The Universal Turing Machine
A Half-Century Survey

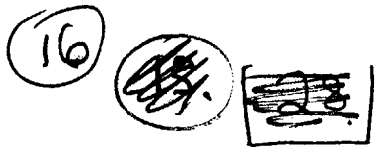
edited by

Rolf Herken

Institut für Theorie der Elementarteilchen
Freie Universität Berlin

REPRINTED WITH PERMISSION

OXFORD UNIVERSITY PRESS
1988



Turing's Analysis of Computability, and Major Applications of It

Stephen C. Kleene

The aim of this article is to present to readers, not assumed to have any previous acquaintance with Turing's work, both (1) the basic ideas in Turing's analysis of computability, and (2) what seems to me the most straightforward way of developing from it some significant features of the structure of mathematics. The application I use of his computability, and the path I follow in the developments from it, are somewhat different than in Turing's own writing. Other investigators contributed. I shall annotate this as we proceed.

1. *Algorithms (Decidability and Computability)*

In mathematics, we are often interested in having a general *method* or *procedure* for answering any one of a certain class of questions; for example, the class of questions, "Is n a prime number?". It is an infinite class of questions, one question for each positive integer $1, 2, 3, \ldots$ as the value of n. A question is selected from the class by picking such a value of n. And as a general method to test whether the n picked is a prime number, the mathematician answers "No" if $n = 1$; "Yes" if $n = 2$ or $n = 3$; and if $n \geq 3$, tries to divide n evenly by successive positive integers ≥ 2 up (if necessary) to the last one whose square is $\geq n$, answering "No" if he succeeds with one of them, otherwise answering "Yes".

In this example, the questions are yes-or-no questions. We may have the like with what-questions. For example, "What is the nth prime number?". The obvious method involves determining the n-th prime for $n = 1, \ldots, m$ before seeking it for $n = m+1$. It is a theorem of Euclid that, to any number a, there is a prime between $a+1$ and $a!+1$ inclusive. So, if we have already found the m-th prime, call it p, we can find the $m+1$-st prime by testing for primality the numbers $p+1, p+2, \ldots, p!+1$ in order till we find one that is prime. Actually, the upper bound $p!+1$ for the numbers we may

need to test is unnecessary. It is used in Euclid's proof of his theorem that there are infinitely many primes, which I presupposed in posing the class of questions, "What is the n-th prime number?" with the tacit understanding that there is one for each positive integer n. And when we are determining the primes in order, we only need to use previous primes as the divisors in testing for primality.

Of course, we have been taking it for granted that we already have a method for answering all questions of the infinite class, "Does b divide a?" for positive integers a and b. This method uses the method we have for finding the quotient and remainder on dividing a by b. This indeed is the long-division process that we learned in our elementary schooling. If the remainder is 0, the answer to "Does b divide a?" is "Yes"; if not, "No".

Actually, there is a more efficient arrangement of the work of constructing a list of the primes among the positive integers up to a given one. This is the sieve of Eratosthenes (c. 225 B.C.). We start with a list of those positive integers, omitting 1. Then we cancel the multiples of 2 after 2 itself, then the multiples of the next uncancelled number (which is 3) after itself, and so on. The numbers that do not fall through the sieve, i.e., are not cancelled, are the primes.

What do we mean by a method for answering any one of a given infinite class of questions? I think we can agree that such a method is given by a set of rules or instructions, describing a procedure that works as follows. *After* the procedure has been described, if we select *any* question from the class, the procedure will then tell us how to perform successive steps, so that after a finite number of them we will have the answer to the question selected. In particular, immediately after selecting the question from the class, the rules or instructions will tell us what step to perform first, unless the answer to the question selected is immediate. After our performing any step to which the procedure has led us, the rules or instructions will *either* enable us to recognize that now we have the answer before us and to read it off, *or else* that we do not yet have the answer before us, in which case they will tell us what step to perform next. In performing the steps, we simply follow the instructions like robots; no ingenuity or mathematical invention is required of us.

Such methods as I have just described have had a prominent role in mathematics. They have been called "algorithms". The term "algorithm" is a corruption of the last part of the name of the ninth-century Arabian mathematician Abu Abdullah abu Jafar Muhammad ibn Musa al-Khowarizmi from the Khowarizm oasis in central Asia. Also the term "algebra" comes from the words "al jabr" occurring in the Arabic title of one of his writings.

Algorithms and algebra did not originate with al-Khowarizmi, though he made respectable contributions to both subjects. The recognition of algorithms goes back at least to Euclid (c. 330 B.C.). There is a method, called "Euclid's greatest common divisor algorithm", for finding the greatest common divisor of two positive integers a and b; and this is used in an algorithm for answering the questions, "Does the equation $ax + by + c = 0$, where a, b, c are positive integers, have a solution in integers for x and y?". Equations whose solutions are sought in integers are called "Diophantine", after Diophantus (c. 250 A.D.).

When we have an algorithm for an infinite class of yes-or-no-questions, the property $R(a)$ or relation $R(a,b)$ or $R(a_1,\ldots,a_n)$ whose holding is in question is called *decidable* and the algorithm is also called a *decision procedure*. When we have an algorithm for an infinite class of what-questions, the function $\phi(a_1,\ldots,a_n)$ whose value is in question is called *computable* and the algorithm is also called a *computation procedure*.

An algorithm must be described fully *before* we start picking questions from the class; so its description must be finite. Thus an algorithm is a *finitely* described procedure, sufficient to guide us to the answer to any one of *infinitely* many questions, by *finitely* many steps in the case of each question.

With a finite class of questions, the situation is trivial: in principle, an algorithm always exists, consisting simply of a list or table of the answers to the questions.

We don't have to go far to find examples of infinite classes of questions for which the existence of algorithms is problematical.

From now on, I shall take my variables like $a, b, c, \ldots, x, y, z, \ldots$ to range over the *natural numbers* (or *nonnegative integers*) $0, 1, 2, \ldots$, except as may be indicated otherwise. I shall use "(Ex)" to abbreviate "there exists an x such that" or "for some x".

Suppose $R(a,x)$ is some decidable relation between two natural numbers a and x; i.e., an algorithm exists to decide, for each choice of values of a and x, whether or not $R(a,x)$ holds. What about $(Ex)R(a,x)$? This is a property of one natural number a. Is this property decidable? An algorithm for it is not given simply by the meaning of the symbols. All that the symbols suggest directly is that, to try to find out whether or not $(Ex)R(a,x)$ holds for a given a, we ask successively for $x = 0, 1, 2\ldots$ (with that value of a) whether or not $R(a,x)$ holds. We can answer each of these questions, so far as we pursue them, since (as we assumed) $R(a,x)$ is decidable. If eventually we obtain "Yes" for some x, we answer "$(Ex)R(a,x)$?" with "Yes". But suppose we keep on getting "No" till doomsday. Then we fail to answer "$(Ex)R(a,x)$?". For we won't know whether doomsday simply came too soon, or if there is indeed no natural-number value of x making $R(a,x)$ true (with the given value of a).

Although the meaning of $(Ex)R(a,x)$ does not provide an algorithm directly, it may be possible, for a particular choice of the relation $R(a,x)$, to

prove some theorems which will lead to an equivalent formulation of when $(Ex)R(a,x)$ holds from which an algorithm can be recognized. For example, if (with a given decidable $R(a,x)$) it can be proved that, whenever (for a given a) $R(a,x)$ holds for some x, there is such an $x \leq \phi(a)$ where $\phi(a)$ is a given computable function of a, then we would have an algorithm for $(Ex)R(a,x)$ by searching just (if need be) till we reach $\phi(a)$.

2. The Totality of Algorithms

Various problems whether algorithms exist for given infinite classes of questions have engaged the attention of mathematicians from early mathematical history. For indeed, mathematicians can claim the greatest success in treating a mathematical subject represented by an infinite class of questions when they have devised a method for answering all the questions of the class, that is an algorithm. Why not the best? But mathematicians haven't always succeeded in their quests for algorithms.

So it is surprising that only in the present century did people begin to think intensively about exactly what an algorithm is, or about what decidability or computability really mean. The idea of an algorithm, or of a decision or computation procedure, is sufficiently real so that in two thousand years of examples mathematicians have had no trouble in agreeing in each particular case of an infinite class of questions with a procedure they had in hand that the procedure is an algorithm for the class of questions or is not. But this history—these examples—do not provide a concrete picture of what the totality of all possible algorithms looks like. Without such a picture there is no possibility of showing that for some infinite class of questions an algorithm does not exist.

Such a picture was first provided in precise terms by several investigators in the 1930's, actually in three equivalent ways (and several further equivalents surfaced in the 1940's and 1950's). Here I shall paint the picture along the lines used by Alan M. Turing, a British mathematician who lived from 1912 to 1954.

Let me first sharpen our project a bit. As remarked, the case of a class of only finitely many questions doesn't interest us. For a class of infinitely many questions, I shall suppose the questions can be listed or "enumerated" using the natural numbers as the indices, thus: "Q_0?", "Q_1?", "Q_2?",..., "Q_a?", ... (Infinite classes of questions which can't be so listed—which are "uncountable"—arise in mathematics; but the consideration of them is beyond the scope of this article.) Say, first, the questions are what-questions. The answers to the questions must (for this article) come from a given class of objects that can be likewise listed, thus: $c_0, c_1, c_2, \ldots, c_b, \ldots$. So it suffices

to consider classes of questions of the form "$\phi(a) = ?$" with answers "b", where a is the subscript of "Q_a?" locating the question in the first list, and b is the subscript for the object c_b in the second list used in answering it. Actually, it is convenient to consider more generally functions $\phi(a_1, \ldots, a_n)$ with n natural-number variables (for any positive integer n) taking natural numbers as values (which we call *number-theoretic functions*). For, such functions are frequently what we want algorithms for primarily, and not just by going over from our original questions to their subscripts a in a list "Q_0?", "Q_1?", "Q_2?",..., "Q_a?",..., and from our original answers "c_b" to their subscripts b.

For yes-or-no-questions, by letting 0 be a code for the answer "Yes" and 1 for the answer "No", we correlate what-questions (with only two possible answers, "0" and "1").

So our project can be narrowed to: *What number-theoretic functions $\phi(a_1, \ldots, a_n)$ are computable?*

3. Turing Computability of Number-theoretic Functions

I answer this question using Turing's ideas. Actually he worked instead primarily with: *What real numbers have computable binary decimal expansions?* He used his machines to print the successive digits ad infinitum on alternate squares of a 1-way infinite tape, while the intervening squares were reserved for temporary notes serving as scratchwork in the continuing computation.

The application, as we give it now, of Turing machines to characterize the computable number-theoretic functions $\phi(a_1, \ldots, a_n)$ was worked out in detail by Kleene for his seminar at Wisconsin in the spring term of 1940–41, and published in Chapter XIII of his *1952* book.[1] It is closer in some respects to Post *1936* than to Turing *1936-7*.

