Computability Theory

by

Daniele Mundici and Wilfried Sieg

June 1994

Report CMU-PHIL-54

Philosophy
Methodology
Logic

Pittsburgh, Pennsylvania 15213-3890

COMPUTABILITY THEORY

The ejfective calculability of number theoretic functions like addition and multiplication has always been recognized, and for that judgment a rigorous notion of 'computable function' is not needed. A sharp mathematical concept was defined only in the twentieth century, when the decision problem for predicate logic and other issues required a precise delimitation of functions that can be viewed as effectively calculable. Predicate logic emerged from Frege's fundamental 'Begriffsschrift' (1879) as an expressive formal language and was described with mathematical precision by Hilbert in lectures given during the winter term 1917-18[1]*. The logical calculus Frege had also developed allowed proofs to proceed as computations in accord with a fixed set of rules; in principle, the rules could be applied according to Gödel 'by someone who knew nothing about mathematics, or by a machine'.*

Hilbert grasped the potential of this mechanical aspect and formulated the decision problem for predicate logic as follows: 'The Entscheidungsproblem is solved if one knows a procedure that permits the decision concerning the validity, respectively, satisfiability of a given logical expression by a finite number of operations.' Some, e.g., von Neumann in 1927, believed that the inherent freedom of mathematical thought provided a sufficient reason to expect a negative solution to the problem. But how could a proof of undecidability be given? The unsolvability results of other mathematical problems had always been established relative to a determinate class of admissible operations, e.g., the impossibility of doubling the cube relative to ruler and compass constructions. A negative solution to the decision problem obviously required the characterization of 'effectively calculable functions'; cf. ***Church*s theorem and the decision problem.***

For two other important issues a characterization of that informal notion was needed, namely, the general formulation of the incompleteness theorems (cf. ***GödeVs Theorems)*** *and the effective unsolvability of mathematical problems (e.g., of Hubert's Tenth Problem). The first task of computability theory was thus to answer the question: What is a precise notion of effectively calculable function? Many different answers to this question invariably*

---

[1] **Notes of these lectures were written by Beraays and formed the basis for Hilbert and Ackermann's book 'Grundzüge der theoretischen Logik' (1928).**

characterized the same class of number theoretic functions: the partial recursive ones. Today recursiveness or, equivalently, Turing computability are considered as the precise mathematical counterparts to 'effective calculability'; cf. ***Church's Thesis****. Relative to these notions undecidability results have been established, in particular, the undecidability of the decision problem for predicate logic. The notions are idealized in the sense that no time or space limitations are imposed on the calculations; the concept of 'feasibility' is crucial in computer science when trying to capture the subclass of recursive functions whose values can actually be determined (cf.* ***Complexity theory*** *and* ***Computability and information****).*

1 Primitive recursive functions

A precise definition of the class of primitive recursive functions (from n-tuples of natural numbers to natural numbers) was given by Gödel in 1931; there it is still called the class of recursive functions. The main definitional schema of *primitive recursion* used in the generation of the elements of this class was well-known in mathematics. Dedekind (1888) had presented it most clearly and had given a set-theoretic reconstruction that turned out to be of importance also in computability theory. Most functions in elementary number theory are primitive recursive; that was established by Skolem in 1923. The foundational significance of this function class was emphasized by Hilbert and Bernays: the values of the functions (for any argument) can be determined in finitely many steps, proceeding purely "mechanically". That point was also expressed by saying that the functions are effectively calculable.

The class $\mathfrak{PR}$ of primitive recursive functions is specified inductively and contains as its initial functions the zero-function Z, the successor function S, and the projection functions P_i^n for each n and each i with $1 \leq i \leq n$. These functions satisfy the equations $Z(x)=0$,

