CHAPTER V

COMPUTABILITY AND DECIDABILITY

§ 40. Decision and computation procedures. Consider a given countably infinite class of mathematical or logical questions, each of which calls for a "yes" or "no" answer.

Is there a method or procedure by which we can answer any question of the class in a finite number of steps?

In more detail, we inquire whether for the given class of questions a procedure can be described, or a set of rules or instructions listed, once and for all to serve as follows. If (*after* the procedure has been described) we select *any* question of the class, the procedure will then tell us how to perform successive steps, after a finite number of which we will have the answer to the question we selected. In performing the steps, we have only to follow the instructions mechanically, like robots; no insight or ingenuity or invention is required of us. After any step, if we don't have the answer yet, the instructions together with the existing situation will tell us what to do next.[162] The instructions will enable us to recognize when the steps come to an end, and to read off from the resulting situation the answer to the question, "yes" or "no".

In particular, since no human performer can utilize more that a finite amount of information, the description of the procedure, by a list of rules or instructions, must be finite.

If such a procedure exists, it is called a *decision procedure* or *algorithm* for the given class of questions. The problem of discovering a decision procedure is called the *decision problem* for this class.

For example, there is a decision procedure for the class of questions "Does a divide b?" (or "Is a a factor of b?") where a and b are any positive integers. It consists in performing the ordinary long division of b by a and observing whether or not the remainder is 0.

Likewise, there is an algorithm for determining whether or not an algebraic equation

$$(1) \qquad a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n = 0 \qquad (a_0 \neq 0,\ n > 0)$$

[162] In practice such procedures are often described incompletely, so that some inessential choices may be left to us. For example, if several numbers are to be multiplied together, it may be left to us in what order we multiply them.

with integral coefficients $a_0, a_1, \ldots, a_{n-1}, a_n$ has a rational root. It depends on the theorem that, if such an equation (1) has a rational root p/q (p, q integers), then p must be a factor of a_n and q a factor of a_0. So only finitely many rational numbers p/q are candidates for roots; and we can try them each in turn.

As a third illustration, there is a decision procedure for deciding whether or not, for given integers a, b and c, the equation $ax + by + c = 0$ has integral solutions for x and y. For this procedure, which is based on Euclid's "greatest common divisor algorithm", we refer to textbooks on elementary number theory (e.g. MacDuffee 1954 § 9).

For a given formal system **S**, consider the following three general questions, i.e. (countably infinite) classes of particular questions: "Is a given formal expression a formula?", "Is a given finite sequence of formulas a proof?", "Is a given formula provable?".

As we illustrated in § 38, any particular question of the first class can be answered by seeking a proper pairing of the parentheses in the expression. If one is found, then with the help of it we can attempt to retrace the steps by which the expression, if it is a formula, was built up under the definitions of "term" and "formula". In so doing, we shall find out whether the expression can or cannot be thus constructed. To answer any given question of the second class, we merely consider each formula in the given sequence in order, and examine whether it is an axiom or follows from earlier formula(s) by one of the rules of inference. The objects which must be examined to answer a question of either of these two classes are contained as parts of the finite object to which the question applies.

The third class of questions is fundamentally different. To show directly from the definition that a formula is provable, one must exhibit a proof of it. But the proof, if there is one, need not be made up only of parts of the formula itself. One must therefore look elsewhere than within the given object to answer the equation. The definition of a proof of a given formula sets no bound on the length of the proof. To examine all possible proofs without bound on their length is not a procedure which leads to the answer to the question in finitely many steps in the case the formula is not provable. So this third decision problem, i.e. the decision problem for provability in the system **S**, unlike the first two, is not trivial. If there is a decision procedure, it is one which is not afforded almost immediately by the definition of "provable formula". Also this decision problem for a formal system **S** is of especial interest. So it is often called *the* decision problem for the formal system.

For provability in the propositional calculus, there is the decision procedure found by Post in 1921; i.e., to determine whether or not $\vdash$ E, it

suffices to compute the truth table for E, and see whether or not it has all t's (for, by Theorems 14 and 12, $\vdash$ E if and only if $\models$ E).

The recognition of the decision problems for formal systems goes back to Schröder 1895, Löwenheim 1915, and Hilbert 1918.

It would be of especial import to have a decision procedure for the system N of elementary number theory (§ 38). For then the solution to many long-unsolved particular problems of elementary number theory could be obtained mechanically. This is assuming that only formulas true under the interpretation are provable in N. For example, we could then settle the problem of Fermat's "last theorem". About 1637 Fermat claimed he had a proof that the equation $x^n + y^n = z^n$ has no solution in positive integers x, y, z, n for $n > 2$. No one since has succeeded in proving or disproving this proposition. Fermat's "last theorem" can be expressed in N by the formula

$$\text{F:} \qquad \neg\exists x\exists y\exists z\exists n[x>0 \,\&\, y>0 \,\&\, z>0 \,\&\, n>2 \,\&\, \mathrm{A}(x, y, z, n)]$$

where $\mathrm{A}(x, y, z, n)$ is obtained by paraphrasing $x^n + y^n = z^n$ to avoid the exponent function (cf. end § 38). Thus to get $\mathrm{A}(x, y, z, n)$, we first find by known methods (Footnotes 155 and 156 § 38) a formula $\mathrm{E}(x, n, u)$ expressing $x^n = u$; then $\mathrm{A}(x, y, z, n)$ can be

$$\exists u\exists v\exists w[\mathrm{E}(x, n, u) \,\&\, \mathrm{E}(y, n, v) \,\&\, \mathrm{E}(z, n, w) \,\&\, u+v=w].$$

If Fermat's "last theorem" is false, this fact can be shown by computation with a suitable quadruple (x, y, z, n) as counterexample. Corresponding to this informal remark, it can be shown that in this case the formula $\neg$F is provable in N. We cannot give the details in this brief treatment; it comes under the claim we made in § 38 that N is adequate for the usual elementary number theory. If Fermat's "last theorem" is true, then, by the supposition we made here that in N only true formulas are provable, $\neg$F is unprovable. So a decision procedure for provability in N would enable us to decide by a finite number of mechanical steps whether Fermat's last theorem is true or false, by applying the procedure to determine whether $\neg$F is unprovable or provable, respectively.[163]

[163] This would constitute at least a significant *theoretical* gain over the present situation (1967), in which no sequence of mechanical steps is known to lead after finitely many steps to the answer to the question whether Fermat's "last theorem" is true or false.

Practically, the answer to the question might still be beyond our reach by the means described. For, in applying the given decision procedure for the system N to the formula $\neg$F, more space and time than we have available might be required to carry out the finitely many steps that lead to the decision.

For what we do in this book, it is not essential that we concern ourselves with this matter of whether a given decision procedure for a class of questions is or is not practical to use in answering certain questions of the class. (That falls under "computer sciences".)

The efforts expended over centuries in searching for solutions to this and other famous problems of number theory make it implausible that a decision procedure for N should exist. It might have seemed equally implausible in 1918 that mathematics could find a way to *prove* that there can be no decision procedure for N. But exactly this was done by Church in 1936 on the basis of a thesis of his which we shall propound in § 41.

We begin by observing that, just as we may have a decision procedure or algorithm for a countably infinite class of questions each calling for a "yes" or "no" answer, we may have a *computation procedure* or *algorithm* for a countably infinite class of questions which require as answer the exhibiting of some object.

For example, there is a computation procedure for the class of questions "What is the sum of two natural numbers a and b?". We learned this procedure in elementary school when we learned to add. The long division process constitutes an algorithm for the class of questions "For given positive integers a and b, what are the natural numbers q (the quotient) and r (the remainder) such that $a = bq + r$ and $r < b$?". Thirdly, there is the algorithm which bears Euclid's name for the class of questions, "What is the greatest common divisor of two positive integers a and b?". There is also an algorithm for the class of questions "Given a formula E in the propositional calculus and a list $P_1, \ldots, P_n$ of distinct atoms including all occurring in E, what is the truth table for E entered from these atoms?". The last three of these algorithms for what-questions are incorporated into three of the algorithms mentioned above for yes-or-no-questions.

The problem of finding a computation procedure or algorithm for a class of what-questions is the *computation problem* for the class of questions.

We have chosen here to formulate the idea of decision problems and computation problems only for countably infinite classes of questions (yes-or-no-questions or what-questions, respectively).

For a finite class of questions, the decision or computation problem (analogously formulated) is trivial from the classical standpoint. For (theoretically at least), it could be solved simply by preparing a list of the answers to all the questions of the class.

"What is the shortest distance by major highways between any two of the principal cities of the United States?" Assuming agreement on what are major highways and which are the principal cities, one can find the answer to any question of the class from the tabulations given on some highway maps.

"Given any permissible position in chess, can white win (irrespective of black's responses)?" It is notorious that in this case, although a finite list of the answers for all positions exists theoretically, for practical purposes it is unavailable. If it were, the fun in chess would be lost.

"Is a given one of a certain finite set of propositions $\{A, B, C, D, E\}$ true?" Here A, B, C, D, E are supposed to be five fixed propositions, so there are only five questions in our class. An algorithm would be provided by the right list of five yes's and no's. This would be short and simple. However, if A is Fermat's "last theorem", no one to date can provide the algorithm. Only a classical mathematician considers it established that the algorithm exists. An intuitionist would consider the question whether the algorithm exists to be open until someone has solved the problem of Fermat's "last theorem". (Cf. § 36.)

The results to be developed in the rest of this chapter are independent of such differences of opinion between classical and intuitionistic mathematicians on when an algorithm exists.

Returning to our case of algorithms for countably infinite classes of questions, they would likewise always exist trivially from a classical standpoint, if we allowed the (descriptions of) algorithms to be infinite objects; we could again simply list all the answers. But we said an algorithm has to a method or procedure or set of rules which can be used, and which therefore must be finitely described. A *finite* set of instructions must be provided that will suffice to guide one to the answer of any one of an *infinite* set of questions.

A discussion of algorithms for uncountably infinite classes of questions is outside the scope of this book.

When we have a countably infinite class of questions, the different questions of the class will usually arise by giving different values ("arguments") to one or several variables or "parameters" in the general statement of the (class of) questions. In several examples above, a and b (or a, b and c) play the role of these variables. In our second example of a decision procedure, $a_0, \ldots, a_n$ are such variables but their number n also varies.

Because our class of questions is always (outside of the digression above) to be countably infinite, we can always enumerate the questions, say as $Q_0, Q_1, Q_2, \ldots, Q_a, \ldots$. Then a can be the variable or parameter. In the case of yes-or-no-questions, by putting $P(a) \equiv$ {the answer to Q_a is "yes"}, the infinite class of yes-or-no-questions becomes a one-place number-theoretic predicate $P(a)$.

Suppose instead the questions are what-questions. We now further assume that the objects to be produced in answering the questions come

from a countably infinite class, which we can enumerate as y_0, y_1, $y_2, \ldots, y_b, \ldots$. By putting $f(a) = b$ if the answer to Q_a is b, the infinite class of what-questions becomes a one-place number-theoretic function $f(a)$.

So via an enumeration, each countably infinite class of yes-or-no questions we are to consider can be reduced to the form "Is the value of the number-theoretic predicate $P(a)$ for a as argument true?"; of what-questions, to "What is the value b of the number-theoretic function $f(a)$ for a as argument?".

In cases when the class of questions is already given by using a fixed number of variables with suitable ranges, it is often more convenient to render it directly by a number-theoretic predicate or function. For example, "Does a divide b?" can be rendered by a predicate $P(a, b)$, which indeed is commonly written "$a|b$". To standardize in the following treatment, we take our variables $a, b, c, \ldots, x, y, z$ to range over the natural numbers. For the present example, we can extend the "divides" notion to include 0 ($a|0$ is true for every a; $0|b$ is false except for $b = 0$). "What is the sum of a and b?" is obviously rendered by the function $a+b$.

Inversely, starting with any number-theoretic predicate $P(a_1, \ldots, a_n)$, we have the countably infinite class of questions "For given $a_1, \ldots, a_n$, is $P(a_1, \ldots, a_n)$ true?". Starting with any number-theoretic function $f(a_1, \ldots, a_n)$, we have the questions "For given $a_1, \ldots, a_n$, what is the value of $f(a_1, \ldots, a_n)$?".

Thus it comes to the same thing whether we talk about countably infinite classes of questions or about number-theoretic predicates or functions. If there is a decision procedure for a predicate (or for the class of questions arising from it), we call the predicate (or class of questions) *decidable*. Similarly, if there is a computation procedure for a function, we call the function *computable*.

Furthermore, the case of predicates can be reduced to the case of functions by defining the *representing function* $f(a_1, \ldots, a_n)$ of a predicate $P(a_1, \ldots, a_n)$, thus:

$$f(a_1, \ldots, a_n) = \begin{cases} 0 \text{ if } P(a_1, \ldots, a_n) \text{ is true,} \\ 1 \text{ if } P(a_1, \ldots, a_n) \text{ is false.} \end{cases}$$

In the one-variable case, $f(a)$ is what in § 33 we called the "representing function" of the set of a's for which $P(a)$ is true (in symbols, the set $\hat{a}P(a)$). It comes to the same thing to compute the value of $f(a_1, \ldots, a_n)$ and observe whether it is 0 or 1 as to decide whether $P(a_1, \ldots, a_n)$ is true

or false. In other words, a decision procedure can be handled as a computation procedure by taking 0 for "yes" and 1 for "no".[164]

To emphasize the idea of an algorithm by contrast, we consider some more cases in which it is not clear that there is one. We have already mentioned that there is an algorithm for "Does $ax + by + c = 0$ have a solution in integers?", or written as a predicate $(Ex)(Ey)[ax + by + c = 0]$. Here we are using "$(Ex)$" and "$(Ey)$" in the same meaning as $\exists x$ and $\exists y$ in Chapter II; but we prefer henceforth to keep the logical symbolism we use outside a given formal system distinct from the symbolism we use inside, as far as is feasable.[165] The existence of this algorithm is not obvious from the start (unlike those for $a|b$ and $a+b$). Some theory due to Euclid is used in setting it up.

Let us generalize this example to consider, instead of the first-degree equation $ax + by + c = 0$, the second-degree equation $ax^2 + bxy + cy^2 + dx + ey + f = 0$ (a, b, c not all 0) or even any algebraic equation (polynomial equation) of any degree $n > 0$ in any number $m > 0$ of variables $x_1, \ldots, x_m$, and make our class of questions "Does any given algebraic equation with integeral coefficients have a solution for its variables in integers?". The decision problem for this class of questions is "Hilbert's tenth problem", included in his famous list of 23 "future" problems of mathematics 1900a. It has not been solved.

To make matters as simple as possible, consider some 2-place predicate $P(a, x)$ for which we do have an algorithm. It will not directly follow that there is an algorithm for $(Ex)P(a, x)$. The only procedure the *definition* of this predicate suggests *directly* is, after choosing a value of a, to start trying $P(a, 0)$, $P(a, 1)$, $P(a, 2)$, . . . in the hope of finding one which is true. We can't be sure we will by this procedure find out in a finite number of steps whether $(Ex)P(a, x)$ is true or not. For (as we already noted for "$\exists x P(x)$" in § 36), if after any finite number of steps we haven't already found an x which makes $P(a, x)$ true, we won't know whether it's because

[164] Since we are concerned here only with the questions whether the propositions taken as values of the predicates are true or false, the predicates in this theory are treated extensionally, i.e. not distinguished from the corresponding logical functions, which become their representing functions on changing t, f to 0, 1.[92]

Below we emphasize that sometimes the definition of a predicate or function does not directly give an algorithm for it, but an algorithm may be afforded by some theory about the function or predicate. For predicates, if we restore the intensional concept of them, we could then say that an (intensional) predicate P may be undecidable while an equivalent predicate P_1 is decidable.

[165] Beginning with this chapter (and in a few instances before)[23] we use the following informal logical symbolism: $\equiv$ ("equivalent"), $\rightarrow$ ("implies"), & ("and"), $\vee$ ("or"), $\bar{}$ (thus $\bar{A}$; "not"), (x) ("for all x"), (Ex) ("(there) exists (an) x (such that)")

$(Ex)P(a, x)$ is false for that a, or because we haven't gone far enough in our search.

To give this example in terms of number-theoretic functions rather than predicates (i.e. propositional functions), suppose we have a computation procedure for $f(a, x)$. We do not at once know whether there is one for the function

$$f(a) = \begin{cases} 0 \text{ if } (Ex)f(a, x)=0, \\ 1 \text{ otherwise.} \end{cases}$$

If $f(a, x)$ is the representing function of $P(a, x)$, then $f(a)$ is the representing function of $(Ex)P(a, x)$.

The recognition that for some (countably infinite) classes of mathematical questions we have algorithms, and that for others we at least don't know any, goes far back in mathematical history. The Greeks, including Euclid, sought algorithms; and the term "algorithm" is derived from the name of the ninth century Arabian arithmetician al-Khuwarizmi.

To recapitulate, we have seen that computation and decision problems can all be reduced to the computation problems for number-theoretic functions.

So to advance toward our objective (Church's theorem), we have to deal with the question: For what number-theoretic functions are there computation procedures (or algorithms)? Briefly: What is the class of "computable" functions?

What we have to go on in considering this question is a somewhat vague intuitive idea of what constitutes a computation procedure. A computation procedure may consist in essentially a direct application of the definition of the function, or it may consist in something considerably different which mathematical theory shows must lead to the same function values as the original definition requires.

Although our intuitive notion of a computation procedure is vague, it is nevertheless real, as is shown by two circumstances. First, it does not leave mathematicians in any doubt or disagreement that they do have computation procedures for many particular functions, e.g. $a+1$, $a+b$, $a \cdot b$, a^b, $a!$, $\max(a, b)$ (= the maximum of a and b, i.e. the greater if $a \neq b$, the common value if $a = b$), $\min(a, b)$ (= the minimum of a and b), $a \dot{-} 1$ (= $a-1$ if $a \geq 1$, 0 if $a = 0$), $a \dot{-} b$ (= $a-b$ if $a \geq b$, 0 if $a < b$), $[a/b]$ (= the quotient when a is divided by b if $b \neq 0$, 0 if $b = 0$), $[\sqrt{a}]$ (= the greatest natural number whose square is $\leq a$), $[e^a]$ (= the greatest natural number $\leq e^a$), etc. Second, there is likewise no doubt, in other particular cases, that the definition of a function or a given equivalent of its definition does not directly give a computation procedure. For

example, we agree that the definition of $f(a)$ displayed above does not give a computation procedure for it.

We are only recapitulating what has not been disputed among mathematicians for over two millenia (if we correctly understand mathematical history).

This intuitive notion of a computation procedure, which is real enough to separate many cases where we know we do have a computation procedure before us from many others where we know we don't have one before us, is vague when we try to extract from it a picture of the totality of all possible computable functions. And we must have such a picture, in exact terms, before we can hope to prove that there is no computation procedure at all for a certain function, or briefly to prove that a certain function is uncomputable. Something more is needed for this.

Most mathematical logicians agree that it was found in 1935 (and published in 1936), as we shall see in the next section. —

Hereafter, we may say that something can be done "effectively", or that an operation or process is "effective", as a brief way of saying that there is an algorithm for it (i.e. a decision or computation procedure).

EXERCISES. 40.1. For the following predicates and functions, or classes of questions, do we have an algorithm or do we lack such (at least without more knowledge)? For (a) and (b), assume $P(a, x)$ decidable.

(a) $f(a) = \begin{cases} \text{the least } x \text{ such that } P(a, x) \text{ if } (Ex)P(a, x), \\ 0 \text{ otherwise.} \end{cases}$

(b) $f(a, b) = \begin{cases} \text{the least } x \leq b \text{ such that } P(a, x) \text{ if } (Ex)[x \leq b \ \& \ P(a, x)], \\ b+1 \text{ otherwise.} \end{cases}$

(c) Is a a prime number?

(d) What is the nth prime number? (Assume Euclid's theorem that there are infinitely many primes, § 38.)

