COMPUTING MACHINES

Entry for the second edition

of the Encylopedia of Philosophy

Wilfried Sieg and Rosella Lupacchini

July 20, 2005

Technical Report No. CMU-PHIL-173

Philosophy

Methodology

Logic

Carnegie Mellon

Pittsburgh, Pennsylvania 15213

COMPUTING MACHINES

Wilfried Sieg and Rossella Lupacchini

Introduction. Any thorough discussion of computing machines requires the examination of rigorous concepts of computation and is facilitated by the distinction between *mathematical, symbolic* and *physical* computations. The delicate connection between the three kinds of computations and the underlying questions, "What are machines?" and "When are they computing?", motivate an extensive theoretical and historical discussion. The relevant outcome of this discussion is formulated at the very beginning of section 3 that is entitled *Physical realization.*

The paradigm of the first kind of computation is given when a human calculator determines, by finitely many and mathematically meaningful steps, the values of number-theoretic functions for particular arguments. The informal concept of such effectively calculable functions is thought to be captured by Gödel's concept of general recursive functions. The latter notion was introduced in 1934 and arose in an intellectual context that includes the contemporaneous development of Hilbert's Program as well as earlier steps towards modern logic and abstract mathematics.

Turing and Post initiated in 1936 a shift from mathematically meaningful steps to basic, not further analyzable ones that underlie mathematical computations. They investigated symbolic processes carried out by human calculators and proposed essentially the same model of symbolic computation that is mathematically presented now by a *Turing machine.* Turing took, however, an additional, most significant step: he devised a *universal machine* that can execute the program of any Turing machine, and he had it carry out the necessary symbolic operations. This construction allowed him to prove the effective unsolvability of the halting and decision problems.

The physical details of how a universal machine could actually be constructed did not matter for Turing's theoretical investigations in 1936, but

obviously did when he was involved in designing and building an Automatic Computing Engine (ACE). In modern digital computers controlled physical processes are used to realize, efficiently, the stepwise operations of a universal machine. That seems to be true even for quantum computing. In analog computing physical processes are used in a different way. However, quite independently of the mode of the physical computation, the question can be raised, whether there are physical processes that are carried out by a "computing machine", but do not fall under Turing computations.

1. Mathematical computations. Human calculating provides a rich prehistory for the development of machines that can take over routine computational tasks. This prehistory points to the pervasive impact computing machines will develop: from the broadly intellectual and social/economic to the highly focused scientific. Before coming to the technological challenge of building machines that mimic processes on symbolic configurations, we have to address the problem of determining the nature of such processes and those aspects that are crucial for their machine implementation. After all, physical representations of the symbolic configurations are needed, and machines have to perform on them physical operations that correspond to the symbolic ones.

1.1. Prehistory. In the 16th and 17th century Schickard, Pascal, Leibniz and others constructed mechanical calculators to carry out basic arithmetical operations. The calculatory roots go back, however, not just to ancient Greece, but even to Egypt and Mesopotamia; important developments took place also in China, India and in many different parts of the world under Arab influence. It is no accident for the evolution of computing that *algebra* and *algorithm* etymologically come from the same Arab source: the title of a widely used book and the agnomen of its author (al-Kwarizmi).

The construction of mechanical calculators is prima facie narrower than the development of other scientific tools and yet it was pursued as having, potentially, a much broader impact through the intimate connection of computing with mathematics and logic. That was clearly sensed and expressed

with great expectations by Leibniz. Of course, there had been aids to computation in the form of neatly arranged configurations of pebbles, for example. Another quite efficient aid had been the Chinese abacus that allows, via a good representation of natural numbers, the human calculator to add, subtract, multiply and divide through strictly local manipulations of beads. The configurations of the abacus serve as the representation of input, intermediate results and output of the calculation; they are essentially aids to memory.

The difference between abacus-like devices and mechanical calculators (as developed by Schickard, Pascal, and Leibniz) is formulated in an illuminating way by Babbage:

> Calculating machines comprise various pieces of mechanism for assisting the human mind in executing the operations of arithmetic. Some few of these perform the whole operation without any mental attention when once the given numbers have been put into the machine.
>
> Others require a moderate portion of mental attention: these latter are generally of much simpler construction than the former, and it may also be added, are less useful. (*Babbage 1864*, p.30)

The abacus certainly requires a moderate portion of mental attention, whereas Babbage's difference engine is perfectly in line with the development of automatic computing machines. The difference engine was intended to determine the values of polynomials for given arguments by the method of finite differences; the results were to be printed by the machine and create reliable tables useful for astronomy as well as for navigation. The evolution of the difference engine brought to light the economic importance of computing and the consequent governmental support of related research. (The British government sponsored Babbage's work; the Swedish government supported the work of Scheutz who was inspired by a description of Babbage's machine and constructed a difference engine in 1834; an improved version was built between 1851 and 1853 with funds from the Swedish Academy.)