$S(x)=x'$ and $P_i^1(x_1,\ldots,x_n)=x_i$, for all $x,x_1,\ldots,x_n$; x' is the successor of x. The class is, first of all, closed under the *schema of composition:* Given an m-place function ψ in PR and n-place functions $\varphi_1,\ldots,\varphi_m$ in PR, the function ϕ defined by

$$\phi(x_1,\ldots,x_n) = \psi(\varphi_1(x_1,\ldots,x_n), \ldots, \varphi_m(x_1,\ldots,x_n))$$

is also in PR; ϕ is then said to be obtained by composition from ψ and $\varphi_1,\ldots,\varphi_m$. PR is also closed under the *schema of primitive recursion:* Given an n-place function ψ in PR, and an n+2-place function χ in PR, the function ϕ defined by

$$\phi(x_1,\ldots,x_n,0) = \psi(x_1,\ldots,x_n)$$
$$\phi(x_1,\ldots,x_n,y') = \chi(x_1,\ldots,x_n,y,\phi(x_1,\ldots,x_n,y))$$

is a function in PR; ϕ is said to be obtained by primitive recursion from ψ and χ. Thus, a function is primitive recursive if and only if it can be obtained from the initial functions by finitely many applications of the composition and recursion schemas.

The primitive recursive functions do not exhaust the class of effectively calculable functions; that was shown by Ackermann, who constructed a function that is obviously effectively calculable, but not primitive recursive. The *Ackermann-function* can be defined by the following recursion equations:

$$\varphi_0(x,y) = S(y)$$
$$\varphi_n(x,0) = \begin{cases} x & \text{if } n=0 \\ 0 & \text{if } n=1 \\ 1 & \text{otherwise} \end{cases}$$
$$\varphi_{n'}(x,y') = \varphi_n(x,\varphi_{n'}(x,y))$$

Notice that φ_1 is addition, φ_2 is multiplication, φ_3 is exponentiation, etc; i.e., the next function is always obtained by iterating the previous one. For each n, the function $\varphi_n(x,x)$ is primitive recursive, but $\varphi(x,x,x)$ is not! We can construct another example of an effectively calculable, but not primitive recursive function ψ by effectively listing all primitive recursive unary functions $\varphi_0, \varphi_1, \varphi_2, \ldots$ and then setting $\psi(x) = \varphi_x(x)+1$. Since ψ differs from every φ_i, ψ cannot be primitive recursive; but the effectiveness of our listing of the functions φ_i ensures that ψ can be effectively calculated.

For working with this class of functions it is indispensable to establish further closure properties. To allow, for example, directly general *explicit definitions* (without the cumbersome detour of projections) of functions like $f(x,y,z) = ((xy)+z)^2 + 789$, one establishes that variables can be permuted, substitutions can be carried out, and that the constant functions $con^n_m(x_1,...,x_n)=m$ are primitive recursive. Beyond addition, multiplication, and exponentiation it is useful to know that the following functions are also in $\mathfrak{P}\mathfrak{R}$: x! (factorial), pred(x) (predecessor), $x \dot{-} y$ (arithmetic subtraction), $\overline{sg}(x)$ $(=1 \dot{-} x)$, $sg(x)$ $(=1 \dot{-} \overline{sg}(x))$. Then it is very easy to define the bounded summation of an n+1-function ϕ in $\mathfrak{P}\mathfrak{R}$ as follows:

$$\psi(x_1,...,x_n,0) = 0$$
$$\psi(x_1,...,x_n,y') = \psi(x_1,...,x_n,y) + \phi(x_1,...,x_n,y)$$

ψ is usually denoted by $\sum_{x<y} \phi(x_1,...,x_n,x)$; bounded productation is defined similarly.

The functions we listed are useful for the investigation of primitive recursive relations; an n-place relation R between natural numbers is called *primitive recursive* if and only if its characteristic function[2] χ_R is in $\mathfrak{P}\mathfrak{R}$. It is now direct to show that the primitive recursive relations are closed under Boolean operations (particularly, $\neg$--not, &--and, v--or, ->--if then) and bounded quantification (i.e., $(\forall x<y)$--for all x less than y and $(\exists x<y)$--for some x less than y). Finally, we discuss bounded minimization: the function $\mu x<y.R(x_1,...,x_n,x)$ yields the smallest x less than y, if for some x less than y $R(x_1,...,x_n,x)$, otherwise the function yields y. Define first

$$\phi_1(x_1,...,x_n,x) = \begin{cases} 0 & \text{if } (\exists z \leq x)\ R(x_1,...,x_n,z) \\ 1 & \text{otherwise} \end{cases}$$

and notice that $\mu x<y.R(x_1,...,x_n,x) = \sum_{x<y} \phi_1(x_1,...,x_n,x)$. Thus, if the relation R is primitive recursive, bounded minimization leads to a function in $\mathfrak{P}\mathfrak{R}$.