Let us recall my description in § 1 of a method or procedure for answering any one of a given (countably) infinite class of questions. I talked about "steps". What are we manipulating in these "steps"? We might say mental objects. This is so in the case of computations so simple that we perform them mentally, maybe with some motions of the lips. But except in very simple calculations, we need to work with symbols on paper (or some equivalent).

How many symbols do we need? Or indeed, better, how many can we have? Surely we can learn to recognize unambiguously only a finite number

1 An italicized date in conjunction with the name of an author constitutes a reference to the References at the end of this article.

of symbols. In his pioneering paper *1936-7*, Turing wrote, "If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent." In contrast to *analog* computing, as for example by a slide rule, where there is a whole continuum of positions on each of the sliding scales, the computations we are considering are of number-theoretic functions $\phi(a_1, \ldots, a_n)$, where the possible values of the variables $a_1, \ldots, a_n$, and the possible answers b, are discrete (they do not merge with one another). So using symbols differing to an arbitrarily small extent would be inappropriate to our project. We are considering *digital* computation rather than *analog* computation.

The symbols (that is, occurrences of symbols from a given finite list) are what we observe before, and again after, any step in a computation. In actual practice, as for example in performing a long division, we may have an array of symbols, and shift our attention from one place on the paper to another. If the array is large, we may not be able to take it all in at once. Let us consider the paper, or whatever we use instead, as a space to accommodate symbols, divided into parts which I call *cells*, each of which is what we observe at a given moment. We can construe what is written on a cell as a single symbol, even if it is a (finite) complex of our primary symbols.

The cells must have some kind of a regular geometrical arrangement, or our procedure in computing would become chaotic, frustrating its progress toward a determinate result. If one thinks hard about this, and entertains various alternative arrangements of the cells which come to mind, one can see that whatever can be done with any reasonable arrangement can be done with a linear arrangement, open-ended say to the right.[2] Let us call the cells then *squares* on a *tape*, beginning with a first (*leftmost*) square and with successive squares ad infinitum to the right.

A person in computing is not constrained to working from just what he sees on the square he is momentarily observing. He can remember information he previously read from other squares. This memory consists in a state of mind, his mind being in a different state at a given moment of time depending on what he remembers from before. 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."

From such reflections on how a human computer works, Turing formulated his concept of computing machines (now called *Turing machines*), to which I proceed without further ado.

A Turing machine is an agency or apparatus or black box with the following features.

2 This is elaborated in Kleene *1952* under (c) on pp. 379–381.

In describing its operation, I distinguish discrete *moments* of time, numbered $0, 1, 2, \ldots$.

The machine is supplied with a linear *tape*, ruled in *squares*, infinite to the right. The tape is threaded through the machine so that at each moment just one square is observed by the machine or *scanned*.

The machine at each moment of time is in one of a fixed list of $k + 1$ (≥ 2) *states* $q_0, \ldots, q_k$. (These correspond to the states of mind of a human computer.) I call q_0 the *passive* state, and $q_1, \ldots, q_k$ *active* states.

Each square of the tape is capable of having printed on it any one of a fixed list of l (≥ 1) *symbols* $s_1, \ldots, s_l$, or of being *blank*, which I write s_0; so there are $l + 1$ possible *square conditions* $s_0, \ldots, s_l$.

At each moment of time, the total *(tape vs. machine) situation* consists of, **first**, a particular printing on the tape (i.e., which squares are printed, and each with which of $s_1, \ldots, s_l$), **second**, a particular position of the tape in the machine (i.e., which square is scanned), and **third**, a particular machine state (i.e., which of $q_0, \ldots, q_k$ is the state of the machine). I call the situation *active*, if in it the state is active; *passive*, if in it the state is passive.

As we shall use the machines, only a finite number of the squares will be printed at any moment; so the situations will constitute finite objects.

If the situation at a given moment t is active, the machine performs an *act* between the moment t and the next moment $t + 1$, whereby it is determined what the situation will be at Moment $t + 1$. The act consists of three parts, each of which possible changes one element of the situation (these being the only changes between Moment t and Moment $t + 1$); **first**, the choice of one of $s_0, \ldots, s_l$ to be the condition $s_{i'}$ at Moment $t + 1$ of the square which was scanned at Moment t (say that square was in condition s_i at Moment t); **second**, a possible shifting of the tape in the machine, so that at Moment $t + 1$ the scanned square is one square left of (L), the same as (C for "center"), or one square right of (R), the square scanned at Moment t; and **third**, the choice of one of $q_0, \ldots, q_k$ to be the machine state $q_{j'}$ at Moment $t + 1$ (say the machine state was q_j at Moment t). A machine act can thus be described by a triple

$$s_{i'} \quad \begin{matrix} L \\ (C) \\ R \end{matrix} \quad q_{j'} \qquad \text{or briefly} \qquad i' \quad \begin{matrix} L \\ (C) \\ R \end{matrix} \quad j' .$$

If the situation at Moment t is passive, no act is performed, so the situation at Moment $t + 1$ is the same as at Moment t.

What act does a given machine perform from a given active situation? The act performed is determined normally (i.e., outside of one exceptional

kind of situation) by the scanned square condition s_i and the machine state q_j at that moment. This pair s_iq_j, or briefly ij, I call the *configuration* (*active*, if q_j is active). The exceptional kind of situation is when the configuration s_iq_j calls for a move left (L), as part of the act normally performed, but the scanned square is already the leftmost square of the tape. Then the machine instead "jams", i.e., it goes to the passive state q_0 with nothing else changed.

In fact, as we shall ordinarily use the machines, it will never happen that the configuration calls for a move left from the leftmost square.[3]

With $l + 1$ scanned square conditions and k active states, there are $(l + 1)k$ active configurations. A particular Turing machine is specified (in its behavior, which is all we care about) by a *machine table*, showing for each active configuration what act shall be performed (if possible).

I illustrate this by giving (in Figure 1) at the top a table for a machine, call it "$\mathfrak{S}$", with eleven active states (I write them simply "1",...,"11" instead of "q_1",...,"q_{11}") and one symbol s_1, which is a tally mark "|". To follow the action of this machine, we must assume some situation as obtaining at Moment 0. I take it to be as shown first at the bottom left, with the machine in State 1, scanning the rightmost of two squares each printed with a tally, the tape being otherwise blank. This situation is active; and we find the act Machine $\mathfrak{S}$ will perform from Moment 0 by entering the table in the first row (since its state is 1) and the second column (since the scanned square bears "|"). There we find "1R2"; so the machine leaves that tally unchanged (1), moves right on the tape (R), and assumes its second active state (2). Thus we get the tape vs. machine situation at Moment 1 shown next. Similarly we arrive at the next two situations.

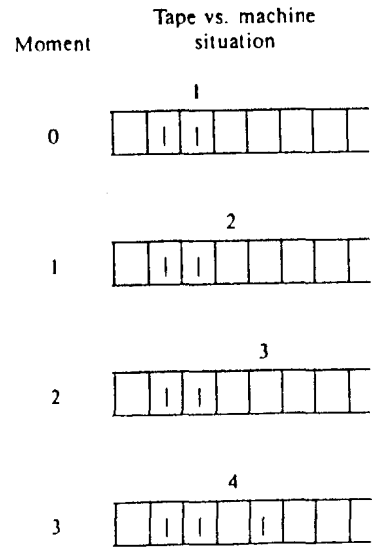
To save space in showing situations, let me change my format to omit the picture of the tape, simply write 0's and 1's for blank and tally-printed squares, and indicate the state by a superscript.

At the same time, I change to indicate the conditions of the squares only as far out as the rightmost one thus far scanned in the operation of the machine (all the rest we know are blank). As we remarked, our tape vs. machine situations are finite. In this format, I show in Figure 1 at the right the situations all the way down to Moment 23. Since the state at Moment 23 is the passive one 0, the situation will be the same at all subsequent moments.

Our announced objective was to apply the Turing machine concept to

3 Without relying on this, we could easily fix matters so that the configuration determines the act in all situations without exception. Let the leftmost square bear a special mark $*$ doubling the list of square conditions to $s_0,\ldots,s_l,*s_0,\ldots,*s_l$. From a configuration $*s_iq_j$, the act is then $*s_1Cq_0$ if s_iq_j calls for a motion left, and otherwise the same as from s_iq_j except with a $*$ on the first member of the triple unless s_iq_j calls for a motion right.

Machine state	Scanned square condition 0	1
1	0C0	1R2
2	0R3	1R9
3	1L4	1R3
4	0L5	1L4
5	0L5	1L6
6	0R2	1R7
7	0R8	0R7
8	0R8	1R3
9	1R9	1L10
10	0C0	0R11
11	1C0	1R11



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

Figure 1. Table for Machine $\mathfrak{S}$ and example of its action.

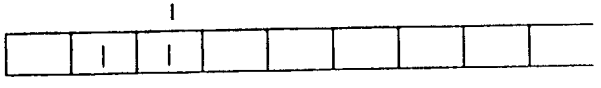
characterizing the computation of a number-theoretic function $\phi(a_1, \ldots, a_n)$. First, I take $n = 1$. The idea is that an algorithm or computation procedure for $\phi(a)$ can be embodied in a Turing machine which, when we feed it a tape on which a particular value of the variable a is represented, will perform acts terminating in a situation in which the corresponding value $b = \phi(a)$ of the function is also represented.

First, we must agree on how to represent the natural numbers $0, 1, 2, \ldots$ on the tape. I take the most primitive representation, using only the tally symbol "|". However, because I don't want 0 to be represented by no tallies (which would be invisible on the tape), I use one tally to represent 0, two to represent 1, three to represent 2, etc., thus:

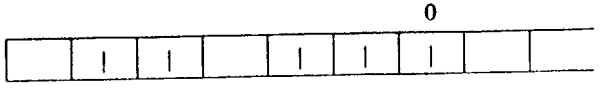
0 1 2
| , | | , | | | , ...

For a given machine to *compute* a given function $\phi(a)$, it must behave as follows. After picking a value of a, we start the machine off at Moment 0 by putting a tape in it with a represented by $a + 1$ tallies on consecutive squares beginning with the next-to-the-leftmost square, with the tape otherwise blank, and with the machine in its first active state 1 scanning the rightmost of those $a + 1$ tallies. If it is a machine that computes our function, it must eventually stop (that is, assume the passive state 0) with the corresponding function value $\phi(a)$ represented by $\phi(a) + 1$ tallies on squares occurring consecutively to the right after leaving one square blank following the representation of a, with the tape otherwise blank, and with the scanned square the one bearing the rightmost of the $\phi(a) + 1$ tallies. This must happen for each choice of the natural number a.

I illustrate this taking $\phi(a)$ to be the successor function $a + 1$ (simple, but important!). First, take $a = 1$. So the machine, started off thus



should eventually stop thus



($a = 1$, $\phi(a) = 2$). Indeed, the machine $\mathfrak{S}$, whose table I gave in Figure 1, does just this, giving us the answer "2" to our question "1 + 1 = ?" at Moment 23. If Machine $\mathfrak{S}$ does the like for every value of a, then it computes the function $\phi(a) = a + 1$.

