BULLETIN (New Series) OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 5, Number 1, July 1981

THE THEORY OF RECURSIVE FUNCTIONS, APPROACHING ITS CENTENNIAL[1]

(*Elementarrekursiontheorie vom höheren Standpunkte aus.*[2])

BY STEPHEN C. KLEENE

ABSTRACT. An algorithm is a procedure, given by a finite set of instructions, to serve as follows in relation to a given infinite class of questions. (a) If we select any question from the class, the instructions will tell us how to perform a step. (b) After any step, if we do not receive the answer then, the instructions together with the existing situation will tell us what step to take next. (c) The instructions will enable us to recognize when a situation is reached in which the answer is before us, and to read it off then; and this will eventually happen if the question has an answer. In "steps" and "situations", what are we handling? Since there must be no ambiguity, surely some kind of regular complexes of occurrences of symbols from a given finite list. Such complexes can be coded by positive integers. Consider specifically an algorithm for computing a functional $\phi(\Theta; \mathfrak{A})$ where Θ are number-theoretic function variables and $\mathfrak{A}$ are natural number variables. A question is selected from the infinite class "What is the value of $\phi(\Theta; \mathfrak{A})$?" by specifying $(\Theta; \mathfrak{A})$. I gain some space for maneuver by using instead $(\Theta; \mathfrak{A}, 0)$. After any step, the situation will be represented by $(\Theta; \mathfrak{A}, b)$ where b is the code for the complex of symbols in it. By (c), there must be a functional $\chi(\Theta; \mathfrak{A}, b)$, for which we already know how to get the values, such that, in the situation represented by $(\Theta; \mathfrak{A}, b)$, $\chi(\Theta; \mathfrak{A}, b) = 0$ if the answer is not before us, and otherwise $\chi(\Theta; \mathfrak{A}, b) = \phi(\Theta; \mathfrak{A}, b) + 1$ where $\phi(\Theta; \mathfrak{A}, b)$ is the answer; and $\chi(\Theta; \mathfrak{A}, 0) = 0$. By (b) and (a), there must likewise be a functional $\rho(\Theta; \mathfrak{A}, b)$ such that in the situation represented by $(\Theta; \mathfrak{A}, b)$, if $\chi(\Theta; \mathfrak{A}, b) = 0$ then $(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, b))$ represents the situation after the next step; and $(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, 0))$ represents the situation after the first step. Now, putting $\phi(\Theta; \mathfrak{A}, b) \simeq \phi(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, b))$ if $\chi(\Theta; \mathfrak{A}, b) = 0$, $\simeq \chi(\Theta; \mathfrak{A}, b) \dot{-} 1$ if $\chi(\Theta; \mathfrak{A}, b) > 0$, we have a definition of $\phi(\Theta; \mathfrak{A}, b)$ of the form $\phi(\Theta; \mathfrak{A}, b) \simeq \psi(\lambda\mathfrak{A}b\phi(\Theta; \mathfrak{A}, b), \Theta; \mathfrak{A}, b)$ as in Kleene's first recursion theorem [**1952**, p. 348]; and $\phi(\Theta; \mathfrak{A}) \simeq \phi(\Theta; \mathfrak{A}, 0)$. Thence it is argued that the first recursion theorem, in a proper setting, enables all functionals $\phi(\Theta; \mathfrak{A})$ to be defined for which there are algorithms; and consequences are deduced therefrom.

The theory of recursive functions is nearly one hundred years old. For nearly the first fifty years it was the theory of what are now called "primitive

This paper was presented as an address to the Society at its meeting in Kenosha, Wisconsin (The University of Wisconsin, Parkside) in 1980; received by the editors November 15, 1980.

1980 *Mathematics Subject Classification.* Primary 03D20; Secondary 03D10, 03A05.

Key words and phrases. The first recursion theorem, schemata for recursive definitions, primitive recursion, algorithms, Herbrand-Gödel general recursiveness, λ-definability, Turing computability, partial recursiveness, Church's thesis, Church's theorem, Gödel's theorem.

[1]I prepared this paper for a general audience. Then I was amazed to find on the program twenty-three other papers (including four special sessions organized by Richard A. Shore) on recursion theory, which illustrates the health of the nonagenarian.

[2]Kleene is a North German form of Klein.

0002-9904/81/0000-0303/$05.75

recursive functions" and some extensions thereof. A little under fifty years ago general recursive functions and equivalents came on the stage.

The simplest infinite mathematical system is that of the natural numbers, i.e. the nonnegative integers

$$0, 1, 2, \ldots .$$

I choose to deal with these rather than with the positive integers

$$1, 2, 3, \ldots ,$$

and I shall transpose to the natural numbers the part of the work I review that was phrased in terms of the positive integers.[3]

On Tuesday, September 21, 1886, Kronecker declared, "Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk (God made the integers, all the rest is work of man)." We know so well the natural number system from Peano's five axioms, published by him in **[1889]** and **[1891]**. In fact, as Peano acknowledged in **[1891**, p. 93], these axioms come from the definition of a simply infinite system in Dedekind, "Was sind und was sollen die Zahlen?" **[1888]**. I think we can say that recursive function theory was born there ninety-two years ago with Dedekind's Theorem 126 ("Satz der Definition durch Induktion") that functions can be defined by primitive recursion, as I shall presently illustrate.[4]

Under Dedekind and Peano's treatment, the natural numbers constitute the system of objects obtained by starting with an object 0 ("zero"), and repeatedly generating a next object by an operation $'$ ("successor" or "$+1$").[5] That one thus generates only natural numbers is stated in the first two Peano axioms. That objects thus differently generated are distinct is stated in the third and fourth Peano axioms. That all the natural numbers are so generated is given by the fifth Peano axiom, which says that we can parallel the generation of the natural numbers using 0 and $'$ by proofs that all the natural numbers have a mathematical property, i.e. proofs by mathematical induction. Indeed, in the set-theoretic terminology which was appearing about the same time, a subset of the natural numbers containing 0 and closed under the operation $'$ is the set of all the natural numbers.

Recursion, or definition by induction, is the principle of definition corresponding to proof by induction. The example that comes to mind first is the definition of the sum function $a + b$, by the two equations

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

---

[3]Of course the two systems by themselves are isomorphic. They differ in their application as cardinal numbers, and in the definitions (suited to that application) adopted for the functions $a + b$, $a \cdot b$ and a^b.

[4]Péter **[1934]** introduced the name "primitive recursion", and the name "primitive recursive functions" came in with my **[1936]** for what had been called simply "recursive functions" by Gödel **[1931]**, **[1934]** and Péter **[1934]**, **[1935]**, **[1936]**. "Recursion" (but not "recursive function") appeared in Skolem **[1923**, p. 11] and Hilbert **[1926]**.

[5]Dedekind and Peano dealt with the positive integers $1, 2, 3, \ldots$; but as stated I am transposing to the natural numbers $0, 1, 2, \ldots$. Other work cited below which was phrased in terms of the positive integers is that of Skolem, Church, Kleene before **[1936]**, Post and Smullyan.

You may wonder why in 1980 I am dwelling on this. There is a point of view (developed by me since 1977) from which the quantum jump at midlife of recursion theory from dealing only with primitive and other special recursive functions to general recursive functions can be based on thinking through the form of a recursion as exemplified by this definition of $a + b$.

However, let me first deal with this less boldly. The definition of $a + b$ comes under the general form of a primitive recursion on b with $n - 1$ ($\geqslant 0$) parameters $a_2, \ldots, a_n$,[4]

$$\begin{cases} \phi(0, a_2, \ldots, a_n) = \psi(a_2, \ldots, a_n), \\ \phi(b', a_2, \ldots, a_n) = \chi(b, \phi(b, a_2, \ldots, a_n), a_2, \ldots, a_n). \end{cases} \tag{2}$$

In our example of $a + b$, there is one parameter a ($n = 2$), and the functions ψ and χ assumed as already known are

$$\psi(a) = a, \qquad \chi(b, c, a) = c'. \tag{3}$$

To complete the definition of the class of the primitive recursive functions, I need to make precise the context in which primitive recursions are to be used. Specifically, what functions are to be taken as known initially, and what other kinds of definition may be interspersed with primitive recursions? I shall make these items precise presently.