Babbage took later another important conceptual step when developing his analytical engine. He followed the lead of Jacquard who had used "cards with holes" as a means of programming a loom to weave intricate patterns. Babbage devised, but never fully constructed, a programmable computing machine with a rather modern organization. In Chapter VIII of *Passages from the*

Life of a Philosopher, Babbage writes after having described the process of the Jacquard loom:

> The analogy of the Analytical Engine with this well-known process is nearly perfect. The Analytical Engine consists of two parts:
> 1. The store in which all the variables to be operated upon, as well as all those quantities which have arisen from the result of other operations, are placed.
> 2. The mill into which the quantities about to be operated upon are always brought.
> Every formula which the Analytical Engine can be required to compute consists of certain algebraical operations to be performed upon given letters, and of certain other modifications depending on the numerical value assigned to those letters. (p. 89)

Evidently, store corresponds to the memory and mill to the central processing unit of a contemporary computer. The programming constructs in Babbage's design are of such a general character that Gandy asserted, the number theoretic functions that are Babbage calculable are precisely those which are Turing computable.

The generality of computational issues, beyond their connection with arithmetic and analysis, is emphasized through the algebraic treatment of logic in the hands of Boole, deMorgan, Peirce and Schröder, among others. In this line of research the decision problem was formulated and considered as a central issue. Even in the traditional Aristotelian presentation of logic computational features were considered to be significant by Lullus and, very importantly, by Leibniz in his project of constructing a universal language and an appropriate *calculus ratiocinator*. A logical machine in that tradition was built by Jevons and described in the *Proceedings of the Royal Society* for January 20, 1870. Finally, it should be mentioned, that Frege claimed in the introduction to *Grundgesetze der Arithmetik* (1893) that in his logical system "inference is conducted like a calculation", but continued: "I do not mean this in a narrow sense, as if it were subject to an algorithm the same as ... ordinary addition and multiplication, but only in the sense that there is an algorithm at all, i.e., a totality of rules which governs the transition from one sentence or from two sentences to a new one in such a way that nothing happens except in conformity with these rules."

Within mathematics at that time, Kronecker insisted on the decidability of mathematical notions and the calculability of functions. These logical and mathematical developments were joined in formal mathematics, when Hilbert

exploited the effective meta-mathematical description of formal theories (in his consistency program) and shifted effectiveness requirements from mathematics to meta-mathematics. It is here that modern computability theory found its ultimate motivation through the emphasis of the decision problem (*Entscheidungsproblem*) in the Hilbert School and the systematic articulation of the significance of Gödel's incompleteness theorems; both issues required a rigorous mathematical concept of effective method or mechanical procedure. Though these issues could have been addressed in their formulation for symbolic configurations, it took a detour through the calculability of number theoretic functions to arrive at sharp mathematical notions.

1.2. Uniform calculability. Dedekind formulated in his 1888 essay *Was sind und was sollen die Zahlen?* the general concept of a primitive recursive function and proved that all these calculable functions can be made explicit in his logicist framework. Dedekind's idea for the proof was very abstract, namely, to show the existence of unique solutions for functional equations of the form

$$\psi(0) = \omega,$$

$$\psi(\varphi(n))=\theta(\psi(n)),$$

where ω is an element of N, φ is the successor function and θ an arbitrary given function from N to N. This general point recurs in the early 1920s, for example, in the work of Hilbert, Skolem and Herbrand. However, the existence of solutions is no longer to be guaranteed by abstract logicist or set theoretic considerations, but rather by the availability of suitable calculation procedures. Implicit in these discussions is the specification of the class PR of primitive recursive functions. Hilbert's 1925 essay *On the Infinite* defines this class inductively, in almost the standard contemporary form, by specifying initial functions and closing under the definitional schemas of *composition* and *primitive recursion*. One shows by an easy inductive argument that the values of primitive recursive functions can be determined by an effective procedure for any given argument. All primitive recursive functions are in this sense calculable, but there are calculable functions that are not primitive recursive. Hilbert discussed an example due to Ackermann prominently already in 1925.

Herbrand viewed the Ackermann function in 1931 as *finitistically calculable.* In his systems of arithmetic he considered different classes F of finitist functions for which recursion equations were available. The defining axioms for the elements in F had to satisfy in particular this calculability condition, which had to be proved by finitist means:

> We must be able to show, by means of intuitionistic [i.e., finitist] proofs, that with these [defining] axioms it is possible to compute the value of the functions univocally for each specified system of values of their arguments. (Letter to Gödel in *Gödel's Collected Works V*, p. 15)

The issue of characterizing classes of finitistically calculable functions was crucial for Herbrand's reflections on Gödel's second incompleteness theorem and its impact on Hilbert's Program. Inspired by Herbrand's formulation, Gödel defined in his Princeton Lectures of 1934 the class of *general recursive functions*; its definition no longer depends on the problematic concept of finitist provability.

