classes. All the inclusions shown above are proper. This follows from the hierarchy theorem below, due to KLEENE [1943].

10.2. THEOREM. *For each k, m, and n:*

(i) *There is a $(k+1, m)$-ary relation in Σ_{n+1} that is universal for (k, m)-ary relations in Σ_{n+1} (i.e., every (k, m)-ary relation is obtainable by holding the first numerical variable fixed as a parameter).*

(ii) *There is a (k, m)-ary relation in Σ_{n+1} that is not in Π_{n+1}.*

Part (ii) follows from part (i) by diagonalization, and part (i) follows from the Normal Form Theorem.

It is easy to see that both Σ_n and Π_n are closed under many–one reducibility. They are not closed under Turing reducibility, since every set of numbers is recursive in its complement.

We can now extend our proof that the set T of sentences true in $\mathfrak{N}$ is not recursive, to show that T is not arithmetical. In place of K, take any Σ_{n+1} subset A of $\mathbb{N}$ that is not Π_{n+1}. Then as before $A \leq_1 T$, so T cannot be Π_{n+1}. Since n is arbitrary, T cannot be arithmetical.

We can relate the arithmetical hierarchy to the jump operation on degrees of unsolvability, thus linking the definability concepts with the ideas of Section 8. The jump operation was there characterized by an existential quantifier, which corresponds to projection of relations. This line of thought leads eventually to the theorem below, which is proved by iteration of Theorem 8.3(ii). Let $\emptyset^{(n)}$ be the result of applying the jump operation n times to $\emptyset$.

10.4. THEOREM (POST [1948]). *A subset of $\mathbb{N}$ is in Σ_{n+1} iff it is r.e. in $\emptyset^{(n)}$ (and this holds iff it is one–one reducible to $\emptyset^{(n+1)}$). It is in Δ_{n+1} iff it is recursive in $\emptyset^{(n)}$.*

Recall that the analytical sets are those definable in $\mathfrak{N}$ by formulas of second order. As in the arithmetical hierarchy, we can use the number of alternations between universal and existential quantifiers (now quantifying function variables) as a measurement of definability. Let

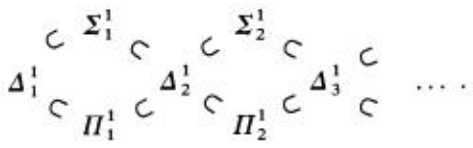
$$\Sigma^1_0 = \Pi^1_0 = \text{the class of arithmetical relations.}$$

Define Σ^1_{n+1} to be the class of all projections along function axes of relations in Π^1_n. Thus a relation in Σ^1_{n+1} is of the form

$$\{\mathfrak{x}\colon \exists\alpha_1 \cdots \exists\alpha_l R(\mathfrak{x}, \alpha_1, \ldots, \alpha_l)\}$$

where R is in Π^1_n. Define Π^1_n to be the class of complements of relations in Σ^1_n, and define Δ^1_n to be $\Sigma^1_n \cap \Pi^1_n$.

The simplest facts about the arithmetical hierarchy can be extended to the analytical hierarchy. We have the inclusions:



All inclusions here are proper, by a hierarchy theorem directly analogous to Theorem 10.2.

These classes in the analytical hierarchy enjoy stronger closure properties than do their counterparts in the arithmetical hierarchy. This is illustrated by Theorem 10.5 below, which is a quantitative version of the transitivity of definability. The classes Σ^β_n and $\Sigma^{1,\beta}_n$ are defined by replacing recursiveness by the relativized notion of recursiveness in β.

10.5. TRANSITIVITY THEOREM. *Assume that $\mathscr{A} \subseteq \mathscr{N}$ and $\beta \in \mathbb{N}^{\mathbb{N}}$.*

(i) $$\mathscr{A} \in \Sigma^\beta_{m+1} \;\&\; \beta \in \Delta_{n+1} \Rightarrow \mathscr{A} \in \Sigma_{m+n+1}$$

(ii) $$\mathscr{A} \in \Sigma^{1,\beta}_m \;\&\; \beta \in \Delta^1_n \Rightarrow \mathscr{A} \in \Sigma^1_{\max(m,n)}$$

The proof of (i) is straightforward. The proof of (ii), due to SHOENFIELD [1962], begins by writing

$$\mathfrak{x} \in \mathscr{A} \Leftrightarrow \exists\gamma\,[\gamma = \beta \;\&\; Q(\mathfrak{x}, \gamma)]$$

where Q is in Σ^1_m. The expression on the right is then put into prenex form. Part (ii) prevents having an analogue of Post's theorem (Theorem 10.4) hold for the analytical hierarchy.