(e) $f(a) = \begin{cases} a+1 \text{ if Fermat's "last theorem" is true,} \\ a \text{ if Fermat's "last theorem" is false.} \end{cases}$

(f) For given formulas $A_1, \ldots, A_m$, B in the propositional calculus, does $A_1, \ldots, A_m \vdash B$ hold in the propositional calculus?

(g) For given formulas $A_1, \ldots, A_m$, B in the predicate calculus, does $A_1, \ldots, A_m \vdash B$ hold in the predicate calculus?

(h) Given formulas $A_1, \ldots, A_m$ in the predicate calculus, is a given finite sequence of formulas a deduction from $A_1, \ldots, A_m$ in the predicate calculus holding are all variables constant?

(i) For a given finite domain D and a given formula E in the predicate calculus, does $\bar{\bar{D}}\text{-}\vdash E$ hold? (Cf. § 17).

(j) For a given formula E in the predicate calculus, does $\vdash E$ hold?

(k) (For calculus students.) Does a given elementary function (say one

composed from rational numbers and the variable x using finitely many times addition, multiplication, division, powers, roots, and trigonometric and exponential functions and their inverses) have an indefinite integral which is elementary?

(‡) (For algebra students.) Does a given system of m linear equations in n unknowns with integral coefficients have a solution?

40.2. Show that $[\sqrt{a}]$ is computable.

40.3*. (For calculus students.) Show that $[e^a]$ is computable. (Use the result of Hermite in 1873 that e is transcendental; cf. § 33.)

§ 41. Turing machines, Church's thesis. The situation in 1935 was that a certain exactly defined class of computable number-theoretic functions considered by Church and Kleene during 1932–35, called the "λ-definable functions", had been found to have properties strongly suggesting that it might embrace all functions which can be regarded as computable under our vague intuitive notion. This result was somewhat unexpected, since initially it was not clear that the class contained even the particular computable function $a \dot{-} 1$ mentioned above, and a proof in 1932 (publ. 1935) that it did was the present author's first piece of mathematical research. Another class of computable functions, called the "general recursive functions", defined by Gödel in 1934 building on a suggestion of Herbrand, had similar properties. It was proved by Church 1936 and Kleene 1936a that the two classes are the same, i.e. each λ-definable function is general recursive and vice versa.

Under these circumstances Church proposed the thesis (published in 1936) that all functions which intuitively we can regard as computable, or in his words "effectively calculable", are λ-definable, or equivalently general recursive. This is a thesis rather than a theorem, in as much as it proposes to identify a somewhat vague intuitive concept with a concept phrased in exact mathematical terms, and thus is not susceptible of proof. But very strong evidence was adduced by Church, and subsequently by others, in support of the thesis.

A little later but independently, Turing's paper 1936–7 appeared in which another exactly defined class of intuitively computable functions, which we shall call the "Turing computable functions", was introduced, and the same claim was made for this class; this claim we call *Turing's thesis*. It was shortly shown by Turing 1937 that his computable functions are the same as the λ-definable functions, and hence the same as the general recursive functions. So Turing's and Church's theses are equivalent. We shall usually refer to them both as *Church's thesis*, or in connection with that one of its three versions which deals with "Turing machines" as *the Church-Turing thesis*. Post in 1936, independently of Turing, published

rather briefly a formulation fundamentally the same as Turing's. In 1943 he published a fourth equivalent, using ideas from unpublished work of his in 1920–22.[166] Still another equivalent formulation is provided by Markov's theory of algorithms 1951c.

Turing's machine concept arises from a direct effort to analyze computation procedures as we know them intuitively into elementary operations. Turing argued that repetitions of his elementary operations would suffice for any possible computation. For this reason, Turing computability suggests the thesis more immediately than the other equivalent notions, and so we choose it for our exposition.

Turing described a kind of theoretical computing machine. This differs in two respects from a human computer working under preassigned instructions or an actual *digital* computing machine (such as a desk calculator, or a high-speed computer with electronic tubes or transistors). These are respects in which we idealize by divesting the human computers and the physical machines of their practical limitations.

First, a "Turing machine" is not liable to errors; i.e. it obeys the intended laws for its action without any deviations.

Second, a "Turing machine" is given a potentially infinite memory. That is, although the amount of information stored at any one time is finite, there is no upper bound on this amount. Information to be stored may include (at one time or another) the statement of the particular question given the machine, the scratch work performed by the machine while arriving at the answer, and the answer. To allow for unlimited storage of such information, we consider as separate the machine proper and a peripheral storage facility, which we will take to be an infinite "tape".

The machine proper, which does the computing and thus determines what function is computed, admits only a fixed finite number of possible "states". It represents the finite list of rules or finite description of a procedure in our intuitive notion of an algorithm, § 40. (Also information, but only up to a fixed amount, can be stored momentarily by the machine's assuming one or another of its "states".)

Now we formulate our notion of a *Turing machine* in detail. We number moments of time for the operation of the machine as 0, 1, 2, ... At any given moment, the machine shall be in one of $k + 1$ *states*, which we number 0, 1, . . . , k. *State* 0 we call the *passive state*; the others, *active states*. A *linear tape* ruled in *squares* passes through the machine (when it is set up for operation). The tape is potentially infinite to the right. Each square is either *blank* s_0 or has printed on it one of a given finite list of *symbols* $s_1, \ldots, s_j$; so $s_0, \ldots, s_j$ are the possible *square*

[166] An account (from 1941) of this work of Post in the 1920's was published posthumously in Davis 1965 pp. 338–433.

conditions. However, only a finite number of squares are printed at any moment in the use of the machine. At each moment, beginning with Moment 0, one square of the tape is *scanned* by the machine.

Now consider any moment when the machine is in one of its active states $1, \ldots, k$. Between this moment and the next, the machine performs an *act* consisting of a sequence of three operations (a), (b), (c), each operation coming from the respective category as follows: (a) print one of the symbols $s_1, \ldots, s_j$ on the scanned square (supposed blank at the given moment), or erase the scanned square (supposed printed at the given moment), or erase and print one of $s_1, \ldots, s_j$ on the scanned square (supposed printed at the given moment), or make no change in the scanned square; (b) move the tape so as at the next moment to scan the square next to the left of the scanned square (briefly, move left), or leave the tape unmoved (briefly, stay centered), or move the tape so as at the next moment to scan the square next to the right of the scanned square (briefly, move right); (c) change to another state, or remain in the same state. What act (within these possibilities) is performed, between a given moment at which the machine state is active (one of $1, \ldots, k$) and the next moment, is determined by the machine state and the scanned-square condition (one of $s_0, \ldots, s_j$) at the given moment, with one exception to be explained presently. We call the machine state together with the scanned-square condition at a given moment the *configuration.* In contrast, the machine state together with the specification of the scanned square and the entire printing on the tape we call the (*machine vs. tape*) *situation.*

The exceptional case in which the configuration at the given moment does not determine the act is that the configuration would call for a motion left when the scanned square is already the leftmost square of the tape. Then the (b) and (c) parts of the act are altered to: stay centered, and assume the passive state (briefly, stop). The machine "jams". We could avoid this exception by assuming a 2-way infinite tape.[167]

If at a given moment the machine is in the passive state 0, no act is performed between this and the next moment, i.e. the machine does not print or erase, does not move, and does not change from state 0.

We shall give an illustration of the operation of a Turing machine presently. However, let us first define how such a machine is to be used to compute a number-theoretic function. (Turing used his machines primarily to carry out continuing computations of decimal fractions for real

[167] We begin thus in IM p. 357; but the results are essentially the same [168,170]

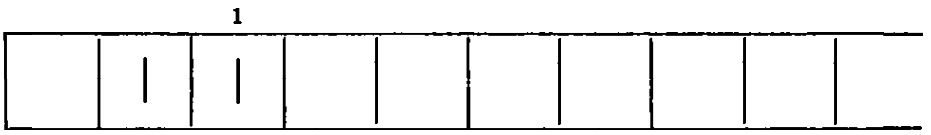
Our discussions of Turing machines (beginning with one in our seminar on the foundations of mathematics at Wisconsin in 1941) follow Turing 1936–7 in the general conception of the behavior of the machines, but not in the detailed formulation and development. Cf. IM top p. 361.

numbers.) For this purpose, we must agree how the argument(s) (i.e. value(s) of the independent variable(s)) are to be represented on the tape, and how the machine is to give us the resulting value of the function. We shall make the supposition that all machines to be considered have among their symbols the tally mark "|"; say it is s_1. We shall represent natural numbers by sequences of tallies, "|" for 0, "||" for 1, "|||" for 2, ... To set up the machine and tape to compute for a given argument a, we shall arrange that: at the moment 0 the system consisting of machine and tape is started off so that the leftmost square of the tape is blank, a is represented by tallies on the next $a+1$ squares, all squares to the right of these are blank, the machine is scanning the rightmost printed square, and is in its first active state 1. In this situation, we say the machine is *applied to a as argument*. We say the machine *computes a value c for a as argument*, if, starting from this situation at Moment 0, the machine at some later moment assumes the passive state 0 ("stops") with a blank and $c+1$ tallies printed on the tape after the $a+1$ tallies representing the argument a, the tape being otherwise blank, and the rightmost printed square being again the scanned square.[168]

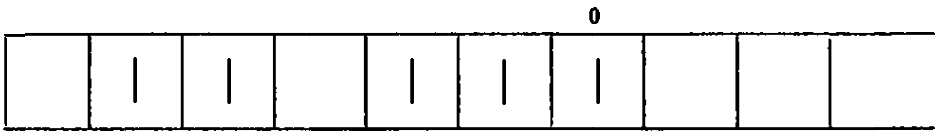
A given machine may compute a value for each natural number a as argument, or for some a's but for others, or for no a's. If, for each a, it computes a value c where $c = f(a)$, we say that the machine *computes* the function $f(a)$, and that $f(a)$ is *Turing computable*.

Similarly, for functions of more than one variable.

For example, a machine is applied to 1 as argument, if at Moment 0 it is in the following situation



where the "1" written over the third square shows that this is the scanned square and the machine is in state 1, and all squares to the right of those shown are blank. A machine computes the value 2 for 1 as argument if, having been stated at Moment 0 in the situation above, it will at some later moment x reach the situation



where again all squares to the right of those shown are blank. If in similar fashion, when the machine is started with any $a+1$ tallies on the

[168] In this case, the machine cannot during the computation attempt a motion leftward from the leftmost tape square; for, in that case State 0 would be assumed without the last of two successions of tallies separated by a blank being scanned.

tape, it will eventually stop with these followed by a blank and $a+2$ tallies (the last of them scanned), the machine computes the function $f(a) = a+1$; the above illustrates this for $a = 1$.

Now we shall describe a machine $\mathfrak{S}$ which does compute this function $f(a) = a+1$ (the successor function), and we shall follow it through its acts in computing for the argument $a = 1$. The machine will have only the one symbol "|". To save drawing a picture of the tape each time, we shall use sequences of numerals "0" and "1" where a "0" stands for a blank square and a "1" for a square printed with tally. Thus in the illustration above we would write the initial situation

$$0 \quad 1 \quad 1^1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \ldots,$$

and the situation at the moment the computation is completed

$$0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1^0 \quad 0 \quad 0 \ldots.$$

To describe a machine, we need merely tell for each of its active states $1, \ldots, k$ and each of the $j+1$ scanned-square conditions (blank s_0 or printed with $s_1, \ldots, s_j$) what act it is to perform. For the machine $\mathfrak{S}$ we describe now, there are to be 11 active states, and 2 conditions of the scanned square, namely blank and printed with "|" (or as we write them more conveniently, "0" and "1"). The acts it is to perform for each of the configurations of these 11 active states and 2 scanned-square conditions can be shown by the machine table at the left below (top p. 237). In the table "*P*" means "print", "*E*" means "erase"; "*L*", "*C*", "*R*" mean "left", "center" (i.e. don't move), "right"; and the number at the end of each table entry is the state the machine is to assume at the succeeding moment.

At the right, we follow out the acts of $\mathfrak{S}$ in computing $a+1$ for the argument $a = 1$. At Moment 0, in the sample computation at the right, we see that a printed square (i.e. a 1) is scanned in state 1; so we enter the machine table in the first row and second column finding "*R*2", i.e. go right and assume State 2. The resulting situation is shown in the sample computation opposite Moment 1. Now we have a blank square scanned in State 2, so we enter the table in the second row and first column, finding "*R*3", i.e. go right and assume State 3. The result is shown opposite Moment 2. Now the scanned square is still blank but the state is 3, so by the table (third row, first column, where it reads "*PL*4"), the machine prints a "|", goes left, and assumes State 4, with result shown at Moment 3. Continuing, we find that at Moment 23 the machine has computed the desired value 2 ($= a+1$ for $a = 1$) shown by three tallies. To see that the machine computes $f(a) = a+1$, we must convince ourselves that it will compute the value $a+1$ for every value of a as argument. We have done

Table for Machine $\mathfrak{S}$ which computes $f(a) = a+1$

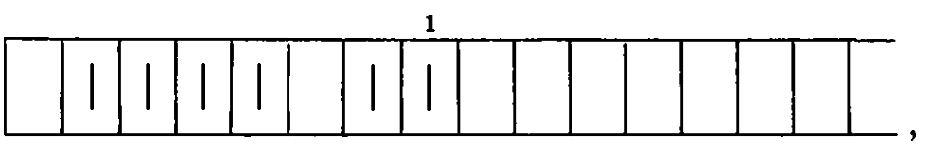
Machine state	Scanned square condition	
	0	1
1	*C*0	*R*2
2	*R*3	*R*9
3	*PL*4	*R*3
4	*L*5	*L*4
5	*L*5	*L*6
6	*R*2	*R*7
7	*R*8	*ER*7
8	*R*8	*R*3
9	*PR*9	*L*10
10	*C*0	*ER*11
11	*PC*0	*R*11

Computation by Machine $\mathfrak{S}$ for $a = 1$

Moment	Machine vs. tape situation						
0	0	1	1^1	0	0	0	0
1	0	1	1	0^2	0	0	0
2	0	1	1	0	0^3	0	0
3	0	1	1	0^4	1	0	0
4	0	1	1^5	0	1	0	0
5	0	1^6	1	0	1	0	0
6	0	1	1^7	0	1	0	0
7	0	1	0	0^7	1	0	0
8	0	1	0	0	1^8	0	0
9	0	1	0	0	1	0^3	0
10	0	1	0	0	1^4	1	0
11	0	1	0	0^4	1	1	0
12	0	1	0^5	0	1	1	0
13	0	1^5	0	0	1	1	0
14	0^6	1	0	0	1	1	0
15	0	1^2	0	0	1	1	0
16	0	1	0^9	0	1	1	0
17	0	1	1	0^9	1	1	0
18	0	1	1	1	1^9	1	0
19	0	1	1	1^{10}	1	1	0
20	0	1	1	0	1^{11}	1	0
21	0	1	1	0	1	1^{11}	0
22	0	1	1	0	1	1	0^{11}
23	0	1	1	0	1	1	1^0
24	0	1	1	0	1	1	1^0

this only for $a = 1$; but the reader should not find it too hard to get the "hang" of how this machine operates to see that it does so for every a.

To illustrate computation of a function of 2 variables, a machine to compute $f(a, b) = a+b$, when started (for $a = 3$, $b = 1$) in the situation



should later stop (i.e. assume State 0) in the situation

The machine that computes $a+1$ is sufficiently complicated so the reader may well wonder how to find machines to compute complicated effectively calculable functions. Of course we are only interested *here* in the theoretical possibility of finding a machine to compute any given effectively calculable function, and not in whether the machine operates efficiently. The business of finding machines can be systematized by starting from the theory of recursive functions.[169] This theory deals with recursive definitions of functions, such as

$$\begin{cases} a+0 = a, \\ a+b' = (a+b)', \end{cases} \quad \begin{cases} a\cdot 0 = 0, \\ a\cdot b' = a\cdot b+a, \end{cases} \quad \begin{cases} a^0 = 1, \\ a^{b'} = a^b\cdot a. \end{cases}$$

How these definitions define the functions was illustrated in § 38. The functions commonly used in number theory are definable by use of such recursions, and proceeding from the recursive definitions one can in a systematic way find corresponding Turing machines, after first setting up Turing machines for such simple operations as filling in with tallies all but the rightmost of a sequence of blank squares preceded and followed by a tally, copying a sequence of tallies, etc.

In giving the arguments to the machines and receiving the values, we have used only one symbol "|" besides the blank (i.e. two square conditions "0" and "1"), and likewise in all the action of Machine $\mathfrak{S}$. The definition of "Turing machine" allows more symbols. To write a *machine table* for a machine with j symbols $\mathrm{s}_1, \ldots, \mathrm{s}_j$, there will be $j+1$ columns. In the case $j > 1$, "P" is ambiguous; then in place of "P" or "E" or the absence of either we can write i in decimal notation to indicate that the square condition at the next moment is to be s_i $(0 \leq i \leq j)$.

While we are thus leaving open the possibility of using $j+1 > 2$ square conditions instead of only 2, the fact is that we get no larger class of computable functions thereby.[170]

It may seem a little anomalous that, after we have argued that any intuitive computation should be performable using only such operations as a Turing machine admits, we should find it quite an exercise to see that a simple function like $a+1$ is Turing computable. However we have

[169] E.g. IM Part III. The three definitions shown next are "primitive recursions" (IM Chapter IX); but the theory extends to more complicated sorts of recursions, all defining "general recursive functions" as already mentioned (IM Chapter XI).

[170] This is shown in IM Chapter XIII, by using only the one symbol | in proving that every general recursive function is Turing computable, while allowing j symbols $\mathrm{s}_1, \ldots, \mathrm{s}_j$ for any $j \geq 1$ in proving that every Turing computable function is general recursive.

The same method shows that we could have allowed a 2-way infinite tape without getting more (or less) computable functions.

started at the very beginning, and we have done it with a Turing machine with only one symbol "|" (besides the blank), though we didn't need to. Use of more symbols would make the Turing machine action better resemble informal computation.

Moreover, our illustration may unduly suggest that the computer is restricted to take an ant's eye view of his work, squinting at one square at a time. However, when more than the one symbol "|" (besides the blank) is allowed, it is possible to interpret what we mean by a symbol liberally. There is no reason why individual tape squares cannot be considered to correspond to whole sheets of paper, each ruled into finitely many squares each of which may receive one of finitely many primary symbols. What can be written on the whole sheet can then be construed as a single symbol for the Turing machine. If the sheets are ruled into 20 columns and 30 lines and 100 primary symbols are allowed, there are 600^{101} square conditions $s_0, \ldots, s_j$ ($j = 600^{101} - 1$), and we are at the opposite extreme. The schoolboy doing arithmetic on $8\frac{1}{2}'' \times 11''$ sheets of ruled paper would never need all this variety. He would usually operate just by changing the condition of the sheet of paper before him (= the scanned square), only occasionally transferring figures from one sheet to the next or preceding sheet in his stack of paper (= moving right or left).

From this point of view, the tape square represents whatever one looks at, at a given moment, which may not be small.

Actually the schoolboy at a given moment would directly perceive only a part of what appears on the sheet of paper; the rest, to the extent it affects his act from that moment to the next, he must remember in the form of a state of mind. Thus, psychologically, it is something between the small squares on the paper and the whole sheet that plays the role of the Turing machine tape square.

Another representation of the Turing machine tape is a stack of IBM cards, each card constituting a single square for the Turing machine.