Given this list of functions in $\mathfrak{P}\mathfrak{R}$ and the closure conditions for primitive recursive relations, we have a convenient framework in which we can show the primitive recursiveness of number theoretic

---

[2] For recursion-theoretic purposes it is convenient to consider 0 to be the truth-value *true* and 1 the truth-value *false*.

relations; e.g., the characteristic function %< of the less-than-relation < is given by %<(x,y)=$\overline{sg}$(y-x); equality, being a divisor of, being prime are all defined easily. This is to indicate that the class is mathematically rather rich.

2 Finite computations

Gödel used in 1931 and in his Princeton Lectures of 1934 primitive recursive functions and relations to describe the syntax of particular 'formal[1] theories — after Gödel-numbering the syntactic configurations that make up the theory. Since he strove to arrive at a general concept of formality through the underlying concept of calculability for functions, there was no reason to focus attention on theories whose syntax could be presented primitive recursively. He viewed primitive recursive definability of formulas and proofs as a precise condition which *in practice* sufficed to describe formal systems, but he was searching for a condition that would suffice *in principle.*

In his Princeton Lectures, Gödel considered it as a very 'important property' that the value of any primitive recursive function, for arbitrary arguments, can be calculated by a 'finite computation'. He added in a footnote to this remark:

The converse seems to be true if, besides recursions according to the scheme (2) [of primitive recursion], recursions of other forms ... are admitted. This cannot be proved, since the notion of finite computation is not defined, but it can serve as a heuristic principle.

In the last part of his lectures Gödel used quite general forms of recursions, when introducing 'general recursive functions'. These functions are obtained as unique solutions of a system E of equations, and their values must be computable in an equational calculus with just two (obviously mechanical) rules[3]: the first rule allows the substitution of numerals for variables in any equation derived from E; the second rule allows the replacement of terms $t(v_i,\ldots,v_n)$ in a derived equation by |x, in case $t(v_i,.,.,v_n)=|i$ is a derived equation. (Lower case Greek letters stand for numerals.)

This class of general recursive functions very quickly turned out to be characterizable in a variety of ways: Church and Kleene

---

3 The relation of GSdel's proposal to suggestions of Herbrand is discussed in (Sieg 1994).

showed the equivalence to λ-definability, Kleene to μ-recursiveness (to be discussed below), Turing to computability by his machines; cf. **Lambda calculus, Combinatory logic, and Turing machines.** The class introduced by Gödel was used by Church in his first formulation of 'Church's Thesis'. The early, pre-Turing attempts to argue for the thesis are captured in a mathematically concise way through the concept of a function reckonable according to rules ("regelrecht auswertbare Funktion"). These functions were introduced by Hilbert and Bernays in the second volume of their *Grundlagen der Mathematik*; they were characterized as being calculable in deductive formalisms satisfying general recursiveness conditions; the critical condition required the proof predicate of the formalisms to be primitive recursive. It was shown that the calculations could be carried out in a particular subsystem of arithmetic; Gödel took this fact in 1946 as the basis for his claim that computability is an "absolute" concept. The analysis of Hilbert and Bernays revealed also clearly the 'stumbling block' all these analyses encountered: they tried to characterize the elementary nature of steps in calculations, but could not do so without recurring to recursiveness (Church), primitive recursiveness (Hilbert and Bernays), or to very specific rules (Gödel).

Only Turing was able, in his ground-breaking 1936 paper *On computable numbers*, to circumvent the stumbling block by focusing on the calculations of a human computor proceeding mechanically. He formulated general boundedness and locality conditions for such a computor and showed that any number theoretic function, whose values can be calculated by a computor satisfying these conditions, can actually be computed by a Turing machine; cf. (Sieg 1994). The latter fact is sometimes called Turing's Theorem. The restricted formulation of Turing machines allows a uniform and simple description of computations; its adequacy for linear computations is guaranteed by Turing's Theorem. The starting point of Turing's analysis was, however, the mechanical behavior of a human computor operating on finite configurations in the plane. This behavior can be described generally and mathematically precisely. To arrive at such a more general description, we take a preliminary step and replace the states of mind by, what Turing described as, "physical and definite counterparts". This is done by considering "state of mind" not as a property of the working computor, but rather as part of the configuration on which he operates. Turing discussed this replacement very vividly in his 1936 paper:

It is always possible for the computer [i.e., in our terminology, the computor] to break off from his work, to go away and forget all about it, and later to come back and go on with it. If he does this he must leave a note of instructions (written in some standard form) explaining how the work is to be continued. This note is the counterpart of the "state of mind". We will suppose that the computer works in such a desultory manner that he never does more than one step at a sitting. The note of instructions must enable him to carry out one step and write the next note.