Gödel's class of functions includes all primitive recursive functions and also those of the Ackermann type. Assume, Gödel suggests, you are given a finite sequence $\psi_1, \ldots, \psi_k$ of known functions and a symbol ϕ for an unknown one. Then substitute these symbols "in one another in the most general fashions" and equate certain pairs of the resulting expressions. If the selected set of functional equations has exactly one solution, consider ϕ as denoting a "recursive" function; the definition of general recursive functions is obtained by insisting on two restrictive conditions. The first stipulates a standard form of certain terms, whereas the second condition demands that for every l-tuple $k_1, \ldots, k_l$ there is exactly one m such that $\phi(k_1, \ldots, k_l)=m$ is a *derived equation.* The set of derived equations is specified inductively. The basic clauses guarantee that all numerical instances of a given equation as well as all true equalities $\psi_{ij}(x_1, \ldots, x_n)=m$ are derived equations. The rules that allow steps from already obtained equations to additional ones are formulated as follows:

(**R.1**) Replace occurrences of $\psi_{ij}(x_1, \ldots, x_n)$ by m, if $\psi_{ij}(x_1, \ldots, x_n)=m$ is a derived equation;

(**R.2**) Replace occurrences of $\phi(x_1, \ldots, x_l)$ on the right-hand side of a derived equation by m, if $\phi(x_1, \ldots, x_l)=m$ is a derived equation.

Gödel emphasized two central features in his definition when comparing it to Herbrand's: first, the precise specification of *mechanical rules* for carrying out numerical computations in a uniform way; second, the formulation of the *regularity condition* requiring calculable functions to be total, but without insisting on a finitist proof of that fact.

1.3. Normal form and the μ-operator. Using Gödel's arithmetization technique to describe provability in the equational calculus Kleene analyzed the class of general recursive functions in 1936. The uniform and effective generation of the derived equations allowed him to establish an important theorem that is called now Kleene's normal form theorem: *for every recursive function* φ *there are primitive recursive functions* ψ *and* ρ *such that* $\varphi(x_1, \ldots, x_n)$ *equals* $\psi(\varepsilon y.\rho(x_1, \ldots, x_n, y)=0)$, *where for every n-tuple* $x_1, \ldots, x_n$ *there is a y such that* $\rho(x_1, \ldots, x_n, y)=0$. The latter equation expresses that y is (the code of) a computation from the equations that define φ for the arguments $x_1, \ldots, x_n$; $\varepsilon y.\rho(x_1, \ldots, x_n, y)=0$ provides the smallest y, such that $\rho(x_1, \ldots, x_n, y)=0$, if there is a y for the given arguments (it yields 0 otherwise). Finally, the function ψ considers the last equation in the given computation and determines the numerical value of the term on the r.h.s of that equation, which is a numeral and represents the value of φ for the given arguments $x_1, \ldots, x_n$. This theorem, or rather its proof, is quite remarkable: it allows to establish equivalences of different formulations with great ease; what is needed for the proof is only that the inference or computation steps are all primitive recursive.

Hilbert & Bernays had introduced in the first volume of their *Grundlagen der Mathematik* (1934) a μ-operator that functioned in just the way the ε-operator did for Kleene. The μ notation was adopted later by Kleene and is still being used in computability theory. Indeed, the μ-operator is at the heart of the definition of a new class of number theoretic functions, the so-called *μ-recursive functions*, and the normal form theorem is the crucial stepping stone in proving that this class of functions is co-extensional with that of Gödel's general recursive ones. The μ-recursive functions are specified inductively in the same way as the

primitive recursive ones, except that a third closure condition is formulated: if $\rho(x_1, \ldots, x_n, y)$ is μ-recursive and for every n-tuple $x_1, \ldots, x_n$ there is a y such that $\rho(x_1, \ldots, x_n, y)=0$, then the function $\theta(x_1, \ldots, x_n)$ given by $\mu y.\rho(x_1, \ldots, x_n, y)=0$ is also μ-recursive.

Gödel's concept characterized a class of calculable functions that contained all known effectively calculable functions. Footnote 3 of the Princeton Lectures seems to express a form of Church's thesis. In a letter to Martin Davis of 15 February 1965, Gödel rejected that interpretation. He wrote:

> ... 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; and in fact the equivalence between my definition and Kleene's ... is not quite trivial.

At that time in early 1934 Gödel was equally unconvinced by Church's proposal that effective calculability should be identified with λ-definability; he called the proposal "thoroughly unsatisfactory". That was reported by Church to Kleene on 29 November 1935. In the following year Gödel observed the *absoluteness* of general recursive functions: if the value of a general recursive function can be computed in a finite or even transfinite type extension of arithmetic, then it can be computed already in arithmetic. Gödel added this important observation to his 1936 paper *On the length of proofs* and viewed it as providing evidence that an important and stable class of functions had been isolated. The next section presents considerations of some of the pioneers, obviously including Gödel, as to their reasons, why the mathematically rigorous notion of machine computation introduced by Turing, and not general recursiveness, was ultimately viewed as providing the correct concept of mechanical procedure.

2. Conceptual analysis. Returning to the beginning, we notice a shift from effective mathematical calculations to mechanical operations of a machine. Church maintained in 1935 that the former are properly captured by the calculations involving general recursive functions. Clearly, if one has appropriate machines that allow the calculation of the base functions and mimic composition, recursion and minimization, then all recursive functions and thus all effectively

calculable ones are seen to be machine computable. Gödel argued in exactly that way in his *193?* and drew broader conclusions. He asserted that the characteristics of his equational calculus "are exactly those that give the correct definition of a computable function". He expanded that assertion by "That this really is the correct definition of mechanical computability was established beyond any doubt by Turing." The equivalence between general recursiveness and Turing computability is taken to support this claim.

