Mathematics Studies Machines.

by

Daniele Mundici and Wilfried Sieg

May 1993

Report CMU-PHIL-36

Carnegie Mellon

Philosophy
Methodology
Logic

Pittsburgh, Pennsylvania 15213-3890

Mathematics Studies Machines.

Daniele Mundici and Wilfried Sieg

To appear (in Italian) in: <u>Le Scienze della Mente</u>, C. Mangione (ed.), volume of the encyclopedia <u>Le Scienze e le Tecnologie</u>, ieri oggi e domani, Grandi Opere

INTRODUCTION. Machines were introduced as calculating devices to simulate operations carried out by human computors following fixed algorithms: this is true for the early mechanical calculators devised by Pascal and Leibniz, for the analytical engine built by Babbage, and the theoretical machines introduced by Turing. The distinguishing feature of the latter is their universality: They are claimed to be able to capture any algorithm whatsoever and, conversely, any procedure they can carry out is evidently algorithmic. The study of such "paper machines" by mathematical means is the topic of our contribution. This is not only in accord with its usual understanding in computer science, but conceptually and historically right, when we recall the purpose for which Turing machines were introduced.

Section one, entitled "STEPS (towards Turing's analysis)", reexamines the analysis of effective calculability as it emerged in the thirties. By way of providing crucial "LOGICAL BACKGROUND" we describe in section zero the decision problem for first-order logic as formulated in the twenties by Hilbert. Then we shall argue that the subsequent work of Gödel, Church & Kleene, and Hilbert & Bernays focused on *one* central informal notion, namely the stepwise calculation (by a human computor) in a symbolic calculus. However, in each case a serious stumbling-block to a convincing analysis emerged, as the restricted nature of the steps in computations was characterized by recourse to (primitive) recursiveness or, in Gödel's case, to very specific and obviously mechanical rules. That stumbling-block was overcome only by Turing, who formulated finiteness conditions and reduced the notion of a mechanical procedure (carried out by a human computor satisfying these conditions) to a mathematically very convenient concept of computability by machines.

For Gödel this mathematical notion of computability captured the informal one of calculability exactly and it allowed him to formulate his Incompleteness Theorems for all "formal" theories (satisfying some basic conditions). This is discussed in section two, "STEPS (towards the mathematical theory)", after describing results that are on the one hand central to recursion theory but on the other hand reinforce the conceptual analysis given in section one. The idea of stepwise calculation is reflected most directly in Kleene's fundamental Normal Form Theorem discussed in section 2.0. With hardly any additional work this theorem implies in section 2.1 the existence of a "universal" machine; such a machine was explicitly constructed by Turing, and the underlying idea turned out to be crucial for the architecture of modern computers. Then we discuss the unsolvability of the Halting Problem together with a concept of effective reducibility: by reducing the Halting Problem to the decision problem Turing established the unsolvability of the latter.

The computability notion based on Turing machines is highly idealized in at least two respects: although the number of different kinds of steps and different types of symbols is strictly bounded, the number of applications of steps and occurrences of symbols that may be used in a computation is not bounded at all. This means more intuitively that the time and space a computation can take are not restricted. By imposing suitable time or space limitations one attempts to characterize "tractable" problems and "feasible" algorithms. That is the central topic of section three, "COUNTING THE NUMBER OF STEPS"; it discusses in particular the functions that can be computed in polynomial time (restricting trivially also space).

In section four, "PHYSICAL LIMITS OF STEPS", we continue the analysis of computation steps. Turing's finiteness conditions are crucially motivated by limitations of the human sensory apparatus, but ultimately, Turing claims, by necessary limitations of human memory. Physical restrictions that may provide reasons for such memory limitations are among those discussed in this section. We argue that, for good physical reasons, steps cannot be accelerated arbitrarily and cannot be made arbitrarily complex. So one is led to a different sense of the mathematical study of machines, namely, the mathematical formulation of physical principles governing the operations of physically realizable machines. Our mathematical inquiry into "paper machines" will lead us to a point where (effective) mathematical descriptions of nature and (natural) computations for mathematical problems coincide.

0 LOGICAL BACKGROUND.

In some respects, the issues go back to Greek mathematics and philosophy, as they concern, on the one hand, the axiomatic presentation of geometry (by Euclid) and, on the other hand, the formalization of logical reasoning (by Aristotle and, for sentential logic, by the Stoics). But it was only Gottlob Frege who provided, with his 1879 "Begriffsschrift", an expressive formal language and a logical calculus that allowed the realization of the earlier intentions with respect to mathematics. Frege required that all assumptions be explicitly formulated in the formal language, and that each step in a proof be taken in accord with one of the antecedently specified rules of the logical calculus. The second requirement was Frege's way of sharpening the axiomatic method explicitly traced back to Euclid. With this sharpening Frege pursued the aim of recognizing the "epistemological nature" of theorems. Because that can be done only if inferences do not require contentual knowledge Frege realized that the application of inferences has to be recognizable as correct on account of the syntactic form of the sentences occurring in them. Indeed, Frege claimed that in his logical system "inference is conducted like a calculation". In formulating such inference rules, Frege went beyond the efforts of his contemporary Peano, who also pursued the goal of expressing mathematics in a formal or symbolic language.

Almost half a century later, in 1933, Gödel referred back to Frege and Peano when he formulated "the outstanding feature of the rules of inference" in a formal mathematical system. The rules, Gödel said, "refer only to the outward structure of the formulas, not to their meaning, so that they can be applied by someone who knew nothing about mathematics, or by a machine." Frege had not considered the possibility of mechanically drawing inferences to be among the logically most significant achievements of his "Begriffsschrift". But Hilbert grasped the potential of this aspect, radicalized it, and exploited it in his formulation and pursuit of the consistency problem; a research program that started in a rough and tentative way around 1900 and was pursued with great intensity in the twenties. Over the years the strict formalization of mathematics seemed to open up new ways of solving mathematical problems -- through calculation. The most famous problem among these is the so-called Entscheidungsproblem or decision problem. It is closely related to the consistency problem and was pursued by some (e.g., Herbrand) on account of this connection. Its classical formulation in terms of validity and satisfiability is found in Hilbert and Ackermann's 1928 book "Grundzüge der Theoretischen Logik": "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."

Hilbert and Ackermann emphasized the fundamental importance of a solution to the decision problem, and researchers in the Hilbert school realized full well that a positive solution for predicate logic would allow the decision concerning the provability of any mathematical statement. For some, e.g. von Neumann in 1927, that was sufficient reason to expect a negative solution; at that time von Neumann also claimed that there was no clue as to how a proof of undecidability would go. In support for this claim, he pointed to the underlying conceptual problem: There were well-known proofs for the unsolvability of certain mathematical problems; but all such impossibility results were given relative to a determinate class of admissible means, e.g., doubling the cube by using only ruler and compass. And exactly here lies the problem: A negative solution to the Entscheidungsproblem required a mathematically precise answer to the question, "What are mechanical procedures?"

Examples of mechanical procedures when operating with natural numbers were familiar from mathematical practice; indeed, the values of all "primitive recursive" functions can be calculated mechanically. And most functions from elementary number theory are in this class, e.g., addition, multiplication, exponentiation, the sequence of prime numbers and the sequence FIB of Fibonacci-numbers. For the latter sequence of 1,1,2,3,5,8,... this means that the values of FIB for the first two arguments are given outright by FIB(0)=1 and FIB(1)=1, and for arguments of the form n+2 they are given by FIB(n+2)=FIB(n+1)FIB(n). I.e., to fix the value for n+2 one recurs on the values of FIB for the preceding arguments (n+1) and n. What is historically most important here is the fact that Ackermann discovered in the mid-twenties a function that was not primitive recursive, but whose values could nevertheless be determined by a mechanical procedure. In his classical 1931 paper "Ueber formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme", Gödel had used the primitive recursive functions to describe, after a number theoretic coding, the syntax of particular "formal" theories. Being concerned to use a general concept of formality, through an underlying concept of calculability, there was no good reason to focus attention on theories whose syntax could be presented primitive recursively.

1 STEPS (towards Turing's Thesis). Let us re-emphasize that for our investigations it is absolutely crucial to have convincingly analyzed notions or, in case no convincing case has been made yet, to be clear about scope and limit of the mathematical concepts that are used as "first approximations" to the informal concept.

According to the conventional view we were given by the work of Gödel, Church, Turing, and others (e.g., Kleene, Post, Hilbert, Bernays) mathematical definitions of mechanical procedures. The fact that these definitions turned out to be equivalent, in the sense that they characterize the same class of number theoretic functions as computable, is taken as significant support for Church's Thesis. The question for us is: What are the grounds for accepting the various notions as constituting a mathematical description of mechanical procedures? (The considerations in this section are based on [Sieg 1993].)