This was achieved quite beautifully by Post in 1947, and Post's approach of describing Turing machine computations is used in (Davis 1958). The configurations on which the machine works are instantaneous descriptions, briefly, *id*'s. These are finite sequences in the alphabet of a Turing machine containing exactly one state symbol; the position of the state symbol indicates which symbol is being scanned. A program of a Turing machine can now be viewed as a set of Post production rules operating on (a single symbol of) such *id*'s. If in this way of describing Turing machines one replaces finite sequences by finite graphs (with a few well-motivated properties) and the simple Post-Turing operations on one symbol at a time by operations on a fixed finite number of distinguished graphs, then one arrives at the notion of a *Kolmogorov Machine*. This latter notion, or rather a general concept of algorithm, was introduced by Kolmogorov and Uspensky in 1958: KMs compute exactly the Turing computable number theoretic functions.

Which number theoretic functions can be computed by Turing machines or other computing devices? Kleene's analysis of the equational calculus led to the introduction of the 'regular minimization' operator and to an inductive characterization of Gödel's class of general recursive functions. Suppose the n+1-place function ϕ has the property that for every $x_1,\ldots,x_n$ there is a y such that $\phi(x_1,\ldots,x_n,y)=0$. Denote by $\mu y.\phi(x_1,\ldots,x_n,y)=0$ the least such y. Under these conditions, the function ψ given by $\psi(x_1,\ldots,x_n)=\mu y.\phi(x_1,\ldots,x_n,y)=0$ is said to be obtained from ϕ by regular minimization. A function is (μ-) *recursive* iff it can be obtained from the initial functions by a finite number of applications of composition, primitive recursion, and regular minimization. I.e., the class $\mathfrak{R}$ of recursive functions is obtained from $\mathfrak{PR}$ by closure under regular minimization.

Up to now we have considered total (i.e., everywhere defined) functions and operations that do not lead outside the class of total functions. Gödel (after Kleene) emphasized the importance of partial functions defined only for a subset of **N** or **N**x ... x**N** if they are unary

or n-ary. For example, while |iy.xy=0 is an equivalent definition of the zero function, an expression like |iy.x+y=0 is undefined for each x=l,2,... . For partial functions ∗ we liberalize the regularity condition for minimization: u=|iy.<|>(xi,...,x_n,y)=O holds if and only if the following two conditions are satisfied:

(i) for any i=0,l,...,u-l, <t>(xi,...,x_n,i) is defined and different from 0;
(ii) $\phi(x_1,...,x_n,u)=0$.

Given xi,...,x_n at most one number u can satisfy both (i) and (ii). If no such u exists then the n-tupel (xi,...,x_n) is outside the domain of the function |iy.<|>(xi,...,x_n,y)=0 or, stated differently, ^y.<|>(xi,...,x_n,y)=0 is undefined at (xi,...,x_n). A function is *partial recursive* iff it can be obtained from the initial functions by a finite number of applications of composition, primitive recursion, and (unrestricted) minimization.

One can explicitly construct Turing machines for the initial functions and for those operations on programs that correspond to composition, primitive recursion, regular, as well as unrestricted minimization. Thus, these operations do not lead outside the class of Turing computable functions. This immediately yields the following result, showing the power of Turing machines:

THEOREM: All partial recursive functions are Turing-computable.

3 **Effective descriptions**

To prove the converse of the last Theorem, we follow Pythagoras[1] ontological prescription and use Gödel's device to assign numbers e and y as *codes* to Turing machines *Tl* and to finite sequences of *ids.* The coding introduced by Gödel for formal theories in 1931 (and adapted in Davis[1] book for Turing machines) will do. Once a machine is coded as a number, the code can be supplied as an input to any machine; this adumbrates the metamorphosis of hardware into software, culminating in the stored-program computer. Programming the first computers virtually amounted to inserting wires into plugboards, but von Neumann — influenced by Turing's treatment of his paper machines — realized that programs can be coded and stored as strings of symbols in the same way as data.