Returning to the schoolboy, it is necessary for our concept that the paper be ruled in squares (at least imaginary squares), and that the symbols be from a given finite list, so as to make it exact what the whole finite range of possibilities is. That is, there must be only finitely many possible conditions for a sheet of paper or a Turing machine tape square. Likewise, there are to be only finitely many states of mind for the human computer or machine states for a given Turing machine. As Turing wrote, "the number of states of mind which need to be taken into account is finite. . . . If we admitted an infinity of states of mind, some of them will be 'arbitrarily close' and will be confused." The finite numbers of square conditions and of states of mind can of course be very large. We are

considering *digital* computation, where the data are discrete. This is in contrast to *analog* computation, such as is performed by a slide rule or a differential analyzer, where the data are positions on a scale or scales, representing real numbers only to within some limit of accuracy. The inputs for our computations are to be natural numbers, or they could be something similar, like words in an ordinary language, or formal expressions in a formal system; likewise, the output. Discreteness must be preserved throughout the whole computation. At each moment, the existing (finite) configuration, comprising both the condition of the paper or the tape square, and the state of mind of the computer or the machine state, must completely determine the next configuration. Only thus will the computation lead to a discrete result, not to something approximate, and to such a result that is fully predetermined (when it exists at all) by the initial situation and the computer's instructions or the machine table.

There is however an intrinsic difficulty that our theory of computation must meet. This arises because we define a function $f(a)$ to be computable exactly if there is a machine which will compute its value for *all* choices of a. Thus we must be prepared to carry out computations for arbitrarily big values of a. The schoolboy computing on sheets of paper is not normally required to work with numbers too big to go on one of his sheets. If he had to do this, he also would then be forced to work in a groping manner as our machine $\mathfrak{S}$ does in computing values of $a+1$.

For further discussion of Church's thesis, or the Church-Turing thesis, we refer to the literature.[171] In the next sections, we deal with consequences of it.

[171] In IM the attempt is made to summarize all the evidence for Church's thesis: § 62 gives a general summary, supplemented by p. 352; and § 70 gives the part of the evidence which pertains to Turing's machines, supplementing the present discussion.

The following argument is given in IM, concluding on p. 352. A great stock of intuitively computable functions (all that have been investigated in this connection) are known to be Turing computable. Likewise, we have a great stock of methods or operations (for obtaining new intuitively computable functions from others) which are paralleled by operations for building new Turing machines from given Turing machines (or the analog in terms of recursiveness). If there were a function which is intuitively computable but not Turing computable, it would have to be "inaccessible" by any process of building up toward it from this stock of functions and operations already mastered. IM § 66 gives a theorem (the RECURSION THEOREM, Kleene 1938) applicable to the building process in general. It is hard to imagine how one could give a description, or a set of instructions, for a computation procedure that a human computer could follow, except by putting the description together out of simpler elements, already known, and then it would come under this theorem.

For an argument against Church's thesis and a reply, see Kalmár 1959 and Mendelson 1963. In reading such discussions, we must not lose sight of our intuitive concept of an algorithm or computation procedure (§ 40). An algorithm in our sense must be fully

EXERCISES. 41.1. (a) Write out the computation by Machine $\mathfrak{S}$ when the argument is 0, i.e. starting from the situation $0\ 1^1\ 0\ 0\ 0\ldots$.
(b) Explain the operation of Machine $\mathfrak{S}$ so as to make it clear that $\mathfrak{S}$ does compute the function $a+1$.

41.2. Construct the table for a machine $\mathfrak{E}$ which decides whether the argument a is even; i.e. $\mathfrak{E}$ is to compute the function

$$f(a) = \begin{cases} 0 \text{ if } a \text{ is even,} \\ 1 \text{ if } a \text{ is odd.} \end{cases}$$

41.3. Modify the table for the machine $\mathfrak{S}$ to obtain:
(a) a machine $\mathfrak{I}$ which computes the identity function $f(a) = a$ (i.e. $\mathfrak{I}$ just copies a).
(b) a machine $\mathfrak{P}$ which computes the predecessor function $f(a) = a \dot{-} 1$ (end § 40).

41.4*. Show that, if $f(a)$ and $g(a)$ are Turing computable, so is the function $h(a) = f(g(a))$.

41.5. Show that no function $f(a)$ would be Turing computable under the above definition modified to allow only a finite tape in the machine.

41.6*. Show that no more functions $f(a)$ would be Turing computable under the above definition modified so as not to require the scratch work to be erased.

---

and finitely described before any particular question to which it is applied is selected. When the question has been selected, all steps must then be predetermined and performable without any exercise of ingenuity or mathematical invention by the person doing the computing. None of the skeptics about Church's thesis has come forward with a plausible suggestion of what an algorithm (in our sense) which would not be mechanizable in Turing's manner could be like. (Of course, Church's thesis would be disproved, if someone should describe a particular function, authenticated unmistakably as "effectively calculable" by our intuitive concept, but demonstrably not general recursive.)

We have been assuming without close examination the CONVERSE OF CHURCH'S THESIS: *If a function is Turing computable (or general recursive, or λ-definable), then it is intuitively computable (or effectively calculable).* In defending this implication to an intuitionist, or to any other kind of constructivist who considers an algorithm to exist only when it is proved by his standards that it always works, we only ask him to accept the following: if the hypothesis that a function is Turing computable holds by his standards, so does the conclusion. Put thus, it is hard to see how it can be questioned. Only if one allows a nonconstructive interpretation of the hypothesis, and yet insists on a constructive interpretation of the conclusion, is the converse of Church's thesis in doubt. ("Church's thesis" is sometimes understood to include this converse, as indeed Church 1936 maintained both implications in proposing to identify the effectively calculable functions with the general recursive functions or the λ-definable functions.)

§ 42. Church's theorem (via Turing machines).[172] We have seen that the pattern of behavior of a given Turing machine is determined by the table for it; if we know the table, essentially we know the machine.

As we remarked in § 41, the use of "*P*" for "print" is ambiguous when there is more than just the one symbol "|" or s_1 ($j = 1$). We now write the table entries using the other method explained there, which is suitable for any number $j \geq 1$ of symbols $s_1, \ldots, s_j$. Under this method, a table entry "7*R*3" would mean that the square scanned at the given moment t is at the next moment $t+1$ to have Condition s_7 (i.e. it is to have the symbol s_7 printed on it), and (as before) the machine is to shift the tape between Moments t and $t+1$ so that at Moment $t+1$ the scanned square is next to the right of the square scanned at Moment t, and at Moment $t+1$ the machine is to be in its state numbered 3. This table entry could appear only in the table for a machine with at least 7 symbols and at least 3 active states.

Rewriting the table for our machine $\mathfrak{S}$ in this manner, it becomes:

Machine state	Scanned square condition	
	0	1
1	0*C*0	1*R*2
2	0*R*3	1*R*9
3	1*L*4	1*R*3
	. . .	
10	0*C*0	0*R*11
11	1*C*0	1*R*11

The table for a machine can be written in code form. Consider the table for $\mathfrak{S}$, originally given in § 41 and repeated now. Using the latter way of writing it, let us insert semicolons at the end of each row of entries, and commas separating entries within a row, and then string the entire body of the table along as one sequence of symbols:

$$0C0,1R2;0R3,1R9;1L4,1R3; \ldots ;0C0,0R11;1C0,1R11$$

This sequence of symbols is the *code* for the machine $\mathfrak{S}$.

The code for any machine can thus be written on a typewriter with the following 15 symbols:

$$L\ C\ R\ ,\ ;\ 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9$$

[172] In this and the next two sections we follow the treatment in Kleene 1958 pp. 145–147 (some of which appeared earlier in 1956a, 1957b).

Such a code does not begin with the symbol L. By reinterpreting these symbols as the digits of a number in the number system based on 15, we get a positive integer which describes the machine table and thence the pattern of behavior of the machine; call this number the *index* of the machine.[173]

Now let $T(i, a, x)$ stand for the following:

> i is the index of a Turing machine (call it "Machine $\mathfrak{M}_i$") which, when applied to a as argument, will at Moment x (but not earlier) have completed the computation of a value (call that value "$\varphi_i(a)$").

This predicate (i.e. propositional function) $T(i, a, x)$ is "decidable". For suppose values of i, a, x are given. Then we can determine whether i, in the 15-system of notation, does describe the table for a machine. If it does not, $T(i, a, x)$ is false. If it does, then we can follow out the operations performed by that machine $\mathfrak{M}_i$ (as in the illustration in § 41), starting it at Moment 0 to compute for a as argument, and continuing to Moment x. Finally, in this case, we can see whether at this moment $\mathfrak{M}_i$ has *just* completed the computation of a value. If so, $T(i, a, x)$ is true; if not, false. (For example, if i is the number shown above in the 15-notation, then $T(i, 1, 23)$ is true, but $T(i, 1, x)$ is false for every $x \neq 23$.)

This should make it clear that $T(i, a, x)$ is decidable in the vague intuitive sense (§ 40). It is then implied by Church's thesis, or the Church-Turing thesis (§ 41), that it is (*Turing*) *decidable* in the strict sense that a Turing machine exists which decides it, i.e. computes its representing function $\tau(i, a, x)$, whose value is 0 when $T(i, a, x)$ is true and 1 when $T(i, a, x)$ is false (§ 40). A full treatment of the subject would call for showing this without appeal to Church's thesis. One would not do this from scratch, but by taking advantage of theory developed for this purpose, and alluded to in § 41. In this way we would establish the Part (A) of the following theorem. (In this chapter we are only giving a survey, in which sometimes we will only be able to describe the considerations on which the full proofs are based.)

[173] We are using the "method of digits" end § 32 (and IM § 1), except not bothering to close up the gaps in the resulting numbers (which there is no need for us to do for our present purposes).

We have chosen this particular method of indexing, in expositions since 1956a, as being easy to explain. If we were to deal now with the subjects below in greater detail, it would be advantageous to select a method to make the work as easy as possible. Some other methods have been more commonly used in the literature. But work done in Smullyan 1961 (not quite in the present connection) establishes that the present system is no harder to use.

THEOREM I. (A) *The predicate* $T(i, a, x)$ *is decidable.*
(B) $\varphi_i(a)$ *as a partial function of* i *and* a *is computable.*

To explain Part (B) of the theorem, first observe that the "quantity" $\varphi_i(a)$ in the definition of $T(i, a, x)$ is not defined for every i and a; indeed it is defined, for a given i and a, exactly if there exists an x such that $T(i, a, x)$, or in symbols exactly if $(Ex)T(i, a, x)$. In Chapter II we would have written this "$\exists x T(i, a, x)$", but here we prefer to save "$\exists$x" for use in formal systems, while employing "(Ex)" informally in discussing those systems.[165]

For a value of i which is the index of a Turing machine that computes a one-place number-theoretic function, φ_i is the function computed. Such values of i are described by $(a)(Ex)T(i, a, x)$, where "(a)" means "for all a".[165]

As a function of i and a both, $\varphi_i(a)$ as we remarked is defined exactly if $(Ex)T(i, a, x)$. So it is a partially defined number-theoretic function of two variables i and a, or briefly a *partial function.*

However, for i and a for which it is defined, we can find its value, thus: given such i and a, from i we find the table for Machine $\mathfrak{M}_i$, then we apply (or imitate) $\mathfrak{M}_i$ by performing its steps, starting at Moment 0 with a as argument, up to the moment x for which $T(i, a, x)$ is true, and finally we read from the resulting situation the value computed. This process is an algorithm or decision procedure in the sense of § 40; here the countably infinite class of questions is "What is the value of $\varphi_i(a)$?" where (i, a) ranges, not over all pairs of natural numbers, but over exactly those pairs for which $(Ex)T(i, a, x)$.

We extend the definition given in § 41 for "total" number-theoretic functions to say now that a partial number-theoretic function of two variables is *computed* by a Turing machine, if that machine computes the value of the function for just those pairs of arguments for which the function is defined, and computes no value for other pairs of arguments. Similarly, for n-place partial functions for any n. The Church-Turing thesis applies to partial functions on the same grounds as to total functions (§ 41).[174]

Via this extension of the thesis, it follows from our having an algorithm for finding the values of $\varphi_i(a)$ that there is a machine $\mathfrak{U}$ which computes $\varphi_i(a)$ as a partial function of i and a. Turing showed directly, without appeal to the thesis, that such a machine $\mathfrak{U}$ exists, though in a little different situation (he dealt with decimal expansions of real numbers); and we can do so in our situation. Doing so constitutes the full proof of Theorem I (B).

A machine $\mathfrak{U}$ which computes $\varphi_i(a)$ as a partial function of i and a we

[174] For a fuller discussion, cf. IM pp. 331–332 and § 68.

call a *universal machine*, since it can be used to compute any computable function $\varphi(a)$. To use it to compute $\varphi(a)$, say that $\varphi(a)$ is computed by Machine $\mathfrak{M}_i$; then apply $\mathfrak{U}$ to compute for i, a as a pair of arguments. Thus i plays the role of a set of *instructions* or *program* for $\mathfrak{U}$ which tells $\mathfrak{U}$ what function of a to compute.

The next theorem we can prove in full detail.

THEOREM II. *The function $\psi(a)$ defined by*

$$\psi(a) = \begin{cases} \varphi_a(a) + 1 & \text{if } (Ex)T(a, a, x), \\ 0 & \text{otherwise} \end{cases}$$

is not computable.

PROOF. Suppose $\psi(a)$ were computable; say machine $\mathfrak{M}_p$ computes it, so that $\psi(a) = \varphi_p(a)$ for all a. Substituting p for a,

$$\psi(p) = \varphi_p(p).$$

But since $\mathfrak{M}_p$ computes $\psi(a)$, we have, for all a, $(Ex)T(p, a, x)$, and in particular $(Ex)T(p, p, x)$. Using this in the definition of $\psi(a)$,

$$\psi(p) = \varphi_p(p) + 1.$$

The two displayed equations contradict each other.

To state the proof a little differently, we can consider each Turing machine $\mathfrak{M}_p$, and see that it will fail to compute $\psi(a)$ correctly for $a = p$, as follows. To begin with, $\mathfrak{M}_p$ may fail to compute any value for p as argument. But if $\mathfrak{M}_p$ does compute a value for p as argument, that value is $\varphi_p(p)$, by the definition of $\varphi_i(a)$; and in this case $(Ex)T(p, p, x)$, so that the correct value of $\psi(p)$ is $\varphi_p(p) + 1$ by the definition of $\psi(a)$. —

The significance of this result comes from the Church-Turing thesis, by which computability in Turing's sense agrees with the intuitive notion of computability. Accepting the thesis, as most workers in foundations do, the director of a computing laboratory must fail if he undertakes to design a procedure to be followed, or to build a machine, to compute this function $\psi(a)$. This refutes the notion, which news reports on modern developments in high-speed computing tend to foster in the public mind, that machines can do everything. The theorem does not assert that there is any particular value of $\psi(a)$ that we cannot learn. But in whatever model we freeze the design of a computing procedure or Turing machine, we will be short of having a procedure or machine that can compute all values of $\psi(a)$. If the values it computes are correct, there must be some values it cannot compute; in particular, it cannot compute the value $\psi(p)$ where p

is its own index (or for a procedure, the index of a machine which mechanizes the procedure). To improve the procedure or machine must take ingenuity, something that cannot be built into the machine.

It should have been apparent that there must be uncomputable number-theoretic functions, as soon as we met the Church-Turing thesis in § 41, by which the set of different possible machines is countably infinite because each machine is describable by a finite table in a fixed symbolism (cf. § 32). The set of the machines being countable, the set of the functions computable by machines is countable, while the set of all the number-theoretic functions is uncountable (§ 33). But it remains of interest to see how simple examples of uncomputable functions we can give. Our example ($\psi(a)$ in Theorem II) is really quite simple, as it is obtained by completing the definition (by the "0 otherwise") of a suitable computable partial function.

Why can't we compute $\psi(a)$ from its definition? Answering this question gives the next theorem.

THEOREM III. *The predicate* $(Ex)T(a, a, x)$ *is undecidable; i.e. the function*

$$\chi(a) = \begin{cases} 0 & \text{if } (Ex)T(a, a, x), \\ 1 & \text{otherwise} \end{cases}$$

is uncomputable.

PROOF. If we could decide $(Ex)T(a, a, x)$, we could compute the function $\psi(a)$ of Theorem II thus: Given a, decide whether $(Ex)T(a, a, x)$ or not. If this decision gives "yes", imitate the behavior of $\mathfrak{M}_a$ for a as argument to compute $\varphi_a(a)$, and add 1 to the result. If this decision gives "no", simply write 0.

Again, as in proving Theorem I, we are giving the idea of the proof, not the full technical details. What needs to be supplied is the hypothetical construction of a machine to compute $\psi(a)$ from one (supposed given) to compute $\chi(a)$. This is pretty easy to give, once one has reached the stage in the development that he would have reached in proving Theorem I.

DISCUSSION. Theorem III is essentially Church's theorem, which appeared with his thesis in his 1936 paper entitled "An unsolvable problem of elementary number theory". The difference is that we have given an example in terms of Turing computability, whereas Church's example was in terms of "λ-definability" (beginning § 41). The problem that is "unsolvable" is to find a decision procedure for the predicate $(Ex)T(a, a, x)$. Of course the problem is solved in another sense, by its being shown that there cannot be the required decision procedure. The problem of trisecting the general angle by a ruler and compass construction is unsolvable in one sense; but in another is solved by its being shown that the required construction cannot exist.

We note the especially simple logical form of the undecidable predicate, namely $(Ex)T(a, a, x)$, where as an almost immediate corollary of Theorem I (A) $T(a, a, x)$ is a decidable predicate. This is what is accomplished in Church's theorem, compared to simply contrasting the nonenumerability of the set of all (extensional) number-theoretic predicates (given by Cantor's diagonal method) with the enumerability of the set of the computable number-theoretic predicates (given by Church's thesis).[164]

EXERCISES. 42.1. Show that there is no algorithm for deciding whether a given Turing machine started in a given situation eventually stops (the "halting problem" for a Turing machine).

42.2. Find an undecidable predicate of the form $(x)P(a, x)$ where $P(a, x)$ is decidable. ("(x)" means "for all x".)

42.3. Show that the function $f(a)$ of Exercise 40.1 (a) (often written "$\varepsilon x P(a, x)$") is not computable when $P(a, x) \equiv T(a, a, x)$.

42.4*. Show that there is no algorithm for telling whether, for given i and j, $\varphi_i(a)$ and $\varphi_j(a)$ are the same partial function (i.e. whether, for each a, both functions have either the same value or no value).

42.5*. Show that the computable partial function $\varphi_i(a)$ of Theorem I (B) cannot be extended to a computable total function; i.e. there is no completely defined number-theoretic function $\varphi(i, a)$ such that: $(i)(a)[(Ex)T(i, a, x) \rightarrow \varphi(i, a) = \varphi_i(a)]$, and φ is Turing computable.[175]

§ 43. Applications to formal number theory: undecidability (Church) and incompleteness (Gödel's theorem). Recall that in the foregoing we have been using "decision procedure" and "decidable" (i.e. "a decision procedure exists") in relation to some given countably infinite class of questions (§ 40). Thus in Theorem I (A), the questions are "Is $T(i, a, x)$ true?" for the various values of i, a, x ($i, a, x = 0, 1, 2, \ldots$); and in Theorem III the questions are "Is $(Ex)T(a, a, x)$ true?" for the various values of a ($a = 0, 1, 2, \ldots$). In Theorem IV below, we shall infer from Church's theorem (Theorem III) that the decision problem for the system N of formal number theory of § 38 is "unsolvable" (or is solved in the negative); here the questions are "Is A provable in N?" where A ranges over all the formulas of N. In Theorem II, which concerns "computation procedure" and "computable", the questions are "What is the value of $\psi(a)$?" for the various values of a ($a = 0, 1, 2, \ldots$).

It should also be recalled that "decidable" or "computable" have both a vague intuitive sense (§ 40), and an exact sense which we have defined via Turing machines (§ 41). The Church-Turing thesis, and its converse that every Turing computable function is intuitively computable, assert

[175] Using recursiveness, with $\Phi(i, a)$ corresponding to the present $\varphi_i(a)$, the solution is in IM p. 341 following Theorem XXII.

that these two senses are equivalent. We understand our theorems now to be proved using the second sense, which requires more of us when we are establishing decidability or computability, and is the only sense in which they can be proved when we are establishing undecidability or uncomputability. But by the Church-Turing thesis, the latter theorems have their significance in terms of the intuitive sense of those terms.