2.1. Church's thesis. Almost a year after his conversation with Gödel, Church came back to his proposal in a letter to Bernays dated 23 January 1935; he conjectured that the λ-calculus may be a system that allows the representability of all constructively defined functions. When Church wrote this letter, he knew that all general recursive functions are λ-definable; the converse was established in collaboration with Kleene by March 1935. This mathematical equivalence and the quasi-empirical adequacy of λ-definability provided the background for the public articulation of Church's thesis. Church announced it in a talk contributed to the meeting of the American Mathematical Society in New York City on 19 April 1935, but formulated it with general recursiveness, not λ-definability as the mathematically rigorous notion.

In his 1936 paper Church restated his proposal for identifying the class of effectively calculable functions with a precisely defined class. To give a deeper analysis Church discussed, in section 7 of his paper, two methods of characterizing the effective calculability of number-theoretic functions. The first of these methods uses the notion of *algorithm*, and the second employs the notion of *calculability in a logic*. He argues that neither method leads to a definition that is more general than recursiveness. These arguments have a parallel structure, and we discuss only the one pertaining to the second method. Church considers a logic **L**, that is 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 defines:

F is *effectively calculable* if and only if there is an expression f in the logic L such that: {f}(μ)=ν is a theorem of L iff F(m)=n; here, μ and ν are expressions that stand for the positive integers m and n.

Church claims that F is recursive, *assuming* that L satisfies a certain step condition that amounts to requiring the theorem predicate of L to be recursively enumerable. The claim follows immediately by an application of the μ-operator; the argument parallels that for Kleene's normal form theorem.

The general concept of calculability is thus explicated by that of derivability in a logic, and Church uses the step condition to sharpen the idea that within such a logical formalism one operates with an effective notion of immediate consequence. The thesis is thus appealed to only in a special case. Given the crucial role this condition plays, it is appropriate to view it as a normative requirement: The steps of any effective procedure (governing derivations of a symbolic logic) must be recursive. If this requirement is accepted and a function is defined to be effectively calculable as above, then Church's step-by-step argument *proves* that all effectively calculable functions are recursive.

Church gave two reasons for the thesis, namely, (i) the quasi-empirical observation that all known calculable functions are general recursive and (ii) the mathematical fact of the equivalence of two differently motivated notions. A third reason comes directly from the 1936 paper, the step-by-step argument from a core conception. However, Church and also Gödel found in the end Turing's machine model of computation much more convincing. Church's 1937 review of Turing's paper for the Journal of Symbolic Logic asserts that Turing computability has the advantage over general recursiveness and λ-definability of "making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately ...".

2.2. Finite machines. Church's more detailed argument for the immediate evidence starts out as follows:

The author [Turing] proposes as a criterion that an infinite sequence of digits 0 and 1 be "computable" that it shall be possible to devise a computing machine, occupying a finite space and with working parts of finite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience,

certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality - in particular, a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a kind of Turing machine.

He then draws the conclusion, "It is thus immediately clear that computability, so defined, can be identified with ... the notion of effectiveness as it appears in certain mathematical problems ..." Why Turing's notion should convey this immediate conviction Church does not explain; the step from a computing machine "occupying a finite space and with working parts of finite size" to Turing machines is not deepened.

Gödel commented on Turing's notion in his 1951 Gibbs lecture publicly for the first time and made remarks similar to Church's. He explores there the implications of the incompleteness theorems, not in their original formulation, but rather in a "much more satisfactory form" that is "due to the work of various mathematicians". He stresses, "The greatest improvement was made possible through the precise definition of the concept of finite procedure, which plays such a decisive role in these results." There are, Gödel points out, different ways of arriving at a precise definition of finite procedure, which all lead to exactly the same concept.

However, and here is Gödel's substantive remark on Turing, "The most satisfactory way ... [of arriving at such a definition] is that of reducing the concept of finite procedure to that of a machine with a finite number of parts, as has been done by the British mathematician Turing." (*Collected Works III*, pp. 304-5) Gödel does not expand on this brief remark. In particular, he gives no hint of how reduction is to be understood or why the concept of such a restricted machine is equivalent to that of a Turing machine. At this point, it seems, the ultimate justification lies in the pure and perhaps rather crude fact that finite procedures can be reduced to computations of finite machines.

In a deep sense, neither Church nor Gödel seem to have recognized the distinctive character of Turing's analysis, i.e., the move from arithmetically motivated calculations to general symbolic processes that underlie them. Most importantly in the given intellectual context, these processes have to be carried out programmatically by human beings: the *Entscheidungsproblem* had to be

solved by us in a mechanical way; it was the normative demand of radical intersubjectivity between humans that motivated the step from axiomatic to formal systems. For this reason Turing brings in human computers and exploits the limitations of their processing capacities, when proceeding mechanically. The Turing machine is in the end nothing but, as Gandy put it, a codification of the human computer. Let us see how that comes about.