For any effective Gödel-numbering it is easy, though somewhat tedious, to establish that simple syntactic notions and operations concerning machines and their *ids* can be represented by number-theoretic, indeed primitive recursive predicates and functions. For instance, the ternary predicate T(e,x,y) expressing that y is (the code of) a computation of Turing machine (tt with code) e for input x is primitive recursive; also, the unary 'result-extracting[1] function U is primitive recursive, where U(y) is either the number on the tape of the last *id* of the computation y or, in case y is not the code of a computation, U(y) is 0, (Our choice of the default value 0 is only to guarantee that U is a total function.)

Given a Turing-computable function ϕ and a machine *ft* with code e computing ϕ, there are two possibilities for any input x:
Case 1: *fl* terminates. Then there is a (first) computation y* of *Tt* for input x, i.e., y* is the smallest number y such that T(e,x,y); the number U(y*) read on the tape in the final configuration of y* coincides with the value ()>(x); writing jxy.T(e,x,y) as an abbreviation for |iy.x_T(e,x,y)=0, we have

(**) <Kx)=U(^y.T(e,x,y))

Case 2: tt does not terminate. Then for any y, the predicate T(e,x,y) will be false, i.e., %$_T$(e,x,y)=l for all y, and condition (ii) in the definition of the unrestricted |i-operator will never hold. Thus, both (iy.T(e,x,y) and U((iy.T(e,x,y)) are undefined.
This establishes (**) for both cases, and we have proved Kleene's Normal Form Theorem for unary Turing-computable functions. Kleene's Theorem holds also, with the same proof, for functions having any finite number of arguments.

THEOREM (Kleene's Normal Form) Let 9 be an n-place number theoretic function that can be computed by a Turing machine with code e; then for each xi,...,x_n in the domain of 9 we have

(p(xi,...,x_n)=U(^y.T(e,xi,...,x_n,y)).

Kleene established this Normal Form Theorem not for Turing-computable functions, but for total functions that can be calculated in Gödel's equational calculus. He concluded that Gödel's general recursive functions are recursive, a conclusion we can draw now for Turing-computable partial functions.

COROLLARY Every Turing-computable function is partial recursive.

Using the Theorem formulated at the very end of section 2, it follows that Turing-computability for number theoretic functions is equivalent to partial recursiveness and, indeed, to all the other characterizations discussed in that section. Kleene's Theorem has most interesting additional consequences to be discussed in the next section. This is done not just to present elegant mathematical results, but to reinforce the conceptual analysis, as these results (are taken by some to) lend additional support to Church's Thesis.

4 Basic results

Kleene's Normal Form Theorem has important consequences for the theory of computability. Consider, first of all, the two-place function $\psi(e,x)=U(\mu y.T(e,x,y))$; ψ is partial recursive and provides an enumeration of all unary partial recursive functions. This is Kleene's Enumeration Theorem. The theorem -- together with the equivalence of partial recursiveness and Turing-computability -- guarantees the existence of a *universal Turing machine.* Indeed, consider a Turing machine $\mathfrak{M}[\psi]$ computing ψ. For any pair of arguments e and x, $\mathfrak{M}[\psi]$ interprets its first argument as the code of a Turing machine $\mathfrak{M}[\varphi]$ (computing a unary partial recursive function φ) and its second argument as the input for $\mathfrak{M}[\varphi]$. Then $\mathfrak{M}[\psi]$ proceeds to compute $\varphi(x)$ following the program of $\mathfrak{M}[\varphi]$. Since $\mathfrak{M}[\varphi]$ was arbitrary, $\mathfrak{M}[\psi]$ is able to simulate any Turing machine. Similarly, for each n=2, 3, 4,... , there is a partial recursive (n+1)-place function $\psi^{(n)}$ such that every n-place partial recursive function φ can be written as $\varphi(x_1,...,x_n) = \psi^{(n)}(e,x_1,...,x_n)$ for a suitable index e for φ.