We are now in a position to achieve our goal (§ 40) of showing there is no decision procedure for N (Theorem IV). Continuing the analysis a little further, we shall obtain Gödel's famous incompleteness theorem in a generalized version, as well as his second theorem (in § 44), which will clarify the situation described at the end of § 38.

We defined the predicate $T(i, a, x)$ (§ 42) informally in quite elementary terms, though its definition fully written out (with our explanation of Turing machine tables and the indexing of them) is long. It would therefore be disappointing if $T(i, a, x)$ could not be expressed in the symbolism of N. Indeed, if it could not, the first part of our claim of the adequacy of N for the usual elementary number theory (end § 38) would be false. For, though $T(i, a, x)$ is not commonly included in number-theoretic texts, there is nothing in its definition to prevent it from being included. In fact, the results cited in § 38 do enable us by established methods to find a formula T(i, a, x) (containing free exactly three distinct variables i, a, x) in the symbolism of N which expresses $T(i, a, x)$ under the intended interpretation of the symbols.[176] The like, as we have just argued, would have to be the case for any formal system N which we would consider adequate for the usual elementary number theory.

In N, the particular natural numbers 0, 1, 2, . . . are expressed by the respective terms 0, (0)′, ((0)′)′, . . . (which in § 38 we abbreviated "0", "1", "2", . . .); these terms we call the *numerals* (*for the respective natural numbers* 0, 1, 2, . . .). For any natural number a, we denote the numeral for a by "$\boldsymbol{a}$". Now, for each $a = 0, 1, 2, \ldots$, the proposition $(Ex)T(a, a, x)$ is expressed under our interpretation of the symbolism of N by the formula $\exists x T(\boldsymbol{a}, \boldsymbol{a}, x)$. Call this formula "$C_a$". Similarly, in *any* adequate formal system of elementary number theory, given any value of a, we can find effectively (end § 40) a closed formula C_a expressing under the intended interpretation the proposition $(Ex)T(a, a, x)$.

[176] To give somewhat more specific references for this than in Footnote 155 § 38, one can show by IM that $T(i, a, x)$ is primitive recursive (Chapter IX), using the technique of § 69, and then apply Corollary Theorem I p. 242. The predicate written "$(Ey)T_1(x, x, y)$" in Chapter XI pp. 281 ff. plays a role for general recursiveness analogous to that of the $(Ex)T(a, a, x)$ of this book for Turing computability (In some papers "T_1" is written simply "T"; the notation goes back to Kleene 1936, which slightly antedates Turing 1936–7.)

If, for a given x, the proposition $(Ex)T(a, a, x)$ is true, then we can prove it informally, by the mechanical process of exhibiting the computation steps of the machine $\mathfrak{M}_a$ applied to a up to the moment x at which a value will have just been computed. This will prove $T(a, a, x)$ for that x, and $(Ex)T(a, a, x)$ will follow by informal $\exists$-introduction. Now this informal proof can also be carried out within N (i.e. it can be "formalized in N"). Thus

(a) $$(Ex)T(a, a, x) \to \{\vdash \mathrm{C}_a \text{ in } \mathbf{N}\}.$$

Of course to show this would require a detailed investigation of the proof theory of N, which we are not giving in this book. But if it were not the case, the deductive apparatus of N (i.e. the list of its axiom schemata, axioms, and rules of inference) would be inadequate for the usual elementary number theory, contrary to the second part of our claim in § 38.[177]

Now let us assume about N that in it only true formulas are provable. Since under our interpretation C_a expresses $(Ex)T(a, a, x)$, this gives in particular

(b) $$\{\vdash \mathrm{C}_a \text{ in } \mathbf{N}\} \to (Ex)T(a, a, x).$$

In the next section we shall see why at this stage we simply *assume* (b). Clearly, if (b) were not so, we should reject N as a formal system for number theory. We certainly believe (b); indeed a proof of sorts, though not a finitary one (§ 36) and thus not one in metamathematics, is available, as follows. Under the usual interpretation, the axioms of N are true; and each of the rules of inference, when applied to one or two true formula(s) as premise(s) produces a true formula as conclusion. Thus all provable formulas are true. Hence, (b). (Cf. § 38 (B).)

THEOREM IV. *There is no decision procedure for provability in the formal system* N *of* § 38; *or briefly,* N *is undecidable.*

More generally, this applies not just to the formal system N *of* § 38, *but to any formal system* N *in which, to each a, there can be found effectively a closed formula* C_a *such that* (a) *and* (b) *hold.*

PROOF. Suppose there were a decision procedure for provability in N. Then we could, given a, decide as follows whether $(Ex)T(a, a, x)$ or not, which would contradict Theorem III. Given a, find (as we can effectively) the formula C_a, and apply the assumed decision procedure for provability in N to settle whether this formula C_a is provable. By (b) and (a), according as C_a is or is not provable, $(Ex)T(a, a, x)$ holds or does not hold. —

[177] If the method of finding T(i, a, x) proposed in Footnote 176 is used, then (a) will follow by IM Corollary Theorem 27 p. 244 and $\exists$-introduction.

Preparatory to the next theorem, we further apply our supposition that in N only true formulas are provable to the formulas $\neg C_a$ ($a = 0, 1, 2, \ldots$). Thus we assume (where "$\overline{}$" expresses "not")[165]

(c) $$\{\vdash \neg C_a \text{ in } \mathbf{N}\} \rightarrow \overline{(Ex)}T(a, a, x).$$

The same remarks we made about (b) apply to (c).

We now ask whether we can prove $\neg C_a$ when it is true; that is, whether the converse of (c), namely

(*) $$\overline{(Ex)}T(a, a, x) \rightarrow \{\vdash \neg C_a \text{ in } \mathbf{N}\},$$

also holds. If it did, we should have a decision procedure for $(Ex)T(a, a, x)$ as follows, again contradicting Theorem III. First, observe that all the proofs in the N of § 38 can be written on a typewriter with the 41 formal symbols of N and the comma to separate successive formulas in a proof, making 42 symbols in all. So the proofs are enumerable (say using the method of digits, end § 32), and indeed effectively (since there is a decision procedure for being a proof, § 40). So, given a, we could search through an enumeration of the proofs in N for one of C_a or of $\neg C_a$. By the classical law of the excluded middle with (a) and (*), we shall find one or the other. By (b) and (c), according to which we find we shall learn that $(Ex)T(a, a, x)$ is true or not true.

Thus (*) does not hold for all a, which gives us the next theorem.

THEOREM V. *In the system* N *of* § 38 *there is a closed formula* C_p *such that* (i) $\neg C_p$ *is true*, (ii) *not* $\vdash \neg C_p$ *in* N, *and* (iii) *not* $\vdash C_p$ *in* N.

More generally, this applies to any formal system N *in which, to each* a, *there can be found effectively a closed formula* C_a (*expressing* $(Ex)T(a, a, x)$) *such that* (a)–(c) (*or* (b)–(d) *below*) *hold.*

FIRST PROOF, COMPLETED. Since (*) does not hold for all a, there is some number p such that $\overline{(Ex)}T(p, p, x)$ (i.e. (i), by what $\neg C_p$ expresses) but not $\vdash \neg C_p$ in N (i.e. (ii)). By $\overline{(Ex)}T(p, p, x)$ with (b), then not $\vdash C_p$ in N (i.e. (iii)).

REMARKS. This gives us Gödel's famous incompleteness theorem 1931, generalized to apply to all formal systems N satisfying very general conditions, and with the "formally undecidable sentence" C_p expressing the value of a preassigned predicate $(Ex)T(a, a, x)$ for an argument p depending on the particular system. This generalized form of Gödel's theorem (with a predicate written "$(Ex)T(a, a, x)$") is due to Kleene 1943.[178]

[178] Kleene used "general recursive functions" (§ 41) instead of Turing machines. The first use of Church's thesis to give a generalized version of Gödel's theorem was by Kleene in 1936.

In a formal system with notation like that of **N**, we say a formula E is *formally decidable* if $\vdash$ E or $\vdash \neg$E; and we say the system is *simply complete*, if every closed formula E is formally decidable. Thus by (iii) and (ii) of Theorem V, **N** is simply incomplete, with C_p as an example of closed formally undecidable formula. This notion of "formal decidability" applies to a particular formula, while decidability in the sense of § 40 or of § 41 applies to an infinite class of questions (or propositions). We restrict the E in the definition of simple completeness to be closed, because e.g. we would not wish to have $\vdash 2|a$ or $\vdash \neg 2|a$ (cf. § 38). For, under the generality interpretation (which applies to free variables of provable formulas, § 38), $2|a$ as a provable formula would say "all natural numbers are even" and $\neg 2|a$ would say "all natural numbers are odd". We likewise confine the definition to systems with notation like that of **N**, excluding systems like the propositional and predicate calculi, because e.g. neither $\vdash$ P $\supset$ Q nor $\vdash \neg$(P $\supset$ Q) in those calculi. Although P $\supset$ Q is a closed formula, its atoms P and Q function in the definition of validity like variables ranging with the generality interpretation over all propositions.

The foregoing proof of Theorem V is indirect, the existence of the p being inferred from the absurdity of (*) holding for all a. We now give a direct proof.

SECOND PROOF OF THEOREM V. Let $\mathfrak{M}_p$ be a Turing machine which, applied to a as argument, searches through an enumeration of the proofs in **N** for one of $\neg C_a$, and if one is found writes 0, but otherwise never computes a value (indeed, never stops). Considerations employed in similar situations above should make it clear that such a machine $\mathfrak{M}_p$ exists. A detailed proof for the particular **N** of § 38 is available on the basis of the developments in Turing-machine theory to which we referred late in § 41 and early in § 42. From what the machine $\mathfrak{M}_p$ does, with the definition of $T(i, a, x)$,

$$(d) \qquad (Ex)T(p, a, x) \equiv \{\vdash \neg C_a \text{ in } \mathbf{N}\}.$$

Now (b)–(d) give us the three parts of Gödel's theorem, thus.

(i) Suppose $(Ex)T(p, p, x)$; then by (d)

$$\vdash_{\mathbf{N}} \neg C_p.$$

Thence by (c)

$$\overline{(Ex)T(p, p, x)}.$$

This contradicts $(Ex)T(p, p, x)$; so by reductio ad absurdum, $\overline{(Ex)T(p, p, x)}$, i.e. $\neg C_p$ is true.

(ii) By (i) $\neg C_p$ is true, i.e. $\overline{(Ex)T(p, p, x)}$. Thence by (d)

$$\text{not } \vdash_{\mathbf{N}} \neg C_p.$$

(iii) Suppose $\vdash_{\mathbf{N}} C_p$; then by (b)

$$(Ex)T(p, p, x),$$

i.e. C_p is true. This contradicts (i); so not $\vdash_{\mathbf{N}} C_p$.

DISCUSSION. Here we have used the feature of formal systems, essential to the purposes which they are intended to serve (§§ 36, 37), that a proof of a formula can be effectively recognized as being such (and also that C_a can be effectively found from a). Without this feature, we would have a trivial counterexample to Theorem V by taking all the true closed formulas as the axioms of N. With it, by Church's thesis we conclude that, for any such system, an $\mathfrak{M}_p$ exists.[179] Here the computability notion can be applied directly to the linguistic symbolism of the system, or that symbolism can be converted to natural numbers, e.g. by the method of digits with Footnote 173, as we have already done with machine tables. The numbers correlated to the linguistic objects are called *Gödel numbers*, and the correlation is called a *Gödel numbering*, after Gödel, who introduced this device in 1931.[180]

The application of Church's thesis, by which we obtain Theorem V for all systems N, can be avoided for a particular system by actually constructing the $\mathfrak{M}_p$ for it. This *in effect* Gödel did in 1931 in proving his theorem for a particular system before Church's thesis had appeared (1936);[181] and this, as we have implied, can be done by known methods to get the theorem, without leaning on Church's thesis, for the particular system N of § 38.

Substituting p for a in (d) and negating both sides (as we already did in proving (ii)),

$$(\bar{d}) \qquad \overline{(Ex)T(p, p, x)} \equiv \{\text{not } \vdash \neg C_p \text{ in } \mathbf{N}\}.$$

Remembering that under the interpretation $\neg C_p$ expresses $\overline{(Ex)T(p, p, x)}$, we thus have that $\neg C_p$ *is a formula which, under the interpretation, expresses (a proposition equivalent to) its own unprovability.* This was the point of

[179] "In particular, the nature of the intuitive evidence for the deductive processes which are formalized in the system plays no role.

"Let us imagine an omniscient number theorist, whom we should expect, through his ability to see infinitely many facts at once, to be able to frame much stronger systems than any we could devise. Any correct system which he could reveal to us, telling us how it works without telling us why, would be equally subject to the Gödel incompleteness." (Kleene 1943 p. 65.)

[180] Godel used another method of numbering in 1931; and Hilbert and Bernays 1939 and IM use still another method. In 1931–2, Gödel used what we are calling the "method of digits". This type of numbering is studied in detail in Smullyan 1961, which gives a fresh approach to formal systems from ideas of Post 1943.

[181] In the period immediately after Gödel's results appeared, there was some uncertainty among logicians whether there might not be an escape from them by using some other particular system quite different in its details from the system Gödel used.

departure in Gödel's original proof, or at least in the heuristic explanation he gave for it; namely, he constructed a formula expressing its own unprovability. This is very close to the paradox of the Liar (§ 35), where we have a sentence expressing its own falsity. Only now, by Gödel's substitution of "unprovability" for "falsity", there is a way out. We wish (and believe in the case of N) that all provable formulas are true. Then, if all true formulas were provable, we would have "unprovable ≡ false", so we would have the paradox of the Liar. The way out now is that not all true formulas are provable; in particular, $\neg C_p$ is unprovable though true.[182]

The first part of Hilbert's program (§§ 36, 37) called for formalizing number theory, analysis, and a suitable part of set theory in a formal system S. Gödel's theorem shows this cannot be done completely even for number theory. For $\neg C_p$ expresses a number-theoretic proposition which by the theorem is true, yet unprovable, provided of course S satisfies the hypotheses for the N of the theorem. However these hypotheses are connected with the purposes for devising formal systems, so we have no prospect of escaping them. These hypotheses are simply consequences of the structural feature of formal systems discussed above and of the supposition that N is adequate and correct for some elementary number theory.

We do not consider that Theorem V means we must give up our emphasis on formal systems. The reasons which make a formal system the only accurate way of saying explicitly what assumptions go into proofs are still cogent. Rather, Theorem V indicates that, contrary to Hilbert's program, the path of mathematical conquest (even within the already fixed territory of arithmetic) shall not consist solely in discovering new proofs from given axioms by given rules of inference, but also in adducing new axioms or rules. There remains the question whether mathematicians can agree on the correctness of the new axioms or rules.

In Theorem V, no sooner are we aware that $\neg C_p$ is unprovable than we also know that $\neg C_p$ is true, so we can extend N (call it "N_0") by adding $\neg C_p$ as a new axiom. But then Gödel's theorem will apply to the extended system N_1, and we shall have in this system a true but unprovable formula $\neg C_{p_1}$. The process can be repeated to obtain successively stronger formal systems N_0, N_1, N_2, ... We can combine these to form another formal system, if these systems are formed systematically enough so that after they are combined it will be decidable which formulas are axioms, and hence

[182] Nagel and Newman 1956, 1958 give a popular exposition of Gödel's theorem along the lines of Godel's original proof 1931. They give a misleading impression that the generalized Gödel theorem is implied by Gödel's 1931 reasoning and thus without the Church-Turing thesis.

In Popper 1954, Theaetetus explains Gödel's theorem to Socrates.

which finite sequences of formulas are proofs (otherwise we could not consider the result a formal system). Then, starting from this system, we can again use the extension process based on Gödel's theorem. Proceeding in this way does not provide an escape from the consequences of Gödel's theorem.[183]

§ 44. Applications to formal number theory: consistency proofs (Gödel's second theorem). In establishing (i) in Theorem V we have proved that $\neg C_p$ is true, though by (ii) $\neg C_p$ cannot be proved in **N**. It is illuminating to consider wherein our intuitive proof of (the truth of) $\neg C_p$ transcends the resources of **N**.

In our second proof of (i) in Theorem V we used (c), which we regarded as an assumption. Using (a), this assumption (c) can be derived from the assumption that **N** is *simply consistent*, i.e. that for no formula E do both $\vdash E$ and $\vdash \neg E$ hold in **N** (Exercise 44.1 (A)).

Alternatively, we can recast the second proof of (i) to use simple consistency directly instead of (c), thus.

(i) Suppose $(Ex)T(p, p, x)$; thence by (a) and (d)

$$\vdash_{\mathbf{N}} C_p \text{ and } \vdash_{\mathbf{N}} \neg C_p,$$

contradicting the simple consistency. So by reductio ad absurdum, $(\overline{Ex})T(p, p, x)$, i.e. $\neg C_p$ is true.

In either version of the second proof of (i), the part written out in detail is entirely elementary; indeed, it is only informal use of the predicate calculus. Also the reasoning by which (a) and (d) can be established for the **N** of § 38, or a similar particular system, is elementary, on the level of informal number theory, though it is quite long when executed in full detail. There remains (c) or the simple consistency as the sole component not known to be elementary.

In summary, our informal proof of (i) is entirely elementary except for

[183] Classes of such systems are called "ordinal logics" because the systems are indexed by finite and infinite (or "transfinite") "ordinal numbers". Ordinal logics have been studied by Turing 1939, Feferman 1958 abstracts, 1962, and Kreisel 1958c.

The familiar natural numbers serve both as finite cardinals (e.g "There are 10 houses on this street.") and as finite ordinals (e.g. "He lives at 10 Downing Street."). Cantor gave a theory in which *ordinal numbers* can be defined as equivalence classes of "well-ordered" sets under the relation of admitting a 1–1 correspondence preserving the ordering. A *well-ordered* set is a "linearly ordered" set each nonempty subset of which possesses a least element. A *linearly ordered* set is a set S together with an "order" relation $<$ which is irreflexive, transitive, and total (for each a and b in S, $a < b \vee a = b \vee a > b$). Under an obvious definition of $<$ for ordinal numbers, the ordinal numbers themselves are well-ordered.

Ordinal logics use the theory of ordinal numbers in a constructive or computable version, due to Church and Kleene 1936, Church 1938 and Kleene 1938.

our having to use in it (c) or the simple consistency of N. Thus it is indicated that (c) or that simple consistency is the respect in which it transcends N.

Preparatory to formulating this more carefully, we note for reference that the following can be shown in an entirely elementary way:

(1) $\{\mathbf{N} \text{ is simply consistent}\} \rightarrow \overline{(Ex)}T(p, p, x)$.

The proposition that N is simply consistent can be expressed in the symbolism of N. Let $\mathfrak{M}_r$ be a Turing machine which, applied to any number a, searches through an enumeration of the proofs in N for a proof of a formula of the form $\mathrm{E} \,\&\, \neg\mathrm{E}$, and if it finds one writes 0, but otherwise never computes a value. The consistency of N is equivalent to $\overline{(Ex)}T(r, a, x)$, for any a, and in particular to $\overline{(Ex)}T(r, r, x)$, which is expressed in N by $\neg\mathrm{C}_r$. Call this formula "Consis".[184]

Now the informal implication (1) can be translated into the symbolism of N as the formula

$$\text{Consis} \supset \neg\mathrm{C}_p,$$

since $\neg\mathrm{C}_p$ expresses $\overline{(Ex)}T(p, p, x)$.

THEOREM VI. (Gödel's second theorem.) *In the system* N *of* § 38, *not* $\vdash$ Consis; *i.e. the formula* Consis *expressing in* N *the consistency of* N *is unprovable in* N.