1.0 RECURSIVENESS. In his 1934 Lectures at Princeton Gödel strove to make his incompleteness results less dependent on particular formalisms, but he did not succeed to his own satisfaction in resolving the main conceptual issue, namely, to give a general notion of "formal theory". He still viewed the primitive recursive definability of formulas and proofs as a precise condition which in p r a c t i c e suffices to describe particular formal systems, though he was clearly looking for a condition that would suffice in p r i n c i p l e. But in what direction could one search? Gödel considered it as an "important property" that, for any argument, the value of a primitive recursive function can be computed by a "finite procedure" and he added in a footnote: "The converse seems to be true if, besides recursions according to the scheme (2) [of primitive recursion, M&S], 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 section of the Lecture Notes Gödel described "general recursive functions"; they are obtained as unique solutions of certain functional equations, and their values must be calculable in an "equational calculus". For Gödel the crucial point of his proposal was the specification of particular mechanical rules for the computation of function values. Though the footnote we just quoted may seem to express a form of Church's Thesis, Gödel emphasized in a 1965 letter to Martin Davis that no formulation of Church's Thesis was intended. He wrote in that letter: "The conjecture stated there only refers to the equivalence of 'finite (computation) procedure' and 'recursive procedure'. However, I was, at the time of these lectures, not at all convinced that my concept of recursion comprises all possible recursions; ... "

At the time, Gödel was equally unconvinced by Church's proposal to identify effective calculability with lambda-definability. Church recalled (in a letter to Kleene dated November 29, 1935) a conversation with Gödel in early 1934 , when Gödel called this proposal "thoroughly unsatisfactory". Nevertheless, Church announced his thesis in a talk at the meeting of the American Mathematical Society on April 19, 1935; he formulated it in terms of recursiveness, not lambda-definability. In his subsequent famous 1936 paper "An unsolvable problem of elementary number theory" Church wrote: "The purpose of the present paper is to propose a definition of effective calculability which is thought to correspond satisfactorily to the somewhat vague intuitive notion in terms of which problems of this class are often stated, and to show, by means of an example, that not every problem of this class is solvable." Church proposed again to identify effective calculability with recursiveness. The fact that lambda-definability was known to be an equivalent concept simply added for Church "... to the strength of the reasons adduced below for believing that they [these precise concepts] constitute as general a characterization of this notion [i.e., effective calculability] as is consistent with the usual intuitive understanding of it."

1.1 STEP-BY-STEP. To give a deeper analysis Church pointed out that the notion "calculability in a logic" suggests itself as one way to characterize effective calculability of number theoretic functions, and he argued that it does not lead to a definition more general than recursiveness. Let us indicate briefly the argument: Church considered a logic **L**, i.e. a system of symbolic logic whose language contains the equality symbol =, a symbol { }( ) for the application of a unary function symbol to its argument, and numerals for the positive integers. He called unary functions F *effectively calculable* if and only if there is an expression f in **L** such that {f}(M)=N is a theorem of **L** exactly when F(m)=n; M and N are expressions of **L** that stand for the positive integers m and n. Church claimed such F are recursive, when **L** satisfies conditions that guarantee essentially the recursive enumerability of **L**'s theorem predicate; the claim follows by an unbounded search. The crucial condition in Church's list requires the STEPS in derivations of equations to be, well, recursive! Here we hit on a serious stumbling-block for Church's analysis, since an appeal to the thesis when arguing for it is logically circular. And yet, Church's argument achieves something: The general concept of calculability is explicated as derivability in a symbolic logic, and the step-condition is used to sharpen the idea that we operate by effective rules in such a formalism. We suggest that the claim "Steps of any effective procedure must be recursive!" be called **Church's Central Thesis**.

The concept of "calculability in a logic" used in Church's argument is an extremely natural and fruitful one. It is directly related to decidability (Entscheidungsdefinitheit) for relations and classes introduced by Gödel in his 1931 paper and to "representability" as used in his Princeton lectures. It was used also in other analyses: Gödel defined that very notion in his 1936 note "On the length of proofs" and emphasized its (type-) absoluteness. In his contribution to the Princeton Bicentennial Conference (1946) Gödel reemphasized absoluteness in a more general sense and took it as the main reason for the special importance of recursiveness. Here we have according to Gödel the first interesting epistemological notion whose definition is not dependent on the chosen formalism. But the stumbling-block Church had to face shows up also here: absoluteness is achieved only relative to the description of f o r m a l systems. The more general definition of absoluteness Gödel gave in 1946 is already explicit in the work of Hilbert and Bernays in the second volume of their book "Grundlagen der Mathematik". They called a number-theoretic function reckonable according to rules ("regelrecht auswertbar"), if it is computable in some deductive formalism. Then they formulated three recursiveness conditions for such formalisms and showed: (i) a function that is computable in some deductive formalism satisfying the recursiveness conditions can be computed in a very restricted number theoretic formalism, and (ii) the functions computable in the latter formalism are exactly the recursive functions.

Hilbert and Bernays' analysis is a natural and satisfactory capping of the development from Entscheidungsdefinitheit to an "absolute" notion of computability. But their analysis does not overcome the major stumbling-block; rather, it puts the stumbling-block in plain view through the recursiveness conditions that deductive formalisms must satisfy. The crucial condition requires the proof predicate for such formalisms to be primitive recursive! We want to show now, how Turing got around this fundamental difficulty and start out by describing Turing machines; in the presentation of these machines we follow Davis [1958], not Turing's original paper.

1.2 MACHINES & WORKERS. A Turing machine consists of a finite, but potentially infinite tape; the tape is divided into squares, and each square may carry a symbol from a finite alphabet, say, just the two-letter alphabet consisting of 0 and 1, or B(lank) and |. The machine is able to scan one square at a time and perform, depending on the content of the observed square and its own internal state, one of four operations: print 0, print 1, or shift attention to one of the two immediately adjacent squares. The operation of the machine is given by a finite list of commands in the form of quadruples $q_i s_k c_l q_m$ that express: if the machine is in internal state q_i and finds symbol s_k on the square it is scanning, then it is to carry out operation c_l and change its state to q_m. The deterministic character of the machine operation is guaranteed by the requirement that a program must not contain two different quadruples with the same first two components.

In 1936, the very year in which Turing's paper appeared, Emil Post published a strikingly similar computation model in a brief note that appeared in the Journal of Symbolic Logic under the title "Finite Combinatory Processes - Formulation 1". Here we have a worker who operates in a *symbol space* consisting of "a two way infinite sequence of spaces or boxes, i.e., ordinally similar to the series of integers ... . The problem solver or worker is to move and work in this symbol space, being capable of being in, and operating in but one box at a time. And apart from the presence of the worker, a box is to admit of but two possible conditions, i.e., being empty or unmarked, and having a single mark in it, say a vertical stroke". (Post remarks that the infinite sequence of boxes can be replaced by a potentially infinite one, expanding the finite sequence as necessary.) The worker can perform a number of *primitive acts*; namely, make a vertical stroke [V], erase a vertical stroke [E], move to the box immediately to the right [M_r] or to the left [M_l] (of the box he is in), and determine whether the box he is in is marked or not [D]. In carrying out a particular combinatory process the worker begins in a special box (the *starting point*) and then follows directions from a finite, numbered sequence of instructions. The i-th direction, i between 1 and n, is in one of the following forms: (i) carry out act V, E, M_r, or M_l and then follow direction j_i, (ii) carry out act D and then, depending on whether the answer was positive or negative, follow direction j_i' or j_i''. (Post has a special stop instruction, but that can be replaced by the convention to stop, when the number of the next direction is greater than n.)

Are there intrinsic reasons for choosing Formulation 1, except for its simplicity and Post's expectation that it will turn out to be equivalent to recursiveness? An answer to this question is not clear (from Post's paper). Post wrote at the very end of his paper: "The writer expects the present formulation to turn out to be equivalent to recursiveness in the sense of the Gödel-Church development. Its purpose, however, is not only to present a system of a certain logical potency but also, in its restricted field, of psychological fidelity. In the latter sense wider and wider formulations are contemplated. On the other hand, our aim will be to show that all such are logically reducible to formulation 1. We offer this conclusion at the present moment as a *working hypothesis.* And to our mind such is Church's identification of effective calculability with recursiveness." Investigating wider and wider formulations and reducing them to Formulation 1 would change for Post this "hypothesis not so much to a definition or to an axiom but to a *natural law"* .

1.3 CONCEPTUAL ANALYSIS. It is methodologically remarkable that Turing proceeded in *exactly* the opposite way when trying to justify that all computable numbers are machine computable or, in our way of speaking, that all effectively calculable functions are Turing computable: He did not try to extend a narrow notion reducibly and obtain in this way additional quasi-empirical support, but analyzed the intended broad concept and reduced it to a narrow one -- once and for all. Turing's classical paper "On computable numbers" opens with a description of what is ostensibly its subject, namely, "real numbers whose expressions as a decimal are calculable by finite means". Turing is quick to point out that the fundamental problem of explicating "calculable by finite means" is the same when considering, e.g., computable functions of an integral variable. Thus it suffices to address the question: "What does it mean for a real number to be calculable by finite means?" In §9 he argues that the operations of his machines "include all those which are used in the computation of a number". But he does not try to establish the claim directly; he rather attempts to answer what he views as "the real question at issue", i.e., "What are the possible processes which can be carried out [by a human computor, M&S] in computing a number?"