Dedekind [**1888**] and Peano [**1889**] followed the recursion for the sum $a + b$ by those for the product $a \cdot b$ and the exponent function a^b,[5]

$$\begin{cases} a \cdot 0 = 0, \\ a \cdot b' = (a \cdot b) + a, \end{cases} \tag{4}$$

$$\begin{cases} a^0 = 1, \\ a^{b'} = a^b \cdot a. \end{cases} \tag{5}$$

The characteristic feature of primitive recursion is that, e.g. with one parameter a, the value of the function ϕ being defined for any given pair (b, a) of arguments with $b > 0$ is made to depend via a previously known function on its value for the pair $(b - 1, a)$ (besides on b and a), and so by iteration on its values successively for $(b - 2, a)$, $(b - 3, a)$, ... and ultimately for $(0, a)$, which value is given by a previously known function of a. This makes Dedekind's [**1888**] theorem that a function is defined by the recursion quite transparent.

The functions commonly used in arithmetic or elementary number theory are primitive recursive. Something of a calculus of primitive recursive functions was developed by Skolem [**1923**], Gödel [**1931**] and Péter [**1934**], from which my exposition in [**1952**, Chapter IX] drew heavily.

In [**1928**] Ackermann gave an example of a recursion on two variables n and b simultaneously (with one parameter a). In this recursion, the value of the function ξ in question for a triple (n, b, a) of arguments with n, b both > 0 is made to depend on its value for certain triples $(\bar{n}, \bar{b}, a)$ with $(\bar{n}, \bar{b})$ preceding (n, b) in the ordering by the ordinals $n\omega + b$, and is given by a

known function if either $n = 0$ or $b = 0$,

$$\begin{cases} \xi(0, b, a) = a + b, \\ \xi(n', 0, a) = \alpha(n, a) \quad (= 0 \text{ if } n = 0, 1 \text{ if } n = 1, a \text{ if } n > 1), \\ \xi(n', b', a) = \xi(n, \xi(n', b, a), a). \end{cases} \tag{6}$$

Ackermann proved that this function ξ is not (as we now say) primitive recursive. Indeed, $\xi(0, b, a) = a + b$, $\xi(1, b, a) = a \cdot b$, $\xi(2, b, a) = a^b, \ldots$; and $\xi(b, b, b)$ majorizes every primitive recursive function $\phi(b)$.

Generalizing from the $k = 1$ case (primitive recursion) and the $k = 2$ case (exemplified by Ackermann's double recursion), Péter [**1936**] studied the hierarchy of increasing classes of functions definable using k-fold recursions for $k = 1, 2, 3, \ldots$. So much for the first phase of recursive function theory.

A half century ago there was a great ferment in thinking about the foundations of mathematics. Stimulated by Cantor's development of set theory ([**1874**], [**1895–7**]) and the ensuing paradoxes (from 1895 on), the schools of logicism, intuitionism and formalism had taken the field. The formalists (after Hilbert) had put portions of classical mathematics into the setting of fully formalized systems, and proposed to study these systems (in particular to prove their consistency) by "finitary" methods, indeed by the methods of elementary intuitionism (after Brouwer). Hilbert had posed in [**1918**] the problem of the solvability in principle of each mathematical question, and the problem of the decidability of a mathematical question through a finite number of operations. Applied to a formal system, we thus have the problem of finding a decision procedure (Entscheidungsverfahren) whereby, given any formula of the system, its provability or unprovability in that system can be decided in finitely many steps. Thus arises the Entscheidungsproblem or decision problem, which had also appeared in Schröder [**1895**] and Löwenheim [**1915**].

What is all this really about? Let us try to view it from a standpoint above the details of one or another particular formal system. What does a formal system really do, and what would a decision procedure be?

The germ of the concept which gives us the overview we want now has been in mathematics for more than two millenia: the idea of algorithms.

An *algorithm* is a method or procedure, established by a finite set of rules or instructions, to serve as follows in relation to a given infinite class of questions. (a) *After* the procedure has been described, if we then select *any* question from the class, the procedure will apply and tell us how to perform a step, the first of a sequence of one or more steps. (b) After any step, if we do not then receive the answer to the question selected, the instructions together with the existing situation (to which that step led) will tell us what step to take next. (c) The instructions will enable us to recognize when a situation is reached in which the answer is before us, and to read it off then; and this will eventually happen (after a finite number of steps). In performing the steps and reading off the answer, we have only to follow the instructions, like robots; no insight or ingenuity or invention is required of us.[6]

[6]This clearly paraphrases the description of an algorithm as it naturally came to my mind in [**1967**, p. 223] and [**1969**, p. 335].

To be more specific, consider various sorts of infinite classes of questions. To begin with, let us take as our object domain the natural numbers, or a similar countably infinite domain of objects each finitely describable by its generation or position as a member of the domain.

As we know, we can define functions of one or more variables, each ranging over the natural numbers, e.g. by recursion. We can also define properties and relations (i.e. propositional functions, or as I will preferably say "predicates") of natural-number variables, by defining functions and setting them equal to 0, and further by applying logical operations (including the quantifiers "for all x" and "(there) exists an x (such that)", or in symbols "(x)" and "(Ex)") to predicates already defined.

If we have a predicate $P(a)$ or $P(a_1, \ldots, a_n)$ with natural-number variables, we may have an algorithm for the infinite class of questions "Is $P(a)$ true?" or "Is $P(a_1, \ldots, a_n)$ true?". We get a particular question of the class by selecting a particular natural number as the value of a or a particular n-tuple of natural numbers as the values of $a_1, \ldots, a_n$. These questions call for "yes" or "no" as answers. The algorithm is then also called a *decision procedure* for the predicate P. If such exists, the predicate is called *decidable*.

Similarly with a number-theoretic function $\phi(a)$ or $\phi(a_1, \ldots, a_n)$, an algorithm for the infinite class of questions "What is the value of $\phi(a)$?" or "What is the value of $\phi(a_1, \ldots, a_n)$?" is also called a *computation procedure* for the function ϕ. If such exists, the function is called *computable*.

Since to any predicate $P(a)$ or $P(a_1, \ldots, a_n)$ we can correlate a function $\phi(a)$ or $\phi(a_1, \ldots, a_n)$ taking the value 0 or 1 according as the proposition taken as value by the predicate is true or false, decision procedures are encompassed under computation procedures.

The example of Euclid's greatest common divisor algorithm (fourth century B.C.) illustrates that algorithms were present in Greek mathematics. The name "algorithm" is a corruption of the last part of the name of Abu Abdullah abu Jafar Muhammad ibn Musa al-Khowarizmi, the ninth century Arabian mathematician who came from the Khowarizm oasis in central Asia.[7]

The objective of formalizing a mathematical theory a la Hilbert is to remove all uncertainty about what constitutes a proof in the theory, of course only after likewise fully specifying what are the formulas expressing propositions of the theory. Given a proposed proof, it must be possible by application of the rules defining the formal system to check in finitely many steps whether or not it really is a proof.

In brief, disregarding differences among various ways of carrying out the details of formalization, the aim of embodying a theory in a formal system is to establish an algorithm for the notion of proof (besides of formula) in the theory. And the decision problem for a given formal system is the problem of

[7]I have in my hand an autographed picture of Al-Khowarizmi.–Actually, it is likeness circulated at the Symposium dedicated to Al-Khowarizmi, at Urgench in Uzbekistan, September 16–22, 1979, inscribed in Arabic with his name by Heinz Zemanek of Vienna, an Al-Khowarizmi scholar.

finding an algorithm for the notion of provability (i.e. of the existence of a proof) in it.

So we fall back on the two-thousand-year-old idea of algorithms. What this mathematical tradition gives is numerous examples in which mathematicians agree that an algorithm is or is not provided. For example, a primitive recursion (2) provides an algorithm for the function ϕ defined by it (assuming we already have algorithms for the functions ψ and χ). But a definition of the form $\phi(a)$ = (the least x such that $\psi(a, x) = 0$ if such an x exists, and 0 otherwise), where there is a known algorithm for ψ, does not of itself provide an algorithm for ϕ. With a particular ψ, some theory might be developed that would lead to an equivalent formulation that would provide an algorithm. For example, this happens if we succeed in proving that, if there is an x such that $\psi(a, x) = 0$, there is such an $x \leqslant \theta(a)$ where θ is a function for which we have an algorithm.