In fact it does! To see this, let us break down its action into groups of steps fitting a pattern, as I shall first illustrate for the case $a = 1$ which I displayed in Figure 1.

Pat. 1 (used only initially). Machine $\mathfrak{S}$, if it is in State 1 scanning a rightmost printed square (at Moment 0), thereupon goes right two squares (it keeps count by changing successively to States 2 and 3), prints there, and returns left one square, assuming State 4 (at Moment 3).

Pat. 2. What does Machine $\mathfrak{S}$ do from a situation in which it has just become in State 4 (as at Moments 3 and 10)? It goes left till it first encounters a printed square after one or more blank squares (it changes to State 5 just after being as at Moments 3 and 11 on a blank square). Then it probes the square next left of that printed square (going to State 6 there) to see whether it is not blank or is blank, going respectively to State 7 or 2 on the aforesaid printed square (as at Moment 6 or 15).

Pat. 3. Starting in State 7 on a printed square with a blank square next right and some printed squares further right (Moment 6), it erases that printed square, goes right till it comes to a blank square after one or more printed squares, prints on it, and returns left one square assuming State 4 (Moment 10).

Pat. 4 (used only terminally). Starting in State 2 on a printed square with a blank square next right and some printed squares further right (Moment 15), it goes right printing on all but the last of the consecutive blank squares that were just right of it, and continues over the next group of printed squares to print on the first blank square following, where it goes to State 0 (Moment 23).

Let us apply this analysis of our machine's operation to a little more complicated example, say with $a = 3$, not showing all the consecutive situations but just those at the beginnings and ends of the parts which follow a pattern (Figure 2).

In summary our machine $\mathfrak{S}$, after printing a 1 to start the second group of 1's or tallies, copies the 1's of the first group into the second group. It keeps track of the 1's copied by erasing each as it copies it, except the last (leftmost) 1, upon encountering which it restores those erased before copying that.

To compute a function $\phi(a_1, \ldots, a_n)$ with any $n \geq 1$ variables, we start the machine at Moment 0 with the n-tuple $a_1, \ldots, a_n$ represented on the tape by groups of $a_1 + 1, \ldots, a_n + 1$ consecutive tallies, preceded and separated by single blank squares, with all other squares blank, and the square bearing

	0	1	1	1	1^1						
Pat. 1											
	0	1	1	1	1	0^4	1				
Pat. 2											
	0	1	1	1	1^7	0	1				
Pat. 3											
	0	1	1	1	0	0	1^4	1			
Pat. 2											
	0	1	1	1^7	0	0	1	1			
Pat. 3											
	0	1	1	0	0	0	1	1^4	1		
Pat. 2											
	0	1	1^7	0	0	0	1	1	1		
Pat. 3											
	0	1	0	0	0	0	1	1	1^4	1	
Pat. 2											
	0	1^2	0	0	0	0	1	1	1	1	
Pat. 4											
	0	1	1	1	1	0	1	1	1	1	1^0

Figure 2. Action of Machine $\mathfrak{S}$ in computing $a + 1$ for $a = 3$.

the rightmost of these tallies scanned in State 1. To compute ϕ, the machine must, whatever the choice of the values of $a_1, \ldots, a_n$, eventually stop with the $n + 1$-tuple $a_1, \ldots, a_n, \phi(a_1, \ldots, a_n)$ similarly represented.

Although we are using only one symbol, the tally "|", in questioning a Turing machine with an n-tuple of natural numbers and in receiving the answer, our definition of Turing computability permits the machine a list of $l \geq 1$ symbols $s_1, \ldots, s_l$. We take s_1 to be the tally "|". If $l > 1$, the other symbols $s_2, \ldots, s_l$ are allowed to be printed on the tape at intermediate moments of a computation. In fact, it can be proved that every function $\phi(a_1, \ldots, a_n)$ thus computable can be computed by a Turing machine with only the one symbol "|", as in the above illustration and those I shall give presently.[4]

4 See Kleene *1952*, the bottom of p. 361.

4. *Turing's Thesis*

What I am claiming after Turing—call it "Turing's thesis"—is that every number-theoretic function $\phi(a_1, \ldots, a_n)$ for which there is an algorithm – which is intuitively computable (one sometimes says "effectively calculable") – is *Turing computable*; that is, there is a Turing machine which computes it in the manner I have described.

It may seem anomalous, then, that it should be so much of an exercise to see that such a simple function as $a + 1$ is Turing computable. However, we undertook to treat $a + 1$ right at the beginning of our study of Turing machines. And we used only one symbol "|". Using more symbols would tend to make the Turing machine action better resemble computation as we are used to doing it.

Our illustration may unduly suggest that the computer is restricted to taking an ant's eye view of its work, squinting at the symbol on one square at a time. However, when more than the one symbol "|" is used, we can interpret what we mean by a symbol liberally. Thus the Turing-machine squares can correspond to whole sheets of paper. If we employ sheets ruled into 20 columns and 30 lines, and authorize 99 primary symbols, there are $100^{600} = 10^{1200}$ possible square conditions, and we are at the opposite extreme. The schoolboy doing arithmetic on 8½ by 12″ sheets of ruled paper would never need, and could never utilize, all this variety.

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

There is, however, an intrinsic difficulty which the theory of computing must face. We are defining a function $\phi(a)$ to be Turing computable if there is a Turing machine which will compute its value for *all* choices of the value of a, and similarly with $\phi(a_1, \ldots, a_n)$. The schoolboy computing on sheets of paper is not normally asked to work with numbers too big to go on one of his sheets. If he had to do that, he too would have to work in a groping manner as our machine $\mathfrak{S}$ does in computing $a + 1$, though using more than the one symbol "|" he would not need to do so for as small values of a.

Turing wrote,

> ... we cannot regard the process of recognition as a simple process. This is a fundamental point and should be illustrated. In most mathematical papers the equations and theorems are numbered. Normally the numbers do not go beyond (say) 1000. It is, therefore, possible to recognize a theorem at a glance by its number. But if the paper was very long, we might reach Theorem 157767733443477; then, further on in the paper, we might find "... hence (applying Theorem 157767733443477) we have ...." In order to make sure which was the relevant theorem we should have to compare the two numbers

figure by figure, possibly ticking the figures off in pencil to make sure of their not being counted twice.

Incidentally, you surely recognize that Turing machines can be used to perform computations on all sorts of finite sequences of symbols, not just strings of tallies representing natural numbers. This is an elegant feature of his machine concept.

Mathematicians deal with idealized systems of objects, obtained by extrapolating for the purposes of their thought from what people encounter in the real world. They imagine the infinite sequence $0, 1, 2, \ldots$ of the natural numbers, and thus they get a beautiful theory with an elegant logical structure, though in the actual counting of discrete objects we can never use more than finitely many of them.

Computation, theoretically considered (to be performable for all possible values of the independent variables), is idealized. Turing's analysis takes this idealized aspect of it into account. A Turing machine is like an actual digital computing machine, except that (1) it is error free (i.e., it always does what its table says it should do), and (2) by its access to an unlimited tape it is unhampered by any bound on the quantity of its storage of information or "memory".

5. *Some Turing Computable Operations*

As we have illustrated by our 12-state machine $\mathfrak{S}$ to compute $a + 1$, it can be a by no means trivial exercise to construct a Turing machine to compute even a quite simple function for which we have an algorithm. Scholars have verified that Turing machines exist embodying all sorts of algorithms.

One must begin by handling various simple operations.

For example, the table for a machine to copy a number a represented leftmost on the tape (otherwise blank) and scan the copy in *standard position* (i.e., on the rightmost tally) is obtained from the table for our machine $\mathfrak{S}$ (which copies a number adding 1 to it) simply by changing the entry in line 11 and column 0 from $1C0$ to $0L0$.

We can similarly construct a machine to copy not the leftmost of a sequence of numbers, but say the m-th from the left. Or the copy may be put after a one-space gap, meaning that two blank squares separate the copy from the preceding number (which was rightmost before the copying). We may later erase all the numbers from a given space back to such a one-space gap and close up the resulting many-space gap.

Such simple operations constitute useful building blocks in rendering by Turing machines algorithms as they are commonly encountered. I shall illustrate.

One common form of definition is a *primitive recursion*, such as

$$\begin{cases} \phi(0, b) & \psi(b), \\ \phi(a+1, b) & \chi(a, \phi(a, b), b). \end{cases}$$

Specifically, this is a (primitive) recursion on a with one parameter b. (In general, there may be n parameters $b_1, \ldots, b_n$ for any $n \geq 0$.) If $\psi(b)$ and $\chi(a, c, b)$ are previously defined functions, this defines the function $\phi(a, b)$. If we already have algorithms for $\psi(b)$ and $\chi(a, c, b)$, an algorithm for computing $\phi(a, b)$ consists in computing successively (using the algorithms for ψ and χ)

$$\begin{array}{lll} \phi(0,b) & \text{as } \psi(b) & (\text{briefly } f_0 = \psi(b)), \\ \phi(1,b) & \text{as } \chi(0, \phi(0,b), b) & (\text{briefly } f_1 = \chi(0, f_0, b)), \\ \phi(2,b) & \text{as } \chi(1, \phi(1,b), b) & (\text{briefly } f_2 = \chi(1, f_1, b)), \\ \ldots & & \end{array}$$

until we get $\phi(a, b)$ (briefly f_a) for the given value of the recursion variable a, this all being done with a fixed given value of the parameter b.

Dedekind in *1888* and Peano in *1889* used primitive recursions to define successively the sum $a + b$, the product $a \cdot b$ and the exponentiation function a^b of two natural numbers a and b, thus (where b is the recursion variable and a the parameter),[5]

$$\begin{cases} a + 0 &= a, \\ a + (b+1) &= (a+b) + 1, \end{cases}$$

$$\begin{cases} a \cdot 0 &= 0, \\ a \cdot (b+1) &= (a \cdot b) + a, \end{cases}$$

$$\begin{cases} a^0 &= 1, \\ a^{b+1} &= a^b \cdot a. \end{cases}$$

Let us consider how a Turing machine $\mathfrak{M}_\phi$ can be constructed to compute the function $\phi(a, b)$ defined by the primitive recursion displayed above, assuming we already have machines $\mathfrak{M}_\psi$ and $\mathfrak{M}_\chi$ to compute $\psi(b)$ and $\chi(a, c, b)$.

The machine $\mathfrak{M}_\phi$ I have in mind, started out in the standard way with any given choice of values of a and b represented on the tape, will print more

5 The term "recursion" was employed by Skolem in *1923* and Hilbert in *1926*, and "primitive recursion" was introduced by Péter in *1934* to distinguish it from other kinds of recursion. Dedekind, Peano and Skolem worked with the positive integers.

and more numbers to the right after leaving a one-space gap, until finally it prints the desired value f_a of $\phi(a,b)$. Then all the numbers between that and the gap will be erased, and the resulting many-space gap will be closed up.