Turing imagines a computor writing symbols on paper that is divided into squares "like a child's arithmetic book". As the two-dimensional character of this computing space is taken not to be essential, Turing takes a one-dimensional tape divided into squares as the basic computing space and formulates one important restriction. That restriction is motivated by definite limits of our sensory apparatus to distinguish -- at one glance -- between symbolic configurations of sufficient complexity. It states that only finitely many distinct symbols can be written on a square. Turing suggests as a reason that "If we were to allow an infinity of symbols, then there would be symbols differing to an arbitrarily small extent", and we would not be able to distinguish at one glance between them. A second (and clearly related) way of arguing the point uses a finite number of symbols and strings of such symbols. E.g., Arabic numerals like 9979 or 9989 are seen by us at one glance to be different; however, it is not possible for us to determine immediately that 9889995496789998769 is different from 98899954967899998769.

Now let us turn to the question: "What determines the steps of the computor, and what kind of elementary operations can he carry out?" The behavior is *uniquely* determined at any moment by two factors: (i) the symbols or symbolic configuration he observes, and (ii) his "internal state". This uniqueness requirement may be called the **determinacy condition (D);** it guarantees that computations are deterministic. Internal states are introduced to have the computor's behavior depend possibly on earlier observations and, thus, to reflect his experience. Since Turing wants to isolate operations of the computor that are "so elementary that it is not easy to imagine them further divided", it is crucial that symbolic configurations relevant for fixing the circumstances for the actions of a computor are immediately recognizable. So we are led to postulate that a computor has to satisfy two **finiteness conditions:**

(F.1) *there is a fixed finite number of symbolic configurations a computor can immediately recognize;*

(F.2) *there is a fixed finite number of internal states that need be taken into account*

For a given computor there are consequently only finitely many different relevant combinations of symbolic configurations and internal states. Since the computer's behavior is -- according to (D) -- uniquely determined by such combinations and associated operations, the computor can carry out at most finitely many different operations. These operations are restricted as follows:

(0.1) *only elements of observed symbolic configurations can be changed;*

(0.2) *the distribution of observed squares can be changed, but each of the new observed squares must be within a bounded distance L of an immediately previously observed square.*

Turing emphasizes that "the new observed squares must be immediately recognisable by the computer[11], and that means that the distributions of the new observed squares arising from changes according to (0.2) must be among the finitely many ones of (F.1). Clearly, the same must hold for the symbolic configurations resulting from changes according to (0.1). Since some of the operations may involve a change of internal state (or, as Turing also puts it, "state of mind"), Turing concludes: "The most general single operation must therefore be taken to be one of the following: (A) A possible change (a) of symbol [as in our (0.1)] together with a possible change of state of mind. (B) A possible change (b) of observed squares [as in our (0.2)] together with a possible change of state of mind." With this restrictive analysis of the possible steps of a computor, the proposition that his computations can be carried out by a Turing machine is established rather easily. Indeed, Turing first "constructs" machines that mimic the work of computors directly and then observes: "The machines just described do not differ very essentially from computing machines as defined in § 2, and corresponding to any machine of this type a computing machine can be constructed to compute the same sequence, that is to say the sequence computed by the computer [in our terminology: computor]." Thus we have **Turing's Theorem:** Any number theoretic function F that can be computed by a computor, satisfying the determinacy condition (D) and the conditions (F) and (O), can be computed by a Turing machine.

1.4 TURING'S THESIS. Turing's analysis and his theorem can be generalized by making an observation concerning the determinacy condition: (D) is not needed to guarantee the Turing computability of F in the theorem. Computors that do not satisfy (D) can be mimicked by non-deterministic Turing machines and thus, exploiting the reducibility of non-deterministic to deterministic machines, by deterministic Turing machines. That allows us to connect Turing's considerations with those of Church we discussed earlier. Consider for that purpose an effectively calculable function F and a non-deterministic computor who calculates the value of F in a logic L.

Using the generalized form of Turing's Theorem and the fact that Turing computable functions are recursive, F is recursive. This argument for F's recursiveness does no longer appeal to Church's Central Thesis; rather, such an appeal is replaced by the assumption that the calculation in the logic is done by a computor satisfying the conditions **(F)** and **(O)**. If that assumption is to be discharged, then a substantive thesis is needed again. And it is this thesis we call **Turing's Central Thesis**. It expresses the fact that a mechanical computor indeed satisfies the finiteness conditions **(F)** and that the elementary operations he can carry out are restricted as conditions **(O)** require.

Church wrote in a 1937 review of Turing's paper when comparing Turing computability, recursiveness, and lambda-definability: "Of these, the first has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately ..." For Gödel Turing's work provided "a precise and unquestionably adequate definition of the general concept of formal system". In the historical and systematic context Turing found himself, he asked exactly the right question: "What are the possible processes a human computor can carry out in computing a number?" The general problematic *required* an analysis of the idealized capabilities of a mechanical computor. Let us emphasize that the separation of conceptual analysis (leading to the axiomatic conditions) and rigorous proof (establishing Turing's Theorem) is essential for clarifying on what the correctness of his general thesis rests; namely, on recognizing that the axiomatic conditions are true for computors who proceed mechanically. We have to remember that quite clearly when moving to methodological discussions in artificial intelligence and cognitive science. Even Gödel got it wrong, when he claimed that Turing's argument in the 1936 paper was intended to show that "mental processes cannot go beyond mechanical procedures".

2 STEPS (towards the mathematical theory). We focus on results that are central for recursion theory and computer science; these results reinforce our conceptual analysis and are frequently appealed to in support of Church's Thesis. If we take for granted a representation of natural numbers in the two-letter alphabet of Turing machines and a straightforward definition of when to call a number-theoretic function *Turing computable,* we can recast our earlier question: Why does this notion provide "an unquestionably adequate definition of the general concept of formal system"? Is it at all plausible that every effectively calculable function is Turing computable?

2.0 NORMAL FORM. It seems that a n a i v e inspection of the very restricted notion of Turing computability should lead to "No!" as a tentative answer to the second and, thus, also to the first question. However, a systematic development of Turing computability convinces one quickly that it is indeed a powerful notion. One goes almost immediately beyond the examination of particular functions and the writing of programs for machines computing them; instead, one considers machines that correspond to operations on functions and that yield, when applied to computable functions, ones that are also computable. Three such functional operations are crucial, namely, composition, primitive recursion, and minimization. The latter operation allows an unbounded search for a solution to equations of the form g(y, x1, ... , xn)=0: given a recursive function g such that for every x1, ... , xn there is a y with g(y, x1, ... , xn)=0 one can define (by minimization) a new computable function f(x1, ... , xn) whose value for the indicated arguments is the smallest y with g(y, x1, ... , xn)=0. This is usually indicated by f(x1, ... , xn)=muy.g(y, x1, ... , xn)=0.

Kleene established in 1936 the equivalence between Gödel's recursiveness (defined via an equational calculus) and "mu-recursiveness[11]. The latter notion is characterized by an inductive definition that is obtained from that for the primitive recursive functions by adding just one additional clause - for minimization as described above. This characterization of the recursive functions together with the closure of computable functions under the above functional operations and the computability of a few simple initial functions implies the computablity of *all* recursive functions. Conversely, we can use the idea underlying Church's argument for the recursiveness of functions "calculable in a logic" (discussed at the end of section 1.1) and Kleene's argument for the recursiveness of functions "calculable in Gödel's equational calculus" to show that every total computable function is actually recursive! This is an instance of an argument, where conceptual analysis and "technical" work go fortuitously hand in hand.

Gandy called ChurclVs argument for his thesis the "step-by-step argument": if the steps in "logical calculations" are recursive, then the functions being calculated are recursive. But in what sense can steps be "recursive", as they are taken in a logical calculus, whereas recursiveness is a property of number theoretic functions and predicates? It was Goedel who had shown in his 1931 paper, how to "code", "arithmetize", or -- as we often put it in recognition of the fundamental character of his technique -- "Goedel-number" the finite syntactic objects of a logical calculus; given the arithmetization of syntactic objects, it is tedious, but not difficult to see that the syntactic notions (like formula or derivation) for "standard" formal theories are indeed (primitive) recursive. Computations of Turing machines can be described in exactly the same way to yield ultimately that every Turing computable function is recursive.

This result was generalized by Kleene, who associated in the late thirties partial recursive functions with computations; i.e., the domain of a function is now taken to coincide with the set of arguments for which the corresponding Turing machine computation terminates. The mathematical essence of the earlier observations is captured for such partial recursive functions by Kleene's Normal Form Theorem. In the formulation (for a one-place function f to keep matters simple, but without losing generality) one uses a particular three-place predicate T and a one-place function U. The T-predicate applied to numbers e,x,y expresses that y is the code of a computation of a Turing machine with code e for the numerical argument x; U extracts from the computation, in case it does terminate, the numerical result. Kleene's Theorem can now be stated as follows: For every partial recursive function f there is a natural number e, such that for all x in the domain of f,
f(x) = U(muy. T(e,x,y)).

2.1 UNIVERSAL MACHINES. Kleene's theorem exploits the uniform description of Turing machine computations and has most important consequences. Consider, first of all, the two-place function g(e,x)=U(muy. T(e,x,y)); g is partial recursive and provides an enumeration of all one-place partial recursive functions (Kleene's Enumeration Theorem). Together with the fact that the partial recursive functions are exactly the Turing computable ones this theorem guarantees the existence of a universal Turing machine. Such a machine was explicitly constructed by Turing, and the idea underlying his construction was fundamental for the development of the architecture of digital computers through von Neumann; see [von Neumann 1958]. How is the existence established here by using Kleene's mathematical work? Note that by the above observations the two-place function g can be computed by a Turing machine M[g]. M[g] interprets its first argument as the code e of a Turing machine M[f] computing the one-place function f; its second argument is taken by M[g] as the argument for f or rather M[f]. Then M[g] proceeds to compute f(x) by following the program for M[f]; thus, M[g] is able to duplicate the computation of ANY Turing machine.