What is in Δ^1_1? A clue is provided by descriptive set theory. When we relativize recursiveness to $\mathbb{N}^{\mathbb{N}}$, then Σ^1_1 becomes the class of projections of Borel sets of finite rank. These are the *analytic* or A-sets of descriptive set theory. Souslin's theorem SOUSLIN [1917] shows that a set and its complement are both analytic iff the set is a Borel set. This suggests, by analogy, that Δ^1_1 consists of the sets obtained by extending the arithmetical hierarchy into the transfinite. This is exactly correct; the hierarchy is called the *hyperarithmetical* hierarchy. It can be constructed by suitably iterating the jump operation over all ordinals that are "recursively countable" in the sense of being the order type of some recursive well ordering of numbers.

Determining the location of a given set in these hierarchies reduces to two problems. First there is the (usually easy) matter of establishing the

positive fact that the set in question does belong to, say, Σ^1_2. Then there is the negative task of showing that this is the best possible result, by showing that the set does *not* belong to the dual class Π^1_2. Take for example the subset R of $\mathbb{N}^{\mathbb{N}}$ consisting of the total recursive functions from $\mathbb{N}$ into $\mathbb{N}$. Then $R \in \Sigma_3$, because

$$\alpha \in R \Leftrightarrow \exists e\, \forall x\, \exists y\, [T(e, \langle x \rangle, y)\ \&\ U(y) = \alpha(x)]$$

and the part in brackets is recursive. SHOENFIELD [1958] has shown that R is not in Π_3.

We have seen that the set of true sentences of arithmetic is not arithmetical. This set does lie in the hyperarithmetical hierarchy, at level ω. This is a sharpening of Gödel's incompleteness theorem, which asserts that the set is not r.e. But now consider the characteristic function τ of the set of true sentences. Then the singleton $\{\tau\}$, as a subset of $\mathbb{N}^{\mathbb{N}}$, is in Π_2. To prove this we look closely at the definition of satisfaction in $\mathfrak{N}$ (Definition 3.8 of Chapter A.1). Viewed as conditions on τ, the definition is a Π_2 statement that is true of τ and nothing else. Since $\{\tau\}$ is in Π_2, it follows easily that truth is in Δ^1_1:

$$s \text{ is true} \Leftrightarrow \forall \alpha\, [\alpha \in \{\tau\} \Rightarrow \alpha(s) = 1]$$

$$\Leftrightarrow \exists \alpha\, [\alpha \in \{\tau\}\ \&\ \alpha(s) = 1].$$

The class of well orderings of $\mathbb{N}$ can be identified with a subset of $\mathbb{N}^{\mathbb{N}}$, namely

$$\{\alpha \in \mathbb{N}^{\mathbb{N}}\colon \{(x, y)\colon \alpha(\langle x, y \rangle) = 0\} \text{ well orders } \mathbb{N}\}.$$

It is easy to see that this set is in Π^1_1; we just write out its definition carefully and then count quantifiers. The set is not in Σ^1_1; in fact by a classical result it is not even analytic.

11. Recursive analogues of classical objects

The recursive functions are those you can actually write computer programs for; the others are more slippery and elusive. If one wants to approach mathematics from a constructive viewpoint, the recursive functions have a firmer ontological status than the others. One can, in principle, approach a mathematical object (the set of countable ordinals and the set of real numbers will be featured examples), and extract that part of it that is sufficiently constructive to be treated by recursion theory. This constructive part can then be viewed as a recursive analogue of the original.

As a preliminary example, let us consider the extent to which the collection R of total recursive functions on $\mathbb{N}$ is an analogue for the collection $\mathbb{N}^{\mathbb{N}}$ of all functions on $\mathbb{N}$. There is the obvious disparity of size; $\mathbb{N}^{\mathbb{N}}$ is uncountable whereas there are only countably many recursive functions. But the complete analogue of the uncountability of $\mathbb{N}^{\mathbb{N}}$ would say that there is no recursive one-to-one correspondence between the natural numbers and indices of total recursive functions. One formulation of this phenomenon is the following simple result.