In the world of mathematical ideas existent in 1930, one had no basis for establishing the impossibility of there being an algorithm for a given class of questions. For that one would need, further than particular examples, some characterization of the shapes of all possible algorithms on a given domain.

The second half-century of recursive function theory is marked by the introduction of such a characterization, in a number of equivalent versions. At the beginning of the 1930's, no overview was possible on the most fundamental problems of the foundations of mathematics without this step.

We have already seen that primitive recursions, and more generally k-fold recursions, define functions algorithmically, while descriptive definitions like "the least x such that $\psi(a, x) = 0$ if such an x exists, and 0 otherwise" do not, at least not without supplementation.

This suggests trying to define all possible algorithms on the natural numbers by generalizing from the primitive and k-fold recursions.

Gödel in [**1934**], building on a suggestion of Herbrand in a letter in 1931, gave a definition of "general recursive functions" which took for generalization the feature that the equations giving the values of the function defined by such a recursion are formally derivable from the equations of the recursion by using a substitution rule and a replacement rule. However, Gödel, according to a letter he wrote to Martin Davis on 15 February 1965, "was, at the time of [his 1934] lectures, not at all convinced that [this] concept of recursion comprises all possible recursions".

Church had been pushing on the problem of characterizing all number-theoretic functions for which there are algorithms ("effectively calculable" as he phrased it); and as a graduate student under him I had been finding in example after example of functions for which there are algorithms that their definitions could be expressed in a certain formalism affording algorithms (arising out of Church [**1932**], [**1933**]) which we called the "λ-calculus", i.e. those functions are "λ-definable". This work of mine was mainly done in 1932 (published in [**1935**]). Thereafter Church announced his "thesis" (published in [**1936**], and so named in my [**1943**], [**1952**]) that all the functions for which there are algorithms are Church-Kleene λ-definable, or equivalently (as was proved in his [**1936**] and my [**1936a**]) Herbrand-Gödel general recursive.

Turing in [1936–7] reached the same conclusion independently, using as a third equivalent notion computable by idealized computing machines of a certain kind (error-free and with unbounded memory), or "Turing computable". Post in a brief note [1936] gave the same idea independently.

Several other equivalents have appeared subsequently, in particular a formulation by Post using his canonical systems [1943], Markov's theory of algorithms [1951], [1954], and a formulation of Smullyan using his elementary formal systems [1961].

These various equivalent formulations have respective merits (discussed for some in my [1981a]). Turing's computability is intrinsically persuasive in the sense that the ideas embodied in it directly support the thesis that the functions encompassed are all for which there are algorithms; λ-definability is not intrinsically persuasive (the thesis using it was supported not by the concept itself but rather by results established about it)[8] and general recursiveness scarcely so (its author Gödel being at the time not at all persuaded).

What I propose to do in the rest of this talk is to develop another equivalent formulation, which comes out of generalizing from primitive and other special recursions in a different direction than Gödel did in [1934], and which I believe I can say is intrinsically persuasive.

In my development I shall need the function cs (for "case") defined by

$$\mathrm{cs}(a, b, c) = \begin{cases} b & \text{if } a = 0, \\ c & \text{if } a > 0, \end{cases} \tag{7}$$

and the predecessor function $\dot{-}\ 1$ (sometimes written pd), defined by

$$a \dot{-} 1 = \begin{cases} a - 1 & \text{if } a > 0, \\ 0 & \text{if } a = 0. \end{cases} \tag{8}$$

Now the pair of recursion equations (1) for $a + b$ can be written as one,

$$a + b = \mathrm{cs}(b, a, (a + (b \dot{-} 1))'). \tag{9}$$

The general primitive recursion (2) can similarly be written

$$\phi(a, \mathfrak{B}) = \mathrm{cs}(a, \psi(\mathfrak{B}), \chi(a \dot{-} 1, \phi(a \dot{-} 1, \mathfrak{B}), \mathfrak{B})), \tag{10}$$

where I have changed the notation to use "a" as the recursion variable and "$\mathfrak{B}$" for a list of $n - 1 \geqslant 0$ other distinct variables as parameters. (German capitals are used from here on in abbreviating lists of distinct number variables.)

In (10), we can think of the right side as the result of using a known functional ψ with one n-place function variable η, for which in the recursion we substitute the function ϕ being defined, and n number variables $\mathfrak{A}$. Thus we construe (10) as having the form

$$\phi(\mathfrak{A}) \simeq \psi(\phi; \mathfrak{A}). \tag{11}$$

Similarly, a double recursion such as (6) can be put in the form (11).

We may want to use several such recursions, perhaps with intervening composition steps, in a row, so that the ψ's for later ones are defined by

[8] It was not clear initially that even the predecessor function $a \dot{-} 1$ is λ-definable; cf. my [1981a].

earlier ones. This leads us to generalize from (11) to allow besides η also more function variables $\theta_1, \ldots, \theta_l$ ($= \Theta$ briefly), thus

$$\begin{aligned} \phi(\Theta; \mathfrak{A}) &\simeq \psi(\phi, \Theta; \mathfrak{A}) \\ &\simeq \psi(\lambda\mathfrak{A}\, \phi(\Theta; \mathfrak{A}), \Theta; \mathfrak{A}). \end{aligned} \tag{12}$$

The third expression makes it explicit (using Church's λ-notation [**1932**]) that the ϕ in the second is being considered as a function of the number variables $\mathfrak{A}$ for given values of the function variables Θ.

In (11) when its ψ is given by the right side of the primitive recursion (10), the recursion (with the definitions of cs, $\dot{-}\, 1$, ψ, χ) determines the function ϕ completely, the values being obtained by repeated applications of the equation. Similarly with the double recursion (6) put in the form (11). But this is not always the case. For example, if the ψ in (11) is like the right side of (10) but with a primitive recursive function π replacing $\dot{-}\, 1$ where only for some values of a does a, $\pi(a)$, $\pi(\pi(a))$, $\pi(\pi(\pi(a)))$, ... include 0, then $\phi(a)$ is defined by the recursion (and the definitions of cs, π, ψ, χ) exactly for those values of a. In this paragraph, I am assuming ψ and χ to be completely defined.

Consequently, in generalizing from primitive recursions etc. in the present direction, we shall allow our functions, such as ϕ in (11), to be *partial*; i.e. for each tuple of natural numbers as the values of the variables $\mathfrak{A}$, $\phi(\mathfrak{A})$ is either a natural number (defined) or is undefined. And our functionals, such as $\phi(\Theta; \mathfrak{A})$ in (12), are likewise *partial*, being possibly undefined for some tuples of partial functions of the right numbers of number variables as the values of Θ and of natural numbers as the values of $\mathfrak{A}$. Similarly with ψ in (11) or (12).

Partial number-theoretic functions were first explicitly introduced into recursive function theory in my [**1938**], where I partialized the functions for Herbrand-Gödel general recursiveness (Gödel [**1934**] and my [**1936**]). Thereby the theory was unburdened of the extraneous requirement that each function considered be completely defined (total); and we could have my "first recursion theorem" [**1952**, p. 348], and the "recursion theorem" of [**1938**, last two lines of **2**] and [**1952**, pp. 352–353]. The "first recursion theorem" in [**1952**] states that the minimal solution of (12) as a functional equation in ϕ when ψ is partial recursive is partial recursive. In (11) and (12), I use "$\simeq$" rather than "$=$" to express that both sides are defined with the same value or both are undefined. (I prefer to reserve "$=$" for the partial recursive predicate which is defined as usual when both sides are defined and is undefined otherwise.)

Now instead I am taking (12) as a schema of definition to generate a partial recursive functional ϕ, or with Θ empty (i.e. (11)) a partial recursive function ϕ. I use (12) (or (11)) with the understanding that $\phi(\Theta; \mathfrak{A})$ (or $\phi(\mathfrak{A})$) is defined only as the equation requires it to be in a certain manner that can be reduced to the application of certain computation rules (my [**1978**, 2.2–2.4]).