In Figure 3, I show the numbers in the order in which they will be printed on the tape. Each letter like "c" stands for $c+1$ consecutive tally-printed squares, and each comma "," stands for one blank square. After a number has just been printed, its rightmost tally will be scanned.

$$,a,b,,a,b,f_0 \qquad (\text{where } f_0 = \psi(b))$$

$$,a,\begin{cases} f_0 & \text{if } a = 0. \\ 0,f_0,b,f_1 & (\text{where } f_1 = \chi(0,f_0,b)) \quad \text{if } a \neq 0 \end{cases}$$

$$,a-1,\begin{cases} f_1 & \text{if } a-1 = 0. \\ 1,f_1,b,f_2 & (\text{where } f_2 = \chi(1,f_1,b)) \quad \text{if } a-1 \neq 0 \end{cases}$$

$$,a-2,\begin{cases} f_2 & \text{if } a-2 = 0. \\ 2,f_2,b,f_3 & (\text{where } f_3 = \chi(2,f_2,b)) \quad \text{if } a-2 \neq 0 \end{cases}$$

$$,a-3,\begin{cases} f_3 & \text{if } a-3 = 0. \\ 3,f_3,b,f_4 & (\text{where } f_4 = \chi(3,f_3,b)) \quad \text{if } a-3 \neq 0 \end{cases}$$

...

Figure 3. Action of Machine $\mathfrak{M}_\phi$ in computing $\phi(a,b)$.

Notice the one-space gap represented by the double comma in the first line. The machine $\mathfrak{M}_\psi$ which computes $\psi(b)$, started on a tape bearing just ",b" leftmost, will come to ",b,f_0" where $f_0 = \psi(b)$, and in doing so it doesn't look left of the blank square preceding "b". So, started instead in the situation ",$a,b,,a,b$", the ",$a,b,,a$" to the left of the ",b" won't distract it in its computation of f_0. Hence, after arranging for our desired machine $\mathfrak{M}_\phi$ to begin by copying "a,b" to obtain ",$a,b,,a,b$" we can hitch that part of it into $\mathfrak{M}_\psi$. To do so, we annex the table for $\mathfrak{M}_\psi$ with its active states renumbered to run up not from 1 but from the next number after those used to program the copying of "a,b". The first of these new state numbers is

substituted for the passive state 0 of a machine which would stop after that copying. So we get to f_0 (end of the first line), and it constitutes no problem to add further lines to the table we are building for $\mathfrak{M}_\phi$ so that $\mathfrak{M}_\phi$ will then copy a (the 3rd number back), and test whether a is 0 or not. The result of this test is expressed by one of two different states being assumed next.

The state to be assumed in the case that $a = 0$ will lead to the copying of the second number back (which in the situation we have just reached is f_0), and thereafter (not shown in Figure 3) to the erasing of all printing to the left of that copy back to the one-space gap, and to the closing up of the resulting many-space gap with the machine assuming the passive state scanning the rightmost tally. Thus, as desired, we get ",$0,b,f_0$" when $a = 0$.

If our machine $\mathfrak{M}_\phi$ has gone to the state representing $a \neq 0$, it is to print 0, copy the 3rd number back (f_0 in our situation) and b (the 5th number back), feed into the machine $\mathfrak{M}_\chi$ for computing $\chi(a,c,b)$ with result f_1 in our situation. The part of our machine $\mathfrak{M}_\phi$ thus far set up, which has gotten us to f_1 in our present case that $a \neq 0$, will for this case be hitched into a machine which copies the 4th number back (a in our situation) decreased by 1 (which it can do since we are in the case $a \neq 0$), tests whether the result ($a-1$) is 0 or not, if it is copies the 2nd number back (f_1), after which it proceeds as our machine did with f_0 in the upper case of the second level of Figure 3; but if it is not 0, copies the 5th number back (0) increased by 1, the third number back (f_1) and the fifth number back (b), feeds into $\mathfrak{M}_\chi$ getting f_2 and finally feeds into itself by assuming the same state as we had when we first got f_1 after finding that $a \neq 0$. So it will copy the 4th number back (which is now $a-1$) decreased by 1 (using $a-1 \neq 0$) with result $a-2$, test whether that is 0 or not, etc., performing for the 4th level of Figure 3 the operations corresponding to the 3rd level; and so on at every level, until finally the process is terminated after the upper case ($=0$) obtains.[6]

Note that, if the like is true of $\mathfrak{M}_\psi$ and $\mathfrak{M}_\chi$, then $\mathfrak{M}_\phi$ in its action in computing $\phi(a)$ never tries to go left from the leftmost square, and uses no symbol other than the tally "|".

Scholars in the manner illustrated have convinced themselves that all possible algorithms for computing number-theoretic functions can be embodied in Turing machines (Turing's thesis). The ingredients that are basically necessary were all provided by Turing: a fixed finite number of symbols, a fixed (perhaps very large) number of states, actions determined by the condition of one scanned square and the state in accordance with the constitution of the particular machine (i.e., its table), unlimited space (on the tape) for receiving the questions and reporting the answers and tem-

6 The construction of a machine $\mathfrak{M}_\phi$ is spelled out fully in Kleene *1952*, p. 373, with preceding material (with "$y, x_2, \ldots, x_n$" for any $n \geq 1$ instead of "a,b").

porarily storing scratchwork, and of time (the moments) for completing the computations.

6. The Existence of Unsolvable Decision and Computation Problems

We know of problems on which mathematicians have worked unsuccessfully, sometimes even for thousands of years, which later were shown to be unsolvable. One of these is to find a ruler and compass construction à la Euclid for trisecting any angle. Only after about 2,250 years, in the 1930's, was it proved that such a construction does not exist. Surely, it is a major advance in mathematics when mathematicians get a grip on an enterprise on which they have been engrossed that tells them that what they were trying so hard to do can't be done, or, in other cases, that something can't be done with a certain body of methods they were using, so that others will have to be brought to bear if they are to have any hope of success.

A result of the introduction of an accepted exact characterization (as by Turing computability) of the totality of all algorithms is that it then became possible to prove that there are infinite classes of questions for which no algorithms exist: i.e., that there are *unsolvable decision problems* and *unsolvable computation problems*. What was being sought as a solution does not exist, though of course that fact itself is a solution in another sense. That is, the problem "Find an algorithm for this class of questions" is unsolvable, but the problem "Tell me whether an algorithm exists for this class of questions and if so find one" is solved, with the answer "No".

Let us interrupt the development to sketch the history. As remarked in § 2, Turing *computability* was one of three equivalent ways of characterizing exactly the functions for which algorithms exist which appeared in the 1930's. The concepts used in the other two were Church-Kleene *λ-definability* (developed at Princeton University in 1932 and 1933) and Herbrand-Gödel *general recursiveness* (presented by Gödel in his *1934* lectures at the Institute for Advanced Study in Princeton). Their equivalence was established by Kleene in *1936a* (with his *1936*). Church in *1936* proposed that the number-theoretic functions for which there are algorithms (the "effectively calculable" functions) are exactly the λ-definable functions, or equivalently the general recursive functions, which has come to be known as "Church's thesis".[7] Post in *1936*, who knew of the work at Princeton but not of Turing's work, gave much the same analysis as Turing *1936-7* (without detailed development). Church in *1936* gave examples of unsolvable

7 All this is elaborated in Kleene *1981*.

decision problems for an infinite class of questions arising in the theory of λ-definability (the existence of which is "Church's theorem") and Kleene in *1936* (knowing of Church's result) gave ones in the theory of general recursiveness, and Turing in *1936-7* in his theory of computability. From these results it was a straightforward step to the unsolvability of some decision problems arising in logic. Specifically, both Church in *1936a* and Turing in *1936-7* showed the unsolvability of the classical decision problem for provability in the pure predicate calculus.

Turing's part in this was done independently of (and slightly later than) the work at Princeton. Indeed, Turing learned of that work (including the results of Church *1936* and *1936a*) only in May 1936 or slightly earlier just as he was ready to send off the manuscript of his *1936-7* for publication, when an offprint or offprints arrived at Cambridge University of Church *1936* or of *1936* and *1936a* both. Thereupon he added references to it in his manuscript (received May 28, 1936), and an appendix (dated August 28, 1936) sketching a proof of the equivalence of his computability to λ-definability, which he expanded in *1937*.

Church wrote me on May 19, 1936 (thus shortly after he had enunciated his thesis and obtained his first unsolvability result), "What I would really like 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 (USA) and Markov (USSR), independently of each other, showed "the word problem for semi-groups" to be unsolvable. The "word problem for groups", a celebrated problem for algebraists, who had failed in intensive efforts to solve it, was shown to be unsolvable in a 143 page paper by Novikov (USSR) in *1955*, and the "homeomorphism problem for four-dimensional manifolds" in topology by Markov in *1958*. Thus the undecidability results of the 1930's began a development encompassing a wide range of applications in the mainstream of algebra and topology.

I proceed now to extract consequences from Turing's thesis as I have formulated it for number-theoretic functions, beginning with the existence of an unsolvable computation problem in Theorem 2 and of an unsolvable decision problem in Theorem 3.

We have seen that a particular Turing machine, so far as we are interested in it, is fully described by a table, giving the act it is to perform from each of the $(l+1)k$ active configurations for it.

Now I assign a positive integer, called its "index", to any Turing machine with a given table. For any natural numbers $b_0, \ldots, b_n$, let $< b_0, \ldots, b_n > = p_0^{b_0} \cdot \ldots \cdot p_n^{b_n}$ where $p_0, p_1, \ldots, p_i, \ldots = 2, 3, \ldots, p_i, \ldots$ are the prime numbers in order of magnitude. The *index* of a machine $\mathfrak{M}$ shall be $< c_1, \ldots, c_k >$ where k is the number of $\mathfrak{M}$'s active states, each $c_m = < d_{m0}, \ldots, d_{mt} >$

where l is the number of $\mathfrak{M}$'s symbols, and each

$$d_{mn} = < i, \begin{matrix} L \\ (C), \\ R \end{matrix} j > \quad \text{for} \quad \text{“}i, \begin{matrix} L \\ (C), \\ R \end{matrix} j\text{”}$$

the respective entry in its table with L,C,R rendered now as 0, 1, 2. Thus the index of the machine $\mathfrak{G}$ with the table given in Figure 1 is

$$<<< 0,1,0 >, < 1,2,2 >>, << 0,2,3 >, < 1,2,9 >>, \ \dots\ ,$$
$$<< 1,1,0 >, < 1,2,11 >>> .$$

Likewise I assign an *index* to a computation carried through any moment m. It is $< u_0, \dots, u_m >$ where each u_t represents the situation at Moment t, being $< v_t, r_t, q_t >$ where v_t represents the printing on the tape (that is, what we show at the right in the example in Figure 1 without the superscript, written now as a single number), r_t gives the position of the scanned square (the one bearing the superscript in Figure 1) counting the leftmost square as the 0th, and q_t indicates the machine state (the superscript in Figure 1). For example in Figure 1, $u_6 = << 0,1,1,0,1 >, 2, 7 >$.

We have been applying Gödel's method of numbering to our complexes of symbols. Turing did the like with another method of numbering.