11.1. THEOREM. *There is no total recursive function f on $\mathbb{N}$ such that $\{\{f(n)\}\colon n \in \mathbb{N}\}$ coincides with the collection R of total recursive functions.*

PROOF. As in the proof of the uncountability of $\mathbb{N}^{\mathbb{N}}$, we diagonalize. The equation $g(x) = \{f(x)\}(x) + 1$ defines a total recursive function that f has omitted. There is another proof that gives more quantitative information. From the fact (see Section 10) that R is not Π_3, we conclude that there cannot be any Σ_2 set A such that $R = \{\{x\}\colon x \in A\}$. □

The operations of addition, multiplication, and composition of functions are applicable to R as well as to $\mathbb{N}^{\mathbb{N}}$. But from the recursive viewpoint we need additional information: it must be possible, given indices of f and g, to find effectively indices for $f + g$, $f \cdot g$, and $f \circ g$. To prove that this can indeed be done, observe for example that $\{x\}(z) + \{y\}(z)$ is a recursive partial function of x, y, and z, and apply the parameter theorem.

We can summarize this by saying that the operations of addition, multiplication, and composition of functions are "effective on the indices". That is, each of our functions has indices that denote it, and operations on the functions are induced by operations on the indices, which in the present case are recursive operations. This phenomenon will recur in the subsequent examples: the constructive members of the classical object come with indices that denote them, and effective operations must work with these names.

For a more serious example of a recursive analogue of a classical object, take (as a classical object) the set of countable ordinals. Knowing that ordinal numbers are useful in transfinite constructions (as in the Borel hierarchy), we should not be surprised to find that a recursive analogue would be useful in transfinite constructions in recursion theory (as in the hyperarithmetical hierarchy, mentioned in the preceding section).

A cheap analogue is provided by the set

$$W = \{x\colon \{x\}^{2,0} \text{ encodes a well ordering}\},$$

where f is said to encode a well ordering if f is a total function from $\mathbb{N} \times \mathbb{N}$ into $\{0, 1\}$ and $\{(x, y): f(x, y) = 0\}$ is a well ordering of a subset of $\mathbb{N}$. More briefly, we can refer to W as the set of indices of recursive well orderings. For x in W, let $\|x\|$ be the ordinal number of the corresponding well ordering. Since an initial segment of a recursive well ordering is another such ordering, the set $\{\|x\|: x \in W\}$ is an initial segment of the countable ordinals. Its least upper bound will be called "recursive ω_1". Although recursive ω_1 is countable, it is the first ordinal that is not recursively countable.

Unfortunately, W has some deficiencies that impair its usefulness. For example, given x in W, we cannot tell effectively whether $\|x\|$ is a successor ordinal. Even if it is a successor ordinal, we cannot effectively find a member of W denoting the predecessor.

A better quality analogue is provided by a system of ordinal notations introduced by KLEENE [1938]. The set O is built up from below, unlike the set W. We will give an inductive definition simultaneously for the set O, the binary relation $<_O$, and a map $x \mapsto |x|$ from O into the ordinals. (The definition differs only slightly from Kleene's version.)

(i) $1 \in O$ and $|1| = 0$. Thus 1 is the unique number denoting the ordinal 0.

(ii) If $x \in O$ then $2^x \in O$, $x <_O 2^x$, and $|2^x| = |x| + 1$. This is the successor step.

(iii) If $x <_O y <_O z$ then $x <_O z$. The relation $<_O$ will in the end be a partial well ordering on O.

(iv) If $\{e\}$ is a total recursive increasing sequence of notations (i.e., $\{e\}: \mathbb{N} \to O$ and $\{e\}(n) <_O \{e\}(n+1)$ for each n), then $3 \cdot 5^e \in O$, $\{e\}(n) <_O 3 \cdot 5^e$ for each n, and $|3 \cdot 5^e| = \sup\{|\{e\}(n)|: n \in \mathbb{N}\}$.

This inductive definition assigns notations to an initial segment $\{|x|: x \in O\}$ of the countable ordinals. It turns out that the least ordinal not receiving a notation is again recursive ω_1. In fact there are several other ways to characterize recursive ω_1: it is the the least ordinal that is not the order type of any Σ^1_1 well ordering of numbers, and it is the least admissible ordinal after ω (in the sense of Chapter C.5).

As a final example of recursive analogues, we will consider recursive real numbers. This subject was first treated in TURING [1936]; a more recent reference is RICE [1954]. Any real number can be approximated by rationals, but we want to single out those numbers for which we can effectively generate rational approximations, and with a known estimate of the error. These are the realest of the reals.