2.3. Computors. Call a human computing agent who proceeds mechanically a *computor*; such a computor operates on finite configurations of symbols and, for Turing, deterministically so. At issue is then, how do we step from calculations of computors to computations of Turing machines? Turing explores, as he put it, the extent of the computable numbers (or, equivalently, of the effectively calculable functions) by considering two-dimensional calculations in a child's arithmetic book. Such calculations are reduced to symbolic steps on linear configurations of such a simple character that a Turing machine operating on strings (instead of letters) can carry them out. Turing's argument concludes: "We may now construct a machine to do the work of the computer. ... The machines just described [string machines] do not differ very essentially from computing machines as defined in § 2 [letter machines], 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."

It is important to recall Turing's goal of isolating the basic steps of computations, that is, steps that need not be further subdivided. This leads to the demand that the configurations, which are operated on, must be *immediately recognizable* by the computor. Combined with the evident limitation of the computor's sensory apparatus, this demand motivates convincingly two restrictive conditions:

(**B**) (*Boundedeness*) There is a fixed finite bound on the number of configurations a computor can immediately recognize.

(**L**) (*Locality*) A computor can change only immediately recognizable (sub-) configurations.

Turing's considerations, sketched above, lead rigorously from operations of a computor on linear configurations to operations of a letter machine and can be generalized to other syntactic or graphic configurations. It should be noted that these constraints apply to Turing machines, but are violated by Gödel's equational calculus, as the replacement operations quite naturally involve terms of arbitrary complexity.

Turing's analysis secures the generality of mathematical results (for example of the incompleteness theorems) and their conclusiveness (for example of the undecidability of predicate logic) by respecting the intellectual context that appealed to effective operations carried out by humans without invoking higher mental capacities. It was after all the decision problem, the *Entscheidungsproblem* in the title of Turing's *1936*, that motivated Turing's work. Its positive solution required "a procedure ... that permits - for a given logical expression - to decide the validity, respectively satisfiability, by finitely many operations." Hilbert & Ackermann gave that formulation in their book *Grundzüge der theoretischen Logik* (1928) and considered the decision problem as the main problem of mathematical logic. Why *that* problem should be considered as the main problem of mathematical logic is stated clearly in their remark:

> The solution of this general decision problem would allow us to decide, at least in principle, the provability or unprovability of an arbitrary mathematical statement. (p. 86)

Taking for granted the finite axiomatizability of set theory or some other fundamental theory in first-order logic, the general decision problem is solved when that for first-order logic has been solved.

A negative solution of the decision problem required, however, a rigorous characterization of finite procedures and a proof that none of them answers Hilbert & Ackermann's demand. Turing did both, as he gave a convincing conceptual analysis, established the effective unsolvability of the halting problem (or rather of the equivalent printing problem), and showed how to reduce it to the decision problem. Thus, if the latter were effectively solvable, then the halting problem would be; but as it is not, we have a contradiction. The proof of the unsolvability of the halting problem makes crucial use of a particular

Turing machine, the universal machine U that, when presented on its tape with the program of a Turing machine M and an input, executes M's program for that input. This particular machine will play a special role in the next section.

3. Physical realization. For our further considerations, the most significant outcome of the above historical and conceptual examination can be restated sharply as follows: Turing's notion of machine computation is obtained by an analysis of symbolic calculations carried out by computors. To put it negatively, Turing's notion is *not* obtained by an independent analysis of physical devices with the goal of, first, defining a general notion of machine and, second, introducing an appropriate concept of computation for such machines. It was only in 1980 that Gandy gave an analysis of machines and the deterministic computations they can carry out. This is presented in the second subsection below and will be followed, in the last subsection, by a description of the special features of quantum computers. However, we discuss first what amounts to the physical implementation of Turing's universal machine U. That is an absolutely central step in the development of modern computing machines and was taken in intricately intertwined ways, it seems, by Turing and von Neumann; their work shaped the architecture of modern computers.

3.1. Implementing U. In the years following the Second World War Turing worked on various aspects of the design and the actual building of a practical version of his universal machine U. During the last three months of 1945 he wrote a remarkable document entitled *Proposal for development ... of an Automatic Computing Engine (ACE)* and connected, in a lecture to the London Mathematical Society of 20 February 1947, the work on the ACE explicitly with his early theoretical work:

> Some years ago I was researching on what might now be described as an investigation of the theoretical possibilities and limitations of digital computing machines. I considered a type of machine which had a central mechanism, and an infinite memory which was contained on an infinite tape [...] It was essential in these theoretical arguments that the memory should be infinite. It can easily be shown that otherwise the machine can only execute periodic operations. Machines such as the ACE may be regarded as practical versions of this same type of machine. (*Turing 1947*, pp. 106-7)

Turing characterized the ACE in his lecture as a typical *large-scale electronic digital* computing machine. From a mathematical perspective, Turing viewed being *digital* as the most relevant property of the ACE, since digital machines can work to any desired degree of accuracy and are *not* restricted, as analog machines are, to a particular type of computational problem.