The existence of a universal machine is established here without the elaborate construction of Turing's; clearly, if one desires to do so, one can extract from the above argument a program for a universal machine. In a similar way it is possible to avoid a detailed construction of von Neumann's to create a "self-reproducing" automaton; cf. the discussion of cellular automata in section 4.2. First one notices the possibility of performing operations on machines effectively (indeed, primitive recursively) on the codes of their programs; the most pervasive operation is captured in Kleene's S-m-n-Theorem, that allows to determine the code of a machine for n arguments from a machine for (m+n) arguments, when m arguments are fixed. Then one can establish a fundamental fact, the so-called Recursion or Fixed-Point Theorem; and that in turn allows the proof of the existence of a self-replicating automaton in the following sense: there is an m, such that $g(m,y)=m$ for all y. For details concerning these arguments we have to point to the literature; a very good presentation can be found in [Davis 1958] and [Cutland].

For our purposes some "negative" results are most important. In contrast to Kleene's Enumeration Theorem for partial recursive functions, we can show by a classical "diagonal argument" that the one-place total recursive functions cannot be enumerated by a total two-place recursive function. The argument proceeds as follows: Assume (to obtain a contradiction) that there is a total recursive function g enumerating all one-place total recursive functions f; i.e., for every function f there is a natural number e such that $g(e,x)=f(x)$ for all x. Clearly, as g is (assumed to be) recursive the one-place function f* defined by

(1) $f^*(x) = g(x,x)+1$

is also recursive; thus, as g is an enumeration, we have for f* an e* such that for all x :

(2) $g(e^*,x)=f^*(x)$.

For x equals e* we obtain from (2):

(3) $g(e^*,e^*)=f^*(e^*)$;

and from (1) we obtain by setting x equal to e*:

(4) $f^*(e^*)=g(e^*,e^*) +1$.

Obviously, (3) and (4) imply the contradiction $g(e^*,e^*)=g(e^*,e^*)+1$. Thus we know that an enumeration function for the total recursive functions cannot be recursive. And, similarly, functions enumerating Turing machines that compute total number theoretic functions cannot be calculated by Turing machines.

2.2 REDUCIBILITY. The diagonal method of proof goes back to Cantor who used it to show that the set of real numbers is not enumerable. A modification of the above argument shows that particular questions concerning Turing machines cannot be answered by Turing machines; the most famous question is this: Does the computation of machine M for input x terminate or halt? This is the Halting Problem as formulated by Turing in his 1936 paper; it is clearly a fundamental issue concerning computations. Turing used the unsolvability of this particular problem to establish the unsolvability of related machine problems, e.g. the Self-halting Problem and the Printing Problem. For that purpose Turing made implicit use of a notion of (effective) reducibility; a problem P, identified with a set of natural numbers, is reducible to another problem Q iff there is a recursive function f, such that for all x: P(x) iff Q(f(x)). Thus, if we want to see that x is in P we compute f(x) and test whether that number is in Q! In order to obtain his negative answer to the decision problem Turing reduced (in a most elegant way) the Halting Problem to the Decision Problem; thus, if the Decision Problem were solvable, the Halting Problem would be. This idea of Turing's will re-appear in our discussion of complexity, in particular in the context of Cook's Theorem in section 2.2.

The self-halting problem K is the simplest in an infinite sequence of increasingly complex and clearly undecidable problems, the so-called jumps. First of all notice that for a machine M with code e the set K can be defined arithmetically by the statement "there exists a y, such that T(e,e,y)". K is indeed "complete" for sets A definable by a formula that is obtained from a recursive one by just prefixing one existential quantifier; i.e. any such A is reducible to K. 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.

2.3 INCOMPLETENESS. The above considerations underly the "arithmetic hierarchy" introduced by Kleene and Mostowski in the fifties. But there are sets of natural numbers that are not definable by arithmetic formulas; a particular example is the set of Gödel-numbers of sentences of true arithmetic statements. If that set were arithmetically definable, one could formulate arithmetically the "liar sentence" that expresses its own falsity. This observation of Tarski and Gödel is the cornerstone for proving that any sound formal theory of arithmetic is incomplete: No matter which true statements we choose as axioms and no matter which inferences leading from true statements to true ones we select, there will be true statements that are not provable in the theory. That is correct as long as formality requirements are imposed on the theory, meaning — in mathematical terms and using the identification of formality with recursiveness -- that the set of theorems can be listed by a recursive function or that it is "recursively enumerable". Recursively enumerable sets can be defined by a formula that is obtained from a recursive one by just prefixing one existential quantifier, and thus the sets of theorems of such formal theories are reducible to K . In all of this it has to be assumed that the formal theory is strong enough to allow the calculability of all recursive functions. To summarize: This argument, under the explicit assumptions on the theory, establishes incompleteness.

Gödel, in his 1931 paper, constructed a particular unprovable statement for a theory PM inspired by Russell and Whitehead's "Principia Mathematica", namely, the self-referential statement G expressing that it itself is not provable in PM. This is Gödel's First Incompleteness Theorem for PM; recall that the latter theory was taken not only as a fundamental, but indeed universal theory for all of mathematics. Using an improvement of Gödel's construction due to Rosser, one can formulate a sentence R whose independence from PM is established under the sole assumption of PM's syntactic consistency -- without appealing to its semantic soundness. In this form the First Incompleteness Theorem is taken to refute an underlying assumption of Hilbert's Program, namely that formal theories like PM can capture "completely" mathematical practice. The aim of establishing the consistency of formal theories by restricted mathematical, so-called finitist, means was taken to be unreachable on account of GödePs Second Incompleteness Theorem. This theorem states that the proposition "PM is consistent", when formulated in the language of PM, cannot be proved in PM. To reach a definite verdict on Hilbert's Program, as formulated in the twenties, it has only to be assumed that finitist mathematics can be formalized in PM. Indeed, it is generally assumed that finitist mathematics can already formalized within elementary number theory and coincides, possibly, with a weak fragment of number theory, primitive recursive arithmetic.

In order to formulate the theorems for arbitrary "formal systems" satisfying certain basic requirements Gödel needed a general concept of "formality", and he tried to approach such a general notion in his Princeton lectures of 1934; but it was only Turing's notion that provided it in a convincing way. Given the background for Turing's work on the decision problem and Gödel's work on incompleteness, it is quite clear that they needed as broad a notion as possible. Now we turn our attention to contexts that require more restricted notions of computability.

3 COUNTING THE NUMBER OF STEPS. To re-emphasize, Turing computability is a highly idealized concept, because it disregards limitations on two resources, namely, space and time. The former is disregarded, as the number of tape squares involved in a computation is unbounded; the latter is disregarded, as the number of computation steps is not limited. If we insist that the number of steps be bounded, we automatically insist on a bound for the number of tape squares that can be used in a computation. Yet it seems only too necessary to impose such bounds: Our lifespan is limited, and the size of the physical universe is bounded.

3.0 INTRODUCTION TO TRACTABILITY. Draw an edge between each pair of distinct vertices of a hexagon. The resulting graph is known as the 6-clique and is denoted by K6. Now follow freely your inspiration and color the 15 edges of K6 red or blue. Then, necessarily, there will exist a monochromatic 3-clique, i.e. a triangle with only blue or only red sides. For a proof of this claim, choose your favorite vertex V in K6 and suppose for the moment that there are at least three red edges VA, VB, VC. If the triangle ABC is blue, we are done, otherwise one of the triangles VAB, VAC, or VBC must be red, and we are done. What, if our hypothesis fails? In this case V will be a vertex with at least three blue edges, and a photocopy of the above argument, possibly printed in red ink, yields a monochromatic triangle. Q.E.D.

Who would set out to check all possible 2^15 colorings, now that the existence of a monochromatic triangle is guaranteed by the above argument? In a similar way, it makes little sense to inspect, whether energy is conserved through all stages of the evolution of a physical system, such as a pendulum, once its complete description is achieved by solving a simple differential equation. From the solution one immediately gets energy conservation, just as from the above argument one immediately gets a direct and fast way of finding a monochromatic triangle for each possible coloring of the hexagon. Passing now from six to forty five points and coloring all edges of the 45-clique K45 red or blue, consider the question, whether all such colorings have a monochromatic 5-clique. Experimentally, the answer seems to be affirmative, yet nobody has an argument to exclude the existence of a counterexample -- and nobody is willing to check all possible colorings. This problem originated from a theorem of Ramsey and is just one among many famous problems that can be easily explained to the layman, but that cannot be solved even by the experts of the Government with their latest supercomputers.

Intuitively speaking, a problem is tractable if it can be solved "quickly" — possibly via some instructive shortcut argument using new and sophisticated concepts, such as symmetry, counting, induction, continuity. Such concepts may take care of an enormous number of elementary subcases, just as a single multiplication takes care of many additions. Can the notion of tractability be given a mathematically precise sense in the same way as it was done for the general notion of computability in section one?