Let us return to my description of an algorithm, in the paragraph above beginning with the words "An *algorithm* is ...". There I talked about "situations" and "steps". What are we handling? Surely, the situations are some kind of finite complexes of symbols in the context of a question picked

for the algorithm. As we must have discreteness, these complexes must be built in some regular manner from finitely many occurrences of symbols from a given finite list. They may be simply finite (linear) sequences of occurrences of those symbols, or they may be other regular finite arrangements of occurrences, as in elementary school arithmetic, or in the computation trees of my [**1959**] and [**1978**] when only number variables (called there "type-0 variables") and θ_t's with type-0 variables are allowed. For any suitable geometry of a symbol space with countably many cells, rules can be given for representing the finite arrangements of symbol occurrences on cells of the space as simple finite sequences. Either using such rules first, and then coding the resulting sequences by positive integers, or directly, the symbol complexes can be coded by positive integers. In particular, if distinct positive integers have been assigned to the symbols, the finite sequence of symbol occurrences $s_0, \ldots, s_l$ can be represented by the code $p_0^{s_0} \cdot \ldots \cdot p_l^{s_l}$ (abbreviated $\langle s_0, \ldots, s_l \rangle$) where $s_0, \ldots, s_l$ are the positive integers assigned to the symbols $s_0, \ldots, s_l$ and $p_0, p_1, p_2, \ldots$ are the consecutive prime numbers 2, 3, 5, ... (as in Gödel's numbering [**1931**]).

Now let me pin down in arithmetical terms what an algorithm for computing the values of a functional $\phi(\Theta; \mathfrak{A})$ does. The Θ may be function variables or fixed functions. Now, unless $\mathfrak{A}$ is empty and Θ are fixed, we have an infinite class of questions "What is $\phi(\Theta; \mathfrak{A})$?", determined by the various choices of tuples of natural numbers as values of the variables $\mathfrak{A}$, and, if the Θ are not fixed, of tuples of functions as their values. We select a particular question of this class by naming (the values of) $(\Theta; \mathfrak{A})$. I shall gain some space for maneuver by using $(\Theta; \mathfrak{A}, 0)$ instead. The 0 conveys that we are just about to start a computation by our algorithm. And after a situation (with a complex of symbols) has been established by the first step, or reestablished by a later step, that situation will be represented by $(\Theta; \mathfrak{A}, b)$ where b is the code (> 0) for the complex of symbols then before us. As you see, I take Θ, $\mathfrak{A}$ to be still directly available after each step (although alternatively the $\mathfrak{A}$ could be represented within the b). By (c) (in the cited paragraph), the algorithm "will enable us to recognize when a situation is reached in which the answer is before us, and to read it off then". So the algorithm must provide us with a functional χ (for which we already know how to get the values) with the following features. When the symbol complex before us is coded by b (> 0), $\chi(\Theta; \mathfrak{A}, b) = 0$ if the answer is not yet before us, and $\chi(\Theta; \mathfrak{A}, b) = \phi(\Theta; \mathfrak{A}, b) + 1$ if the answer $\phi(\Theta; \mathfrak{A}, b)$ is before us; and $\chi(\Theta; \mathfrak{A}, 0) = 0$. And in the case the symbol complex before us is coded by b (> 0) and $\chi(\Theta; \mathfrak{A}, b) = 0$, by (b) the algorithm "will tell us what step to take next"; and likewise, by (a), right after picking the $(\Theta; \mathfrak{A})$ the algorithm "will ... tell us how to perform a step, the first". So there is also a functional ρ (for which we already know how to get the values) such that, when the symbol complex before us is coded by b (> 0) and $\chi(\Theta; \mathfrak{A}, b) = 0$, then $(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, b))$ represents the next situation with $\rho(\Theta; \mathfrak{A}, b)$ coding the symbol complex in it; and likewise $(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, 0))$ represents the situation after the first step. For any $(\Theta; \mathfrak{A})$, I write $\phi(\Theta; \mathfrak{A}, 0)$ as synonymous with $\phi(\Theta; \mathfrak{A})$; and when b (> 0) codes the symbol complex in the situation after any step in the application of the

algorithm begun by picking $(\Theta; \mathfrak{A})$, I write $\phi(\Theta; \mathfrak{A}, b)$ for the answer toward which (hopefully) we are being led. Thus, above, when the answer is before us, I wrote it $\phi(\Theta; \mathfrak{A}, b)$. Now everything can be brought together into an equation for a functional $\phi(\Theta; \mathfrak{A}, b)$,

$$\phi(\Theta; \mathfrak{A}, b) \simeq \begin{cases} \phi(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, b)) & \text{if } \chi(\Theta; \mathfrak{A}, b) = 0, \\ \chi(\Theta; \mathfrak{A}, b) \dot{-} 1 & \text{if } \chi(\Theta; \mathfrak{A}, b) > 0, \end{cases}$$
$$\simeq \mathrm{cs}(\chi(\Theta; \mathfrak{A}, b), \phi(\Theta; \mathfrak{A}, \rho(\Theta; \mathfrak{A}, b)), \chi(\Theta; \mathfrak{A}, b) \dot{-} 1). \quad (13)$$

This fits the first recursion theorem (12) with "$\mathfrak{A}, b$" as its "$\mathfrak{A}$". And the functional $\phi(\Theta; \mathfrak{A})$ for which we had the algorithm is

$$\phi(\Theta; \mathfrak{A}) \simeq \phi(\Theta; \mathfrak{A}, 0). \quad (14)$$

When I described an algorithm above ("An *algorithm* is . . . "), I had not yet introduced partial functionals. Now, even when $\phi(\Theta; \mathfrak{A})$ is total (i.e. completely defined), $\chi(\Theta; \mathfrak{A}, b)$ and $\rho(\Theta; \mathfrak{A}, b)$ may not be total. They only need to be defined, with a given $(\Theta; \mathfrak{A})$, for the b's for which we need them in pursuing the algorithm; specifically, for 0, $b_0 = \rho(\Theta; \mathfrak{A}, 0)$, $b_1 = \rho(\Theta; \mathfrak{A}, b_0)$, . . . up to the first b_x such that $\chi(\Theta; \mathfrak{A}, b_x) > 0$ (inclusive for χ, exclusive for ρ). When $\phi(\Theta; \mathfrak{A})$ is not total, its indefinition for a given $(\Theta; \mathfrak{A})$ may come about through indefinition of some needed value of the χ or ρ, or through there being an infinite sequence $b_0, b_1, \ldots$ with $\chi(\Theta; \mathfrak{A}, b_x) = 0$ for all x. For a partial functional $\phi(\Theta; \mathfrak{A})$, in my description of an algorithm each of (a), (b) and (c) should be qualified by adding "if the question selected has an answer".[9]

Because an algorithm can be thus analyzed as resting on the application of (13) with functionals χ and ρ for which we already know how to get the values, I am led to argue that, if we start out with the functionals that we must regard as known initially, and repeatedly use (12) with ψ composed from the functionals we have already, followed each time if necessary by further steps of composing functionals, we will get all the functionals for which there are algorithms.

What functionals should we start with? We know the natural number sequence 0, 1 $(= 0')$, . . . , a, a', . . . by its generation from 0 using the successor operation $'$. Surely then we know the following functionals:[10]

$$\phi(\Theta; \mathfrak{A}) \simeq 0. \qquad \text{S2.0}$$

$$\phi(\Theta; a, \mathfrak{B}) \simeq a. \qquad \text{S3.}$$

$$\phi(\Theta; a, \mathfrak{B}) \simeq a' \quad [= a + 1]. \qquad \text{S1.0}$$

$$\phi(\Theta; a, \mathfrak{B}) \simeq a \dot{-} 1 \quad \left[= \begin{cases} a - 1 & \text{if } a > 0, \\ 0 & \text{if } a = 0 \end{cases}\right]. \qquad \text{S1.1}$$

Thus, we know outright 0 (as a constant functional); and, given a number argument a, we can have it as an identity functional, also its immediate

[9]However by the result embodied in (15)–(17) below, the χ and ρ can be picked so that, when Θ is empty or consists of total functions only, they are total and only (c) needs to be qualified (indefinition coming about then only by having $(x)\chi(\Theta; \mathfrak{A}, b_x) = 0$).