From a practical point of view, the property of the ACE to be an *electronic* machine Turing considered as extremely important: it was to guarantee high speed and thus make it possible to execute complex procedures. The latter possibility requires, beyond the speed of basic operations, an appropriate organization of the machine, so that it can proceed fully *automatically* - without having to interact with a human operator - while executing a procedure. Turing emphasized, alluding to his universal machine:

> It is intended that the setting up of the machine for new problems shall be virtually only a matter of paper work. Besides the paper work nothing will have to be done except to prepare a pack of Hollerith cards in accordance with this paper work, and to pass them through a card reader connected with the machine. There will positively be no internal alterations to be made even if we wish suddenly to switch from calculating the energy levels of the neon atom to the enumeration of groups of order 720. It may appear somewhat puzzling that this can be done. How can one expect a machine to do all this multitudinous variety of things? The answer is that we should consider the machine as doing something quite simple, namely carrying out orders given to it in a standard form which it is able to understand. (*Turing 1946*, p. 21)

In the 1947 lecture he made the connection to the universal machine quite explicit; after discussing memory extensively, he claims that digital computing machines such as the ACE are just "practical versions of the universal machine". He continues, "There is a certain pool of electronic equipment, and a large memory. When any particular problem has to be handled the appropriate instructions for the computing process involved are stored in the memory of the ACE and it is then 'set up' for carrying out that process."

The requirements for building a universal machine can in the end only be satisfied, if the machine is not only digital and electronic, but also *large-scale*, as they involve demands for "storage of information or mechanical memory". Indeed, Turing pointed out already in the ACE Report that "the memory needs to be very large indeed ... ". The principled as well as the practical issues of implementation overlapped at this point with developments in America. Indeed,

Turing recommended reading his report "in conjunction with J. von Neumann's *Report on the EDVAC*". (Goldstine and Hodges present complementary views on the tenuous connection between the two projects; a balanced perspective is given in Hodges' note 5.26, pp. 555-6.)

von Neumann completed a first draft of his report on 30 June 1945; the report emerged out of work with the group of Eckert and Mauchly at the Moore School of Electrical Engineering (University of Pennsylvania, Philadelphia). The group had built one of the first electronic calculators, the *Electronic Numerical Integrator and Computer* (ENIAC), and was evaluating a new memory system for a second, more sophisticated calculator, the *Electronic Discrete Variable Calculator* (EDVAC). The demand for a large, readily accessible memory emerged out of computational practice, namely, the need to have very fast access to instructions, but also to fixed constant parameters and statistical data. That was to be achieved by storing them in the machine; von Neumann writes:

> The device requires a considerable memory. While it appeared, that various parts of this memory have to perform functions which differ somewhat in their nature and considerably in their purpose, it is nevertheless tempting to treat the entire memory as one organ.

von Neumann shifted the attention from the technological problems of having a larger memory to logical ones concerning the basic structure of machines with a central control mechanism and extensive memory. This structure is discussed in detail pp. 204-210 of Goldstine's book.

A higher level of generality was attained in the *Electronic Computer Project* at the Institute for Advanced Study in Princeton; this project was begun in March of 1946 and directed by von Neumann. The resulting IAS Computer can be viewed as a prototype of all modern computers (the "von Neumann machine"). Its basic architecture, however, is similar to that of the ACE; it is the balance between arithmetical and fundamental logical operations that is distinctive. Goldstine describes the issue as follows:

> The work of Post and Turing made it very clear that from the point of view of formal logics there was no problem to devise codes which were "in abstracto adequate to control and cause the execution of any sequence of operations which are individually available in the machine and which are, in their entirety, conceivable by the problem planner." The problem is of a practical nature and is closely allied to that connected with the choice of elementary operations in the arithmetic organ. (p. 258)

Turing and von Neumann made different compromises between simplicity of basic machine operations and complexity of programs needed to execute mathematical or symbolic procedures. These choices were obviously informed by their different computational experience and goals, but also by their broader philosophical outlook. (That is quite movingly described in Hodges's book on pp. 320 - 333.)

3.2. Discrete machines. Turing's U can be realized within practical limits by physical devices, and we can raise the question, whether these devices are just doing things faster than we do, or whether they are in a principled way computationally more powerful. Church, as we recalled earlier, asserted in 1937 that finite machines are essentially Turing machines; in Gödel's remarks (from *193?* and *1951*) that assertion is taken for granted. The claim seems to be plausible, but it does require an argument. On the one hand, there may be physical systems that do not obey the same restrictions as computors and consequently may be able to carry out computations not possible for a computor. On the other hand, there may be physically grounded limits for machines in the same way that there are psychologically based constraints for computors.