More generally, this applies to any simply consistent formal system N, *with any choice of a formula* Consis *expressing in* N *the consistency of* N, *such that there can be found effectively, to each a, a closed formula* C_a, *and a number p, for which* (a) *and* (d), *and* (2) *below, hold.*[185]

PROOF (COMPLETED). Because (1) is established by elementary intuitive reasoning, our belief in the adequacy for elementary number theory of the system N of § 38 gives us reason to believe that the formula Consis $\supset \neg\mathrm{C}_p$ expressing (1) can be proved in N, i.e. that the informal proof of (1) in elementary number theory can be formalized in N. A person having some familiarity with the development of number theory in N

[184] Alternatively, we can correlate Gödel numbers to the objects in the symbolism of N (end § 42), and take as Consis a formula which *directly* translates the consistency property into a statement about Gödel numbers. Cf. IM p. 210.

[185] So much has been put into the hypotheses of the generalized version of this theorem (the second paragraph) that it only amounts to an application of modus ponens to (2) with the fact that simple consistency, (a) and (d) imply Theorem V (ii). The significance of this second paragraph is that the hypotheses are ones that would be satisfied by any system of number theory, and any choice of Consis for it, that we would normally consider. A fuller treatment of the domain of applicability of Godel's second theorem was first given by Feferman 1960.

(further than we went in § 38) would find it hard to doubt this. Hilbert and Bernays did verify it to be the case.[186] That is,

$$\vdash_N \text{Consis} \supset \neg C_p. \tag{2}$$

Now suppose $\vdash_N$ Consis. By $\supset$-elimination (modus ponens) from (2), then $\vdash_N \neg C_q$, contradicting (ii) of Theorem V.

DISCUSSION. The second part of Hilbert's program for foundations called for showing by finitary metamathematical reasoning that a formal system chosen as a formalization of classical mathematics is consistent. As the mathematics formalized in N § 38 is not all finitary, the hope must have been that the part of the methods formalized in N obtained by excluding the nonfinitary ones would suffice for the consistency proof. Gödel's second theorem shows that not even all the methods formalizable in N, i.e. a metalanguage isomorphic to N itself, can prove the consistency of N, if N is consistent (as we have been assuming).

Some mathematicians judged that this ended forever the hope of getting a guarantee for classical mathematics by a metamathematical consistency proof.

Others thought it possible that methods could be found which could be considered as finitary even though not formalizable in N. Then the set of the finitary methods F and the set of the methods formalized in N could be pictured as overlapping circles; the law of the excluded middle for countably infinite sets (§ 36) would be in N but not in F, while some new finitary method would be in F but not in N.

Progress toward proving the consistency of N had been stalled since the consistency proof for a subsystem of N by Ackermann in 1924–5 (end § 38). Interesting new proofs, but still for essentially the same subsystem, were given by von Neumann in 1927, Herbrand in 1931–2, and Gentzen in 1934–5. After the necessity of using some nonelementary method was revealed by Gödel's second theorem in 1931, it was not too long before Gentzen in 1936 gave a consistency proof for N. This employed, as its method not formalizable in N, induction over a certain segment of the finite and transfinite ordinal numbers, which Cantor had obtained by an extension of the counting process or ordinal use of numbers beyond the natural numbers, in association with "well-ordered" sets (analogously to his introduction of transfinite cardinal numbers in association with unordered sets, § 34).[183] The induction was over the ordinals less than the ordinal called "ε_0" by Cantor.[187] A different consistency proof for N was

[186] Hilbert and Bernays 1939 pp. 283 ff., especially pp. 300–324. Their work is done in a system essentially equivalent to the N of § 38, though using a somewhat different choice of Consis and $\neg C_p$ than the ones we are basing on Turing machines.

[187] For a little more of an indication, cf. IM pp. 476–478.

given by Ackermann in 1940, also using transfinite induction over the ordinals $< \varepsilon_0$. Schütte 1960 gives such a proof in a new, quite perspicuous form. Denton and Dreben 1968 greatly simplify the Ackermann proof by using ideas from Herbrand 1930.

It is a rather subjective matter whether this should make us feel safer about N than we already felt on the basis of its axioms being true, and its rules of inference preserving truth, under an interpretation ("truth definition") that as classical mathematicians we presumably accept (§ 43 following (b)). By a simple reduction of classical logic to intuitionistic logic given by Gödel 1932–3, Gentzen 1936 and Bernays, the consistency proof by a truth definition can even be managed intuitionistically.[188] Tarski, asked whether he felt more secure about classical mathematics from Gentzen's consistency proof, replied, "Yes, by an epsilon". (Calculus students will recognize epsilon "ε" as commonly used for a small positive number.)

It is clear that the consistency proofs by induction over the ordinals less than the ordinal ε_0 work very hard to accomplish something, but less clear what.

Kreisel (1951–2, 1958) finds the significance of the consistency proofs by transfinite induction up to ε_0 in by-products. Suppose that, for a given natural number i, the formula $\forall a \exists x T(\boldsymbol{i}, a, x)$ is provable, where $\boldsymbol{i}$ is the numeral for i. Then, assuming that in N only true formulas are provable, $(a)(Ex)T(i, a, x)$ holds, and thus $\varphi_i(a)$ is a computable total function (cf. § 42). It is not hard to see that the computable (total) functions which can thus be proved to exist in N constitute a proper subclass (i.e. not all) of the computable functions (Exercise 44.2). Indeed Kleene 1936 gave a proof of Gödel's incompleteness theorem from this idea.[178] Kreisel however extracts from Ackermann's consistency proof 1940 a different characterization (not directly from N) of this subclass of the computable functions. The possibility thus appears that some true formula $\forall a \exists x T(\boldsymbol{i}, a, x)$ might be shown to be unprovable in N because for that i the computable function $\varphi_i(a)$ is not in this subclass.

Gentzen had already claimed in the first (1936) version of his consistency proof to have established a property of provable formulas of classical formal number theory N that can be regarded as an intuitive interpretation. But the property was complicated, and received little attention after his proof appeared in another version 1938a easier to follow and not adducing the property.

To see that there is a problem of interpretation, we recall from § 36 that Hilbert 1926, 1928 made a distinction between "real" statements having a clear intuitive meaning, and other statements called "ideal". In classical mathematics, the "ideal" statements are adjoined to the "real". One

[188] Cf. IM § 81. A related reduction was given by Kolmogorov 1924–5.

might have supposed that the "real" statements would include all the propositions of elementary number theory.

However the picture has not turned out to be this simple. For, in elementary number theory there are statements proved classically that are not true on the basis of their meanings for an intuitionist. Kleene 1943 argued this as follows.

The intuitionist understands an existential statement $(Ey)P(y)$ to mean that one can actually find a y such that $P(y)$. From this standpoint, what can $(a)(Ey)P(a, y)$ mean? Only that there is an effective procedure by which, given any a, one can find a y such that $P(a, y)$. By the Church-Turing thesis, this must mean that the y is a computable function of a. Thus we are led to the thesis that intuitionistically $(a)(Ey)P(a, y)$ only if there is a computable function $g(a)$ such that $(a)P(a, g(a))$.

By the classical law of the excluded middle (which intuitionists decline to affirm), for each a, $(Ex)T(a, a, x) \vee \overline{(Ex)}T(a, a, x)$.
Thence $[(Ex)T(a, a, x) \,\&\, 0=0] \vee [\overline{(Ex)}T(a, a, x) \,\&\, 1=1]$.
Thence $(Ey)[(Ex)T(a, a, x) \,\&\, y=0] \vee (Ey)[\overline{(Ex)}T(a, a, x) \,\&\, y=1]$.
Thence $(Ey)\{[(Ex)T(a, a, x) \,\&\, y=0] \vee [\overline{(Ex)}T(a, a, x) \,\&\, y=1]\}$.
This is for each a, so we have proved classically

$$(\alpha) \qquad (a)(Ey)\{[(Ex)T(a, a, x) \,\&\, y=0] \vee [\overline{(Ex)}T(a, a, x) \,\&\, y=1]\}.$$

We have presented this proof informally, but it is a simple matter to formalize it in the system N of § 38, so that we have

$$(\beta) \qquad \vdash_{\mathrm{N}} \forall a \exists y\{[\exists x T(a, a, x) \,\&\, y=0] \vee [\neg \exists x T(a, a, x) \,\&\, y=1]\}.$$

Let us abbreviate (α) as $(a)(Ey)P(a, y)$. By the above thesis, (α) holds intuitionistically only if $(a)P(a, g(a))$ for some computable function $g(a)$. But from what $P(a, y)$ is in this example, the only $g(a)$ for which $(a)P(a, g(a))$ holds is the representing function of $(Ex)T(a, a, x)$; and by Theorem III § 42, this $g(a)$ is not computable.

Summarizing: (α) holds in classical informal number theory, and translates into a formula which (as (β) states) is provable in N, but (α) cannot be affirmed as true intuitionistically.

Specker 1949 gave similar examples in which the propositions (α) which hold classically but not intuitionistically (if the above thesis of Kleene 1943 is accepted) are instances of well-known theorems of analysis, e.g. the theorem that a bounded monotone sequence of rationals is convergent. For definiteness, take a monotone nondecreasing sequence. Using variables n, b, m, n_1, n_2 ranging over the natural numbers, and writing $f(0), f(1)$,

$f(2), \ldots$ for any sequence of rationals, the theorem can be stated thus:

If

(A) $(n)f(n) \leq f(n+1)$ (f is monotone nondecreasing),

and

(B) $(Eb)(n)f(n) \leq b$ (f is bounded),

then

(C) $(m)(En)(n_1)(n_2)\{n_1, n_2 \geq n \rightarrow |f(n_1)-f(n_2)| < 1/2^m\}$ (f converges).

It should be reasonably obvious that the notion of Turing computability can be applied to functions $f(n)$ with rational numbers as values, either directly or by talking about the natural numbers which index the rationals in some fixed enumeration of the rationals § 32. Specker gave a particular sequence $f(0), f(1), f(2), \ldots$, for which the function f is Turing computable,[189] and such that (A) and (B) hold but

(C′) $(m)(n_1)(n_2)\{n_1, n_2 \geq g(m) \rightarrow |f(n_1)-f(n_2)| < 1/2^m\}$

holds for no computable function g. Thus, for this particular f, (A)–(C) are expressible in the elementary theory of natural and rational numbers (which is in the common part of the classical and intuitionistic languages of mathematics), the hypotheses (A) and (B) are true both intuitionistically and classically, but the conclusion (C) is true only classically.

Kreisel's proposal uses Ackermann's consistency proof for N to correlate to (α) on the basis of (β) a considerably more complicated statement, which is both meaningful and true for a "finitist". A "finitist", if not an intuitionist, is at least of similar ilk.[190]

Formal systems can be set up to study the foundations of intuitionistic mathematics, as Heyting did in 1930, 1930a, though the intuitionists have always maintained on philosophical grounds (from before Gödel's theorem was known) that such systems cannot be complete. (Cf. ends §§ 12, 25, 39.) Kleene, David Nelson and others have since 1941 been applying computable (or general recursive) functions to elucidate the differences between intuitionistic and classical formal systems.[191]

EXERCISES. 44.1. Using (a), show that: (A) the simple consistency of N implies (c), and (B) conversely (using weak ¬-elimination § 11).

44.2. Show that $\vdash_N \forall a \exists x T(i, a, x)$ does not hold for all computable total functions $\varphi_i(a)$. (Assume that only true formulas are provable in N.)

44.3*. Assuming N simply consistent, show that there is a formal system

[189] In fact, via indices in a standard enumeration of the rationals, f is primitive recursive IM Chapter IX.

[190] Introductions to these ideas of Kreisel are 1953, 1958. Kreisel's current thinking on a broad range of foundational problems is in 1965.

[191] Cf. IM § 82, and Kleene and Vesley 1965.

M of number theory which is simply consistent but in which not all provable formulas are even classically true under the usual interpretation.[192]

44.4*. Let **S** be a system satisfying the hypotheses of Gödel's second theorem (except not assumed to be simply consistent). Let **T** be **S** minus some of its postulates. What would have been an appropriate response by a logician in 1930 to the following report by one of his colleagues? In 1932?
(a) "In **S**, I have proved **S** consistent."
(b) "In **T**, I have proved **S** consistent."

★§ 45. Application to the predicate calculus (Church, Turing).

THEOREM VII. *There is no decision procedure for provability in the (pure) predicate calculus; briefly, the predicate calculus* **Pd** *is undecidable.* (Church 1936a, Turing 1936–7.)

PROOF. Theorem IV followed from Theorem III because the theory of the predicate $(Ex)T(a, a, x)$ can be formalized in **N** to the following extent: to each natural number a, a formula C_a can be found effectively such that C_a is provable in **N** if and only if $(Ex)T(a, a, x)$ ((a) and (b) in § 43).

Theorem VII will follow from Theorem III because the theory of $(Ex)T(a, a, x)$ can similarly be formalized in simply the pure predicate calculus **Pd** (end § 39). Our argument that the theory of $(Ex)T(a, a, x)$ can be formalized in **Pd** will be in three parts. (α) Only *finitely* many function symbols $f_1, \ldots, f_k$ and ("nonlogical") axioms have to be added to the predicate calculus with the predicate symbol $=$ to obtain a formal system S_k in which the theory of $(Ex)T(a, a, x)$ can be formalized to the extent required. (β) The system S_k can be transformed into an essentially equivalent system **S** with the symbolism of the pure predicate calculus and still only finitely many nonlogical axioms. (γ) The closed nonlogical axioms of **S** can be regarded as assumption formulas for deductions in **Pd**, to which the deduction theorem can be applied.

(α) To outline the first part, we start with the fact that the representing function (§ 40)

$$\tau(i, a, x) = \begin{cases} 0 & \text{if } T(i, a, x), \\ 1 & \text{otherwise} \end{cases}$$

of the predicate $T(i, a, x)$ can be defined by the last of a finite list of (primitive) recursive definitions, beginning with those of $+$ and $\cdot$ (§§ 38, 41). The reader must take our word for this part of the detailed proof.[193]

Now we describe the system S_k. We start with the predicate calculus, using the symbolism of the system **N** of § 38 (i.e. we start with the system

[192] Solution in § 47 below.

[193] A fair amount of technique such as is provided in IM Chapters IX, X and XIII is helpful in constructing this list of recursive definitions.

[20] of § 39). To this we add function symbols to express the functions after + and · which are defined by the rest of the recursive definitions culminating in the definition of τ. Say that altogether the (individual and) function symbols expressing 0, ′, +, ·, . . . , τ are $f_1, \ldots, f_k$ (so "f_1" is a name for 0, "f_2" for ′, etc.). As the axioms $A^k_1, \ldots, A^k_{m_k}$ we add to the axioms of the predicate calculus, we take the last six particular axioms 16–21 of N, also the pairs of equations of the additional recursive definitions, and finally the open equality axioms for the function symbols $f_3, \ldots, f_k$ (§ 29). For f_3 (i.e. +) these are the two formulas which were proved in N in § 38 Example 2 and Exercise 38.2 (but now they are to be axioms).

In S_k the equations expressing the values of the functions +, ·, . . . , τ will all be provable, as is illustrated for + by Exercise 38.4. Thus, when i, a, x are natural numbers such that $T(i, a, x)$ holds, $f_k(i, a, x)=0$ will be provable in S_k. So when $(Ex)T(a, a, x)$ holds, we can in S_k prove the equation $f_k(a, a, x)=0$ for a suitable x, and thence by ∃-introduction prove $\exists x f_k(a, a, x)=0$; call this formula "D^k_a". Thus

(1) $$(Ex)T(a, a, x) \rightarrow \{\vdash D^k_a \text{ in } S_k\}.$$

We get an easy proof of the converse of (1) by applying the theory of valid consequence in the predicate calculus with functions (§§ 20, 28). For, similarly to (B) in § 38, if $\vdash D^k_a$ in S_k, then $\forall A^k_1, \ldots, \forall A^k_{m_k} \models D^k_a$. Now take as the domain D the natural numbers {0, 1, 2, . . .}, and consider the assignment in D in which $=, f_1, f_2, f_3, f_4, \ldots, f_k$ are interpreted by the equality predicate =, the natural number 0, and the functions ′, +, ·, . . . , τ respectively. Under this assignment, $\forall A^k_1, \ldots, \forall A^k_{m_k}$ are all t, so (by $\forall A^k_1, \ldots, \forall A^k_{m_k} \models D^k_a$) D^k_a is also t. But under this assignment, D^k_a being t means that $(Ex)T(a, a, x)$. Thus

(2) $$\{\vdash D^k_a \text{ in } S_k\} \rightarrow (Ex)T(a, a, x).$$

This proof of (2) is not metamathematical (cf. § 37). A metamathematical proof can be given.[194] Combining (1) and (2),

(3_k) $$\{\vdash D^k_a \text{ in } S_k\} \equiv (Ex)T(a, a, x).$$

(β) There is a method by which, starting with S_k, we can replace successively the function symbols $f_k, \ldots, f_4, f_3, f_2, f_1$ expressing $\tau, \ldots, \cdot, +, ′, 0$

[194] The metamathematical proof of (2), and the details missing in our outline of the argument, appear e.g. in Kleene IM Part IV, with certain results of Parts II and III (cf. p. 434 Remark 2), though with a different $T(i, a, x)$ than the present one. (There the treatment is based on general recursiveness instead of on Turing computability; cf. § 41 above.)

Thus for Theorem VII we don't need a nonelementary assumption, like the assumption of (b) for Theorem IV.

by respective predicate symbols $F_k, \ldots, F_4, F_3, F_2, F_1$ expressing the representing predicates of $\tau, \ldots, \cdot, +, ', 0$ (cf. end § 38). We shall give the idea by discussing the step in which f_4 (i.e. $\cdot$) is replaced, assuming that the function symbols $f_k, \ldots, f_5$ have already been replaced by predicate symbols $F_k, \ldots, F_5$. (By giving just this step, we can illustrate the process, while keeping the notation simple.)

So we suppose that, as a result of the previous replacement steps, we have a system $\mathbf{S}_4$ in which the only function symbols are f_4, f_3, f_2, f_1 (i.e. $\cdot, +, ', 0$) and which has only finitely many nonlogical axioms $A^4_1, \ldots, A^4_{m_4}$. Furthermore in $\mathbf{S}_4$, to each natural number a, there is a formula D^4_a such that

$$(3_4) \qquad \{\vdash D^4_a \text{ in } \mathbf{S}_4\} \equiv (Ex)T(a, a, x).$$

We seek a system $\mathbf{S}_3$ with only the function symbols f_3, f_2, f_1, with only finitely many nonlogical axioms $A^3_1, \ldots, A^3_{m_3}$, and in which to each a there is a formula D^3_a such that

$$(3_3) \qquad \{\vdash D^3_a \text{ in } \mathbf{S}_3\} \equiv (Ex)T(a, a, x).$$

To obtain $\mathbf{S}_3$ from $\mathbf{S}_4$ we alter the symbolism by replacing the 2-place function symbol $\cdot$ by a 3-place predicate symbol which we shall also write $\cdot$ (i.e. we use $\cdot$ for both f_4 and F_4). Thus in the formation rules, we omit the clause which says that whenever r and s are terms (r)$\cdot$(s) is a term, but add a clause which says that whenever r, s and t are terms $\cdot$(r, s, t) is a formula. For the interpretation, the formula $\cdot$(r, s, t) of $\mathbf{S}_3$ has the same meaning as (r)$\cdot$(s)=t had in $\mathbf{S}_4$. That is, $\cdot(a, b, c)$ expresses the representing predicate $a \cdot b = c$ of the product function $a \cdot b$ (cf. end § 38).

To get the axioms of $\mathbf{S}_3$ from those of $\mathbf{S}_4$, we replace 20 and 21 by

(A) $\cdot(a, 0, 0)$, (B) $\exists c(\cdot(a, b, c) \,\&\, \cdot(a, b', c+a))$.

We replace the two equality axioms for $\cdot$ in $\mathbf{S}_4$ by

(C) $a=b \supset (\cdot(a, c, d) \supset \cdot(b, c, d))$, (D) $a=b \supset (\cdot(c, a, d) \supset \cdot(c, b, d))$.

We add the two axioms

(E) $a=b \supset (\cdot(c, d, a) \supset \cdot(c, d, b))$,
(F) $\exists c(\cdot(a, b, c) \,\&\, \forall d(\cdot(a, b, d) \supset c=d))$.