Given a function $\varphi(p_1,...,p_m,x_1,...,x_n)$ of m+n variables, let us 'parametrize' φ, i.e., assign values $p_1^*,...,p_m^*$ to the first m variables. We then obtain a new n-place function ϕ as follows:

$$\phi(x_1,...,x_n)=\varphi(p_1^*,...,p_m^*,x_1,...,x_n).$$

Intuitively, ϕ is the "slice" of φ with coordinates $(p_1^*,...,p_m^*)$. By Kleene's Enumeration Theorem we can write for suitable indices e and e*:

$$\varphi(p_1,...,p_m,x_1,...,x_n) = \psi^{(m+n)}(e,p_1,...,p_m,x_1,...,x_n)$$

and

$(Kxi,...,x_n) = y^{(n)}(e^*,xi,...,x_n)$.

The following theorem tells us that the slicing operation leading from e to e* can be performed uniformly and effectively:

PARAMETER (or S&-) THEOREM For each m,n > 0 there is a primitive recursive (n+l)-place function S& such that

$$\psi^{(m+n)}(e,p_1,...,p_m,x_1,...,x_n)=\psi^{(m)}(S^n_m(e,p_1,...,p_m),x_1,...,x_n).$$

This theorem was established by Kleene at first for GödeFs general recursive functions; it amounts in that context to replacing effectively the first m variables of the function <p computed from equations with code e by numerals for $pi,...,p_m$- This theorem is used to prove the following central result, also due to Kleene:

RECURSION THEOREM Let ϕ be a partial recursive function with (m+l)-places. Then there is a number e such that for all $xi,...,x_m$

$$\backslash|f^{(m)}(e,xi,...,x_m) = (|>(e,xi,...,x_m).$$

The theorem finds many uses in establishing that implicitly defined functions are actually partial recursive. For example, letting m=l and writing \|f instead of $y^{\wedge 1}$), an application of the Recursion Theorem to the function g(y,x)=y immediately yields:

FIXPOINT THEOREM There is a constant e* such that for all x

$y(e^*,x) = e^*$.

Intuitively, for every possible input x, the program with number e* uses any input to output a copy of itself — just as a self-replicating organism. More generally, for every recursive function r(z), we obtain by applying the recursion theorem to the function \p(r(z),x) a number e such that for all x, \j/(e,x)=y(r(e),x).

Kleene[1]s Enumeration Theorem allows us to reformulate properties of partial recursive functions as number theoretic properties of their codes. A set P of natural numbers is called an *input-output property* (of unary partial recursive functions) iff, whenever e^f is in P and e" codes the same function as e^f (in the sense that $v(e^f,x)=v(e^{lf},x)$ for all x in their common domain), then e" is also

in P. The following theorem states that non-trivial input-output properties are not recursive.

RICE'S THEOREM If P is an input-output property such that neither it nor its complement is empty, then P is not recursive.

There are important non-trivial input-output properties of programs. For instance, the set of codes of Turing machines computing total functions is one such property; as a consequence of Rice's theorem we know that there is no Turing machine that decides, whether or not a given Turing-computable function is total. We will discuss additional undecidable problems in the next section; however, undecidability will be established there in quite different ways.

5 Undecidable problems

"Problems" are identified with (characteristic functions) of one place predicates or, equivalently, with subsets of N. Let K be the problem corresponding to the question: Does the computation of machine *Tt* with code e terminate, if the input is the natural number e? This problem was formulated by Turing and is known as the Self-Halting Problem. A diagonal argument (using the existence of a universal machine) settles the undecidability of K without resorting to Rice's theorem.

The Self-Halting problem is the progenitor of a multitude of undecidable problems, arising from virtually all fields of mathematics. Consider, for instance, the following problem. INSTANCE: a presentation P of a group G via generators and relations (P looks like a multiplication table for G) and two expressions v and w built up from these generators, using the multiplication operation and its inverse. QUESTION: do v and w represent the same element of G? This is known as the word problem for groups. Novikov and, independently, Boone proved that the problem is undecidable. — Markov proved the undecidability of the following problem for 4-dimensional manifolds. INSTANCE: a suitable combinatorial description of two such manifolds. QUESTION: are they homeomorphic? In the 2-dimensional case the homeomorphism problem becomes decidable; it is not known whether the 3-dimensional problem is decidable or not.