Now I let "$T(i, a_1, \dots, a_n, x)$" stand for the following $n + 2$-ary relation:

> i is the index of a Turing machine (call it "Machine $\mathfrak{M}_i$") and x is the index of a computation by $\mathfrak{M}_i$ for $a_1, \dots, a_n$ as the arguments down through a moment at which it has just been completed to give a value (call that value "$\phi_i(a_1, \dots, a_n)$").

Much of the time I shall be employing these notations for $n = 1$, writing "$a_1, \dots, a_n$" then as "a".

The quantity $\phi_i(a_1, \dots, a_n)$ here is not defined for every $n + 1$-tuple of values of $i, a_1, \dots, a_n$. So it is what I call a *partial* number-theoretic function of $i, a_1, \dots, a_n$. Indeed, for each $i, a_1, \dots, a_n$, it is defined exactly when $T(i, a_1, \dots, a_n, x)$ holds for some x. Thus, recalling from § 1 that "(Ex)" is short for "there exists an x such that ",

$$\{\phi_i(a_1, \dots, a_n) \quad \text{is defined}\} \leftrightarrow (Ex)T(i, a_1, \dots, a_n, x).$$

With a fixed value of i, $\phi_i(a_1, \dots, a_n)$ as a function of $a_1, \dots, a_n$ may not (depending on i) be defined for every n-tuple of values of $a_1, \dots, a_n$. So ϕ_i is a partial function of n variables—the partial function which $\mathfrak{M}_i$ computes, if i is the index of a Turing machine. I use the term "partial function" to include the ordinary or "total" number-theoretic functions. A given partial number-theoretic function $\phi(a_1, \dots, a_n)$ is computed by a Turing machine $\mathfrak{M}$ exactly if, whenever $\mathfrak{M}$ is applied (in our standard way) to an n-tuple $a_1, \dots, a_n$, then $\mathfrak{M}$ will eventually stop with $a_1, \dots, a_n, \phi(a_1, \dots, a_n)$ represented on the tape (in our standard way) in case $\phi(a_1, \dots, a_n)$ is defined, and $\mathfrak{M}$ will fail to stop with the n-tuple $a_1, \dots, a_n, b$ for any b represented on the tape in case $\phi(a_n, \dots, a_n)$ is undefined. In the latter case, $\mathfrak{M}$ either runs forever or stops in some other kind of situation. If i is not the index of a Turing machine, ϕ_i is the totally undefined function of n variables.

Theorem 1: *The $n + 2$-ary relation $T(i, a_1, \dots, a_n, x)$ is decidable.*

Proof. I begin by taking "decidable" in the intuitive sense. I must show that there is a decision procedure (intuitively considered) by which, given any $n + 2$-tuple $i, a_1, \dots, a_n, x$ of natural numbers, we can decide whether or not $T(i, a_1, \dots, a_n, x)$ holds for it.

To decide, first we test whether i has the right structure to be the index of a Turing machine $\mathfrak{M}_i$. If it doesn't, we declare that $T(i, a_1, \dots, a_n, x)$ is false. If it does, we can read from it the table for $\mathfrak{M}_i$, using which we can apply $\mathfrak{M}_i$ to compute for $a_1, \dots, a_n$ as the arguments, to see that we obtain successively the situations represented by successive positive exponents in x (which, for $T(i, a_1, \dots, a_n, x)$ to be true, must be of the form $p_0^{u_0} \cdot \dots \cdot p_m^{u_m}$ with $u_0, \dots, u_m$ all > 0), up to the last one (represented by u_m), at which we look to see if the printing on the tape is of the form "$a_1, \dots, a_n, b$" for some b (all of this being leftmost) with the tape otherwise blank and the rightmost of the tallies scanned in State 0 (assumed at Moment m for the first time). If so, $T(i, a_1, \dots, a_n, x)$ is true (with $\phi_i(a_1, \dots, a_n) = b$); otherwise $T(i, a_1, \dots, a_n, x)$ is false (with $\phi_i(a_1, \dots, a_n)$ undefined). So $T(i, a_1, \dots, a_n, x)$ is decidable intuitively.

So now, by Turing's thesis, Theorem 1 holds with "decidable" meaning precisely Turing decidable. It is a straightforward exercise (using techniques such as were illustrated in § 5) to construct a Turing machine which calculates the (total) function $r(i, a_1, \dots, a_n, x)$ taking 0 or 1 as value according as $T(i, a_1, \dots, a_n, x)$ is true or false.

Theorem 2: *The function $\psi(a)$ defined by*

$$\psi(a) = \begin{cases} \phi_a(a) + 1 \text{ if } (Ex)T(a, a, x) \text{ (Case 1)}, \\ 0 \text{ otherwise (Case 2)} \end{cases}$$

is uncomputable.

Remark. The displayed equation does define $\psi(a)$ as an ordinary ("total") number-theoretic function, using in Case 1 the implication

$(Ex)T(a,a,x) \rightarrow \{\phi_a(a) \text{ is defined }\}$

which is included in the equivalence displayed preceding Theorem 1 (for $n = 1$ and $i = a$).

Proof of Theorem 2. The proof will necessarily rest on Turing's thesis.

Assume, for reductio ad absurdum, that ψ is computable. Then by the thesis, there is a machine $\mathfrak{M}_p$ which computes it. So ψ is the function ϕ_p computed by $\mathfrak{M}_p$, i.e., for all a, $\psi(a) = \phi_p(a)$, and in particular, $(*)\ \psi(p) = \phi_p(p)$. Since $\phi_p(p)$ is defined, using the other implication

$\{\phi_a(a) \text{ is defined }\} \rightarrow (Ex)T(a,a,x)$

we get $(Ex)T(p,p,x)$. So Case 1 of the definition of ψ applies for p as the value of a, and thus $\binom{*}{*}\ \psi(p) = \phi_p(p) + 1$. The equations $(*)$ and $\binom{*}{*}$ contradict each other. So, by reductio ad absurdum, ψ is not computable.

Theorem 3: *The property* $(Ex)T(a,a,x)$ *is undecidable.*

Proof. Why doesn't the definition of $\psi(a)$ in Theorem 2 give us an algorithm for computing it? Only because of our want of an algorithm for deciding, for any a, which case applies, i.e., whether $(Ex)T(a,a,x)$ holds (Case 1) or not (Case 2).

To elaborate, suppose there were such an algorithm. Take any a. If that algorithm gives "Yes" for "$(Ex)T(a,a,x)$?" (Case 1), then we can run the situations for Machine $\mathfrak{M}_a$, started with a on the tape at Moment 0, out to the moment at which $T(a,a,x)$ holds for x the index of the computation, read off from the tape the value $\phi_a(a)$ thus computed, and add 1 to get $\psi(a)$. If we get "No" for "$(Ex)T(a,a,x)$?" (Case 2), then immediately $\psi(a) = 0$.

So by Theorem 2, there is no algorithm for deciding $(Ex)T(a,a,x)$.▽

Theorem 3 is a counterpart of Church's theorem, based on his thesis, from *1936a*. His unsolvable problems are easily seen to be likewise of the form $(Ex)R(a,x)$ with $R(a,x)$ decidable. (That $T(a,a,x)$ is decidable is immediate from Theorem 1.) Many of the decision problems on which mathematicians had been working can be put in the same form. So further developments such as I sketched earlier in this section could be expected.

7. *Gödel's Theorems*

The part of the mathematical community attuned to foundational issues was much shaken, several years before the appearance of Church's thesis

and theorem in *1936* and the like in Turing *1936-7*, by Gödel's *1931* results. I shall indicate now how the work of Church or Turing provided the means for putting Gödel's results in a general setting.

I begin by observing that an "algorithmic theory" is one kind of a theory the Greeks had, for certain quite restricted mathematical subjects or classes of questions. Another is an "axiomatic-deductive theory", with which students are familiar from the example of Euclid's geometry. In such a theory, one starts with some propositions called *axioms* which are accepted as true immediately from their meanings, and *deduces* others called *theorems* (whose truth we might not have been ready to accept immediately) by using *logical inferences* which we accept as propagating truth forward. Thus an axiomatic-deductive theory provides a means for recognizing the truth of various propositions.

Of course algorithmic theories may receive vindication from axiomatic-deductive theories which establish the correctness of the answers they provide.

Later we shall touch on the role of axiomatic-deductive theories as methods for finding answers to questions of a class.

In modern times (beginning in the 1880's and 1890's) axiomatic-deductive theories, as typified by Euclid's geometry (c. 300 B.C.), have been sharpened up.

Some of Euclid's proofs were criticized as defective in that the theorems did not follow from only the mathematical assumptions formulated in the axioms. Unstated assumptions had been sneaked in through Euclid's use of figures. (Thus some anti-Euclidean jokesters proved "false" theorems like "Every triangle is isosceles" by using "fudged" figures, which nothing stated in Euclid's axioms prohibited.) These gaps in Euclid's treatment were mended, mainly by Pasch in *1882* and Hilbert in *1899*.

If one is going to police a theory so that all the mathematical assumptions are explicitly stated, there is no reason for not being equally careful in stating the logical principles used in its deductions or proofs. One can achieve rigor in both these components of the theory only after being equally careful about the syntax of the language that is used in stating the propositions. In doing all this, we wind up by employing a symbolic language specially designed for the theory in hand. You are familiar with the use of formulas in mathematics, which you are used to seeing run in with ordinary English words. But now we shall put everything into formulas. We call the result a *formal system*.

In a formal system, our purpose would not be accomplished if we had to rely on the meanings, rather than just the forms, of the symbols and the expressions built from them. For the question would arise whether unstated assumptions are being sneaked in through the meanings. The symbols, and

formulas suitably built from them, will have meanings, providing an *interpretation* of the formal system, which makes the system interesting to us and by which we recognize it to be a *formalization* of some portion of the preexisting "informal" mathematics. But we will have rules specifying from only their forms as linguistic objects which strings of (occurrences of) the symbols are *formulas*, and which strings of formulas are *proofs* (of the last formula of the string). Formulas of which proofs exist are *provable* (or are *theorems*) in the formal system.

Since these rules must provide us with complete fully-usable definitions of the classes of the formulas and of the proofs, they must afford us algorithms applying to the symbolic objects for the classes of questions, "Is this string of symbols a formula?" and "Is this string of formulas a proof?". The provision, in the process of establishing a formal system, of such algorithms is a feature of formal systems essential to our purposes in formalizing. This was perhaps only recognized explicitly since the late 1930's after the general concept of algorithm came into currency.

Formal systems formalizing portions of mathematics as it had been developed informally were given by Frege (*1883, 1903*), by Peano (*1894-1908*), and by Whitehead and Russell in their monumental *Principia Mathematica, 1910-13*.

As the most elementary nontrivial domain for applying these ideas, I take elementary number theory. This subject, in which systems of objects like the positive integers, or the natural numbers, or all the integers, are investigated, has had a long history going back to Euclid and before. In this century, especially since the two volumes of Hilbert and Bernays, *Grundlagen der Mathematik (Foundations of Mathematics), 1934, 1939*, we have become used to having the elementary theory of the natural numbers represented in a formal system. I shall call this system, in any one of several familiar versions, N.[8] I will speak more generally of a formal system S embracing elementary number theory, which can be just N or may include a formalization of other parts of mathematics, as for example real-number analysis.