[10]The designations "S2.0", "S3", etc. for these schemata are the ones used in my [1978], [1980].

successor a', and also $a \dot{-} 1$ which is its immediate predecessor if it is > 0. These are fundamental to finding our way around in the natural number sequence. Furthermore, we can assume we will always know whether we have before us 0 or a successor, and act accordingly. Acting accordingly can consist in then writing the respective one of two given numbers b and c. So we adopt the schema

$$\phi(\Theta;\ a,\ b,\ c,\ \mathfrak{B}) \simeq \mathrm{cs}(a,\ b,\ c) \quad \left[= \begin{cases} b & \text{if } a = 0, \\ c & \text{if } a > 0 \end{cases}\right]. \qquad \text{S5.1}$$

Let $\Theta = (\theta_1, \ldots, \theta_l)$ be variable or fixed functions of $m_1, \ldots, m_l$ variables, respectively. Then (for $t = 1, \ldots, l$) θ_t shall be used only by taking its value (i.e. the value of whatever function we have at the moment as the value of θ_t, if θ_t is not fixed) for a given m_t-tuple of arguments for which it is defined. Thus we have the schema

$$\phi(\Theta;\ \mathfrak{B},\ \mathfrak{C}) \simeq \theta_t(\mathfrak{B}) \qquad \text{S0.}$$

where $\mathfrak{B}$ consists of m_t variables. It is fundamental to this theory that a function variable (or an assumed fixed function) θ_t is used only thus. We utilize no global information about the θ_t. We only grope for one value at a time, by using S0 with a given m_t-tuple of values of $\mathfrak{B}$.[11] Turing **[1939]**, in a somewhat different context, described this as appealing to an oracle for the function θ_t, who, questioned with a tuple of arguments, reveals the corresponding value.[12]

These are what we must start with.

Furthermore, we want to be able at any stage to throw together functionals already available to us in any combinations, under the usual practice with notation for functions (composition of functionals). E.g. if we already have ϕ, ψ, χ as known functions of 2, 1, 1 variables, respectively, $\phi(\phi(\psi(b), \chi(a)), \psi(b))$ is a known function of a, b. The rule for computing it can be written in closed form (with no need for a new recursion), thus: having picked values of a and b, compute $\psi(b)$ and $\chi(a)$, call the results c and d; then compute $\phi(c, d)$, call the result e; and finally compute $\phi(e, c)$. This kind of definition is called "explicit definition". It is easily seen (in my **[1978**, 3.1]) that our class of functionals becomes closed under explicit definition when we add the following schemata in which ψ and χ are to be previously defined functionals:

$$\phi(\Theta;\ \mathfrak{A}) \simeq \psi(\Theta;\ \chi(\Theta;\ \mathfrak{A}),\ \mathfrak{A}). \qquad \text{S4.0}$$

$$\phi(\Theta;\ \mathfrak{A}) \simeq \psi(\Theta;\ \mathfrak{A}_1) \qquad \text{S6.0}$$

where $\mathfrak{A}$ comes from $\mathfrak{A}_1$ by moving one of the variables in $\mathfrak{A}_1$ to the front of the list. The second of these schemata, S6.0, offsets the limitation that some of

[11] So when we have completed the computation of a functional $\phi(\Theta;\ \mathfrak{A})$ with θ_t one of its function arguments Θ, we will have used via S0 the values of θ_t only for certain m_t-tuples of numbers as its arguments. The result will then be good also with θ_t replaced by the partial function $\bar{\theta}_t$ which coincides with θ_t for those m_t-tuples of arguments and is undefined for all other m_t-tuples; and likewise by every extension of $\bar{\theta}_t$. This constitutes a monotonicity property of our functionals.

[12] Turing dealt with (total) number-theoretic properties (predicates) of one variable, rather than with partial functionals of l function and n number variables.

the schemata above have been stated relative to a certain ordering of the variables; e.g. in S3, a is the first variable. It is also essential here that the schemata have been stated so that a functional $\phi(\Theta; \mathfrak{A})$ can be had also as $\phi(\Theta; \mathfrak{A}, \mathfrak{B})$ with additional variables $\mathfrak{B}$ (the lists of variables in our schemata are open ended). Identifications of two occurrences of a variable are effected using S4.0 together with S3 and if necessary S6.0.

Now let us start with the class of the functionals definable by repeated uses of the schemata I have now listed. Thus a functional ϕ is in this class if ϕ is ϕ_p where $\phi_1, \ldots, \phi_p$ are successive functionals, each one ϕ_i $(i = 1, \ldots, p)$ defined by a schema, either outright (as by S2.0, S3, etc.) or from one or two of the preceding functionals (as by S6.0, S4.0). Then apply the first recursion theorem (12) as a schema

$$\phi(\Theta; \mathfrak{A}) \simeq \psi(\lambda\mathfrak{A}\, \phi(\Theta; \mathfrak{A}), \Theta; \mathfrak{A}), \qquad \text{S11.}$$

with ψ in the aforesaid class. Adding this ϕ, and using explicit definitions anew, we have an in general larger class of functionals ψ available for a second application of the first recursion theorem S11; and so on.

I claim there is no way to establish an algorithm for a functional $\phi(\Theta; \mathfrak{A})$ which cannot thus be analyzed as the result of a finite number of applications of the first recursion theorem, after introducing our initial functionals, and with preceding, intervening, and final uses of explicit definition. I call the functionals $\phi(\Theta; \mathfrak{A})$ so obtainable the *partial recursive functionals*.

We saw earlier that a primitive recursion can be construed as a simple application of the first recursion theorem. Suppose we add to our list of schemata the schema of primitive recursion (for Θ empty, (2) or (10) above)

$$\begin{cases} \phi(\Theta; 0, \mathfrak{B}) \simeq \psi(\Theta; \mathfrak{B}), \\ \phi(\Theta; a', \mathfrak{B}) \simeq \chi(\Theta; a, \phi(\Theta; a, \mathfrak{B}), \mathfrak{B}). \end{cases} \qquad \text{S5.}$$

If we then omit the first recursion theorem as a schema S11 (or equivalently, without adding S5 use S11 only to effect primitive recursions), we generate a smaller class of functionals (all totally defined if the functions Θ are), which I call the *primitive recursive functionals*.[13]

How often may we need to use the first recursion theorem S11 (our most powerful method of definition) other than, if we do not have S5 as a separate schema, in defining primitive recursive functionals? With a given list Θ of function variables (as fixed by l and $m_1, \ldots, m_l$), only once![14] There is a fixed primitive recursive functional ψ^* such that, applying the first recursion

---

[13]For Θ consisting of total functions only, this is in my [**1978**, p. 213].