3.1 POLYNOMIAL-TIME AND ITS ROBUSTNESS. In a letter to von Neumann of March 20, 1956, Goedel considered a Turing machine T with the property that, for every formula F of the predicate calculus and every natural number n, T decides whether F has a proof of length n. Let TIME(F,n) be the number of steps T requires to decide, whether F has such a proof; let MAXTIME(n) be the maximum of TIME(F,n) as F ranges over all formulas. Goedel asked von Neumann to confirm his belief that MAXTIME(n) is a slow growing function, perhaps growing proportionally to n or to the square of n. Goedel noted that, if his belief were true, then computers could somehow replace human reasoning for yes-no problems -- in spite of the undecidability of the Entscheidungsproblem. Goedel asked von Neumann also about the possibility of fast procedures for deciding such questions as to whether a given number is prime. We don't know about von Neumann's answer to Goedel's letter. Von Neumann was seriously ill at the time and died on February 8,1957. In any event, Goedel adumbrated a quantitative measure for the complexity of solutions to decidable problems and, thus, a measure for their tractability.

The step from decidability to tractability is the birth of complexity theory: Instead of asking whether a problem is algorithmically solvable, attention is focused on the resources needed for a solution. Today we know that MAXTIME(n) is bounded by a polynomial iff P = NP (see below). Goedel's optimism on the growth of MAXTIME(n) was not shared by the Russian cybernetic school: That school investigated, around 1956, the role of exhaustive search ("perebor") and came to the conclusion that for several important problems perebor seems to be inevitable. However, a formal definition of a "universal perebor problem" was introduced by Levin only in 1972, almost at the time of Cook's theorem (see below). Why might we be curious about a theory whose decidability is established, thus making problem-solving in the theory seemingly a matter of mechanical computation and not a matter of human ingenuity? Well, let us consider the theory of addition, i.e. the set of all first-order statements which are true in the additive semigroup of natural numbers. Presburger proved in 1929 that this theory is decidable, and Hilbert and Bernays regarded this result as a significant building block in the foundations of mathematics. However, Fischer and Rabin proved in 1974 that the theory has very high complexity. More precisely, they showed that there exists a real number c > 0 such that for every Turing machine T deciding the theory, there exist true statements s, say of length n, for all sufficiently large n, such that T will recognize their truth only after a number of steps larger than 2^(2^(cn)). (We say that s is of length n if it consists of n occurrences of symbols.) Thus, the theory of addition is quite intractable, despite its decidability. (For further information, see Rabin's chapter in the "Handbook of Mathematical Logic".) It was in 1960, that Rabin made explicit for the first time the notion of computational complexity for functions. Let us turn to the central definitions as they are standard now.

In the following complexity-theoretic definitions a word over a finite set A is simply a finite string of elements of A; A is conveniently regarded as an alphabet, and its elements are the alphabet's symbols. We denote by A* the set of all words. A problem L concerning A* is identified with a subset of A*. Thus, for A = (0, 1,..., 9) the problem of recognizing composite numbers is the set COMPOSITE = (0, 1, 4, 6, ...), and the problem of recognizing prime numbers is the set PRIME = (2, 3, 5, 7, 11, ...). For the alphabet A'

= (0,1, ..., 9, *) the KNAPSACK problem is the set of all strings of A^1 of the form n1*n2*...*nk*m such that it is possible to obtain a total weight of m kilograms by a suitable choice among the weights n1,...,nk. The satisfiability problem SAT is the set of Boolean formulas having a satisfying assignment. We say that a problem L is decidable in polynomial time (for short, L is in P) iff there is a polynomial r and a Turing machine T with the following property: Having as its input an arbitrary word x with n (occurrences of) symbols, T decides membership of x in L within at most r(n) steps.

In the mid-sixties, Cobham noted that many problems can be computed in polynomial time, and he noted also that the notion of polynomial time computability is robust, i.e. it is invariant under many models of computation. Edmonds called an algorithm good whose complexity grows polynomially with the length of the input.
As above, P denotes the class of problems that can be solved by a Turing machine working in polynomial time. The reader who has consulted at least two textbooks on Turing machines may have observed that there is no unanimity about the equipment of Turing machines: some books use quadruples, others use quintuples, or right-infinite tapes, or double tapes, etc... . Fortunately, a simple simulation argument shows that if a problem is solvable in polynomial time by machines in one model, then it is also solvable in polynomial time by machines in any other reasonable model. Technically speaking, the class P is "robust", or textbook-invariant. It would not be as simple to define a robust notion of quadratic, or cubic time.

There is a very long list of problems, coming from all fields where computation is done, for which no polynomial time algorithm is known. The list includes PRIME, KNAPSACK, SAT, and, a fortiori, the decision problem of every theory in first-order logic. A closer examination shows that for every formula F, the task of deciding whether F is in SAT can be split into two sub-tasks:

(i) magically guessing a satisfying assignment for F,
(ii) trivially checking that the guessed assignment satisfies F.

Replacing assignments with choices of weights, with divisors, or more generally with "certificates", one easily sees that (i) and (ii) still hold for KNAPSACK, COMPOSITE, and many other problems. Edmonds called a problem well characterizable iff every solution has a polynomial time checkable certificate.

To enable a Turing machine T to perform subtask (i), we drop the determinacy condition that whenever $q_is_kc_lq_m$ and $q_is_kc_{l''}q_{m''}$ are instructions of T, then $c_lq_m= c_{l''}q_{m''}$; in other words, we allow T to choose non-deterministically its next state. Given input x, the possible configurations of T are conveniently located in a tree: The initial configuration is at the bottom of the tree, and at each branching point there is a configuration C having finitely many possible immediate successors C',C", ... as given by the quadruples of T taking effect in configuration C. T accepts x iff the tree has a path of successive configurations ending with a halting configuration. We say that a problem L is in NP iff there is a polynomial r and a nondeterministic Turing machine T such that for any n and string x of length n, T has an accepting computation of less than r(n) steps, if x is in L, and T has no accepting computation at all, if x is not in L. It follows from the definition that SAT, COMPOSITE, and KNAPSACK are all in NP. In 1975 Pratt observed that PRIME is in NP because of the following well-known number-theoretic result: n is prime iff for some a, a^(n-1) is congruent to 1 modulo n, and a^(n-1)/q is not congruent to 1 modulo n for each prime divisor of n-1.

3.2 REDUCTIONS AND NP-COMPLETE PROBLEMS.. By analogy to section 2.2 we now introduce the appropriate dynamic machinery that enables us to pass from one problem to another. Suppose we need a prime number recognizer, but the best we can get is a machine R that recognizes successors of prime numbers: Thus, over input 3,4,6,8,12,... R outputs a green light, while over input 0,1,2,5,9,... R outputs a red light. Still, we can efficiently use R to recognize, whether a number y is prime by transforming y into y' = y+1 (here is the dynamics) and then looking at the output of R over input y'. Needless to say, y will be prime iff the output is green. The transformation from y to y' quickly reduces the problem of prime number recognition to prime successor recognition.

The following definition is due to Karp: A subset L of A* (i.e. a problem L over alphabet A) is polynomially reducible to a subset M of B* iff there exists a polynomial r and a deterministic Turing machine T such that, for every string x of A* of length n, T outputs a string x' of B* in not more than r(n) steps with the property that x is in L iff x' is in M. When problem L is polynomially reducible to problem M we naturally regard M as being at least as difficult as L. In the class NP there exist problems M that are maximally difficult in the sense that every problem in NP is polynomially reducible to M. These problems are called NP-complete, and are currently thought to be intractable by the majority of computer scientists. In this sense, a problem in P is regarded as not as intractable as an NP-complete problem -- but see our remarks below.

Cook proved in 1971 the NP-completeness of SAT as follows: Let L be a problem in NP, let r be a polynomial, and let T be a nondeterministic Turing machine recognizing, whether a string s of length n belongs to L in at most d = r(n) steps. We must quickly transform s into a Boolean formula which is satisfiable iff s is in L. Since T can visit at most d consecutive squares to the right or to the left of the scanned square, we may safely use a tape with just 2d+1
squares. Since the alphabet of T and the number of states of T are both finite, a suitably chosen Boolean formula B1 can express that T is in its initial state, that its scanner is placed over the central square of the tape containing the first symbol of the input, and that the tape contains only the input surrounded by blank squares. There is no difficulty in writing down a Boolean formula B2 saying that in the t-th configuration, t = 0, 1,...,d, T is in a particular state, its scanner is placed over a particular square, and each square contains a particular symbol. Satisfying assignments of B2 are in one-one correspondence with sequences C0, C1,...,Cd of configurations. It remains to be ensured that each Ct+1 follows from Ct in accord with the rules imposed by the quadruples of T. For this purpose we let a Boolean formula B3 say that if the scanner of T at time t is on the x-th square, x = 0, 1,..., 2d, then the remaining squares will maintain their symbols at time t+1, while the symbol on the x-th square at time t+1 (or the next move of the scanner of T) and T's state will be as prescribed by the quadruples of T. Finally, we let Boolean formula B4 say that T is in the halting state at time d. Now it is straightforward to show that satisfying assignments for the Boolean formula S given by B1 & B2 & B3 & B4 are in one-one correspondence with accepting computations C0, ..., Cd for s. The reduction of s to S can be performed by a deterministic Turing machine working in polynomial time. This yields the required fast reduction of L to SAT. Q.E.D.