Just as there are several ways to construct the reals from the rationals, so

are there several equivalent ways of making precise the concept of a recursive real. For example, we could require that the binary expansion of the fractional part of the number be given by a recursive funtion. Equivalently, we can adapt the Cauchy sequence construction. To each natural number n assign the rational number $r(n) = ((n)_0 - (n)_1)/((n)_2 + 1)$. Then say that e denotes a recursive real if $\{e\}$ is total and whenever $m > n$ then

$$|r(\{e\}(n)) - r(\{e\}(m))| < \frac{1}{2^n}.$$

The essential feature of both definitions is that we can effectively produce both upper and lower rational bounds converging to the number.

The recursive reals form a field, since for $x \neq 0$ we can get rational bounds of the same sign which then convert to rational bounds on $1/x$. (But given that e denotes a recursive real, there is no effective way to decide whether that real is zero.) Not only do we get a field, but the algebraic operations are "effective on the indices", e.g., there is a recursive function f such that if a and b denote recursive reals then $f(a, b)$ denotes their sum.

The algebraic real numbers are all recursive. In fact it is not too hard to see that the recursive reals form a real closed ordered field, i.e., an ordered field in which any polynomial that changes sign has an intervening root. Consequently the result of adjoining $\sqrt{-1}$ to the field is algebraically closed. (See VAN DER WAERDEN [1953], section 70.) Furthermore by Tarski's theorem, the field of recursive reals is elementarily equivalent to the field of all real numbers (see Chapter A.2).

In addition to the algebraic numbers, various transcendental numbers such as e and π are recursive, since there are well-known methods of churning out convergent upper and lower rational bounds.

The recursive reals are not as kind to analysts as they are to algebraists. The following example occurs in RICE [1954]; see also SPECKER [1949]. Start with a non-recursive r.e. set K. There is a one-to-one total recursive function f whose range is K (think of $f(n)$ as the $(n+1)$-st distinct number to emerge in an effective enumeration of K). Consider the set of rational numbers:

$$\frac{1}{2^{f(0)}}, \quad \frac{1}{2^{f(0)}} + \frac{1}{2^{f(1)}}, \quad \frac{1}{2^{f(0)}} + \frac{1}{2^{f(1)}} + \frac{1}{2^{f(2)}}, \quad \ldots.$$

This is a bounded recursive set of rational numbers. But its least upper bound (which is the only limit point of the set) is not a recursive real number, lest K be recursive.

Finally, let us consider the analogues of functions of a real variable. A function F from the set of recursive reals into itself is called a *recursive*

operator if there is a recursive partial function f such that whenever e denotes a recursive real x then $f(e)$ is defined and denotes $F(x)$. Thus f, working on the indices, performs F. We state without proof the following result.

11.2. THEOREM (KREISEL, LACOMBE and SHOENFIELD [1959], CEĬTIN [1959]). *Every recursive operator is continuous. Moreover, given an e denoting a recursive real and a positive rational ε, we can effectively find a δ that works.*

References

BLUM, M.
[1967] A machine-independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.*, **14**, 322–336.

BRADY, A.H.
[1966] The conjectured highest scoring machines for Rado's $\Sigma(k)$ for the value $k = 4$, *IEEE Trans. Computers*, EC–**15**, 802–803.
[1975] The solution to Rado's "busy beaver game" is now decided for $k = 4$ (abstract), *Notices Am. Math. Soc.*, **22**, A–25.

CEĬTIN, G.S.
[1959] Algorithmic operators in constructive complete separable metric spaces (Russian), *Dokl. Akad. Nauk SSSR*, **128**, 49–52.

CHURCH, A.
[1935] An unsolvable problem of elementary number theory (abstract), *Bull. Am. Math. Soc.*, **41**, 332–333.
[1936] An unsolvable problem of elementary number theory, *Am. J. Math.*, **58**, 345–363.

DAVIS, M.
[1958] *Computability and Unsolvability* (McGraw-Hill, New York).

FEFERMAN, S.
[1957] Degrees of unsolvability associated with formalized theories, *J. Symbolic Logic*, **22**, 161–175.

FRIEDBERG, R.M.
[1957] Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944), *Proc. Nat. Acad. Sci. U.S.A.*, **43**, 236–238.

GÖDEL, K.
[1931] Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I. *Monatsh. Math. Phys.*, **38**, 173–198.
[1965] On undecidable propositions of formal mathematical systems, in: *The Undecidable, Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions* (Raven Press, Hewlett, NY) pp. 41–71.

GREEN, M.W.
[1964] A lower bound on Rado's sigma function for binary Turing machines, in: *Switching Circuit Theory and Logical Design*, Proceedings of the fifth annual symposium, The Institute of Electrical and Electronics Engineers, pp. 91–94.