[14]The question was asked by Andrei Ershov at the 1979 Urgench Symposium, and answered by me there (cf. [**1981b**, §5]) but without mentioning that (when S5 is separately available) we do not need different applications of S11 for the various lists $\mathfrak{A}$ of number variables, as comes out of my [**1978**, 3.2]. Indeed, with a little more work, we can get by with just one use of the recursion theorem (when S5 is separately available) for all lists Θ (as well as for all lists $\mathfrak{A}$). To do this, we represent lists Θ of l functions of $m_1, \ldots, m_l$ variables by single one-place functions $\langle\Theta\rangle^*$, just as in (16) we represent lists $a_1, \ldots, a_n$ of n numbers by single numbers $\langle a_1, \ldots, a_n\rangle$, where $a_i = (\langle a_1, \ldots, a_n\rangle)_i$ for $i = 1, \ldots, n$, using my [**1952**, p. 230, #19] (and $(0)_i = 0$ for all i). If z is an index of a functional $\phi(\Theta; \mathfrak{A})$ where Θ is l functions of $m_1, \ldots, m_l$ variables, the l and $m_1, \ldots, m_l$ are given by z. Now, e.g. with $l = 3$, $m_1 = 2$, $m_2 = 3$, $m_3 = 1$, we can take

theorem to define

$$\phi^*(\Theta; z, a, b) \simeq \psi^*(\lambda zab\ \phi^*(\Theta; z, a, b), \Theta; z, a, b) \tag{15}$$

and putting for each n (where $\langle a_1, \ldots, a_n\rangle = p_0^{a_1} \cdot \ldots \cdot p_{n-1}^{a_n}$)

$$\{z\}^{\Theta}(a_1, \ldots, a_n) \simeq \phi^*(\Theta; z, \langle a_1, \ldots, a_n\rangle, 0), \tag{16}$$

each partial recursive functional $\phi(\Theta; a_1, \ldots, a_n)$ is given for some fixed z as

$$\phi(\Theta; a_1, \ldots, a_n) \simeq \{z\}^{\Theta}(a_1, \ldots, a_n). \tag{17}$$

The method of the proof is this. A partial recursive functional $\phi(\Theta; a_1, \ldots, a_n)$ is defined by a sequence of schema applications, whereby $\phi_1, \ldots, \phi_p$ with $\phi = \phi_p$ are defined successively. A system of indexing can be established so that an *index* z of ϕ represents this sequence of schema applications (details in my [**1978**, 1.3]), and thus encapsules the definition of ϕ. Now, if we take z as a variable, we can devise an algorithm which is an algorithm for all algorithms (with the list Θ): a universal algorithm.[15] In some detail: Having an index z, we can give numbers e (called *Gödel numbers* in [**1978**, 3.2]) to expressions (0-*expressions* in [**1978**, 2.2]) such as arise in computing the functional $\phi(\Theta; a_1, \ldots, a_n)$ with the index z for given values of $\Theta, a_1, \ldots, a_n$. We can further build the Gödel numbers e into *codes* $b > 0$ for the complexes of symbols in computational situations (*computations not necessarily completed*, in the terminology of my [**1980**, 7.2]). Now there are primitive recursive functionals $\chi^*(\Theta; z, a, b)$ and $\rho^*(\Theta; z, a, b)$ with the following properties. Take any choice of $(\Theta; z, a)$, i.e. of values of the variables Θ, z, a, to select a question for our universal algorithm, Θ being l functions of $m_1, \ldots, m_l$ variables respectively. If z is an index of a partial recursive functional ϕ of l function variables with $m_1, \ldots, m_l$ number variables respectively and n number variables (if so, n is given by z), then $\chi^*(\Theta; z, a, 0) = 0$ and $\rho^*(\Theta; z, a, 0)$ is the code of the computation just begun with $\phi_p(\beta_0^0, \ldots, \beta_{n-1}^0)$ as the initial 0-expression.[16] Thus we take $\beta_0^0, \ldots, \beta_{n-1}^0$ as the formal number variables for the functional $\phi = \phi_p$ in whose computation for $a_1, \ldots, a_n$ as their values (and for Θ as chosen) we are interested. For $a > 0$, we determine $a_1, \ldots, a_n$ from the chosen a by writing $a = \langle a_1, \ldots, a_k\rangle$ with a $k \geqslant n$, if necessary by taking $a_i = 0$ for all sufficiently big $i \leqslant n = k$; for $a = 0$, we take $a_1 = \ldots = a_n = 0$. In the said computation, we shall need no other number variables than $\beta_0^0, \ldots, \beta_{n-1}^0$, and we shall assign them $a_1, \ldots, a_n$ as their respective values throughout. If z is an index of a partial recursive functional ϕ of l function variables with

$\langle\Theta\rangle^*(a) \simeq \theta_1((a)_1, (a)_2)$ if $(a)_0 = 0$, $\theta_2((a)_1, (a)_2, (a)_3)$ if $(a)_0 = 1$, $\theta_3((a)_1)$ if $(a)_0 > 2$. This spreads the values, and instances of indefinition, of $\theta_1, \theta_2, \theta_3$ on disjoint subsets of the domain of $\langle\Theta\rangle^*$, unlike the method used (with total one-place functions) in [**1959**, 2.1]. Now $\theta_1 = \lambda ab\ \langle\Theta\rangle^*(\langle 0, a, b\rangle)$, $\theta_2 = \lambda abc\ \langle\Theta\rangle^*(\langle 1, a, b, c\rangle)$, $\theta_3 = \lambda a\ \langle\Theta\rangle^*(\langle 2, a\rangle)$. In (15) "$\Theta$" becomes simply "$\theta$" with θ a one-place function, the right side of (16) becomes $\phi^*(\langle\Theta\rangle^*; z, \langle a_1, \ldots, a_n\rangle, 0)$, and (17) is unchanged.

[15]Universal algorithms (in other representations) first appeared in my [**1936**] and Turing [**1936–7**].

[16]Since the list Θ_i for each of $\phi_1, \ldots, \phi_p$ (where $\phi = \phi_p$) is determined by the schema applications, I omit "Θ_i" as argument of ϕ_i in writing the 0-expressions ([**1978**, 2.2]).

$m_1, \ldots, m_l$ number variables and n number variables, and $b > 0$ is the code of a computation begun with $\phi_p(\beta_0^0, \ldots, \beta_{n-1}^0)$ under the assignment $\Theta, a_1, \ldots, a_n$, then, if that computation is uncompleted, $\chi^*(\Theta; z, a, b) = 0$ and $\rho^*(\Theta; z, a, b)$ is the code of the computation immediately extending that one (possibly undefined, if the Θ are not all total); and, if that computation is completed, $\chi^*(\Theta; z, a, b) = 1 +$ (the value computed) and $\rho^*(\Theta; z, a, b) = 0$. Otherwise, $\chi^*(\Theta; z, a, b) = \rho^*(\Theta; z, a, b) = 0$. We thus obtain a primitive recursive functional ψ^* for (15), having the form on the right of (13) with our primitive recursive χ^* and ρ^* and with "z, a, b" as its "$\mathfrak{A}, b$", so that ϕ^* has the property stated in (16)–(17).

An immediate consequence of the characterization by our schemata of all the possible algorithms for computing functionals $\phi(\Theta; \mathfrak{A})$ with given lists of function variables Θ and number variables $\mathfrak{A}$ is that there are such functionals, and, with Θ empty, functions, for which there is no algorithm: uncomputable functionals and functions. For, the characterization makes the class of the algorithms as normalized in our way countable, while, even with Θ empty, the class of the functionals $\phi(\Theta; \mathfrak{A})$ is uncountable (assuming $\mathfrak{A}$ not empty).

How simply can we define an uncomputable function? Take the case Θ is empty and $\mathfrak{A}$ is one variable a, so $\{z\}^{\Theta}(a_1, \ldots, a_n)$ specializes to $\{z\}(a)$. Using (15)–(16) with $n = 1$, $\{z\}(a)$ is defined exactly if the sequence $\chi^*(z, \langle a\rangle, \beta^*(0, z, a))$, $\chi^*(z, \langle a\rangle, \beta^*(1, z, a))$, $\chi^*(z, \langle a\rangle, \beta^*(2, z, a)), \ldots$, where $\beta^*(0, z, a) = \rho^*(z, \langle a\rangle, 0)$ and $\beta^*(x', z, a) = \rho^*(z, \langle a\rangle, \beta^*(x, z, a))$, has a member > 0. Thus $\{z\}(a)$-is-defined is expressible as $(Ex)T(z, a, x)$ where $T(z, a, x)$ is the primitive recursive predicate $\chi^*(z, \langle a\rangle, \beta^*(x, z, a)) > 0$ and "(Ex)" expresses "there exists an x such that". Now if we define

$$\zeta(a) = \begin{cases} \{a\}(a) + 1 & \text{if } (Ex)T(a, a, x) \text{ (Case 1),} \\ 0 & \text{otherwise (Case 2),} \end{cases} \tag{18}$$

ζ is an uncomputable total function. For, if ζ were computable, then, for some number z, $\zeta(a) = \{z\}(a)$ for all a; hence $\{z\}(z)$ is defined, i.e. $(Ex)T(z, z, x)$; so $\{z\}(z) = \zeta(z) = \{z\}(z) + 1$, a contradiction!