Here (C)–(E) are the open equality axioms for the predicate symbol $\cdot$, which express that $\cdot(a, b, c)$ is well-defined as a predicate; and (F) (abbreviated "$\exists! c \cdot(a, b, c)$" in § 29) expresses that, for given a and b, $\cdot(a, b, c)$ is true for one and only one c, i.e. that $\cdot(a, b, c)$ is the representing predicate of a function. Finally, we replace each other axiom of $\mathbf{S}_4$ which has the function symbol $\cdot$ in it by the result of paraphrasing it to use the predicate

symbol ·. Already (A) and (B) are such paraphrases of 20 and 21, and the method of paraphrasing is also illustrated at the end of § 38 for the function symbol !.

Now in the system S_3 which we have just constructed from S_4, the same situation exists relative to the function · that exists in N of § 38 relative to functions like !. By the general theory cited in Footnote 156 § 38, we can in S_3 carry out the reasoning about the function · via the paraphrases.[195] So in this way, from (3_4) it can be inferred that (3_3) holds when D_a^3 is the result of paraphrasing D_a^4 to replace the function symbol · by the predicate symbol ·. (In fact, D_a^4 will not contain ·, so D_a^3 is D_a^4. But D_a^{k-1} will differ from D_a^k, D_a^1 from D_a^2, and D_a^0 from D_a^1.)

The successive replacements (illustrated by the step from S_4 to S_3) lead to a system S_0 with no function symbols, but instead only predicate symbols, and with only finitely many axioms $A_1^0, \ldots, A_{m_0}^0$. In S_0, to each a there is a formula D_a^0 such that

$$(3_0) \qquad \{\vdash D_a^0 \text{ in } S_0\} \equiv (Ex)T(a, a, x).$$

From S_0 we now pass to a system S having the notation of the pure predicate calculus **Pd**, by changing the predicate symbols $=, F_1, \ldots, F_k$ of S_0 to respective predicate letters $P_0, \ldots, P_k$ (used with the same respective numbers of arguments). Let $A_1, \ldots, A_m$ $(m = m_0)$ and D_a result by this change in notation from $A_1^0, \ldots, A_{m_0}^0$ and D_a^0; so $A_1, \ldots, A_m$ are the nonlogical axioms of S. A proof of D_a^0 in S_0 will clearly become a proof of D_a in S when the notation is thus changed (it will make no difference to any step in the proof whether we are using the predicate symbols or the respective predicate letters). Thus[196]

$$(4) \qquad \{\vdash D_a^0 \text{ in } S_0\} \rightarrow \{\vdash D_a \text{ in } S\}.$$

The converse is not quite so simple, since a given proof of D_a in S might use some predicate letters other than $P_0, \ldots, P_k$. (In S we have available infinitely many predicate letters with each number of arguments, but in S_0 only the $k+1$ predicate symbols $=, F_1, \ldots, F_k$.) However a proof of D_a in S will remain one when the atoms in it formed using other predicate letters are changed to $\forall x P_1(x)$. The resulting proof of D_a in S will become one of D_a^0 in S_0 when $P_0, \ldots, P_k$ are changed to $=, F_1, \ldots, F_k$. Thus[196]

$$(5) \qquad \{\vdash D_a \text{ in } S\} \rightarrow \{\vdash D_a^0 \text{ in } S_0\}.$$

[195] The application of the results cited in Footnote 156 § 38 to replace a function symbol by a predicate symbol is discussed in general in Hilbert and Bernays 1934 pp. 460–467 and IM pp. 417–419; and for the case of multiplication · (but not for exactly the system S_4) in IM Example 11 p. 419.

[196] The implications (4) and (5) constitute simple applications of the proof-theoretic substitution rule cited in the second paragraph of Footnote 86 § 25.

Combining (4) and (5) with (3_0),

(6) $$\{\vdash D_a \text{ in } \mathbf{S}\} \equiv (Ex)T(a, a, x).$$

(γ) Now $\{\vdash D_a \text{ in } \mathbf{S}\} \equiv \{\forall A_1, \ldots, \forall A_m \vdash D_a \text{ in } \mathbf{Pd}\}$ (similarly to (A) § 38 p. 208) $\equiv \{\vdash \forall A_1 \supset (\forall A_2 \supset \ldots (\forall A_m \supset D_a) \ldots) \text{ in } \mathbf{Pd}\}$ (by Corollaries Theorems 11_{Pd} and 10_{Pd}). Using this in (6),

(7) $\{\vdash \forall A_1 \supset (\forall A_2 \supset \ldots (\forall A_m \supset D_a) \ldots) \text{ in } \mathbf{Pd}\} \equiv (Ex)T(a, a, x).$

Now we complete the proof, as we planned, by applying Theorem III. Thus, suppose there were a decision procedure for provability in **Pd**. Then, given a, we could decide as follows whether $(Ex)T(a, a, x)$ is true or false, contradicting Theorem III. From a, find the formula $\forall A_1 \supset (\forall A_2 \supset \ldots (\forall A_m \supset D_a) \ldots)$. Apply the supposed decision procedure for provability in **Pd** to the question whether this formula is provable. By (7), according as the answer is "yes" or "no", $(Ex)T(a, a, x)$ is true or false. —

An interesting new proof of Theorem VII (presupposing ideas given here in Chapter VI) is in Büchi 1962.

The undecidability results made possible by the Church-Turing thesis (§ 41) arose first (like Theorem III) directly in connection with the new notions of λ-definability, general recursiveness and Turing computability, and next (like Theorems IV and VII) for decision problems for formal systems.

Church on May 19, 1936, shortly after enunciating his thesis and obtaining his first results (Theorems III, IV and VII, particularly), wrote the author, "What I would really like to see done would be my results or yours used to prove the unsolvability of some mathematical problems of this order not on their face specially related to logic." This hope was fulfilled, beginning in 1947 when Post and A. A. Markov (the younger) independently of each other showed the "word problem for semi-groups" to be unsolvable.[197] This led to the unsolvability of the "word problem for semi-groups with cancellation" in Turing 1950 (with Boone 1958), and thence to the unsolvability of the "word problem for groups" in a 143-page article by Novikov 1955.[198] Simpler proofs of Novikov's result have since been given from other directions by Boone in 1957 of 1954–7, and in 1959, and by Britton in 1958 with 1956–8, and in 1963, and (as a corollary of another theorem) by Higman in 1961.[199] In 1958 Markov established the unsolvability of the "homeomorphism problem for four-dimensional manifolds" in topology. This is reworked and extended in Boone, Haken and

[197] A treatment is given in IM § 71.

[198] For the problem, see Dehn 1912 p. 117.

[199] Britton's 1963 proof has been worked into a textbook: Rotman 1965 Chapter 12.

Poénaru **1967**. Other major publications in this area include Rabin 1958, Clapham **1964**, Shepherdson **1965**, Boone **1966**, **1966a** and **1967** (with full bibliography). In the papers cited in bold face, the inquiry is refined to consider not just solvability vs. unsolvability but also "degrees of unsolvability" (§ 46).

Unsolvability results have begun to appear in real-variable analysis: Scarpellini 1963; Richardson 1966 abstract (which treats a variant of the integrability problem of Exercise 40.1 (k)).

Undecidability results have also been obtained in connection with grammatical problems for languages for use with computing machines and finite automata: Rabin and Scott 1959; Bar-Hillel, Perles and Shamir 1961.

***§ 46. Degrees of unsolvability (Post), hierarchies (Kleene, Mostowski).** We can summarize our proofs of Theorems IV and VII as follows. First, in Theorem III, we established the undecidability of the predicate $(Ex)T(a, a, x)$. Then we "reduced" the decision problem (P) for this predicate to the decision problem (Q) for provability in **N** or in the predicate calculus **Pd**.

To say this in more detail, take the case of **Pd** (Theorem VII). We showed that, if we had a way to answer any question of the class (Q) "Is a given formula E provable in **Pd**?", then we could answer any question of the class (P) "Does a given natural number a have the property that $(Ex)T(a, a, x)$?". But in Theorem III we had shown from the Turing machine concept with the Church-Turing thesis that there can be no algorithm to answer all questions of the class (P). Therefore there can be none to answer all questions of the class (Q).

We reduced (P) to (Q), because (P) is the class of questions we could show directly to be undecidable. However, it is of some interest to see that, inversely, (Q) can be reduced to (P). This is included in the following proposition (and similarly with **N** in place of **Pd**).

(A) *To each predicate of the form* $(Ex_1) \ldots (Ex_m)R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ *where* $R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ *is a given decidable predicate* $(m, n > 0)$, *there is a computable function* $\theta(a_1, \ldots, a_n)$ *such that*

$$(Ex_1) \ldots (Ex_m)R(a_1, \ldots, a_n, x_1, \ldots, x_m) \equiv (Ex)T(\theta(a_1, \ldots, a_n), \theta(a_1, \ldots, a_n), x).$$

Similarly for $n = 0$, *with* $\theta(a_1, \ldots, a_n)$ *becoming simply a natural number* h.

(Proof follows.) Thus, any question of the class "Is $(Ex_1) \ldots (Ex_m)R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ true?" is reduced to answering a corresponding question of the class "Is $(Ex)T(b, b, x)$ true?", namely the

question with $b = \theta(a_1, \ldots, a_n)$ (where, since θ is computable, from any $a_1, \ldots, a_n$ we can effectively find the corresponding b).[200]

To prove (A), take any fixed $a_1, \ldots, a_n$, and consider the following intuitive computation procedure. First, suppose all m-tuples of natural numbers $(x_1, \ldots, x_m)$ have been enumerated, i.e. brought into an infinite list (§ 32). Now, using a Turing machine (supposed given) which decides the predicate $R(a_1, \ldots, a_n, x_1, \ldots, x_m)$, test the m-tuples $(x_1, \ldots, x_m)$ in the order listed, searching for one which makes $R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ true, and write 0 if such an m-tuple is found, but continue the search ad infinitum otherwise.

Now we claim, and the reader will have to take our word for the details, that a Turing machine can be constructed which, applied to *any* number b as argument, carries out the above process and writes 0 when an m-tuple $(x_1, \ldots, x_m)$ is found for which $R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ is true, but otherwise never computes a value. What machine we construct depends on the numbers $a_1, \ldots, a_n$ we started with. We further claim that these various machines can be constructed so that their indices can be given by a Turing computable function $\theta(a_1, \ldots, a_n)$. Thus, by what the machines do and the definition of the predicate $T(i, a, x)$ in § 42,

$$(Ex_1) \ldots (Ex_m)R(a_1, \ldots, a_n, x_1, \ldots, x_m) \equiv (Ex)T(\theta(a_1, \ldots, a_n), b, x)$$

for all $a_1, \ldots, a_n, b$. Taking $b = \theta(a_1, \ldots, a_n)$ in this gives the equivalence in (A).

To apply (A) to see that the decision problem (Q) for provability in the predicate calculus **Pd** (or in **N**) can be reduced to that (P) for $(Ex)T(a, a, x)$, we may first give Gödel numbers to the formulas in the predicate calculus in an effective way (cf. § 43 Discussion). Now "A is provable" can be expressed as $(Ex)\mathrm{Pr}(a, x)$ where a is the Gödel number of A, and $\mathrm{Pr}(a, x)$ is the decidable predicate which says that x is the Gödel number of a proof of the formula with the Gödel number a. Now (A) applies with Pr as the R (with $n = m = 1$).

Summarizing, by (A) with the proofs of Theorems IV and VII, each of our three examples of undecidable predicates (or unsolvable decision problems) is reducible to each of the others.

In Cantor's set theory, when we began comparing infinite sets under one-to-one correspondences, we discovered that not all infinite sets are

[200] Post in 1944 gave the first result of this sort, using a different predicate than $(Ex)T(b, b, x)$. (The earlier sections of Post 1944 are less technical than most papers written in this area.) The result for a predicate in the theory of general recursive functions analogous to the present $(Ex)T(b, b, x)$ appeared in IM p 343. Slightly stretching Post's terminology, $(Ex)T(b, b, x)$ can be called a *complete* predicate for the class of predicates $(Ex_1) \ldots (Ex_m)R(a_1, \ldots, a_n, x_1, \ldots, x_m)$.

equally numerous; indeed, we found a hierarchy of increasing cardinal numbers. Analogously, the question arises now whether all undecidable number-theoretic predicates are equally undecidable, in the sense that the decision problem for any one of them is reducible to the decision problem for any other. As in set theory, the answer is negative; indeed, some predicates are "more undecidable", or (have decision problems) of "higher degree of unsolvability", than others.

So far we have only considered some particular pairs of undecidable predicates which are equally undecidable (the decision problem for either one being reducible in a simple way to that for the other). Now we need a more general notion of when one predicate (or its decision problem) is reducible to another. This requires an extension of Church's thesis or the Church-Turing thesis to decidability or computability relative to a given predicate $Q(b)$.

The concept that we use here is due to Turing 1939, and its use to define "degree (of unsolvability)" to Post 1948 abstract (with some anticipation in 1944).[201] Consider a "Q-machine", which is like an ordinary or "absolute" Turing machine, except that it has access to a second (actually) infinite tape (a "Q-tape") on which are printed the answers to all the questions (for $b = 0, 1, 2, \ldots$) whether $Q(b)$ is true or false. (Turing instead spoke of a machine having access to an "oracle" that would answer any such question the machine asks it about $Q(b)$.) If a suitable Q-machine can answer all the questions (for $a = 0, 1, 2, \ldots$) whether $P(a)$ is true or false, we say (the decision problem for) $P(a)$ is (*Turing*) *reducible* to (the decision problem for) $Q(b)$. If $P(a)$ is reducible to $Q(b)$ and vice versa, we say $P(a)$ and $Q(b)$ (or their decision problems) are of the same *degree* (*of unsolvability*). If $P(a)$ is reducible to $Q(b)$ but not vice versa, we say $P(a)$ is of *lower degree* than $Q(b)$ or $Q(b)$ is of *higher degree* than $P(a)$. Hence, if $P(a)$ is reducible to $Q(b)$, $P(a)$ is of the same or lower degree than $Q(b)$.

Here lower, equal or higher degree between predicates $P(a)$ and $Q(b)$ corresponds to less, equally or more numerous between sets M and N. To put the theory in better form, we should define degrees themselves and not just use the word "degree" to express the above relationships between predicates $P(a)$ and $Q(b)$. Here we can use the same method as in defining "cardinal number" by the Frege-Russell method in § 34. We begin by verifying (Exercise 46.1) that the above "equal-degree" relation "$P(a)$ is reducible to $Q(b)$ and vice versa" is an equivalence relation, i.e. it is reflexive, symmetric and transitive (like the "equally-numerous" relation "M can be put into 1-1 correspondence with N", § 34). So we can define

[201] In IM pp. 314–315 the treatment is based equivalently on the theory of general recursive functions (relativized by Kleene 1943), instead of on Turing computability.

the *degree* of any number-theoretic predicate $P(a)$ to be the equivalence class to which $P(a)$ belongs under this equivalence relation (§ 30). Then to justify the above definition of when one predicate $P(a)$ is of lower degree than another $Q(b)$, which now becomes "the degree of $P(a)$ is $<$ the degree of $Q(b)$", we need to verify (Exercise 46.2) that the result is independent of what particular predicates $P(a)$ and $Q(b)$ are used from their respective equivalence classes.

The relation $<$ between degrees is irreflexive and transitive (Exercise 46.3).

To keep the notation simple, we have spoken just now only of one-place number-theoretic predicates $P(a)$ and $Q(b)$. But the reducibility notion applies likewise to number-theoretic predicates or functions of any numbers ≥ 1 of arguments; and the equivalence classes can be taken in this larger totality.

All the questions about particular values of a decidable predicate $P(a)$ can be answered by a suitable absolute Turing machine. So a suitable Q-machine can answer all those questions, without looking at its Q-tape. Hence each decidable predicate $P(a)$ is of degree $\leq$ the degree of each predicate $Q(b)$. The degree of any decidable predicate we write $\mathbf{0}$ (since each two decidable predicates are reducible each to the other, the decidable predicates do constitute a degree); this we have just seen is the lowest degree of unsolvability ("solvability").

By Theorem III, the predicate $(Ex)T(a, a, x)$ cannot be decided by an absolute Turing machine. So $(Ex)T(a, a, x)$ cannot be decided by a Q-machine when $Q(b)$ is a decidable predicate. For, the values of $Q(b)$ on the Q-tape would give the Q-machine only unnecessary help; a suitable absolute machine could manufacture those values for itself.

Hence $(Ex)T(a, a, x)$ is of higher degree than the decidable predicates. We write $\mathbf{0}'$ $(= \mathbf{1})$ for the degree of $(Ex)T(a, a, x)$. Thus $\mathbf{0}' > \mathbf{0}$.

Now it is easy to see how to proceed to higher degrees than $\mathbf{0}'$. Though we have not spelled out exactly how the Q-tape of a Q-machine is used, it should be fairly obvious that we can do so, and then construct a predicate $T^Q(i, a, x)$ (relative to a given predicate $Q(b)$) which plays the role for Q-machines which $T(i, a, x)$ plays for absolute machines. Then (similarly to Theorem I (A)) $T^Q(i, a, x)$ is decidable by a Q-machine, and $\varphi_i^Q(a)$ as a partial function of i and a is computable by a Q-machine.

By using a Q-machine instead of an absolute machine in the proof of (A), we can establish:

(B) *For any predicate Q,* (A) *holds when $R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ is allowed to be any predicate decidable by a Q-machine, and $T(a, a, x)$ is replaced by $T^Q(a, a, x)$.*

Similarly, paralleling with Q-machines the proofs of Theorems II and III, $(Ex)T^Q(a, a, x)$ is not decidable by a Q-machine. But since $Q(a) \equiv (Ex)(Q(a) \,\&\, x=x)$, by (B) Q is reducible to $(Ex)T^Q(a, a, x)$. Thus $(Ex)T^Q(a, a, x)$ is of higher degree than Q. We can state this result in the following form.

(C) *If $Q(b)$ is a predicate of degree* $\mathbf{d}$, *then $(Ex)T^Q(a, a, x)$ is of degree* $\mathbf{d}' > \mathbf{d}$.

Here by using the notation $\mathbf{d}'$ we include that the degree of $(Ex)T^Q(a, a, x)$ depends only on the degree $\mathbf{d}$ of Q; and furthermore that, when Q *is* decidable, the degree of $(Ex)T^Q(a, a, x)$ is the same as the degree of $(Ex)T(a, a, x)$, which we originally designated as $\mathbf{0}'$. These additional facts are easily proved using (B) and (A) (Exercise 46.4).

Using (C) repeatedly, we get a succession of increasing degrees

$$\mathbf{0} < \mathbf{0}' < \mathbf{0}'' < \ldots < \mathbf{0}^{(n)} < \ldots.$$

It is easy to see that:

(D) *If $P_0(a)$, $P_1(a)$, $P_2(a)$, . . . are predicates of increasing degrees, and $P(n, a) \equiv P_n(a)$, then $P(n, a)$ as a 2-place predicate is of higher degree than any of $P_0(a)$, $P_1(a)$, $P_2(a)$, . . .*.

(C) and (D) play a role in generating increasing degrees like (C) and (D) in § 34 in generating increasing cardinal numbers. This method of generating number-theoretic predicates of increasing degrees is due to Davis 1950, and independently to Kleene and to Post.