Yet another logic-free undecidable problem is given by elementary functions, i.e., those functions f of a real variable x which can be built up using exponentials, logarithms, n-th roots, trigonometric functions and their inverses by addition, multipli-cation, and composition. While the derivative operation does not lead outside the realm of elementary functions, the integral of, say, (sin x)/x is not elementary. Indeed, Richardson proved that the following problem is undecidable. INSTANCE: an elementary function f(x). QUESTION: is the integral of f elementary? Finally, the tenth problem in Hilbert's list presented at the International Congress of Mathematicians in 1900 asks for an algorithm to decide of a polynomial equation $p(x_1,...,x_n)=0$, with integer coefficients, whether or not it has integer solutions. Following important work of J. Robinson, Davis, and Putnam, Matijasevic proved in 1970 that the problem is undecidable. -- Let us go back to the problem K.

Recalling the discussion in section 3, it is clear that the predicate K(e) can be defined by the statement "there exists a y, such that T(e,e,y)". K is thus an example of a *recursively enumerable* or, briefly, r.e. set of natural numbers: it can be defined by prefixing one existential quantifier to a decidable binary predicate. Trying for each n in N all possible pairs (e,y) such that e+y=n and singling out e whenever T(e,e,y) is true, we can effectively list all members of K. Recursively enumerable sets were named because of the following equivalent characterization: a subset of N is r.e. iff it is either empty or the range of a (primitive) recursive function. Furthermore, a set R is recursive iff both R and its complement are recursively enumerable. This latter fact allows a different entry to computability theory: recursive enumerability is taken as the basic concept, and recursive sets are defined as those r.e. sets whose complements are also recursively enumerable; that is done in Post's approach.

The importance of K among r.e. sets stems from its being *complete*, i.e., maximally difficult among all r.e. sets; by this we mean that every r.e. set can be effectively reduced to K. A problem P is effectively reducible to Q iff there is a recursive function f such that for all natural numbers x, x is in P iff f(x) is in Q. Thus, if we want to see that x is a solution of problem P we compute f(x) and test whether f(x) solves Q! Clearly, Q is at least as difficult as P, in the sense that effective solutions for Q yield effective solutions for P. By effectively reducing the self-halting problem K to the Entscheidungs-problem E, Turing concluded from the undecidability of K, that also E cannot be decidable. Turing's reduction is nothing else but a

transcription of the ternary predicate T(e,e,y) into a first-order arithmetical theory with sufficient demonstrative power.

The self-halting problem K, the most complex among r.e. problems, is the simplest in an infinite sequence of manifestly undecidable problems, the so-called jumps. In order to obtain the *jump hierarchy*, the concept of computation is relativized to sets of natural numbers whose membership relations are revealed by "oracles". The jump K' of K, for example, is defined as the self-halting problem, when an oracle for K is available. This hierarchy can be associated in a most informative way to definability questions in the language of arithmetic: all jumps can be defined by increasingly complex arithmetical formulas, and all arithmetically definable sets are reducible to some jump. The above construction underlies the arithmetic hierarchy introduced by Kleene and Mostowski in the fifties.

Certain interesting sets of natural numbers are not definable by arithmetic formulas; one example is the set of all Gödel numbers of true arithmetic statements. If this set were arithmetically definable, one could formulate arithmetically the "liar sentence" that expresses its own falsity. This observation of Gödel and Tarski is the cornerstone for proving the incompleteness of every formal theory of arithmetic that contains symbols for addition and multi-plication and has semantically sound axioms and rules of inference: No matter which (true) statements we choose as axioms, and no matter which inference rules we adopt (provided they are truth-preserving) there are statements which, albeit true for the natural numbers, are not provable in the theory. Arithmetic truth can be defined, however, in second order languages that allow quanti-fication over functions; cf. **Arithmetic and analytic hierarchy.**

6 Physical steps

Turing appealed in his analysis of mechanical calculability to the limitations of the human sensory apparatus; however, he claimed that the justification for his thesis lies ultimately "in the fact that the human memory is necessarily limited". This remark is not expanded upon at all, and we can only speculate as to Turing's understanding of this "fact": Did he have in mind more than the spatial limitations

for "encoding" finite configurations? If not, the restrictions can be motivated by physical considerations.[4]