This brings us to the point which in other treatments led (long ago) to some fundamental results on the foundations of mathematics. I shall quickly recapitulate the arguments (much as in my **[1957]**, **[1958]**, **[1964]**, **[1967]**, **[1969]**).

How does the definition (18) of $\zeta(a)$ fall short of providing an algorithm for it? We would clearly have an algorithm for ζ if only we had one for deciding which of the two cases applies. Thus there can be no algorithm for the predicate $(Ex)T(a, a, x)$! This is a version of Church's theorem **[1936]**. We have an undecidable predicate, obtained simply by prefixing an existential quantifier (Ex) to the decidable predicate $T(a, a, x)$.

It is easy to proceed from this result to two celebrated results concerning formal systems.

I consider how certain propositions, depending on a parameter a, are expressible by formulas in a suitable formal system. The system can be the usual formal system of elementary number theory or various other systems. In a suitable formal system, the propositions $(Ex)T(a, a, x)$ for $a = 0, 1, 2, \ldots$ will be expressed by respective formulas A_a (obtained from a by

an algorithm) such that A_a is provable if and only if $(Ex)T(a, a, x)$ (is true). The "only if" is a consistency property (often reducible to what Gödel called "ω-consistency"). The "if" comes about because a formal proof of A_a should be obtainable, corresponding to the informal proof of $(Ex)T(a, a, x)$ which consists in verifying by the decision procedure for T that $T(a, a, x)$ for the given a and a suitable x.

This being the case for a formal system, the system is undecidable; i.e. there is no decision procedure (Entscheidungsverfahren) for the provability of any formula in the system. For, if there were, by applying it to the formulas A_a (obtainable from a by an algorithm) we would have an algorithm for $(Ex)T(a, a, x)$, which we just saw cannot exist. This reasoning can be used to establish the undecidability of the usual formal system of elementary number theory, and also of the pure first-order predicate calculus (a famous result of Church [**1936a**] and Turing [**1936–7**]).

Consider the negations of the propositions $(Ex)T(a, a, x)$, i.e. $\overline{(Ex)}T(a, a, x)$ or equivalently $(x)\overline{T}(a, a, x)$. In any of the aforesaid systems in which the propositions $(Ex)T(a, a, x)$ are expressed by closed formulas A_a (not the predicate calculus, where A_a have free predicate variables) and which have the symbol $\neg$ for negation, the propositions $(x)\overline{T}(a, a, x)$ are expressed by the formulas $\neg A_a$. Suppose that (as a consistency property) $\neg A_a$ is provable only if $(x)\overline{T}(a, a, x)$. I shall argue that it is not the case that, for every a, $\neg A_a$ is provable if $(x)\overline{T}(a, a, x)$. For, as remarked above, the accomplishment of a formal system is to provide an algorithm for the notion of proof. Applying this to the formulas $\neg A_a$, and using the algorithm for getting these from a, the predicate $\neg A_a$-is-provable is expressible in the form $(Ex)R(a, x)$ where $R(a, x)$ is the decidable predicate that x is the code (say a Gödel number in the manner of Gödel [**1931**]) of a proof of $\neg A_a$. Now if, for every a, $\neg A_a$ is provable if (as well as only if) $(x)\overline{T}(a, a, x)$, we would have that, for all a, $(x)\overline{T}(a, a, x) \equiv (Ex)R(a, x)$. But $(x)\overline{T}(a, a, x)$ is not expressible in the form $(Ex)R(a, x)$ with R decidable. For if it were, we could decide whether or not $(Ex)T(a, a, x)$ by looking for the least x (which must exist by the law of the excluded middle) such that either $T(a, a, x)$ or $R(a, x)$, and answering "yes" or "no" according to whether, for that x, $T(a, a, x)$ or $R(a, x)$. This is a generalized version (given in my [**1943**], and less simply in my [**1936**]) of the famous theorem of Gödel [**1931**] on the existence of formally undecidable sentences in Principia Mathematica and related systems (the first of the two incompleteness theorems of [**1931**]): for some number p, the formula $\neg A_p$ is unprovable, and also true (i.e. $(x)\overline{T}(p, p, x)$) so that A_p is also unprovable.[17] The theorem in this version is generalized from Gödel [**1931**], since it applies outright to all formal systems which simply are adequate for expressing some propositions of elementary number theory

---

[17]What I have just established is that $\overline{(a)}[(x)\overline{T}(a, a, x) \rightarrow (\neg A_a \text{ is provable})]$. This makes it absurd that such a p should not exist. To obtain such a p directly: With R as above, the-least-x-such-that-$R(a, x)$ is a partial recursive function of a with index p, and $(Ex)T(p, a, x) \equiv (\neg A_a \text{ is provable})$. It follows easily that $(x)\overline{T}(p, p, x)$ and that $\neg A_p$ and A_p are unprovable (cf. my [**1967**, p. 251, SECOND PROOF], or [**1957**] or [**1958**]).

(correctly, i.e. with the requisite consistency properties) and satisfy the structural requirement that there is an algorithm for being a proof. Here there is no possibility of escaping the incompleteness (as by seeking to devise a formal system quite remote in its details from Principia Mathematica), since the features of formal systems just named are objectives of ours in formalizing theories including elementary number theory. In this version, the formally undecidable sentences A_p (with different numbers p for different formal systems) express values of the preassigned number-theoretic predicate $(Ex)T(a, a, x)$. As I expressed it in [**1943**], there is neither a complete algorithmic theory for this predicate (i.e. $(Ex)T(a, a, x)$ is undecidable), nor (by the generalized Gödel theorem) a complete formal deductive theory.

These were among the results that ushered in the second half century (nearly) of recursive function theory.

Much has happened since then. Many of the newer developments have been associated with the use of function variables, as illustrated above by the Θ's.

To begin with something not so new, take the idea of a total recursive functional $\phi(\Theta; \mathfrak{A})$ and specialize to the case of one assumed function θ and that one-placed and total and one number variable a. By $\phi(\theta; a)$ being recursive we define when one one-place number-theoretic function $\phi = \lambda a\, \phi(\theta; a)$ is *recursive* in another $\theta = \lambda a\, \theta(a)$. Here ϕ and θ can be the functions (taking only 0 and 1 as values) which represent two predicates $P(a) \equiv \phi(a) = 0$ and $Q(a) \equiv \theta(a) = 0$. The recursiveness of P in Q, formulated differently by Turing in [**1939**], gives a reducibility notion that was used by Post [**1944**], [**1948**] in studying the number-theoretic predicates. Slightly earlier, in my [**1943**] I described a hierarchy of number-theoretic predicates in which two positions are occupied by the decidable predicates $R(a)$ and the predicates expressible in the form $(Ex)R(a, x)$ with R decidable, which we encountered a moment ago. This hierarchy can be extended way upward. One of the ways of extending it beyond arithmetic is to use variables of successive higher finite types 1, 2, 3, . . . consisting of the total one-place functions over the preceding type to the natural numbers (where type 0 is the natural numbers). I took the step to include type 1 (of which a version is given above) in [**1950**] with [**1955**], [**1955a**], [**1955b**]; and to all finite types in [**1959**] and [**1963**]. What I have drawn upon in today's lecture is the use of the first recursion theorem as one of a list of schemata in my revisitation of the resulting generalized recursion theory in [**1978**] and [**1980**]. Here I have focussed on what comes out of that when we specialize to using only number variables and number-theoretic function variables. This concludes my discussion of elementary recursive function theory from a higher standpoint.

References

ACKERMANN, WILHELM

[1928] *Zum Hilbertschen Aufbau der reellen Zahlen*, Math. Ann. **99**, 118–133.

CANTOR, GEORG

[1874] *Über eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen*, J. Reine Angew. Math. **77**, 258–262.

[1895–7] *Beiträge zur Begründung der transfiniten Mengenlehre*, Math. Ann. **46** (1895), 481–512; **49** (1897), 207–246.

CHURCH, ALONZO

[1932] *A set of postulates for the foundation of logic*, Ann. of Math. (2) **33**, 346–366.