Consider any such formal system S embracing elementary number theory—one which, as we may say, is "adequate" for expressing the propositions, and carrying out the proofs, of elementary number theory.

For each particular natural number $0, 1, 2, \ldots$ as the value of a, (Ex) $T(a, a, x)$ is a proposition of elementary number theory. For each of $a = 0, 1, 2, \ldots$, we can find a respective formula C_a of S which expresses the proposition $(Ex)T(a, a, x)$ under the interpretation of S. Indeed, there should be an algorithm for finding C_a from a (it could be embodied in a

8 A treatment is in Kleene *1952* Part II.

Turing machine operating on the natural number a to produce the corresponding formula C_a).

For a given a, if $(Ex)T(a, a, x)$ is true, there is an informal proof of it which consists in verifying for the right x that $T(a, a, x)$ holds. Assuming the adequacy of S for natural number theory, that proof can be "formalized" as a proof in S of the formula C_a expressing $(Ex)T(a, a, x)$. Let me use the convenient abbreviation "$\vdash_S C_a$" to say "C_a is provable in S". So we have, for every a,

(a) $$(Ex)T(a, a, x) \rightarrow \vdash_S C_a.$$

This much, I argue, is true for every formal system adequate for elementary number theory. In a detailed study, it can be established for particular such systems S by elementary reasoning.[9]

We would like S to have the feature that only true formulas are provable in it ("correctness"). Clearly a system not having this feature would be "subversive". This includes the following, for all a, where "$\neg$" is the symbol in S for "not", and "—" expresses "not" in our informal language:

(b) $$\vdash_S C_a \rightarrow (Ex)T(a, a, x),$$

(c) $$\vdash_S \neg C_a \rightarrow \overline{(Ex)}T(a, a, x).$$

Theorem 4[10]: *Let S be a formal system adequate for elementary number theory, as for example N, so that it has the foregoing features up through our statement that (a) holds for all a. A number p can be found such that:*

(i) *If (c) holds for all a, then $\overline{(Ex)}T(p, p, x)$.*
(ii) *If (c) holds for all a, then not $\vdash_S \neg C_p$.*
(iii) *If (b) and (c) hold for all a, then not $\vdash_S C_p$.*

Proof. Since for a particular formal system, there are algorithms for recognizing when a sequence of its symbols is a formula, and when a sequence of its formulas is a proof, we can put all the sequences of formulas into an infinite list, and go through this list checking which are proofs. Thus all the

9 E.g., for the N of Kleene *1952*, on pp. 243-244, using a device of Gödel *1931* (his *1934* β-function) and the "primitive recursiveness" of $T(i, a_1, \ldots, a_n, x)$ (hence of $T(a, a, x)$) mentioned below (late in §8).

10 By incorporating a device of Rosser *1936* using a bit more complicated decidable relation than $T(a, a, x)$, the hypotheses in (i)-(iii) can be simplified to "S is simply consistent" (see before Theorem 5 below).

proofs are brought into a list (obtained by suppressing from the aforesaid list the members which are not proofs). Also we have an algorithm for finding C_a, and hence $\neg C_a$, from a. Hence there is an algorithm which, applied for a given value of a, searches through the proofs in S in their order in the last-mentioned list, looking for a proof of $\neg C_a$, and if it finds one writes 0, but otherwise never stops searching. Now I apply Turing's thesis in the form relating to the computation of a partial number-theoretic function (of one variable, in this application) to claim that there is a Turing machine, say $\mathfrak{M}_p$, which carries out this algorithm. So $\mathfrak{M}_p$, applied to a, eventually stops giving a value (0 in fact) exactly if there is a proof of $\neg C_a$. Thus, for all a,

$$(d) \qquad (Ex)T(p,a,x) \leftrightarrow \vdash_S \neg C_a.$$

Now I prove (i)–(iii) successively.

(i) Assume (c), and for reduction ad absurdum, $(Ex)T(p,p,x)$. Then by (d) $\vdash_S \neg C_p$, whence by (c) $\overline{(Ex)}T(p,p,x)$, contradicting our assumption.

(ii) Assume (c). By (i) $\overline{(Ex)}T(p,p,x)$, when by (d) not $\vdash_S \neg C_p$.

(iii) Assume (b) and (c). By (i), $\overline{(Ex)}T(p,p,x)$, whence by (b), not $\vdash_S C_p$. ▽

Since C_p expresses $(Ex)T(p,p,x)$ under the interpretation of the symbolism of S, (i)–(iii) of the theorem can be combined in the following statement: *If (b) and (c) hold for all a, then a formula C_p of S can be found such that* (by (iii) and (ii)) *neither C_p nor its negation $\neg C_p$ is provable in S* (thus C_p is "formally undecidable" in S), *though* (by (i)) *$\neg C_p$ is true.*

What does the true but unprovable formula $\neg C_p$ express? $\neg C_p$ expresses $\overline{(Ex)}T(p,p,x)$ or equivalently $(x)\overline{T}(p,p,x)$; and by substituting p for a in (d) and negating both sides,

$$\overline{(Ex)}T(p,p,x) \leftrightarrow \{\text{ not } \vdash_S \neg C_p\}.$$

So $\neg C_p$ expresses its own unprovability.

The title of Gödel's *1931* paper, translated into English, is "On formally undecidable sentences of *Principia Mathematica* and related systems I."

In setting up a formal system S, we hope to have encompassed all the methods—the axioms and principles of deduction—for the mathematical theory we are formalizing. Will the body of these methods, as formalized in S, suffice for deciding as to the truth or falsity of each proposition of the theory, by there being a proof in S either of the formula C expressing the proposition or of the formula $\neg C$ expressing its negation? In the case of elementary number theory, "No"! Neither C_p nor $\neg C_p$ is provable in S. Of course, having proved Theorem 4, we then know that $\neg C_p$ is true, so we can get a stronger system than S by adding $\neg C_p$ as a new axiom! This of course is assuming (b) and (c), which are included in our supposition that S respects truth (or is correct).

In brief, S can't be both correct and complete. Assuming its correctness, S is (*simply*) *incomplete*, i.e., there is a pair of formulas C and $\neg C$ of S without free variables neither of which is provable in S.

Our Theorem 4 is a version of Gödel's "(first) incompleteness theorem" of *1931*. In this version, I have generalized his theorem to apply to all formal systems S adequate for elementary number theory, without regard to the details of the formalization. (In describing a particular formal system, there is an immense amount of detail.) We have assumed only the very general features of S outlined before I stated Theorem 4. These features are closely connected with the purposes for which formal systems were devised, in this case ones formalizing (at least) number theory. In *1931*, one could be uncertain whether Gödel's incompleteness theorem applies to systems quite remote in their details from *Principia Mathematica*. Another respect in which the theorem is generalized is that we have a fixed preassigned number-theoretic property $(Ex)T(a,a,x)$ such that the undecidable sentences for all systems S (under our very general conditions) express one or another instance of this property. Thus the theory of just this one property (as I remarked with another example in a talk in 1935) provides inexhaustible scope for the exercise of mathematical ingenuity.

Gödel's (first) *1931* incompleteness theorem presented a very sobering set-back to the program Hilbert had been promoting of embodying all mathematics up to a certain level in a formal system, and proving the system free of contradictions by "safe" methods. Contradictions had turned up in the higher reaches of mathematics around the beginning of this century, so mathematics did have a problem of purging itself of perjury.[11]

A formal system is (*simply*) *consistent* if in it no pair of contradictory formulas C and $\neg C$ are both provable. As a detailed study shows, in any formal system S adequate for elementary number theory, it is possible to find a formula, call it "Consis", expressing the simple consistency of S.

Theorem 5: *If the system S of Theorem 4 is simply consistent, then* not $\vdash_S$ Consis, *i.e., the formula expressing that fact is not provable in S.*

Proof. I begin by establishing

$$(1) \qquad \{S \text{ is simply consistent}\} \rightarrow \{\text{(c) holds for all } a\}.$$

So I assume S is simply consistent, $\vdash_S \neg C_a$ and $(Ex)T(a,a,x)$, and endeavor

11 Cf. e.g., Kleene *1952*, § 11.

to deduce a contradiction. But from $(Ex)T(a,a,x)$ by (a), $\vdash_S C_a$. This with $\vdash_S \neg C_a$ contradicts that S is simply consistent.

Combining (1) with (i) in Theorem 4,

(2) $$\{S \text{ is simply consistent } \} \rightarrow \overline{(Ex)}T(p,p,x).$$

The informal reasoning by which we have established (2) is entirely elementary. So if S is, as we are assuming, adequate for elementary number theory, it should be possible to formalize the proof of this implication in S.[12] Using "$\supset$" as the formal symbol in S for implication "$\rightarrow$" and remembering that Consis expresses "S is simply consistent" and that C_p expresses "$(Ex)T(p,p,x)$", formalizing the proof of (2) in S gives

(3) $$\vdash_S \text{Consis} \supset \neg C_p.$$

To complete the proof of Theorem 5, assume that S is simply consistent, and for reductio ad absurdum that $\vdash_S$ Consis. By the latter with (3), $\vdash_S \neg C_p$. By the former with (1), (c) holds for all a. Thence by (ii), not $\vdash_S \neg C_p$, contradicting $\vdash_S \neg C_p$. ▽

Theorem 5 is Gödel's second incompleteness theorem of *1931*, again generalized in respect to the system S. By this theorem, the "safe" methods that Hilbert had been proposing to use in proving a formal system consistent cannot, for a system embracing at least elementary theory, be methods all of which are formalized in the system. Hilbert's program for proving consistency must find among its "safe" methods ones outside of those formalized in the system.

8. *The Arithmetical Hierarchy*

A property of n natural numbers was called *arithmetical* by Gödel *1931* if it can be expressed in terms of constant and variable natural numbers, the addition and multiplication functions + and ·, equality =, and the logical symbolism of the first-order predicate calculus. From the results of Gödel and Kleene with the Church-Turing thesis, the class of the arithmetical properties of one variable a coincides with those expressible in the first display

12 This was done in detail for their system N by Hilbert and Bernays *1939*, pp. 283ff., especially pp. 306–324.

in Theorem 7 (and similarly with n variables $a_1,\ldots,a_n$).[13]

Theorem 6: *For each decidable relation $R(a_1,\ldots,a_n,x)$, there is a number p such that, for all $a_1,\ldots,a_n$,*

$$(Ex)R(a_1,\ldots,a_n,x) \leftrightarrow (Ex)T(p,a_1,\ldots,a_n,x).$$

Proof. Let $\mu x R(a_1,\ldots,a_n,x)$ be the least x such that $R(a_1,\ldots,a_n,x)$ if $(Ex)R(a_1,\ldots,a_n,x)$, and be undefined otherwise. This is a computable partial number-theoretic function. So by Turing's thesis, it is computed by a Turing machine, say $\mathfrak{M}_p$; so $\mu x R(a_1,\ldots,a_n,x)$ is $\phi_p(a_1,\ldots,a_n)$. Using the condition for $\phi_t(a_1,\ldots,a_n)$ to be defined (displayed preceding Theorem 1),

$$\begin{aligned}(Ex)R(a_1,\ldots,a_n,x) &\leftrightarrow \{\mu x R(a_1,\ldots,a_n,x) \text{ is defined}\}\\ &\leftrightarrow (Ex)T(p,a_1,\ldots,a_n,x).\ \triangledown\end{aligned}$$

We can describe Theorem 6 as an "enumeration theorem" for the n-ary relations of the form $(Ex)R(a_1,\ldots,a_n,x)$ with $R(a_1,\ldots,a_n,x)$ decidable. Thus $(Ex)T(i,a_1,\ldots,a_n,x)$ is an $n+1$-ary relation of like form such that, taking $i = 0,1,2,\ldots$ in it, we get a list or "enumeration" of all those n-ary relations. Thus $(Ex)T(i,a_1,\ldots,a_n,x)$ is an "enumerating relation" for them.