KNAPSACK and many further problems were shown by Karp and others, using polynomial reductions, to be NP-complete. Today the list of NP-complete problems contains hundreds of examples from such diverse areas as logic, combinatorial optimization, number theory, cryptography, algebra, graph theory (see Garey and Johnson). Although all NP-complete problems are polynomially reducible to one another, the fact that their progenitor was the SAT problem is worth mentioning: Indeed, short Boolean formulas describe short non-deterministic computations much more immediately than, say, KNAPSACK weights, just as arithmetic formulas immediately describe Turing computations. Furthermore, although finite Boolean algebras are trivial, their presentations in terms of short formulas in the propositional calculus, or switching circuits, or binary codes, are a source of many important problems having deep relations with various areas of mathematics. To quote Garrett Birkhoff in his article "The Role of Modern Algebra in Computing" (SIAM-AMS Proceedings, "Computers in Algebra and Number Theory", Volume IV 1971, page 5): "...I regarded finite Boolean algebras as trivial because they could all be described up to isomorphism, and completely ignored the basic "shortest form" and "optimal packing" problems described above. "

3.3 RELATIVE INTRACTABILITY. The analogy recursive/r.e. = P/NP is very suggestive, and in several cases recursion-theoretic results have found their complexity-theoretic counterparts. Thus, one can find in particular the analogue of the Kleene-Mostowski hierarchy. However, while we saw that there are r.e. problems that are not recursive, it is not known, whether there are NP problems which are not in P. I.e., the main problem in Goedel's letter to von Neumann is still unsolved. A simple simulation argument shows that every problem in NP can be solved by a deterministic Turing machine working in exponential time. More generally, for every language L virtually the same simulation argument would also hold in the "L-relativized" situation, when Turing machines are allowed to consult an oracle that, in one step, recognizes whether a string is in L. Thus, if a simulation argument were able to prove, say, P = NP, then with the same proof one could also settle the L-relativized version of the problem. Baker, Gill and Solovay found in 1975 two recursive languages A and B yielding opposite answers to their respective relativized versions of the P/NP problem---thus showing the inadequacy of simulation arguments to solve the P/NP problem.

One of the major consequences of Cook's theorem is that, if a polynomial time algorithm were found for just one NP-complete problem, then all problems in NP would be solvable in polynomial time. One might hope that the borderline between tractability and intractability would be sharp in the following sense:
(i) If a problem is tractable, then sooner or later a polynomial time algorithm for it will be found.
(ii) If a problem is intractable, then a proof of intractability will be found.
Unfortunately, the following may also happen:
(iii) The problem can be shown to have a polynomial time decision procedure, but, due to the non-constructive character of the proof of

tractability, nobody has an idea of how to construct a polynomial algorithm for it.
(iv) The existence of a polynomial time algorithm might depend, in a rather embarrassing way, on our adherence to this or that set-theoretic school. (And conversely, as Goedel hinted at, our set-theoretic beliefs might depend on which algorithms we wish to homologate as tractable).
A "for instance" of (iv) is this: Let F be an arbitrary class of graphs G with loops and multiedges and let F have the following property: whenever G is in F and G^f is obtainable from a subgraph of G by a sequence of edge contractions, then G^f is also in F. Robertson and Seymour proved that membership in F is decidable in polynomial time. Their proof, however, is irremediably nonconstructive and gives no hint whatsoever for the construction of a polynomial time algorithm. Similarly, Miller proved in 1976 that a polynomial time decision method for primality exists, provided a certain extended form of Riemann's hypothesis is true - this hypothesis being a generalization of a well-known classical open problem. Thus, one of the problems raised in Goedel's letter to von Neumann depends on a fundamental problem, to which Riemann, Hilbert, and many other mathematicians devoted their best efforts without solving it.

Having been incorporated into the large body of mathematics under the name of "polynomial time computability" the notion of tractability is acted upon by the dynamic elements streaming through the mathematical body. One can clearly investigate different kinds of resources, other than Turing time, and different measures of input complexity. What might be highly interesting is to switch from the "worst case" scenario adopted throughout this subchapter and consider instead the "average case" complexity of a problem -- and try to explain rigorously, why many exponential time algorithms work so well in practice. Alternatively, it is interesting to study algorithms that toss coins during their execution: This turns out to be very efficient for prime number recognition. In the worst case scenario, a randomized algorithm makes it difficult for the malicious adversary to choose instances of maximal difficulty. But whether a mathematical concept can indeed capture all the crucial features of "feasibility" remains to be seen: Conceptual analysis followed by mathematical development and computational experience will be crucial ingredients for an informed judgement.

4 PHYSICAL LIMITS OF STEPS. We discussed limits of computations in the logical sense first and were concerned with the

Incompleteness Theorems and the unsolvability of the Entscheidungsproblem. Then we described limits of computations in the sense of feasibility w.r.t. particular machine models and insisted on space and time limitations. Now we want to give "flesh" to the abstract machines and ask: What are general physical constraints on computational devices? We begin, however, with some remarks about undecidability and unpredictability.

4.0 INCOMPLETENESS (revisited). As shown by Gödel's First Incompleteness Theorem, most mathematically interesting theories contain undecidable statements. And, as the theory of addition shows, even decidable theories may contain practically undecidable statements. In the prototypical case of Peano arithmetic the existence of undecidable sentences was regarded for many years as irrelevant to the working mathematician. Then, in 1977, Paris and Harrington (see their chapter in the "Handbook of Mathematical Logic") exhibited an undecidable sentence in Peano arithmetic that has direct significance, at least, for the mathematician working in Ramsey theory. Their sentence can be promptly decided in, say, Zermelo-Fraenkel set theory with the axiom of choice (ZFC) -- the first-order theory most frequently adopted as the official foundation for the whole edifice of mathematics. However, ZFC is no less incomplete than Peano arithmetic: For instance, as shown by Gödel and Cohen (see the chapters by Devlin and Burgess in the "Handbook of Mathematical Logic"), one cannot settle in ZFC the long-standing conjecture that there is no set whose cardinality lies strictly between the cardinality of the continuum and that of the set of natural numbers.

Even the founders of set theory were not unanimous on the problem, whether the evolution of set theoretic relativism would follow the lines of the evolution of geometry after the discovery of the independence of the parallel postulate. For instance, consider the following concluding remarks in SkolenrTs 1922 paper "Einige Bemerkungen zur axiomatischen Begründung der Mengenlehre": "The most important result above is that set theoretic notions are relative. I had already communicated it orally to F. Bernstein in Göttingen in the winter of 1915-16. There are two reasons why I have not published anything about it until now: first, I have in the meantime been occupied with other problems; second, I believed that it was so clear that axiomatization in terms of sets was not a satisfactory ultimate foundation of mathematics that mathematicians would, for the most part, not be very much concerned with it. But in recent times I have seen to my surprise that so many mathematicians think that these axioms of set theory provide the ideal foundation for mathematics; therefore it seemed to me that the time had come to publish a critique."

This Concluding Remark of Skolem's was preceded by the question: "What does it mean for a set to exist if it can perhaps never be defined?" to which Skolem gave a most interesting answer. "It seems clear that this existence can be only a manner of speaking, which can lead only to purely formal propositions - perhaps made up of very beautiful words -- about objects called sets. But most mathematicians want mathematics to deal, ultimately, with performable computing operations ..." And there are developments in mathematics and proof theory (traceable to some of Skolem's work, contemporaneous work of H. Weyl in "Das Kontinuum", and a long constructivist tradition within mathematics) that develop the scientifically applicable parts of set theoretic mathematics in weak formal theories. And these theories are very much motivated by computational concerns; cf. [Feferman].

It is possible that from a long process of natural selection Euclidean-like set theoretic primates will emerge. But, perhaps, the first-order treatment of sets rather parallels the Hamiltonian treatment of the forced pendulum, where the official a-priori determinism is made ineffective by a teeming microcosm of capricious details. If that were so, the incompleteness of ZFC -- in the framework of a complete logic -- may be ascribed to uncontrollable, uninteresting details in our set theoretic stipulations/observations, just as unpredictability of a forced pendulum -- in the framework of deterministic Hamiltonian mechanics -- is caused by uncontrollable, uninteresting microscopic perturbations in the preparation/measurement of the system. A forced pendulum is a small sphere attached to the bottom end of a string; the sphere may oscillate in any direction, while the top end of the string is forced to oscillate along a horizontal line under the action of a crank shaft. Assume that the forcing frequency is a little higher than the natural frequency of the pendulum. Initially, the sphere will oscillate in parallel to the forcing oscillations, but then a perpendicular component of motion appears. Eventually, the motion becomes stationary along a circle with the period of the rotation being equal to the frequency of the forcing oscillation. But however precise we try to make our knowledge of the initial conditions of the forced pendulum, and notwithstanding the deterministic character of the Hamiltonian theory describing the forced pendulum, we cannot predict, whether it will eventually rotate clockwise or counterclockwise.