The character of individual computational steps was at the heart of the conceptual analysis. Due to physical constraints, such steps cannot be accelerated unboundedly or be made arbitrarily complex (*Mundici & Sieg*, section 3). However, there seems to be the possibility of sidestepping these constraints by using massively parallel operations. Cellular automata, introduced by Ulam and von Neumann, operate in parallel; they do not satisfy the boundedness condition (**B**), as the configurations affected in a single computation step are potentially unbounded. They can simulate universal Turing machines and yield discrete simulations of complex physical processes. Zuse, for example, reflected on digital formulations of physics in his essay *Rechnender Raum* (1967). Fredkin advocated the use of (reversible) cellular automata in physics and conjectured in his *Digital Mechanics* (1990) "that there will be found a single cellular automaton rule that models all of microscopic physics; and models it exactly." The interested

reader should consult Herken's collection *1988*, Toffoli & Margoulis's book *Cellular automata machines* and of course Wolfram's perspective in *A new kind of science*.

Gandy addresses the issue of parallel machine computations in his essay *Church's Thesis and Principles for Mechanisms*, where he proposes a particular mathematical description of *discrete mechanical devices* and their computations. He then follows Turing's three steps of pertinent analysis, articulation of constraints and proof of a reduction theorem. The central and novel aspect of Gandy's formulation lies in the fact that it incorporates parallelism in complete generality. Cellular automata fall directly under Gandy's formulation. And yet, the reduction theorem shows that everything calculable by a device satisfying the constraints, a *Gandy machine*, is already computable by a Turing machine. Here is a sketch of the main considerations.

Gandy introduces the term discrete mechanical device to make it vivid that his analysis is not at all concerned with analog devices, but rather with machines that are discrete and proceed step-by-step from one state to the next. Gandy considers two physical constraints as fundamental for such devices: (1) a lower bound on the size of atomic components and (2) an upper bound on the speed of signal propagation. Together, these constraints guarantee what the sensory limitations guarantee for computors, namely that in a given unit of time there is a bound on the number of different observable configurations and of possible actions on them. However, the incorporation of massive parallelism into the mathematical description takes in Gandy's *1980* a substantial amount of complex mathematical work. In *Sieg 2002* Gandy machines are axiomatized as special discrete dynamical systems, and this presentation makes clear that they are radical generalizations of Turing machines: the latter modify one bounded part of a state, whereas the former operate in parallel on arbitrarily many bounded parts to arrive at the next state of the system.

Discrete computing machines in the broadest sense, when only constrained by physically motivated boundedness and locality conditions, do not reach beyond the computational power of Turing machines; that is the general

moral. - Every mathematical model of physical processes faces at least two questions, namely: How accurately does the model capture physical reality, and how efficiently can the model be used to make predictions? It is distinctive for modern developments that, on the one hand, computer simulations have led to an emphasis of algorithmic aspects of scientific laws and that, on the other hand, many physical systems are being considered as computational devices. But under what conditions can a physical system really be viewed in that way? To have one important data point for reflections on this question, we will look at the case of particular quantum systems.

3.3. Quantum computers. Suppose we have a photon that impinges on a beam-splitter and then propagates *via* two different paths. Quantum theory describes the photon as going partly into each of these two components. The state of the photon is given by the superposition of the two states associated with the two components of the original beam. Any observation of the photon, however, results in either the whole photon or nothing at all. This implies that after a measurement (1) the photon changes its state from being partly in one beam and partly in the other to being entirely in one of the beams, and (2) any interference effect is lost since one of the beams no longer enters into the description of the photon. If a second beam-splitter combines the two beams, then the photon will be observed with probability one in a single beam. This certainty is due to quantum interference. Quantum computation arises from the possibility of exploiting a multiplicity of parallel computational paths in superposition as well as quantum interference to amplify the probability of correct outcomes of computations.

As the photon can be in a coherent superposition of being in two beams, the basic unit of quantum information, a *qubit* (from quantum bit), is a two-state system that can be prepared in a superposition of the two logical states 0 and 1. If a computational state can be reached through several alternative paths, then its probability is the squared modulus of the sum of all the "probability amplitudes" for the constituent paths. (Probability amplitudes determine probabilities and these have to add up to one for any quantum computational state.) Since the

probability amplitudes are complex numbers, they may cancel each other and produce *destructive* interference, or enhance each other and produce *constructive* interference.

Imagine a computation that starts in the *input* state 0 and reaches the *output* state in two steps. Suppose a computational step can mimic the action of a beam-splitter and generate a superposition of two intermediate output states, 0 and 1 with probability amplitudes $\sqrt{\ }$ and $\sqrt{\ }$. Then the probability of each output is the same: $|\ |\ \ |\ |$. However, if the output state is measured after two computational steps, then the probability of the output 1 is *one*: the action of a beam-splitter can be perfectly simulated by quantum computing operations which have no classical analogs. One of these is the $\sqrt{\ \ \ }$, which when applied twice results in the logical operation *NOT*.

Since quantum mechanics describes a state transformation by means of a unitary operator, any quantum computing operation is a unitary transformation on qubits. The description of a *quantum Turing machine* (*QTM*) is derived from a Turing machine, but using quantum theory to define the operations carried out by the computer, which is now a *physical system*. Quantum interference allows a *QTM* to act on coherent superpositions of a given state and evolves them *via* unitary operators into other superpositions, from which the next state results with a certain probability. Any unitary operation on n qubits can be decomposed into simple operations on one or two qubits.