In stating my last theorem, I use "(x)" to express "for all x". The prefixes (Ex), (x), (Ey), (y), etc. are called "quantifiers". In proving the theorem, I use the following laws of logic:

$$\overline{(Ex)}A(x) \leftrightarrow (x)\overline{A}(x),\ \overline{(x)}A(x) \leftrightarrow (Ex)\overline{A}(x),\ \overline{\overline{A}} \leftrightarrow A.$$

Theorem 7: *Consider the following forms of properties of a natural number a, where each R is a decidable property or relation of its variables:*

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

To each form after the first, there is a property of a of that form which is neither expressible in the "dual" form (i.e., the form which is paired with it)

13 Details are in Kleene *1952* §§ 48, 49, 57.

nor in any of the forms with fewer quantifiers. Indeed,

$$(Ex)T(a,a,x) \quad (x)(Ey)T(a,a,x,y) \quad (Ex)(y)(Ez)T(a,a,x,y,z) \quad \ldots$$
$$(x)\overline{T}(a,a,x) \quad (Ex)(y)\overline{T}(a,a,x,y) \quad (x)(Ey)(z)\overline{T}(a,a,x,y,z) \quad \ldots$$

are such properties of the respective forms after the first.

Proof. Take for example the form $(x)(Ey)R(a,x,y)$.

I shall show that the property $(x)(Ey)T(a,a,x,y)$ (of this form) is not expressible in the dual form $(Ex)(y)R(a,x,y)$. Suppose it were, i.e., that, for all a,

$$(A) \qquad (x)(Ey)T(a,a,x,y) \leftrightarrow (Ex)(y)R(a,x,y)$$

for some decidable R. By Theorem 6, for this R there is a p such that, for all a,

$$(B) \qquad (Ey)\overline{R}(a,x,y) \leftrightarrow (Ey)T(p,a,x,y).$$

From (A) and (B) I deduce contradictory expressions for the proposition $(Ex)(y)R(p,x,y)$, thus:

$$(C) \qquad (Ex)(y)R(p,x,y) \leftrightarrow (x)(Ey)T(p,p,x,y),$$

$$(D) \qquad (Ex)(y)R(p,x,y) \leftrightarrow \overline{\overline{(Ex)}}(y)R(p,x,y)$$
$$\leftrightarrow \overline{(x)}(Ey)\overline{R}(p,x,y) \leftrightarrow \overline{(x)}(Ey)T(p,p,x,y).$$

$(x)(Ey)T(a,a,x,y)$ is not expressible in a form with fewer quantifiers, for example as $(x)R(a,x)$ with a decidable $R(a,x)$. For if $(x)(Ey)T(a,a,x,y) \leftrightarrow (x)R(a,x)$, then $(x)(Ey)T(a,a,x,y) \leftrightarrow (Ey)(x)(R(a,x) \;\&\; y = y)$, and $R(a,x) \;\&\; y = y$ is decidable.) ▽

This theorem gives a hierarchy of forms for properties of a natural number a such that increasing collections of properties are expressible using the more complicated forms (and different collections in the two forms of equal complexity).

Church's theorem and Gödel's theorem can be associated with two of the forms.

As an undecidable property for Church's theorem (our Theorem 3), we can take any of the properties shown at the bottom in Theorem 7, in particular the one $(Ex)T(a,a,x)$ we used in Theorem 3.

By arguments given above under the generalized Gödel theorem (our Theorem 4), for any number-theoretic property $P(a)$ reasonably expressible in S by formulas D_a, having all and only the formulas D_a provable which are true amounts to saying that, for all a,

$$P(a) \leftrightarrow \vdash_S D_a \leftrightarrow (EX)[X \text{ is a proof in } S \text{ of } D_a]$$
$$\leftrightarrow (Ex)[x \text{ is the index of a proof in } S \text{ of } D_a]$$
$$\leftrightarrow (Ex)R(a,x) \text{ (for a certain decidable } R).$$

This presupposes a choice of some system of indexing proofs in S by natural numbers, on the basis of proofs being strings of strings of symbols from a given list.[14] We get the last line because a formal system must have an algorithm for recognizing proofs, and for reasonable choices of a property $P(a)$ (such as $(x)\overline{T}(a,a,x)$ at the bottom in Theorem 7) an algorithm for finding D_a from a.

In summary, *having a formal system S in which all and only the true instances of a property $P(a)$ are "provable" (i.e., the formulas D_a expressing them in S are provable in S) entails $P(a)$ being expressible in the form $(Ex)R(a,x)$ with a decidable R.*

By Theorem 7, the property $(x)\overline{T}(a,a,x)$ ($\leftrightarrow \overline{(Ex)}T(a,a,x)$) is not so expressible, and neither are any of the properties shown at the bottom there with two or more quantifiers.

In particular, using $(x)\overline{T}(a,a,x)$ or equivalently $\overline{(Ex)}T(a,a,x)$ as the $P(a)$ and $\neg C_a$ as the D_a, the a's for which $\neg C_a$ is provable do not coincide with those for which $\overline{(Ex)}T(a,a,x)$ is true: there is a p such that *either* $\overline{(Ex)}T(p,p,x)$ and not $\vdash_S \neg C_p$ *or else* not $\overline{(Ex)}T(p,p,x)$ and $\vdash_S \neg C_p$. But (c) eliminates the latter, so we have (i) and (ii) of Theorem 4.

I also included (iii) there (which follows from (i) by the correctness), in keeping with Gödel's theme of "formal undecidability".

For $(Ex)T(a,a,x)$ as the $P(a)$ and C_a as the D_a, by (a) and (b) all and only the true instances are "provable".[15]

---

14 Using the technique of Gödel numbering which we employed in formulating $T_1(a_1,\ldots,a_n,x)$ in § 6.

15 A proof of Theorem 4 based on Theorem 3 is afforded by adding to (a)–(c) the supposition for reductio ad absurdum that, for all a,

$$(\bar{a}) \qquad \overline{(Ex)}T(a,a,x) \rightarrow \vdash \neg C_a,$$

so we would have all and only the true instances "provable" both of $\overline{(Ex)}T(a,a,x)$ (expressed by $\neg C_a$) and of $(Ex)T(a,a,x)$ (expressed by C_a). For then (using the argument of Turing *1936-7*, the fourth paragraph of § 11) we would get an algorithm for $(Ex)T(a,a,x)$,

Thus Church's theorem and the generalized Gödel theorem are two instances of when, if we desire to have a necessary and sufficient condition of a certain kind for a property $P(a)$, it may be impossible to get such a condition if we are asking for it to come from too low in the hierarchy. There are levels of mathematical concepts, concepts at a higher level having no equivalents at lower levels.

Let me continue with a bit more of the history. We could have continued from our proof of Theorem 1 to show easily that the relation $T(i, a_1, \ldots, a_n, x)$ is "primitive recursive", i.e., its "representing function" $r(i, a_1, \ldots, a_n, x)$ is "primitive recursive" (Kleene *1952*, p. 219). Furthermore, there is a primitive recursive function $v(x)$ such that, whenever $T(i, a_1, \ldots, a_n, x)$ holds, $\phi_i(a_1, \ldots, a_n) = v(x)$. This gives the substance, in our development of Turing's computability, of the "normal form theorem" which Kleene gave in *1936* using general recursiveness (and using partial recursiveness in *1952*, p. 330). Indeed, in our present treatment (using "μx" as in the proof of Theorem 6), we have for each n and i, $v(\mu x T(i, a_1, \ldots, a_n, x))$ as the normal form of the partial function $\phi_i(a_1, \ldots, a_n)$ of n variables. Altogether, this material affords a counterpart, in our development of Turing's ideas, of the "universal machine" which he gave in *1936-7*. Our universal machine is, for each n, the one that computes $\phi_i(a_1, \ldots, a_n)$ as a partial function of $n + 1$ variables.

Versions of our Theorems 3 and 4 were given in Kleene *1943* (and less simply in *1936*) in terms of the theory of general recursiveness.

The arithmetical hierarchy (our Theorem 7) was given likewise by Kleene in *1943* (1940 abstract in *Bulletin of the American Mathematical Society*), where as here he placed Church's theorem and the generalized Gödel theorem in relation to it, and independently by Mostowski in *1947*. Since Addison *1958* and Mostowski *1959*, Σ_k^j has become standard notation for the properties expressible in the k-quantifier form with an existential quantifier first, and Π_k^j with a universal quantifier first, with $j = 0$ for the arithmetical hierarchy; and with $j > 0$ for the hierarchies (going much further up) with higher types j of variables quantified which Kleene subsequently introduced and studied, adapting the idea of an "oracle" introduced by Turing in *1939*.[16]

Gödel at first did not accept Church's thesis (cf. Davis *1982*). Turing's arguments eventually persuaded him. Thus, in his Postscriptum to the Davis *1965* printing of his *1934* lectures, he says on p. 71, "... due to A.M. Turing's work, a precise and unquestionably adequate definition of the general concept of formal system can now be given, [and] the existence of undecidable arithmetical propositions and the non-demonstrability of the consistency of a system in the same system can now be proved rigorously for *every* consistent formal system containing a certain amount of finitary number theory."

As reported in Wang *1974*, p. 84,

> Gödel points out that the precise notion of mechanical procedures[17] is brought out clearly by Turing machines producing partial rather than general recursive functions. In other words, the intuitive notion does not require that a mechanical procedure should always terminate or succeed. A sometimes unsuccessful procedure, if sharply defined, still is a procedure, i.e., a well determined manner of proceeding. Hence we have an excellent example here of a concept which did not appear sharp to us but has become so as a result of a careful reflection. The resulting definition of the concept of mechanical by the sharp concept of "performable by a Turing machine" is both correct and unique. Unlike the more complex concept of always-terminating mechanical procedures, the unqualified concept, seen clearly now, has the same meaning for the intuitionists as for the classicists. Moreover it is absolutely impossible that anybody who understands the question and knows Turing's definition should decide for a different concept.