Unpredictability follows from the fact that a multitude of infinitesimal perturbations in the initial conditions causes macroscopic bifurcations already in the short-term evolution of the system. Bifurcation entails a sort of incompleteness of the underlying deterministic theory, at least concerning the problem of inferring from the initial conditions a precise truth value for the proposition "The system shall eventually rotate clockwise". The best technology can't help the Hamiltonian mountain to bring forth the appropriate clockwise or counterclockwise mouse. The reader will recall that Plato and Galileo had different views on the readability of the Book of Nature: The former believed that the plethora of accidental perturbations affecting the physical world and our perception of it would not allow any experimental physical theory at all. The latter regarded the Book of Nature as written in terms of triangles and circles in such a way that we can read it "provando e riprovando" (by repeated experiment).

A typical reaction to incompleteness phenomena in mechanics is to regard them as unphysical -- just as undecidable sentences in mathematics are sometimes regarded as not genuinely mathematical. This attitude is made explicit in quantum statistical mechanics by the slogan "Nature does not have ideals": Whenever a physical system is described by an algebra A of operators, A should have no nontrivial quotient structure, i.e., A should have no nonzero ideal I. As a matter of fact, whenever A has such an ideal, the quotient structure A/I and not A describes the relevant physical system. Now, approximately finite-dimensional C*-algebras A describing the thermodynamic limit process in quantum statistical mechanics turn out to be presentable as sets of axioms in the infinite-valued calculus of Lukasiewicz. Such an A is called "Gödel incomplete" just in case there is a presentation of A as a theory whose set of theorems is recursively enumerable, but not recursive. However, it can be proved that every Gödel incomplete C*-algebra A must have a nontrivial ideal; see [Mundici 1986]. Thus, if Nature does not have ideals, then no natural system can be presented by a recursive set of axioms whose set of consequences is not recursive. A fortiori, Nature must abhor presentations which are essentially Gödel incomplete; for in this case any attempt to complete our knowledge of the system by eliminating all quotient structures will irreparably destroy the original effective presentability of the system.

As for the additive-multiplicative system of natural numbers in Peano's axiomatization, or for the ZFC axiomatization of set theory, the incompleteness of any quantum statistical system will show the incompatibility of two desiderata, namely:
(i) completeness of the information available within the formalization, and

(ii) effective computability of (the consequences of) this information.
It is, perhaps, of interest in this context to mention that the undecidability of the halting problem has been used by da Costa and Doria to show the undecidablity of problems in classical mechanics (and the incompleteness of suitable axiomatizations of the theory). But here we are not so much interested in what computation theory can do for (axiomatizations of) mechanics, but rather what physics has to tell us about limitations of computing devices; and it is to that topic that we turn now.

4.1 ONE PHYSICAL STEP. Recall that Turing appealed in his analysis to the limitations of the sensory apparatus of human computors; however, he claimed that the justification for his (central) 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 spacial limitations for "encoding" finite configurations (discussed below)? If such limitations also hold also for computing devices, are there ways of getting around their effect, e.g. by speeding up operations or by using more complex ones?

Here is the description of a "concrete" Gedanken Turing machine T deciding in less than 60 minutes, whether there exists an exceptional coloring of K45, i.e. a red-blue coloring having no monochromatic 5-cliques:
(1) T systematically tries all possible n=$2^{A}990$ colorings of K45, unless an exceptional coloring is found, in which case T rings a bell and stops;
(2) It is easy to write down the instructions for T in such a way that for each individual coloring, T decides in less than half an hour, whether or not the coloring has a monochromatic 5-clique: the number of 5-cliques to check is (45 binomial 5)=1221759 -- a relatively small number;
(3) Having thus taken care of the software, we accelerate the tape of T, as is frequently done in comic movies, in such a way that the second coloring is checked in (1/4)-th, the third in **(1/8)-th**. . . . .the n-th in 1/(2^{A}n)-th of an hour;
(4) In this way T solves the problem in less than 60 minutes, as required.

What is wrong with this T? Does relativity theory impose an insurmountable upper bound on the number of steps T can perform in one second? Does quantum thermodynamics impose a lower bound on the amount of heat being produced by T during the computation? We can argue as follows: Suppose T is a physical Turing machine satisfying the following two conditions:

(i) time is not recycled, i.e. no portion of the time used for one step can be used for another step -- this is just a reformulation of the sequential behavior of Turing machines;
(ii) energy is not recycled, i.e. no portion of the energy used for one step can be used for another step.
Then, if f is the number of steps performed by T in one second (T's frequency) and if W is the power used by T measured in watt (power = energy per second), T will obey the inequality

f^2 <= (2 Greekpi W)/h,

where h is Planck's constant; that means the power absorbed by T grows at least as fast as the square of its frequency. The argument for this inequality is roughly this (for details, cf. [Mundici 1981]): The portion of the tape that is scanned during a computation step undergoes a noticeable modification of its physical properties. Hence, by the Heisenberg inequality, the energy uncertainty (delta E) of the tape square must be greater than h/(2 Greekpi delta t), where (delta t)=1/f is the time needed for the step. A fortiori, the energy used for the step must be greater than (delta E), and the inequality immediately follows from (i) and (ii).

Albeit small, the multiplication constant h has an effect, and the quadratic lower bound for W can be used as a convincing argument for the unfeasibility of (3) for Turing machines. Of course, one might argue that parallel computers are able to circumvent condition (i), and that a carefully designed Turing machine could, at least partially, circumvent condition (ii). More radically, one might argue that Heisenberg's uncertainty principle need not imply that any amount of energy is "used" for a computation step. In any case, no real computer has so far violated the above lower bound. (There is a large literature on the subject of physical limitations of the computing process. We refer the interested reader to volume 21 of the International Journal of Theoretical Physics, published in 1982; it is entirely devoted to the physics of computation.) In the next subsection we want to explore, how space-time features of computations are physically constrained, and how such constraints prevent us from having "arbitrarily" complex operations.

4.2 ONE PHYSICAL REGION. In the above analysis of steps, there is no allusion to the details of the (physical) construction of Turing machines -- except that each step requires an interaction between the square being scanned and the scanning head. Recall from section 1.3 that the restricted formulation of Turing-machines achieves a mathematically uniform and simple description of computations and recall further that its adequacy is guaranteed by Turing's Theorem. The starting-point of the analysis was, however, the "mechanical behavior" of a human computor operating on finite configurations. This behavior can be described directly and mathematically precisely.

But first we take a preliminary step and replace the states of mind by a "physical and definite counterpart". This is done by considering "states of mind" not as a property of the working computor, but rather as part of the configuration on which he operates. Turing discusses this replacement very vividly in section 9, III, of his classical paper "On computable numbers": "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." Mathematically that is done quite beautifully in [Davis 1958]: instantaneous descriptions of machines, that means finite sequences in the alphabet of a Turing machine, contain exactly one state symbol that indicates also by its position in the sequence which symbol is being scanned; programs of Turing machines can then be viewed as a set of Post production rules operating on (a single symbol of) such instantaneous descriptions.

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 Uspensky Machine". This latter notion, or rather a general concept of algorithm, was introduced by Kolmogorov and Uspensky in 1958; for an informative discussion see [Uspensky 1992]. As it turns out KUMs compute exactly the Turing computable number theoretic functions, i.e. the partial recursive functions. -- The focus on a fixed finite number of distinguished graphs can be motivated by thinking again of a human computor operating according to rules; and with this understanding, the KUMs provide what we called above a mathematically direct and precise description of Turing's computor. But the restriction can also be motivated by physical considerations; let us look at the underlying relativistic limitations, following (Mundici [1981], pp.302 ff, Mundici [1983], pp. 43 ff, and Sieg [1993], section 3.3 for the connection to Turing's claim that memory limitations provide the justification for the finiteness conditions).

Assume a machine to operate on configurations containing z different "symbols", each symbol being physically represented or encoded by at least one atom -- which is an altogether reasonable assumption. Then there must be at least z pairwise disjoint regions containing the codes (for the symbols). Otherwise, the electron clouds of two different codes might overlap, making the codes indistinguishable and leading the machine to "mental confusion". Let c and a respectively denote the speed of light and Bohr's radius of the hydrogen atom, where a/c = 1/5.655 x 10^(-18) seconds. It follows that the codes will be contained in a volume V of at least z (4/3) Pi a^3 cubic meters; that forces the diameter 2r to be larger than 2 a z^(1/3) meters -- the diameter being the largest possible distance between two codes in this volume. Again denoting by f the frequency of our machine, and noting that 1/f is the time available for each computation step, since signals cannot travel faster than light and a computation step involves the whole configuration, it follows that f cannot exceed c/(2 a z^(1/3)) steps per second, whence the product of f and z^1/3 must be smaller than (1/2) x 5.655 x 10^18, which points out a fundamental incompatibility between high number of codes (thus size of configurations) and high computational speed.

The operations of KUMs are thus necessarily restricted in complexity, as they have to lead from distinguished graphs to distinguished graphs; and within the given physical boundaries only finitely many different ones are realizable. One alternative to speeding up computations was mentioned already: parallel operations. Computing devices that operate in parallel are cellular automata introduced by Ulam and von Neumann. The latter used these devices to construct (complicated) self-replicating machines. A particular cellular automaton was made popular by Conway, the so-called game of life.
A cellular automaton is made up of many identical cells. Typically, each cell is located on a regular grid in the plane and carries one of two possible values, say, 0 or 1; after each time unit, its values are updated according to a simple rule, depending only on the previous value and the previous values of neighboring cells. Cellular automata of this sort can simulate universal Turing machines, but they also yield discrete simulations of very general and complex physical processes. It should be noted that cellular automata do not satisfy the finiteness axioms for Turing's computor. The reason is that computation steps may operate on

unbounded regions of the plane. But that does not mean that cellular automata cannot be simulated by Turing machines: Indeed, they can be!