Assume that a Kolmogorov machine operates on configurations containing z different symbols, each symbol being physically "coded" by at least one atom; that is an altogether reasonable assumption. Then there must be at least z pairwise disjoint regions containing the codes (of the symbols). Otherwise, the electron clouds of two different codes might overlap, making the codes indistinguishable. Let c and a denote the speed of light and Bohr's radius of the hydrogen atom, where $a/c=0.176x10^{-18}$ seconds. It follows that the codes will be contained in a volume V of at least $z(4/3)rca^3$ cubic meters; that forces the diameter 2r to be larger than $2az^{1/3}$ meters — the diameter being the largest possible distance between two codes in this volume. Let f be the frequency of our machine; thus, 1/f is the time available for each computation step. Since signals cannot travel faster than light and since a computation step involves the whole configuration, it follows that f cannot exceed $c/(2az^{1/3})$ steps per second. Thus, we obtain the inequality $fz^{1/3} < 2.828x10^{18}$ steps per second, which points out a fundamental incompatibility between high number of codes (i.e., size of configura-tions) and high computational speed. The operations of KMs are thus restricted in complexity, as they have to lead from distinguished graphs to distinguished graphs; and within the given physical boundaries only finitely many different graphs are realizable.

If we focus on physical devices and analyze machine, not human mechanical, computability we have to take into account the possibility of parallel procedures as incorporated, for example, in cellular automata. Gandy provided in his (1980) the first conceptual analysis and a general description of parallel algorithms. These algorithms are thought to be carried out by "discrete deterministic mechanical devices", i.e., machines satisfying the physical assumptions explicit in our discussion above; in Gandy's words: "The only physical assumptions made about mechanical devices ... are that there is a lower bound on the linear dimensions of every atomic part of the device and that there is an upper bound (the velocity of light) on the speed of propagation of changes". He formulated axiomatic principles for these devices and proved that whatever can be calculated by devices satisfying the principles, Gandy Machines, is also computable by a Turing machine.

---

[4] Are there ways of getting around the effect of such physical limitations? That issue is discussed with references to the literature by Mundici and Sieg, (1994).

References and further reading

Cutland, N. (1980) Computability - An introduction to recursive function theory, Cambridge: Cambridge University Press. (An informative introduction to basic results.)

Davis, M. (ed.) (1965) The Undecidable, Hewlett (New York): Raven Press. (Anthology of fundamental papers on the subject by Gödel, Church, Turing, Kleene, Rosser, and Post.)

Davis, M. (1958) Computability and Unsolvability, New York: McGraw-Hill. (A classical, detailed presentation of the basic parts of computability theory.)

Herken, R. (ed.) (1988) The Universal Turing Machine (A half-century survey), Oxford: Oxford University Press. (The book contains informed systematic, enlightening historical, and provocative "futuristic" essays by, among others, Kleene, Gandy, Davis, Feferman, Penrose.)

Mundici, D. and Sieg, W. (1994) 'Paper Machines', Philosophia Mathematica. (A non-technical essay on the development of the main ideas of computability, also covering physical computation.)

Odifreddi, P. (1992) Classical Recursion Theory, Amsterdam: North-Holland Publishing Company, volume I, second reprinting 1992; volume II, 1993. (A comprehensive modern treatise on computability theory.)

Rogers, H. (1967) Theory of recursive functions and effective computability, New York: McGraw Hill. (The standard reference on recursion theory before Odifreddi's and Soare's books.)

Sieg, W. (1994) 'Mechanical Procedures and Mathematical Experience' in *Mathematics and Mind* ed. A. George, Oxford, Oxford University Press. (Examination of the conceptual analyses underlying computability theory, in particular in Turing's fundamental paper; emphasizes connections to investigations in the foundations of mathematics.)

Soare, R.I. (1987) Recursively enumerable sets and degrees, Berlin, New York: Springer Verlag. (Brings the reader to the frontiers of

current research on computable functions, featuring: Post's problem, oracles, priority methods, lattice of r.e. sets.)

van Heijenoort, J. (ed.) (1967) From Frege to Gödel - A source book in mathematical logic, 1879-1931, Cambridge: Harvard University Press. (Anthology of fundamental papers in logic starting with Frege's *Begriffsschrift* (1879) and ending with Gödel and Herbrand's work in 1931.)

DANIELE MUNDICI and WILFRIED SIEG