Partial recursive functions were introduced in Kleene *1938* and Turing machines were applied to compute them in his *1952* Chapter XIII. Under what Turing in *1936-7* § 10 called possibly the simplest way of defining a computable function $\phi(n)$ of an integral variable n, $\phi(n)$ is *computable* if there is a computable sequence in which 0 appears infinitely often, with $\phi(n)$ = the number of figures 1 between the n-th and the $(n + 1)$-th figure 0 [for $n = 0$, before the first]. That method is inapplicable to partial (not necessarily total) functions.

Allowing for the differences in Turing's context (he wants his machines to run ad infinitum printing on alternate squares an infinite sequence of 0's and 1's), in effect his "universal machine" computes a non-total partial function (the values being dual expansions of real numbers). The arguments for this function are the standard descriptions (S.D's) of machines $\mathfrak{M}$ (being the machine tables codified). When the universal machine is supplied a tape at the beginning of which the S.D of a machine $\mathfrak{M}$ is printed, the universal machine prints the sequence of 0's and 1's which is computed by $\mathfrak{M}$ if $\mathfrak{M}$ does compute such a sequence; and otherwise does not print such a sequence. Just as our undecidable property (Theorem 4) was that of a number a being one for which the partial function $\phi_a(a)$ is defined, he has

contradicting Theorem 3, thus. After picking a, search through the proofs in S for a proof either of C_a (finding which we answer "Yes") or of $\neg C_a$ (finding which we answer "No").

16 Some indications of all this are on pp. 63-64 of Kleene *1981*.

17 "alias 'algorithm[s]' or 'computation procedure[s]' or 'finite combinatorial procedure[s]'" (Gödel, in Davis *1965*, p. 72).

that of the S.D of a machine $\mathfrak{M}$ being one for which this partial function is defined.

Later in Wang *1974* (p. 325) Gödel is reported as objecting that "Turing 'completely disregards' that

(G) 'Mind, in its use, is not static, but constantly developing.'

... Gödel granted that

(F) The human computer is capable of only *finitely* many internal (mental) states.

holds '*at each stage* of the mind's development', but says that

(G)' '... there is no reason why this number [of mental states] should not converge to infinity in the course of its development.'"

I have condensed this as in Webb *1980*, p. 222. Another statement of it is in Remark 3 of Gödel's posthumous publication *1972a*. The recent literature has extensive and very meticulous discussions of this, delving into the nature of mental and mechanical procedures and computer theory, as in Webb *1980* and *1987* (which is an introductory note to the publication of Gödel *1972a*), Gandy *1980*, Dahlhaus and Makowsky *1985*, and in the present volume in the article by Penrose. Here I shall confine myself to a brief response, much as in Kleene *1987*.

If one chooses to believe (G)', it could imply that there would be no end to the possibilities for the human mind to invent stronger and stronger formal systems that would, in the face of Gödel's ever-renewing incompleteness theorem, prove more and more number-theoretic propositions of the form $(x)\overline{T}(a, a, x)$.

But, as I have said, our idea of an algorithm has been such that, in over two thousand years of examples, it has separated cases when mathematicians have agreed that a given procedure constitutes an algorithm from cases in which it does not. Thus algorithms have been procedures that mathematicians can describe completely to one another *in advance* of their application for various choices of the arguments. How could someone describe completely to me *in a finite interview* a process for finding the values of a number-theoretic function, the execution of which process for various arguments would be keyed to more than the *finite* subset of our mental states that would have developed by the end of the interview, though the total number of our mental states might converge to infinity if we were immortal. Thus Gödel's remarks do not shake my belief in the Church-Turing thesis in the **Public Processes Version** of Hofstadter *1980*, p. 562 (stated there for deciding whether a property holds rather than for calculating the value of a function).

Gödel continued from (G)', "Now there may be exact systematic methods of accelerating, specializing, and uniquely determining this development,

e.g. by asking the right questions on the basis of a mechanical procedure", though he admitted, "the precise definition of a procedure of this kind would require a substantial deepening of our understanding of the basic operations of the mind." I think it is pie in the sky! And I would add, to do us any good, such a procedure would in turn need to be a process finitely describable in advance of its application (an algorithm), and then the combination of the two algorithms would constitute one algorithm which by my lights would need to be finitely describable in advance, on the basis of a finite budget of presently-known mental states, thus under the Church-Turing thesis.

References

Addison, John W.

1958 Separation principles in the hierarchies of classical and effective descriptive set theory. *Fund. Math.* **46** (1958) 123–135.

Church, Alonzo

1936 An unsolvable problem of elementary number theory. *Am. J. Math.* **58** (1936) 345–363.

1936a A note on the Entscheidungsproblem. *J. Symb. Log.* **1** (1936) 40–41. Correction, ibid., 101–102.

Dahlhaus, E., and J.A. Makowsky

1986 Computable directory queries. In: *Proceedings of CAAP '86*, 11th Colloquium on Trees and Algebra in Programming, Nice, March 24–26, 1986, ed. P. Franchi-Zannettacci, Lecture Notes on Computer Science 214, pp. 254–265. Berlin: Springer-Verlag (1986).

Davis, Martin (ed.)

1965 *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Hewlett, NY: Raven Press (1965).

1982 Why Gödel didn't have Church's thesis. *Inf. Contr.* **54** (1982) 3–24.

Dedekind, Richard

1888 *Was sind und was sollen die Zahlen?* Braunschweig: Vieweg & Sohn (1888).

Frege, Gottlob

1893 *Grundgesetze der Arithmetik, begriffsschriftlich abgeleitet*, vol. 1. Jena: H. Pohle (1893).

1903 Ibid., vol. 2 (1903).

Gandy, Robin

1980 Church's thesis and principles for mechanisms. In: *The Kleene Symposium*, pp. 123–148. Amsterdam/New York/Oxford: North-Holland Publ. Co. (1980).

Gödel, Kurt

1931 Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monats. Math. Ph.* 38 (1931) 173–198.

1934 On undecidable propositions of formal mathematical systems. Mimeographed lecture notes, taken by Kleene and Rosser. Printed with revisions in Davis *1965*, pp. 39–74.

1972a Some remarks on the undecidability results. Forthcoming in *Kurt Gödel, Collected Works*, vol. II. New York: Oxford University Press, and Oxford: Clarendon Press.

Hilbert, David

1899 *Grundlagen der Geometrie*, 7th ed. Leipzig/Berlin: Teubner-Verlag (1930).

1926 Über das Unendliche. *Math. Annal.* 95 (1926) 161–190.

Hilbert, David, and Paul Bernays

1934 *Grundlagen der Mathematik*, vol. I. Berlin: Springer-Verlag (1934).

1939 Ibid., vol. II (1939).

Hofstadter, Douglas R.

1980 *Gödel, Escher, Bach: An Eternal Golden Braid*. Vintage Books Edition. New York: Random House (1980).

Kleene, Stephen C.

1936 General recursive functions of natural numbers. *Math. Annal.* 112 (1936) 727–742. For an erratum, a simplification and an addendum, see Davis *1965*, p. 253 (read "$(Ey)T_1(\theta(q),\theta(q),y)$" in place of "$(y)T_1(\theta(b),\theta(b),y)$").

1936a λ-definability and recursiveness. *Duke Math. J.* 2 (1936) 340–353.

1938 On notation for ordinal numbers. *J. Symb. Log.* 3 (1938) 150–155.

1943 Recursive predicates and quantifiers. *T. Am. Math. S.* 53 (1943) 41–73. For a correction and an addendum, see Davis *1965*, pp. 254 and 287.

1952 *Introduction to Metamathematics*, 9th reprint. Amsterdam: North-Holland Publ. Co. (1988).

1981 Origins of recursive function theory. *Ann. Hist. C.* 3 (1981) 52–67. Corrections in Davis *1982*, footnotes 10 and 12.

1987 Reflections on Church's thesis. *Notre Dame J. Form. Log.* 28 (1987) 490–498.

Markov, A.A.

1947 On the impossibility of certain algorithms in the theory of associative systems. *Comptes rendus (Doklady) de l'Académie des Sciences de l'URSS, n.s.* 55 (1947) 583–586 (tr. from the Russian).

1958 Nerazrešimost' problemy gomeomorfii (Unsolvability of the problem of homeomorphy). In: *Proceedings of the International Congress of Mathematicians*, Edinburgh, 1958, pp. 300–306.

Mostowski, Andrzej

1947 On definable sets of positive integers. *Fund. Math.* 34 (1947) 81–112.

1959 On varying degrees of constructivism. In: *Constructivity in Mathematics, Proceedings of Colloquium, Amsterdam, 1957*, pp. 178–194. Amsterdam: North-Holland Publ. Co. (1959).

Novikov, P.S.

1955 On the algorithmic unsolvability of the word problem in group theory. In: *AMS Transl. (2)* 9 (1958) 1–222. Original in Russian, 1955.

Pasch, Moritz

1882 *Vorlesungen über neuere Geometrie*. Leipzig: Teubner-Verlag (1882).

Peano, Guiseppe

1889 *Arithmetices Principia, Nova Methodo Exposita*. Turin: Bocca (1889).

1894-1908 *Formulaire de Mathématiques*, Introduction and five volumes, edited by Peano and written by him and seven collaborators. Turin (1889).

Péter, Rózsa

1934 Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion. *Math. Annal.* 110 (1934) 612–632.

Post, Emil L.

1936 Finite combinatory processes—formulation I. *J. Symb. Log.* 1 (1936) 103–105.

1947 Recursive unsolvability of a problem of Thue. *J. Symb. Log.* 12 (1947) 1–11.

Rosser, J. Barkley

1936 Extensions of some theorems of Gödel and Church. *J. Symb. Log.* 1 (1936) 87–91.

Skolem, Thoralf

1923 Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich. *Skrifter utgit av Videnskapsselskapet i Kristiana, I. Matematisk-naturvidenskabelig klasse 1923*, no. 6.

Turing, Alan Mathison

1936-7 On computable numbers, with an application to the Entscheidungsproblem. *P. Lond. Math. Soc. (2)* 42 (1936-7) 230–265. A correction, ibid. 43 (1937) 544–546. A useful critique is in the appendix to *Post 1947*.

1937 Computability and λ-definability. *J. Symb. Log.* 2 (1937) 153–163.

1939 Systems of logic based on ordinals. *P. Lond. Math. Soc. (2)* 45 (1939) 161–228.

Wang, Hao

1974 *From Mathematics to Philosophy*. London: Routledge and Kegan Paul, and New York: Humanities Press (1974).

Webb, Judson Chambers

1980 *Mechanism, Mentalism and Metamathematics. An Essay on Finitism*. Dordrecht/Boston/London: D. Reidel (1980).

1987 Introductory note to Remark 3 of Gödel *1972a*. Forthcoming in *Kurt Gödel, Collected Works*, vol. II (see Gödel *1972a*).

Whitehead, Alfred North, and Bertrand Russell

1910-13 *Principia Mathematica*. vol. 1 1910, vol. 2 1912, vol. 3 1913, Cambridge, England: Cambridge University Press (1910-13).