[1933] *A set of postulates for the foundation of logic* (*second paper*), Ann. of Math. (2) **34**, 839–864.

[1936] *An unsolvable problem of elementary number theory*, Amer. J. Math. **58**, 345–363.

[1936a] *A note on the Entscheidungsproblem*, J. Symbolic Logic **1**, 40–41; *Correction*, 101–102.

DEDEKIND, RICHARD

[1888] *Was sind und was sollen die Zahlen*? Braunschweig.

GÖDEL, KURT

[1931] *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*. I, Monatsh. Math. Phys. **38**, 173–198.

[1934] *On undecidable propositions of formal mathematical systems*, mimeographed notes by S. C. Kleene and J. B. Rosser on lectures at the Institute for Advanced Study, 1934; reprinted in Martin Davis, *The undecidable, basic papers on undecidable propositions, unsolvable problems and computable functions*, Raven Press, Hewlett, N. Y., 1965, pp. 39–74.

HILBERT, DAVID

[1918] *Axiomatisches Denken*, Math. Ann. **78**, 405–415.

[1926] *Über das Unendliche*, Math. Ann. **95**, 161–190.

KLEENE, STEPHEN, C.

[1935] *A theory of positive integers in formal logic*, Amer. J. Math. **57**, 153–173; 219–244.

[1936] *General recursive functions of natural numbers*, Math. Ann. **112**, 727–742.

[1936a] *λ-definability and recursiveness*, Duke Math. J. **2**, 340–353.

[1938] *On notation for ordinal numbers*, J. Symbolic Logic **3**, 150–155.

[1943] *Recursive predicates and quantifiers*, Trans. Amer. Math. Soc. **53**, 41–73.

[1950] *Recursive functions and intuitionistic mathematics*, Proc. Internat. Congr. Math. (Cambridge, Mass., U.S.A., 1950), Amer Math. Soc., Providence, R. I., 1952, **I**, pp. 679–685.

[1952] *Introduction to metamathematics*, North-Holland, Amsterdam, P. Noordhoff, Groningen and D. Van Nostrand, Toronto and New York.

[1955] *Arithmetical predicates and function quantifiers*, Trans. Amer. Math. Soc. **79**, 312–340.

[1955a] *On the forms of the predicates in the theory of constructive ordinals* (*second paper*), Amer. J. Math. **77**, 405–428.

[1955b] *Hierarchies of number-theoretic predicates*, Bull. Amer. Math. Soc. **61**, 193–213.

[1957] *Mathematics, foundations of*, Encyclopaedia Britannica, 1957 and subsequent printings. Beginning with 1974, the article was reworked to replace Kleene's section on intuitionism by a fuller treatment by Solomon Fefermann.

[1958] *Mathematical logic: constructive and non-constructive operations*, Proc. Internat. Congr. Math. (Edinburgh, 1958) (J. A. Todd, ed.), Cambridge at the University Press, 1960, pp. 137–153.

[1959] *Recursive functionals and quantifiers of finite types*. I, Trans. Amer. Math. Soc. **91**, 1–52.

[1963] *Recursive functionals and quantifiers of finite types*. II, Trans. Amer. Math. Soc. **108**, 106–142.

[1964] *Computability*, The Voice of America Forum Lectures, Phil. of Science Series, no. 6; reprinted, Philosophy of Science Today (Sidney Morgenbesser, ed.), Basic Books, New York and London, 1967, pp. 36–45.

[1967] *Mathematical logic*, Wiley, New York, London and Sidney.

[1969] *The new logic*, Sigma-Xi-RESA National Lecture, Southeast Tour (Spring 1969), Amer. Sci. **57**, 333–347.

[1978] *Recursive functionals and quantifiers of finite types revisited*. I. Generalized Recursion Theory. II, Proc. 1977 Oslo Sympos. (J. E. Fenstad, R. O. Gandy and G. E.

Sacks, eds.), North-Holland, Amsterdam, New York and Oxford, pp. 185–222.

[1980] *Recursive functionals and quantifiers of finite types revisited.* II, The Kleene Symposium (Madison, Wis., June 1978) (J. Barwise, H. J. Keisler and K. Kunen, eds.), North-Holland, Amsterdam, New York and Oxford, pp. 1–29.

[1981a] *Origins of recursive function theory*, 20th Annual Sympos. Foundations of Computer Science (San Juan, Puerto Rico, 1979), Institute of Electrical and Electronics Engineers (IEEE 79CH 1471-2C), pp. 371–382; Annals of the History of Computing, 1981 (to appear).

[1981b] *Algorithms in various contexts*, Proc. Sympos. Algorithms in Modern Mathematics and Computer Science (dedicated to Al-Khowarizmi) (Urgench, Khorezm Region, Uzbek, SSSR, 1979), Springer-Verlag, Berlin, Heidelberg and New York, 1981 (to appear).

KLEIN, FELIX

[1908–9] *Elementarmathematik vom höheren Standpunkte aus*, Lithographed, Leipzig; English transl. from 3rd German ed. (E. R. Hedrick and C. A. Noble, eds.), Macmillan, New York, 1932.

KRONECKER, LEOPOLD

[1886] The saying quoted in the third paragraph of the text was spoken in his lecture on 21 September 1886 before der Deutschen Naturforscherversammlung zu Berlin, according to H. Weber, *Leopold Kronecker*, Math. Ann. **43** (1893), 1–25 (cf. p. 15). Also cf. Leopold Kronecker's *Werke*, hrg. K. Henzel, vol. 3, pt. 2, p. 203.

LÖWENHEIM, LEOPOLD

[1915] *Über Möglichkeiten im Relativkalkül*, Math. Ann. **76**, 447–470.

MARKOV, A. A.

[1951] *Theory of algorithms*, Amer. Math. Soc. Transl. (2) **15** (1960), 1–14.

[1954] *Theory of algorithms*, The National Science Foundation, Washington, D.C., the Department of Commerce, U.S.A., and the Israel Program for Scientific Translation, Jerusalem, 1961.

PEANO, GUISEPPE

[1889] *Arithmetices principia, nova methodo exposita*, Turin, Bocca.

[1891] *Sul concetto di numero*, Rivista di Matematica **1**, 87–102; 256–267.

PÉTER, RÓZSA

[1934] *Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion*, Math. Ann. **110**, 612–632.

[1935] *Konstruktion nichtrekursiver Funktionen*, Math. Ann. **111**, 42–60.

[1936] *Über die mehrfache Rekursion*, Math. Ann. **113**, 489–527.

POST, EMIL L.

[1936] *Finite combinatory processes–formulation* 1, J. Symbolic Logic **1**, 103–105.

[1943] *Formal reductions of the general combinatorial decision problem*, Amer. J. Math. **65**, 197–215.

[1944] *Recursively enumerable sets of positive integers and their decision problems*, Bull. Amer. Math. Soc. **50**, 284–316.

[1948] *Degrees of recursive unsolvability*, abstract (preliminary report) in Bull. Amer. Math. Soc. **54**, 641–642.

SCHRÖDER, ERNST

[1895] *Vorlesungen über die Algebra der Logik (exakte Logik)*, **3** *Algebra und Logik der Relative*, part 1, Leipzig.

SKOLEM, THORALF

[1923] *Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbare Veränderlichen mit unendlichem Ausdehnungsbereich*, Skrifter utgit av Videnskapsselskapet i Kristiania, I. Matematisk-Naturvidenskabelig Klasse 1923, no. 6; English transl., Jean van Heijenoort, *From Frege to Gödel, a source book in mathematical logic*, 1879–1931, Harvard Univ. Press, Cambridge, Mass., 1967, pp. 302–333.

SMULLYAN, RAYMOND M.

[1961] *Theory of formal systems*, Annals of Math. Studies, no. 47, Princeton Univ. Press, Princeton, N. J.

TURING, ALAN MATHISON

[1936–7] *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. (2) **42**, 230–265; *A Correction*, **43** (1937), 544–546.

[1939] *Systems of logic based on ordinals*, Proc. London Math. Soc. (2) **45**, 161–228.

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF WISCONSIN, MADISON, WISCONSIN 53706