A collection of n qubits constitutes a *quantum register* of size n (the analogue of a Turing machine tape). A quantum register of two qubits can store all four numbers $|\ \rangle, |\ \rangle, |\ \rangle, |\ \rangle$ in superposition. Adding qubits increases the storage capacity of the register exponentially: given a quantum register of size L, a *QTM* can in one computational step perform the same mathematical operation on 2^L numbers; a classical machine has to repeat the same computation 2^L times or has to use 2^L different processors working in parallel. However, if we try to read a number out of a superposition of the 2^L output states then we see just one randomly chosen number. Only after an appropriate number of

computational steps can we obtain a single final result that depends - in constructive ways - on all 2^L intermediate results.

This is how *quantum* algorithms work. Grover's algorithm, as an example, can determine an element from an unsorted list of N items in approximately $\sqrt{\ }$ steps. A classical algorithm that scans the entries one by one requires on average $N/2$ steps. Another quantum algorithm, due to Peter Shor, can factorize large integers efficiently. Here the difference in performance between the quantum and classical algorithms seems exponential. Quantum algorithms solve some important problems more efficiently than classical ones, but they don't increase the class of computable functions

If, using Wittgenstein words, Turing machines are humans who calculate, then quantum Turing machines are physical systems that calculate. What made this shift possible was Deutsch's analysis leading to the assertion, "Every finitely realizable physical system can be perfectly simulated by a universal Turing computing machine operating by finite means." Following Deutsch a computing machine operates by finite means if: (i) only a finite subsystem is in motion during anyone step; (ii) the motion depends only on the state of a finite subsystem; (iii) the rules that specify the motion can be given finitely in the mathematical sense (for example by an integer). "Turing machines", Deutsch asserts, "satisfy these conditions, and so does the universal quantum computer." Thus, boundedness conditions play also a significant role in characterizing the computation of a quantum system.

4. Concluding remarks. Computing machines have taken over the tasks of computors and transcend in important ways (of power and efficiency, for example) human computational capacities. The takeover has two bases: (i) aspects of physical or intellectual reality have a finite symbolic representation, and (ii) machines can take on (part of) the effective manipulation of the physical tokens involved in a representation. The latter may consist of just simulating the mechanical steps in human operations, as Turing machine do, or it may involve complex physical processes that are used in a different way, as in the case of

quantum computers, when a suitable theoretical description allows them to perform a massively parallel calculation, so to speak, in a single step.

The concrete technological and scientific challenges of building a quantum computer seem enormous. Broad issues surrounding computing machines in general are multifarious and reach from the mathematically fundamental to the methodologically problematic. Can representations, for example, contain infinite components? Are there physical processes that can be viewed as computations, but do not fall within the Turing limits? What is the conceptual nature of analogue computations? Do they have to have a mathematical description that allows a calculable determination? What are the critical physical issues concerning measurement?

The ultimate challenge, articulated by Turing, is to have machines exhibit intelligence. Implementing the universal machine U meant for Turing to build a machine with *discipline*; producing intelligence required in addition *initiative*. Here is then the core of Turing's challenge, "Our task is to discover the nature of this residue as it occurs in man, and to try and copy it in machines." Computing machines have become in their modern form scientific tools to explore, in particular, our own intellectual nature.

References

The classical 1936 papers by Church, Kleene, Post and Turing are all reprinted in the volume *The Undecidable* (1965), which was edited by Martin Davis. Gödel's papers are all available in his *Collected Works*, which have been published in five volumes by Oxford University Press. Turing's papers from the late 1940s are collected in *Mechanical Intelligence* (D. C. Ince, ed.); the volume appeared with North-Holland in 1992. The classical textbook presenting the Turing-Post development of computability theory is Davis's 1958 *Computability and Unsolvability*. Finally, the references to von Neumann's papers and reports are all in *Goldstine 1972*.

Babbage, Charles
1864 *Passages from the Life of a Philosopher*; London. Reprinted with an introduction by M. Campbell-Kelly, Rutgers University Press, 1994.

Deutsch, David
1985 Quantum theory, the Church-Turing principle and the universal quantum computer; *Proceedings of the Royal Society*, A **400**.

Gandy, Robin
1980 Church's thesis and principles for mechanisms; in: *The Kleene Symposium*, (J. Barwise *et al.* eds.), North-Holland.

Goldstine, Herman H.
1972 *The Computer from Pascal to von Neumann*; Princeton University Press.

Herken, Rolf (ed.)
1988 *The Universal Turing Machine. A Half-Century Survey*; Oxford University Press.

Hodges, Andrew
1983 *Alan Turing: The Enigma*; Simon and Schuster.

Mundici, Daniele and Wilfried Sieg
1995 Paper Machines; *Philosophia Mathematica* 3.

Sieg, Wilfried
2002 Calculations by man and machine: conceptual analysis; in: *Reflections on the foundations of mathematics*; Lecture Notes in Logic 15, Association for Symbolic Logic.