Hierarchies of number-theoretic predicates were first discovered from another point of view by Kleene in 1943 (1940 abstract), and independently (with a somewhat different method) by Mostowski in 1947. Using Kleene's approach, we first establish:

(E) *To each decidable predicate $R(a, x)$, there is a number f such that*

$$(Ex)R(a, x) \equiv (Ex)T(f, a, x).$$

Similarly, to each decidable predicate $R(a_1, \ldots, a_n, x)$, there is a number f such that

$$(Ex)R(a_1, \ldots, a_n, x) \equiv (Ex)T(f, a_1, \ldots, a_n, x)$$

where $T(i, a_1, \ldots, a_n, x)$ plays the same role for the Turing machine computation of n-place functions as $T(i, a, x)$ for 1-place functions. (Enumeration theorem.)[202]

[202] $(Ex)T(0, a, x)$, $(Ex)T(1, a, x)$, $(Ex)T(2, a, x)$, . . . is an enumeration with repetitions of all predicates of the form $(Ex)R(a, x)$ with $R(a, x)$ decidable (the "enumerating predicate" $(Ex)T(i, a, x)$ being of the same form except for the necessary additional variable i).

To prove (E) for the case of one variable a, we need only let $\mathfrak{M}_f$ be a Turing machine which, applied to a, searches for an x such that $R(a, x)$ and writes 0 if one is found, but computes no value otherwise.

(F_1) *The predicate* $(x)\bar{T}(a, a, x)$ *is not expressible in the form* $(Ex)R(a, x)$ *with* $R(a, x)$ *decidable. The predicate* $(Ex)T(a, a, x)$ *is not expressible in the form* $(x)R(a, x)$ *with* $R(a, x)$ *decidable. A fortiori,* $(x)\bar{T}(a, a, x)$ *and* $(Ex)T(a, a, x)$ *are not decidable.*

For, suppose to the contrary that $(x)\bar{T}(a, a, x) \equiv (Ex)R(a, x)$ for some decidable predicate $R(a, x)$. Then
$\overline{(x)}\bar{T}(a, a, x) \equiv \overline{(Ex)}R(a, x) \equiv \overline{(Ex)}T(f, a, x)$ (using (E)) $\equiv (x)\bar{T}(f, a, x)$.
Substituting f for a, $\overline{(x)}\bar{T}(f, f, x) \equiv (x)\bar{T}(f, f, x)$, which is absurd
($\vdash \neg(\neg P \sim P)$ in the propositional calculus).

Similarly, suppose that $(Ex)T(a, a, x) \equiv (x)R(a, x)$ for some decidable $R(a, x)$. Then $(Ex)T(a, a, x) \equiv (x)\bar{\bar{R}}(a, x) \equiv \overline{(Ex)}\bar{R}(a, x) \equiv \overline{(Ex)}T(f, a, x)$ (using (E) for $\bar{R}(a, x)$ as its $R(a, x)$). Substituting f for a,
$(Ex)T(f, f, x) \equiv \overline{(Ex)}T(f, f, x)$, which is absurd.

To infer that $(Ex)T(a, a, x)$ is not decidable, suppose that
$(Ex)T(a, a, x) \equiv R(a)$ with $R(a)$ decidable. Then
$(Ex)T(a, a, x) \equiv (x)(R(a) \,\&\, x=x)$, which contradicts the second part of (F_1), since $R(a) \,\&\, x=x$ is decidable. Similarly or by $(x)\bar{T}(a, a, x) \equiv \overline{(Ex)}T(a, a, x)$, $(x)\bar{T}(a, a, x)$ is not decidable.

(F_2) $(Ex)(y)\bar{T}(a, a, x, y)$ *is not expressible in the form* $(x)(Ey)R(a, x, y)$ *with* $R(a, x, y)$ *decidable.* $(x)(Ey)T(a, a, x, y)$ *is not expressible in the form* $(Ex)(y)R(a, x, y)$ *with* $R(a, x, y)$ *decidable. A fortiori,* $(Ex)(y)\bar{T}(a, a, x, y)$ *and* $(x)(Ey)T(a, a, x, y)$ *are not expressible using one or zero quantifiers applied to a decidable predicate.*

This is proved similarly to (F_1). Likewise we have propositions (F_3), (F_4), (F_5), . . . concerning the predicate forms with 3, 4, 5. . . . quantifiers (alternating between existential and universal) applied to decidable predicates.[203]

(F_1), (F_2), (F_3), . . . are summarized in the hierarchy theorem of Kleene 1943 (1940 abstract):

[203] Since $R(a, x)$ is decidable, $R(a, x) \equiv \bar{\bar{R}}(a, x)$ holds intuitionistically. Also $\overline{(Ex)}T(f, a, x) \equiv (x)\bar{T}(f, a, x)$ and $(x)\bar{\bar{R}}(a, x) \equiv \overline{(Ex)}\bar{R}(a, x)$ are (informal) applications of *82a, which holds intuitionistically. Thus our proof of (F_1) is good using only intuitionistic logic. Classical logic is required for (F_2), (F_3), (F_4), ...

(F) *Consider the predicate forms*

$$R(a)\quad \begin{matrix} (Ex)R(a,x) & (x)(Ey)R(a,x,y) & (Ex)(y)(Ez)R(a,x,y,z) & \dots \\ (x)R(a,x) & (Ex)(y)R(a,x,y) & (x)(Ey)(z)R(a,x,y,z) & \dots \end{matrix}$$

where in each form R is a decidable predicate. To each form after the first, there is a predicate expressible in that form but not in the form dual to it (i.e. arising from it by interchanging existential and universal quantifiers) nor in any of the forms with fewer quantifiers.

Using a theorem of Post 1948 abstract (IM Theorem XI p. 293), it can be shown that the degrees of a decidable predicate and of the predicates $(Ex)T(a,a,x)$, $(x)(Ey)T(a,a,x,y)$, $(Ex)(y)(Ez)T(a,a,x,y,z)$, . . . are indeed $\mathbf{0}, \mathbf{0}', \mathbf{0}'', \mathbf{0}''', \dots$, the same as those of the predicates obtained by starting with a decidable predicate and using (C) repeatedly.

Consider a formal system N (like that of § 38) in which, to each a, a formula can be found which expresses $(x)\bar{T}(a,a,x)$; indeed since $(x)\bar{T}(a,a,x) \equiv \overline{(Ex)}T(a,a,x)$, this formula can be $\neg C_a$ for the C_a of § 43. Say the Gödel number of $\neg C_a$ is $\alpha(a)$ where α is a computable function. Using the predicate $\mathrm{Pr}(a,x)$ for N mentioned early in this section, $\{\vdash \neg \mathrm{C}_a \text{ in } \mathrm{N}\} \equiv (Ex)\mathrm{Pr}(\alpha(a),x)$. Here $\mathrm{Pr}(\alpha(a),x)$ is a decidable predicate of a and x; write it simply $R(a,x)$. Thus

$$\text{(e)}\qquad \{\vdash \neg \mathrm{C}_a \text{ in } \mathrm{N}\} \equiv (Ex)R(a,x).$$

Now it is immediate from the first part of (F_1) that N cannot be both correct and complete, so that $\neg \mathrm{C}_a$ is provable in N if and only if $\neg \mathrm{C}_a$ is true, i.e. so that

$$(\overset{*}{*})\qquad \{\vdash \neg \mathrm{C}_a \text{ in } \mathrm{N}\} \equiv (x)\bar{T}(a,a,x).$$

For, combining $(\overset{*}{*})$ with (e) would give $(x)\bar{T}(a,a,x) \equiv (Ex)R(a,x)$, contradicting (F_1). Thus $(\overset{*}{*})$ cannot hold for all a. Assuming that N is correct,

$$\text{(c)}\qquad \{\vdash \neg \mathrm{C}_a \text{ in } \mathrm{N}\} \rightarrow (x)\bar{T}(a,a,x),$$

which is one of the implications in $(\overset{*}{*})$. So the other

$$(*)\qquad (x)\bar{T}(a,a,x) \rightarrow \{\vdash \neg \mathrm{C}_a \text{ in } \mathrm{N}\}$$

cannot hold for all a. Since $(x)\bar{T}(a,a,x) \equiv \overline{(Ex)}T(a,a,x)$, this (c) and (*) are the same as in § 43, and by the correctness of N we again have (b). Thus (continuing as in the first proof of Theorem V), Gödel's incompleteness theorem, for the system N of § 38 or any system N that is correct and meets the very general structural condition expressed by (e) holding for some decidable R, is implicit in the first part of (F_1).

This uses classical logic to infer the existence of a value p of a for which (*) is false. (Hierarchy theory is largely classical.) If (instead of assuming (**) for reductio ad absurdum) we apply (E) (as we did in proving (F_1)), we get a number f such that

$$(f) \qquad (Ex)R(f, x) \equiv (Ex)T(f, f, x).$$

Now Theorem V (i)–(iii) with f as the p follow intuitionistically from (b), (c), (e), (f).

Church's theorem (Theorem III) is the third part of (F_1).

In brief, Church's and Gödel's theorems correspond to the two forms $R(a)$ and $(Ex)R(a, x)$ in (F). From this point of view (emphasized in Kleene 1943), the hierarchy theorem (F) can be regarded as a generalization of Church's and Gödel's theorems. Under the other construction using (C), the hierarchy results by the iteration of Church's theorem in a relativized version.[204]

Post in 1944 raised the question whether any predicate of the form $(Ex)R(a, x)$ with R decidable has a degree (strictly) between $\mathbf{0}$ and $\mathbf{0}'$. In 1954, Kleene and Post showed that there exist predicates having degrees between $\mathbf{0}$ and $\mathbf{0}'$; but their method did not show whether any of those predicates are of the form $(Ex)R(a, x)$ with R decidable. In 1956 Friedberg (U.S.A.) then only 20 years old and Mučnik (U.S.S.R.) of similar age, independently of each other, refined the Kleene-Post construction to show that there is a predicate of the form $(Ex)R(a, x)$ with R decidable whose degree is between $\mathbf{0}$ and $\mathbf{0}'$, solving Post's 1944 problem.[205]

There are incomparable degrees (i.e. degrees $\mathbf{a}$ and $\mathbf{b}$ such that neither $\mathbf{a} < \mathbf{b}$ nor $\mathbf{a} = \mathbf{b}$ nor $\mathbf{a} > \mathbf{b}$). The totality of all the degrees possessed by number-theoretic predicates, including degrees jumped over or bypassed in the hierarchies described above, has a very complicated structure.[206]

EXERCISES. 46.1. Show that the relation "$P(a)$ is reducible to $Q(b)$ and vice versa" is reflexive, symmetric and transitive.

46.2. Show that, if $P_1(a)$ and $Q_1(b)$ have the same respective degrees as $P(a)$ and $Q(a)$, then $\{P(a)$ is reducible to $Q(b)$ but not vice versa$\} \equiv \{P_1(a)$ is reducible to $Q_1(b)$ but not vice versa$\}$.

46.3. Prove that for any degrees $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$: (a) Not $\mathbf{a} < \mathbf{a}$. (b) If $\mathbf{a} < \mathbf{b}$ and $\mathbf{b} < \mathbf{c}$, then $\mathbf{a} < \mathbf{c}$.

[204] For a little more of an indication of hierarchy theory, cf. Kleene 1958 §2. Mostowski 1954 and Kleene 1955b give fuller but more technical expositions of results to that time.

[205] Friedberg 1956 article concerning, 1957 (abstract 1956); Mučnik 1956, 1958.

[206] The first results to this effect were in Kleene and Post 1954. Much work has been done since by Spector, Lacombe, Shoenfield, Sacks and others. Cf. Sacks 1963.

46.4. Show that: (a) If Q_1 and Q_2 are of the same degree, so are $(Ex)T^{Q_1}(a, a, x)$ and $(Ex)T^{Q_2}(a, a, x)$. (b) If Q is of degree $\mathbf{0}$, then $(Ex)T^{Q}(a, a, x)$ is of the same degree as $(Ex)T(a, a, x)$.

46.5. Prove a statement like (E) with universal instead of existential quantifiers.

46.6. Show that for any decidable R,
$(x_1) \ldots (x_m)R(a_1, \ldots, a_n, x_1, \ldots, x_m)$ is of degree $\leq \mathbf{0}'$.

46.7. Establish (D) and (F_2).

46.8*. Show that each of the predicates expressible in the symbolism of N § 38 under the usual interpretation (which were called "arithmetical" by Gödel 1931) is expressible in one of the forms of (F).[207]

46.9*. Assume the fact that every decidable predicate is expressible in the symbolism of N § 38.[208] Thence establish:
(a) The converse of Exercise 46.8.
(b) For any fixed effective Gödel numbering of the formulas of N (end § 43), the predicate "a is the Gödel number of a true closed formula of N" is not expressible in the symbolism of N.

*** § 47. Undecidability and incompleteness using only simple consistency (Rosser).** For Theorems IV–VI, we made two non-elementary assumptions (b) and (c) about the system N, because they make the approach to the theorems easy, and because we can hardly doubt that they hold for the N of § 38 or any system we would want to use instead.

However, to formulate Theorems IV–VI as elementary metamathematical theorems, (b) and (c), or something to the same effect, should be included in the hypotheses. This we have done in the second paragraph of each theorem.

Theorem VI followed from Theorem V by noting that (c) can be replaced by the simple consistency and (a).

There remains (b), which we used in proving Theorems IV and V (iii). Consider any N (like that of § 38) in which C_a is of the form $\exists x T(a, a, x)$ where $T(a, a, x) \rightarrow \{\vdash T(a, a, x) \text{ in N}\}$, whence (a), and

(g) $$\overline{T}(a, a, x) \rightarrow \{\vdash \neg T(a, a, x) \text{ in N}\}.$$

We can then replace (b) in proving Theorem V (iii) by the hypothesis that N is "ω-consistent" in the following sense, due to Gödel 1931. A system S whose notation includes the numerals (§ 43) is *ω-consistent* if in it, for no variable x and formula A(x), do all of $\vdash A(0)$, $\vdash A(1)$, $\vdash A(2)$, . . . and

[207] Solution in IM p. 285 Theorem VII (d), using general recursiveness instead of Turing computability.

[208] Included in IM p. 285 Theorem VII (b). Using this to generalize Exercise 46.8 to allow symbols for any decidable predicates, we obtain the proposition which led Kleene in 1940 abstract to consider exactly the list of predicate forms in (F).

$\vdash \neg\forall x A(x)$ hold (otherwise, S is *ω-inconsistent*). (An ω-consistent system S is simply consistent, as follows by writing any formula E as A(x) where x is a variable not occurring in E, so that A(0), A(1), A(2), . . . are all E and $\neg\forall x A(x)$ is equivalent to $\neg E$ by *75.) To prove (iii) from the ω-consistency (with (a), (d) and (g)), suppose $\vdash C_p$, i.é. $\vdash \exists x T(p, p, x)$, whence $\vdash \neg\neg\exists x T(p, p, x)$, whence (using *82a) $\vdash \neg\forall x \neg T(p, p, x)$. But by (i) (already proved in § 44 from (a), (d) and the simple consistency), $\overline{(Ex)}T(p, p, x)$, whence $(x)\bar{T}(p, p, x)$, whence by (g) $(x)\{\vdash \neg T(p, p, x)\}$, i.e. $\vdash \neg T(p, p, 0)$, $\vdash \neg T(p, p, 1)$, $\vdash \neg T(p, p, 2)$, ... These with $\vdash \neg\forall x \neg T(p, p, x)$ above contradict the hypothesis of ω-consistency.[209]

Clearly, an ω-inconsistent system violates the interpretation of x as ranging over the natural numbers, expressed by the numerals 0, 1, 2, ...

It is a consequence of Theorem V (ii) that there are simply consistent but ω-inconsistent systems of number-theory. (In substance, this is Exercise 44.3, which we do now.) For, let N be the number-theoretic system of § 38, whose consistency we assume (on the basis of the interpretation or of Gentzen's proof). Let M come from N by adding C_p as an axiom. Then, by arguments just given (using (g) and Theorem V (i) for N), M is ω-inconsistent. But M is simply consistent; for, if $\vdash E$ and $\vdash \neg E$ in M, then $C_p \vdash E$ and $C_p \vdash \neg E$ in N, whence by ¬-introd., $\vdash \neg C_p$ in N, contradicting Theorem V (ii).

Thus not only does Gödel's second theorem show that simple consistency is hard to prove for number theory, but also his first theorem shows that simple consistency is not the only consistency property that we will ordinarily wish our formal systems for number theory to have. The proofs by

[209] Using (g), ω-consistency implies

(b′) $$\{\vdash C_a \text{ in N}\} \rightarrow \overline{\overline{(Ex)}}T(a, a, x),$$

and hence classically (b). For, assume $\vdash C_a$ and $\overline{(Ex)}T(a, a, x)$. Thence $(x)\bar{T}(a, a, x)$ whence by (g): (A) $(x)\{\vdash \neg T(a, a, x)\}$. But also (as for $a = p$ in the text): (B) $\vdash \neg\forall x \neg T(a, a, x)$. But ω-consistency says (A) and (B) can't both hold; so by reductio ad absurdum $\overline{\overline{(Ex)}}T(a, a, x)$.

The proof of Theorem IV in § 43 can be reworked to proceed intuitionistically from (b′) instead of (b), thus. Suppose there were a decision procedure for provability in N. We could use it to construct a machine $\mathfrak{M}_p$ which, applied to a, attempts (successfully if $(Ex)T(a, a, x)$) to compute $\varphi_a(a)+1$ if $\vdash C_a$ (Case 1), and writes 0 if not $\vdash C_a$ (Case 2). Then as for Theorem II we can deduce a contradiction from $(Ex)T(p, p, x)$ (after using (a) to infer $\vdash C_p$), and also from $\overline{(Ex)}T(p, p, x)$ (after using (b′) to infer not $\vdash C_p$). So by informal use of weak ¬-elim. § 11: (C) $(Ex)T(p, p, x) \rightarrow 0 \neq 0$,

(D) $\overline{(Ex)}T(p, p, x) \rightarrow 0 \neq 0$. From (C) by contraposition twice (*13, *12 in § 24),

(E) $\overline{\overline{(Ex)}}T(p, p, x) \rightarrow 0 \neq 0$. Now we get a contradiction by cases (as above, for $a = p$). In Case 1, by (b′) $\overline{\overline{(Ex)}}T(p, p, x)$, whence by (E) $0 \neq 0$, contradicting $0 = 0$. In Case 2, by (a) and contraposition $\overline{(Ex)}T(p, p, x)$, whence by (D) $0 \neq 0$

Gentzen etc. for N § 38 do actually establish ω-consistency. At the same time, the problem of interpretation, mentioned at the end of § 44, is emphasized, because we have no reason to stop at securing only simple and ω-consistency.

Meanwhile, in 1936 Rosser found another method of proof of Theorem IV, and of Theorem V with a different number q in place of p, in which all the results are obtained with simple consistency as the only nonelementary hypothesis. We shall obtain these results now as corollaries of a more general theorem (Theorem VIII).

We call a non-empty set or class S of natural numbers *computably enumerable* or *recursively enumerable* if there is a computable function φ such that $\varphi(0), \varphi(1), \varphi(2), \ldots$ is an enumeration (possibly with repetitions) of S.[210] Two sets S_0 and S_1 are *disjoint* if they have no common elements, or in symbols if $S_0 \cap S_1 = \varnothing$ (cf. § 26).

THEOREM VIII. *There are two (non-empty) disjoint recursively enumerable sets C^0 and C^1 with the following property. Given any two disjoint recursively enumerable sets D^0 and D^1 including C^0 and C^1 respectively, i.e. any two recursively enumerable sets D^0 and D^1 such that*

$$(1)\quad D^0 \cap D^1 = \varnothing,$$

(2^0) $C^0 \subseteq D^0$, (2^1) $C^1 \subseteq D^1$,

a number f can be found which belongs to neither D^0 nor D^1, i.e. such that

(3^0) $f \notin D^0$, (3^1) $f \notin D^1$.

(A symmetric form of Gödel's theorem, Kleene 1950.)[211]

PROOF, using an idea of D. Lacombe (reported by Rabin 1958a Footnote 6). With $\varphi_i(a)$ as in § 42 and "$\hat{a}$" meaning "the a's such that" (§ 26), let

$$C^0 = \hat{a}\{\varphi_a(a) \text{ is defined and } \varphi_a(a) = 0\},$$

$$C^1 = \hat{a}\{\varphi_a(a) \text{ is defined and } \varphi_a(a) \neq 0\}.$$

Clearly C^0 and C^1 are disjoint (and non-empty).