HANF, W.
[1965] Model-theoretic methods in the study of elementary logic, in: *The Theory of Models*, Proceedings of the 1963 international symposium at Berkeley (North-Holland, Amsterdam) pp. 132–145.

KAHR, A.S., MOORE, E.F. and WANG, H.
[1962] Entscheidungsproblem reduced to the $\forall\exists\forall$ case, *Proc. Nat. Acad. Sci, U.S.A.*, **48**, 365–377.

KLEENE, S.C.
[1936a] General recursive functions of natural numbers, *Math. Ann.*, **112**, 727–742.
[1936b] λ-definability and recursiveness, *Duke Math. J.*, **2**, 340–353.
[1938] On notation for ordinal numbers, *J. Symbolic Logic*, **3**, 150–155.
[1943] Recursive predicates and quantifiers, *Trans. Am. Math. Soc.*, **53**, 41–73.
[1952] *Introduction to Metamathematics* (North-Holland, Amsterdam; 7-th reprint, 1974).

KREISEL, G., LACOMBE, D. and SHOENFIELD, J.R.
[1957] Partial recursive functionals and effective operations, in: *Constructivity in Mathematics*, Proceedings of the colloquium held at Amsterdam, 1957 (North-Holland, Amsterdam) pp. 290–297.

LIN, S. and RADO, T.
[1965] Computer studies of Turing machine problems, *J. Assoc. Comput. Mach.*, **12**, 196–212.

MATIJACEVIČ, YU. V.
[1970] Recursively enumerable sets are Diophantine (Russian), *Dokl. Akad. Nauk SSSR*, **191**, 279–282.

MUČNIK, A.A.
[1956] Negative answer to the problem of reducibility in the theory of algorithms (Russian), *Dokl. Akad. Nauk SSSR*, **108**, 194–197.

MYHILL, J.
[1955] Creative sets, *Z. Math. Logik Grundlagen Math.* **1**, 97–108.

POST, E.L.
[1936] Finite combinatory processes–formulation I. *J. Symbolic Logic*, **1**, 103–105.
[1944] Recursively enumerable sets of positive integers and their decision problems, *Bull. Am. Math. Soc.*, **50**, 284–316.
[1948] Degrees of recursive unsolvability (abstract). *Bull. Am. Math. Soc.*, **54**, 641–642.

RABIN, M.O.
[1960] Degree of difficulty of computing a function and a partial ordering of recursive sets, technical report No. 2, Hebrew University, Jerusalem.

RADO, T.
[1962] On non-computable functions. *Bell System Tech. J.*, **41**, 877–884.

RICE, H.G.
[1953] Classes of recursively enumerable sets and their decision problems, *Trans. Am. Math. Soc.*, **74**, 358–366.
[1954] Recursive real numbers, *Proc. Am. Math. Soc.*, **5**, 784–791.

ROGERS, H. JR.
[1967] *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York).

SACKS, G.E.
[1963] *Degrees of Unsolvability* (Princeton University Press, Princeton, NJ).
[1964] The recursively enumerable degrees are dense, *Ann. of Math.*, **80**, 300–312.

SHOENFIELD, J.R.
[1958] The class of recursive functions, *Proc. Am. Math. Soc.*, **9**, 690–692.
[1962] The form of the negation of a predicate, in: *Recursive Function Theory*, edited by J. Dekker, Proceedings of symposia in pure mathematics, Vol. 5 (Am. Math. Soc., Providence, RI) pp. 131–134.
[1967] *Mathematical Logic* (Addison-Wesley, Reading, MA).
[1971] *Degrees of Unsolvability* (North-Holland, Amsterdam).

SOUSLIN, M.
[1917] Sur une définition des ensembles mesurables B sans nombres transfinis, *C.R. Acad. Sci. Paris*, **164**, 88–91.
SPECKER, E.
[1949] Nicht konstruktiv beweisbare Sätze der Analysis, *J. Symbolic Logic*, **14**, 145–158.
TURING, A.M.
[1936] On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.*, Ser. 2, **42**, 230–265.
[1939] Systems of logic based on ordinals, *Proc. London Math. Soc.*, Ser. 2, **45**, 161–228.
VAN DER WAERDEN, B.L.
[1953] *Modern Algebra*, Vol. 1 (Frederick Ungar, New York, revised English ed.).
YASUHARA, A.
[1971] *Recursive Function Theory and Logic* (Academic Press, New York).

C.2

Unsolvable Problems

MARTIN DAVIS*

Contents

* Work supported by National Science Foundation Grant DCR 71-02039 A05.

HANDBOOK OF MATHEMATICAL LOGIC
Edited by J. Barwise