Another interesting model of parallel computation is provided by Boolean circuits. The very practical justification for this model is that the building blocks of most real machines are Boolean circuits. In their simplest form Boolean circuits are given by formulas of the propositional calculus. A Boolean formula F of q variables outputs 1 or 0 according to whether the assignment b1,...,bq satisfies F or not. The most general formulas contain also other connectives, such as "iff". Furthermore, while the output value of a subformula other than a variable may serve as the input for exactly one (larger) subformula, the output of a subcircuit may serve as the input for many larger subcircuits simultaneously. The only requirement is that there are no loops. In this way the input information b1,...,bq is processed in parallel by many subcircuits, each sending their outputs to other subcircuits, and finally we obtain the output.
In contrast to Turing machines, which can handle inputs of unbounded length, a Boolean circuit accepts as its input only sequences of q bits. There are $2^A q$ such sequences and $2^A(2^A q)$ 0-1-valued functions of q variables - most of them being irreducible, in the sense that the best Boolean circuits computing the function are not very different from the (trivial) listing of the values of the function over each possible input. On the other hand, there do exist complex Boolean functions that can be represented by short circuits. It is a very interesting problem to write down the shortest possible circuits ~ recall Birkhoff's quotation. Curiously enough, nobody knows the shortest circuits for addition and multiplication. In fact, very little is known about the computing power of Boolean circuits.

How can parallelism be captured in a general mathematical way, not restricted to the simple pattern of cellular automata, but clearly encompassing them? Robin Gandy provided for the first time in his [1980] a 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 the above discussion of relativistic limitations. As to such "mechanical devices" Gandy suggested that "the reader may like to imagine some glorious contraption of gleaming brass and polished mahogany, or he may choose to inspect the parts of Babbage^{f}s 'Analytical Engine^1 which are preserved in the Science Museum at South Kensington". And, to give the above "i.e. remark" in Gandy's language, "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 is also computable by a Turing machine; this is a marvel of analysis (though not of exposition). The definitional preliminaries are lengthy; John Sheperdson (1988) wrote: "Although Gandy's principles were obtained by a very natural analysis of Turing's argument they turned out to be rather complicated, involving many subsidiary definitions in their statement. In following Gandy's argument, however, one is led to the conclusion that that is in the nature of the situation ..."

We want to give an informal description that skirts the lengthy definitional preliminaries, focusing rather on the intuitive considerations that underly the mathematical formulations. The configurations on which a Gandy machine GM operates are taken to be hereditarily finite sets over some (potentially infinite) set A of urelements as labels, HF(A). The configurations must satisfy two boundedness conditions: the first, called Limitation of Hierarchy, expresses that all configurations of a given GM must be in an initial segment of the cumulative hierarchy HF(A) containing only sets of rank less than a fixed natural number; the second, called Unique Reassembly, requires that all configurations can be uniquely reassembled from parts of bounded size. The third and central condition, called the Principle of Local Causation, governs the transition from one configuration Cn to the next one Cn+1: Cn can be reassembled from parts of bounded size that fit into a fixed finite number of isomorphism types; Gm operates on these parts in parallel and assembles Cn+1 from locally computed parts. It is here that the relativistic limitation on the speed of light comes in and forces the restriction to parts of bounded size on which the computation is carried out locally.

The succesive states of cellular automata (and the succesive configurations of Gandy machines) can be computed by suitable Turing machines; but how complex is this "serialization"? This is a most interesting question, as the cellular automata are particular kinds of dynamical systems that are used to simulate physical processes! Indeed, Fredkin has been advocating the use of (reversible) cellular automata in physics for some time. In his more recent "Digital Mechanics" he conjectures "that there will be found a single cellular automaton rule that models all of microscopic physics; and models it exactly." The interested reader should delve into the paper by Richard Feynman, "Simulating physics with computers", published in the International Journal of Theoretical Physics in 1982. (See the "Quaderno" of Scientific American quoted in the bibliography, but also [Herken 1988].)

4.3 DYNAMICAL SYSTEMS: irreducible simulation? Ever since the introduction of mathematical models for the purpose of capturing aspects of reality, their computational-predictive features have been absolutely crucial. What is distinctive about the modern developments is this: Computer simulations have led to an emphasis of the algorithmic aspect of scientific laws and, conversely, physical systems are being considered as computational devices that process information much as computers. Let us compare the forced pendulum (resp., the classical pendulum) with a Gedanken-45-clique (resp., with a 6-clique), whose red-blue coloring is assumed to vary with time. Just as a classical exercise in Hamiltonian mechanics immediately yields conservation of energy for the pendulum, similarly the combinatorial argument given at the beginning of section 2.0 yields the conservation of a monochromatic 3-clique in the red-blue 6-clique. By contrast, even the most powerful computer will not be able to decide whether a given forced pendulum will eventually rotate clockwise, or whether a variable red-blue 45-clique will always have a monochromatic 5-clique.

Computational intractability stems, perhaps, from the impossibility of conceptually handling (symbolizing) a virtually infinite amount of relevant information, namely the list of digits of the real numbers measuring the initial conditions of the pendulum, or the list of red and blue edges in all possible colorings of the 45-clique. Owing to this virtual impossibility, the forced pendulum, as well as the variable red-blue 45-clique remain their own best simulators. When no shortcuts are available, and the computer attains the highest degree of resemblance with the simulated system, we regard the system (or, equivalently, the simulation) as being irreducible. Although in some fortunate cases a mathematical theorem can completely describe the evolution of a cellular automaton, it follows from the above discussion that, in general, an automaton is its own best simulator.

The Turing machine model is not well suited to simulations of irreducible systems, as most physical processes seem to correspond to parallel computations. Similarly, if the difficulty of a theory is witnessed by sentences stating their own intractability by sequential computation, we can't hope Turing machines to efficiently handle the decision problem of that theory. Even non-deterministic Turing machines are unsuitable, because they require too much parallelism - for the search of an exceptional coloring of the 45-clique we would replace the above Turing machine T by 2^A990 Turing machines all working in parallel, each checking a different coloring. The setting up of paradigms for parallel computing is expected to afford a more efficient handling, not only of simulations of physical processes, but also of combinatorial and decision problems. It seems, ironically, that our mathematical inquiry into "paper machines" has led us to a point where (effective) mathematical descriptions of nature and (natural) computations for mathematical problems coincide.

BIBLIOGRAPHY

J. Barwise (ed.), Handbook of Mathematical Logic, North-Holland, Amsterdam, 1977.

N.C.A. da Costa, F.A.Doria, A partial answer to Arnol'd's 1974 Hilbert Symposium problems, In: Proceedings of the 3rd Goedel Colloquium in Brno, August 24-27, 1993, A. Leitsch et al., Editors, Springer Lecture Notes in Computer Science, 1994, to appear.

M. Davis (ed.), The Undecidable; Raven Press, Hewlett (New York), 1965.

M. Davis, Computability and Unsolvability; McGraw-Hill, New York, 1958

S. Feferman, Why a little goes a long way: Logical foundations of scientifically applicable mathematics; to appear in: Philosophy of Science (1993).

Edward Fredkin, Digital mechanics; Physica D 45 (1990), 254-270

R. Gandy, Church's Thesis and Principles for Mechanisms; in: The Kleene Symposium (J. Barwise, H.J. Keisler, and K. Kunen, eds.), North-Holland, Amsterdam, 1980, 123-148.

M.R.Garey, D.S.Johnson, Computers and Intractability, W.H.Freeman, San Francisco, 1979.

R. Herken (ed.), The Universal Turing Machine (A half-century survey), Oxford University Press, 1988.

D. Mundici, Irreversibility, Uncertainty, Relativity and Computer

Limitations, Il Nuovo Cimento, Europhysics Journal, 61 B, n.2 (1981) pp. 297-305.

D.Mundici, Interpretation of AF C*-algebras in Lukasiewicz sentential calculus, Journal of Functional Analysis, 65 (1986) pp. 15-63.

D. Mundici, Natural limitations of decision procedures for arithmetic with bounded quantifiers, Archiv fuer mathematische Logik und Grundlagenforschung 23 (1983), 37-54.

D. Mundici (ed.), Le Scienze Quademi (Italian Edition of Scientific American), Vol. 56 "La Scienza dei Calcolatori", October 1990.

W. Sieg, Mechanical Procedures and Mathematical Experience; to appear in: Mathematics and Mind (A. George, ed.), Oxford University Press, 1993.

T. Toffoli, N.Margoulis, Cellular Automata Machines, MIT Press, Cambridge, Massachusetts, 1988.

V.A. Uspensky, Kolmogorov and Mathematical Logic; Journal of Symbolic Logic, 37 (2) (1992), 385-412.

J. van Heijenoort (ed.), From Frege to Goedel; Harvard University Press, Cambridge, 1967

J. von Neumann, The Computer and the Brain; Yale University Press, 1958.