[210] Such sets were first considered in Kleene 1936, using the general recursive functions (afterwards proved equivalent to the Turing computable functions, cf. § 41). The term "recursively enumerable" has become the standard one in the literature (Rosser 1936, Post 1944, Smullyan 1959, etc.). Cf. IM pp. 306–307. Also the empty set is often taken to be "recursively enumerable" (Post 1944).

[211] The sense in which this is a form of Gödel's incompleteness theorem is elaborated in IM § 61, which uses the original proof (Kleene 1950).

The sets C^0 and C^1 are disjoint recursively enumerable sets which are "recursively inseparable" in the following sense: for no general recursive set D, $C^0 \subseteq D$ & $C^1 \subseteq \bar{D}$ (where $\bar{D}$ = the compliment of D).

Suppose given recursively enumerable sets D^0 and D^1 satisfying (1), (2^0) and (2^1). Let a Turing machine $\mathfrak{M}_f$ be constructed (incorporating machines which compute functions φ^0 and φ^1 enumerating D^0 and D^1 respectively) which carries out the following operation. Applied to a, $\mathfrak{M}_f$ searches through the enumerations φ^0 and φ^1 of D^0 and D^1 respectively, looking for a. ($\mathfrak{M}_f$ works alternatively, some on φ^0 and some on φ^1, so that if the search is not stopped, it will eventually reach arbitrarily far out in each enumeration.) If $\mathfrak{M}_f$ finds a in the enumeration φ^0 of D^0, it thereupon writes 1 and stops. If $\mathfrak{M}_f$ finds a in the enumeration φ^1 of D^1, it thereupon writes 0 and stops. If neither of these events occurs, $\mathfrak{M}_f$ continues the search ad infinitum, and so computes no value.

To establish (3^0), assume that $f \in D^0$. Then by (1), $f \notin D^1$. So f is in the enumeration φ^0 of D^0, but not in the enumeration φ^1 of D^1. So $\mathfrak{M}_f$ applied to f will find f in the enumeration φ^0 (it cannot be prevented from finding f there by first finding f in the enumeration φ^1). So $\varphi_f(f)$ (the value computed by $\mathfrak{M}_f$ applied to f as argument) is defined, and $\varphi_f(f) = 1$ by the way $\mathfrak{M}_f$ operates. Hence by the definition of C^1, $f \in C^1$. Hence by (2^1) $f \in D^1$, contradicting $f \notin D^1$ (above). By reductio ad absurdum, $f \notin D^0$; i.e. (3^0) holds.

The proof of (3^1) is symmetric to that just given for (3^0). —

A formal system **S'** is an *extension* of a formal system **S**, and **S** is a *subsystem* of **S'**, if each formula of **S** is a formula of **S'** and each provable formula of **S** is a provable formula of **S'**. (Any formal system **S** is the "improper" extension of itself.)

COROLLARY 1. *Assume the system* **N** *of* § 38 *to be simply consistent. To any simply consistent extension* **N'** *of* **N** (*including* **N** *itself*), *there is a closed formula* C_f^0 *of* **N** *such that*: (i) $\neg C_f^0$ *is true*, (ii) *not* $\vdash \neg C_f^0$ *in* **N'**, (iii) *not* $\vdash C_f^0$ *in* **N'**.

More generally, this applies to any simply consistent formal system **N** *in which, to each a, there can be found effectively formulas* C_a^0 *and* C_a^1 (*expressing* $a \in C^0$ *and* $a \in C^1$ *respectively*) *such that* (a^0), (a^1), (b^0), (b^1) *below hold.*

PROOF. The same methods which enable us to find in **N** § 38 a formula C_a to express $(Ex)T(a, a, x)$ (beginning § 43) enable us now to find formulas C_a^0 and C_a^1 expressing $a \in C^0$ and $a \in C^1$ respectively. Indeed, let

$U(i, a, x) \equiv$ {Machine $\mathfrak{M}_i$, applied to a, at Moment x scans a square next to the right of a blank square}

(cf. § 41). A formula U(i, a, x) expressing $U(i, a, x)$ can be found, as before we found T(i, a, x) to express $T(i, a, x)$. Let

C_a^0 be $\exists x[T(a, a, x) \,\&\, U(a, a, x)]$, C_a^1 be $\exists x[T(a, a, x) \,\&\, \neg U(a, a, x)]$.

The same considerations which before gave (a) now give

(a^0) $a \in C^0 \to \{\vdash C_a^0 \text{ in } N\}$, (a^1) $a \in C^1 \to \{\vdash C_a^1 \text{ in } N\}$,

as can be proved in detail. Also, corresponding to the obvious disjointness of C^0 and C^1, it can be shown that

(b^0) $a \in C^0 \to \{\vdash \neg C_a^1 \text{ in } N\}$, (b^1) $a \in C^1 \to \{\vdash \neg C_a^0 \text{ in } N\}$.

Now let N′ be any simply consistent extension of N. Let

$$D^0 = \hat{a}\{\vdash C_a^0 \text{ in } N'\}, \qquad D^1 = \hat{a}\{\vdash \neg C_a^0 \text{ in } N'\}.$$

Then (1) of the theorem is satisfied, by the simple consistency of N′. Also (2^0) holds; for, if $a \in C^0$, then by (a^0) $\vdash C_a^0$ in N, and hence (since N′ is an extension of N) $\vdash C_a^0$ in N′, i.e. $a \in D^0$. Similarly, using (b^1), (2^1) holds.

Because of the nature of any formal system, here N′, Turing machines $\mathfrak{M}^0$ and $\mathfrak{M}^1$ can be found which compute $\varphi^0(n)$ and $\varphi^1(n)$, where $\varphi^0(n)$ is the a of the nth proof (in some enumeration of the proofs in N′) which is a proof of C_a^0 for some a, and $\varphi^1(n)$ likewise for $\neg C_a^0$. Thus D^0 and D^1 are recursively enumerable.

So all the hypotheses of the theorem are satisfied. Hence there is a number f satisfying (3^0) and (3^1). Then by the definitions of D^0 and D^1, (iii) and (ii) of the corollary hold. By (3^0) and (2^0), $f \notin C^0$; so (i) also holds.[212] —

We call a formal system S *essentially undecidable* (after Tarski 1949 abstract), if S is simply consistent, and every simply consistent extension of S (including S itself) is undecidable.

COROLLARY 2. *Assuming the system* N *of* § 38 *to be simply consistent, it is essentially undecidable.*

More generally, any formal system N *satisfying the second paragraph of Corollary* 1 *is essentially undecidable.*

PROOF. Assume N simply consistent, and let N′ be any simply consistent extension of N. Now we take

$$D^0 = \hat{a}\{\vdash C_a^0 \text{ in } N'\}, \qquad D^1 = \hat{a}\{\text{not } \vdash C_a^0 \text{ in } N'\}.$$

Now (1) is immediate. As before (Corollary 1), (2^0) holds by (a^0). Also (2^1) holds; for, if $a \in C^1$, then by (b^1) $\vdash \neg C_a^0$ in N and hence in N′, so by

[212] Symmetrically, (i)–(iii) hold with C_f^0 replaced by C_f^1 for a different choice of f. At least for the N of § 38, by using (A) in § 46 to write $a \in C^0$ or $a \in C^1$ in the form $(Ex)T(\theta(a), \theta(a), x)$, we have (i)–(iii) as they read in Theorem V but for p replaced by $q = \theta(f)$ with either of the present f's.

the simple consistency of N′ not $\vdash C_a^0$ in N′, i.e. $a \in D^1$. As before, D^0 is recursively enumerable.

Now assume that there is a decision procedure for provability in N′, i.e. there is a machine that decides, for any formula A, whether or not $\vdash$ A in N′. Using this machine, we could find a machine to compute a function φ^1 which enumerates the present D^1, so D^1 would also be recursively enumerable. So by the theorem there would be a number f such that $f \notin D^0$ and $f \notin D^1$. With the present D^0 and D^1, this is absurd.

APPLICATIONS TO THE DECISION PROBLEMS FOR AXIOMATIC THEORIES (TARSKI). Rosser's motivation in 1936 was presumably simply to strengthen Theorems IV and V by using the simple consistency of N in place of the ω-consistency. A fertile area of application of Corollary 2 has been cultivated since 1949 by Tarski and his coworkers.[213] This is to showing the undecidability of various axiomatic theories formalized in a logical calculus, which may be either the predicate calculus or the predicate calculus with equality. (Tarski uses the latter.) That is, the axioms of the theory are stated in the symbolism of the calculus, with predicate, (individual) and function symbols; and the calculus provides the logic. Examples of axiomatic theories thus formalized are the formal system N for number theory in § 38 (whose undecidability we already know) and the systems **G**, **Gp**, **AG**, **AGp** for groups and Abelian groups in § 39. As we remarked for N and **G**, in such systems the deduction theorem and our other introduction and elimination rules hold (Theorems 13, 21).

By the undecidability of a formalized theory or formal system **S** we mean, as above, that there is no decision procedure for answering all of the questions whether a given formula A is provable in the system **S** (formalizing the theory). As we saw in the proofs of Theorems IV and VII, and further in § 46, when we know the decision problem for one class of questions (P) to be unsolvable, we can infer the like for another class (Q) by reducing the questions in the first class to questions in the second (or briefly, by reducing the first decision problem to the second). In particular for formal systems, if $\mathbf{S}_2$ is undecidable, we can infer $\mathbf{S}_1$ to be undecidable if we can find effectively, to each formula B in $\mathbf{S}_2$, a formula B′ in $\mathbf{S}_1$ such that $\vdash$ B in $\mathbf{S}_2$ if and only if $\vdash$ B′ in $\mathbf{S}_1$.

A formal system **S** based on the predicate calculus (without or with equality) as the logic with additional or "nonlogical" axioms is said to be *finitely axiomatizable* (after Tarski 1949 abstract), if the number of those

[213] Theorem VIII itself answered a question concerning analogies between the hierarchies described in § 46 (Kleene 1943, Mostowski 1947, etc) and the hierarchies studied in "descriptive set theory" (Borel 1898, Lusin 1930, etc); cf. Kleene 1950, Addison 1960. Other applications are e.g. in Kleene 1956, Rabin 1958a, Kleene and Vesley 1965 pp. 112, 183.

nonlogical axioms is finite, or all but a finite set of them can be omitted without changing the class of the provable formulas; such a finite set of nonlogical axioms of S (or all, if S has only finitely many nonlogical axioms) we may call a *finite axiomatization of* S.

In the case of a system like N which is given with infinitely many nonlogical axioms (the 8 particular axioms 14–21 and the $\aleph_0$-axioms by the Axiom Schema 13), it may not be obvious a priori whether the system is finitely axiomatizable or not. (The author first heard this question asked about the N of § 38 during 1949; its solution, in the negative, was published by Ryll-Nardzewski in 1952.)[214]

The application of the reduction method to show a system S_1 undecidable depends on having available a system S_2 already known to be undecidable whose "theory" can be developed in S_1 (via some translation of formulas B in S_2 into formulas B′ in S_1). The simpler S_2 is, the better the chance we can do this in a given S_1.

Tarski 1949 abstract recognized that we will be in a particularly favorable position for doing this, if we have first of all *a system* S *which is both essentially undecidable and finitely axiomatizable*, besides being simple. For then *each system* S_1 (*with all the symbols which* S *has*) *is undecidable which has a simply consistent common extension* S_3 *with* S. To prove this, consider the system S_2 which has as its (nonlogical) axioms those of S_1 and those in a finite axiomatization of S (and just the symbols of S_1). Then S_2 is a subsystem of S_3, so S_2 is also (simply) consistent. Also S_2 is an extension of S; so by the essential undecidability of S, S_2 is undecidable. Now consider the axioms of S_2 which are not axioms of S_1. They are finitely many, since they come from a finite axiomatization of S; say they are $A_1, \ldots, A_m$. So $\vdash B$ in S_2 if and only if $\forall A_1, \ldots, \forall A_m \vdash B$ in S_1, which by the deduction theorem etc. (Corollaries Theorems 10 and 11) is the case if and only if $\vdash \forall A_1 \supset (\ldots (\forall A_m \supset B) \ldots)$ in S_1. Thus we have reduced the decision problem for the undecidable system S_2 to that for S_1. So S_1 is undecidable, as was to be shown.

To make things simple, we began with the case S_1 lacks no symbols of S. More generally, S_1 may lack predicate or function symbols of S, provided they can be "defined" or "interpreted" in a consistent common extension S_3 with S. For example, if S_1 has the notation of N while S has the predicate symbol $<$, the consistent common extension S_3 could have provable in it

[214] By Corollary Theorem 31 § 29, the theory of equality for a finite list of predicate and function symbols is finitely axiomatizable in the predicate calculus.

If some finite set of formulas of a system S can be used as the nonlogical axioms instead of the original ones (i.e. without changing the class of the provable formulas), then some finite subset of the original nonlogical axioms can be (so S is finitely axiomatizable, under the above definition). Why?

the formula $a<b \sim \exists c\, c'+a=b$. If S has the symbol !, S_3 could have provable $a!=b \sim F(a, b)$ where $F(a, b)$ expresses the representing predicate $a!=b$ of the function $a!$ (cf. end § 38). The additional details which this requires in the above argument that S_1 is undecidable are outside the scope of this book.[215]

Since Rosser 1936, it has been known that N is essentially undecidable (though the name "essentially undecidable" was first used by Tarski 1949 abstract). However N is not finitely axiomatizable, as we have mentioned (Ryll-Nardzewski 1952). The systems of axiomatic set theory of von Neumann 1925, Bernays 1937–54 and Gödel 1940 (if consistent) are essentially undecidable, since they include N (if the symbolism of N is suitably identified within them), and they have only finitely many axioms (unlike the system of Zermelo and Fraenkel § 35, where the axiom of subsets (II) gives rise to $\aleph_0$ axioms when formalized in the predicate calculus). Tarski's reduction method for decision problems for elementary axiomatic theories however calls for an essentially undecidable and finitely axiomatizable system which is much more elementary in its interpretation than axiomatic set theory.

Such a system was first given by Mostowski and Tarski 1949 abstract, using basically Rosser's 1936 result (included in our Corollary 2 Theorem VIII); their system deals with the arithmetic of the integers . . . , −2, −1, 0, 1, 2, . . . instead of the natural numbers 0, 1, 2, ...

In 1950 abstract R. M. Robinson showed that a certain subsystem of the N of § 38 is essentially undecidable and finitely axiomatizable. Using the predicate calculus as the underlying logic (he used the predicate calculus with equality), it is the system having the following 13 non-logical axioms: 14–21, the four equality axioms for + and ·, and the formula $a=0 \vee a>0$. (This is a subsystem of N, since its five axioms which aren't axioms of N are provable in N.) To see by Corollary 2 above that Robinson's system is essentially undecidable, we need only verify that it satisfies the conditions in the second paragraph of Corollary 1. It is quite clear that, by applying methods used in the proof of Theorem VII from Theorem III (where a system S_1 with finitely many axioms sufficed for the relevant theory of $(Ex)T(a, a, x)$), we could find *a system* with only finitely many axioms that would do. This is the fundamental discovery with which we are concerned now. To make this result more neat, by showing that a *subsystem of* N, and indeed exactly Robinson's system, will do, requires

[215] They involve the material cited in Footnote 156 § 38. See Tarski, Mostowski and Robinson 1953 or IM pp. 437–439. In the case S_1 lacks function symbols of S, the logic should be the predicate calculus with equality, or S_1 should have the symbol = and the equality axioms for the symbols of S_1 should be provable in S_3.

additional detailed work of sorts which we have been omitting in this book.[216]

Using Tarski's method with a simple essentially undecidable and finitely axiomatizable system, Tarski and his coworkers have shown the undecidability of a variety of formalized theories in the arithmetic of integers and reals, rings, groups, fields, lattices and projective geometries. In particular, by these means Tarski showed that **Gp** and hence **G** are undecidable.[217] However **Gp** and **G** are not essentially undecidable; for, their respective extensions **AGp** and **AG** are decidable, as Szmielew 1948, 1955 showed.

EXERCISE 47.1*.[218] For each of the following, say whether a Turing machine $\mathfrak{M}$ can be found which performs the described operation. If "yes", give the idea for its construction (as is done for various machines in §§ 42–47), not the full details (as in § 41). If "no", show why not. (We say a machine $\mathfrak{M}_i$ *enumerates* a set C of natural numbers, if $\mathfrak{M}_i$ computes a total function φ_i such that $\varphi_i(0), \varphi_i(1), \varphi_i(2), \ldots$ is an enumeration of C allowing repetitions.)

(a) Applied to i and n, when $\mathfrak{M}_i$ enumerates an infinite set C, $\mathfrak{M}$ computes the nth number $\varphi(n)$ in an enumeration $\varphi(0), \varphi(1), \varphi(2), \ldots$ of C without repetitions.

(b) Applied to i and n, when $\mathfrak{M}_i$ enumerates a nonempty set C, $\mathfrak{M}$ computes the nth number $\varphi(n)$ in an enumeration $\varphi(0), \varphi(1), \varphi(2), \ldots$ of C which is without repetitions if C is infinite.

(c) Applied to f and n, when $\hat{a}(Ex)T(f, a, x)$ is nonempty, $\mathfrak{M}$ computes the nth number $\varphi(n)$ in an enumeration $\varphi(0), \varphi(1), \varphi(2), \ldots$ of $\hat{a}(Ex)T(f, a, x)$ allowing repetitions.

(d) Similarly, with $\hat{a}(x)T(f, a, x)$ in place of $\hat{a}(Ex)T(f, a, x)$.

(e) Applied to i and j, when $\mathfrak{M}_i$ and $\mathfrak{M}_j$ enumerate sets D^0 and D^1 for Theorem VIII (1), (2^0), (2^1), $\mathfrak{M}$ computes a number f for (3^0), (3^1).

[216] First, we may establish that $T(i, a, x)$ and $U(i, a, x)$ are primitive recursive, and hence (by IM Corollary 27 p. 244 with Lemma 18b) numeralwise expressed in Robinson's system by formulas T(i, a, x) and U(i, a, x).[176] Then we take for C_a^0 and C_a^1 the formulas $\exists x[T(a, a, x) \,\&\, \forall y(y<x \supset \neg T(a, a, y)) \,\&\, U(a, a, x)]$ and $\exists x[T(a, a, x) \,\&\, \forall y(y<x \supset \neg T(a, a, y)) \,\&\, \neg U(a, a, x)]$. (Here, under the usual interpretation, $\forall y(y<x \supset \neg T(a, a, y))$ is redundant, but its presence facilitates proving (b^0) and (b^1) in the weak system of Robinson) Now (a^0) and (a^1) follow at once. Also it is not hard to establish (b^0) and (b^1), using the remark in IM p. 198 after the proof of *169. If we do not want to accept the simple consistency of Robinson's system on the basis of its interpretation or the nonelementary consistency proof of Gentzen for the N of § 38 (end § 44), an elementary consistency proof is available in IM Theorem 53 (a) p. 470.

[217] See Tarski, Mostowski and Robinson 1953 Chapter 3.

The undecidability of **Gp** also follows from the more recently proved result of Novikov 1955 that the "word problem for groups" is unsolvable (end § 45).

[218] Some of the solutions are in IM pp. 306, 307, 346.

(f) Applied to i, when $\mathfrak{M}_i$ enumerates a nonempty set C of members of $\hat{a}(x)T(a, a, x)$, $\mathfrak{M}$ computes another member (not in C) of $\hat{a}(x)T(a, a, x)$.
(g) Applied to a, when a is the Gödel number of a formula A of N § 38 (in a fixed effective Gödel numbering of those formulas), $\mathfrak{M}$ decides whether $A \vdash_N 1=0$.
(h) Applied to a, when a is the Gödel number of a formula A of N (as in (g)), $\mathfrak{M}$ decides whether it is decidable whether $A \vdash_N B$ where B ranges over all formulas of N.