Turing Oracle Machines, Online Computing, and Three Displacements in Computability Theory

Robert I. Soare*

January 3, 2009

Contents

*Parts of this paper were delivered in an address to the conference, *Computation and Logic in the Real World,* at Siena, Italy, June 18–23, 2007. **Keywords:** Turing machine, automatic machine, a-machine, Turing oracle machine, o-machine, Alonzo Church, Stephen C. Kleene, Alan Turing, Kurt Gödel, Emil Post, computability, incomputability, undecidability, Church-Turing Thesis, Post-Turing Thesis on relative computability, computable approximations, Limit Lemma, effectively continuous functions, computability in analysis, strong reducibilities. Thanks are due to C.G. Jockusch, Jr., P. Cholak, and T. Slaman for corrections and suggestions.

Abstract

We begin with the history of the discovery of computability in the 1930's, the roles of Gödel, Church, and Turing, and the formalisms of recursive functions and Turing automatic machines (a-machines). To

whom did Gödel credit the definition of a computable function? We present Turing's notion [1939, §4] of an *oracle machine* (o-machine) and Post's development of it in [1944, §11], [1948], and finally Kleene-Post [1954] into its present form.

A number of topics arose from Turing functionals including continuous functionals on Cantor space and online computations. Almost all the results in theoretical computability use relative reducibility and o-machines rather than a-machines and most computing processes in the real world are potentially online or interactive. Therefore, we argue that Turing o-machines, relative computability, and online computing are the most important concepts in the subject, more so than Turing a-machines and standard computable functions since they are special cases of the former and are presented first only for pedagogical clarity to beginning students. At the end in §10 – §13 we consider three displacements in computability theory, and the historical reasons they occurred. Several brief conclusions are drawn in §14.

1 Introduction

In this paper we consider the development of Turing oracle machines and relative computability and its relation to continuity in analysis and to online computing in the real world. We also challenge a number of traditional views of these subjects as often presented in the literature since the 1930's.

1.1 Terminology: Incompleteness and Incomputability

The two principal accomplishments in computability and logic in the 1930's were the discovery of incompleteness by Gödel [1931] and of incomputability independently by Church [1936] and Turing in [1936]. We use the term "noncomputable" or "incomputable" for individual instances, but we often use the term "incomputability" for the general concept because it is linguistically and mathematically parallel to "incomplete." The term "incomputable" appeared in English as early as 1606 with the meaning, that which "cannot be computed or reckoned; incalculable," according to the Oxford English Dictionary. Websters dictionary defines it as "greater than can be computed or enumerated; very great." Neither dictionary lists an entry for "noncomputable" although it has often been used in the subject to mean "not computable" for a specific function or set analogously as "nonmeasurable" is used in analysis.

1.2 The Goal of *Incomputability* not Computability

For several thousand years the study of algorithms had led to new theoretical algorithms and sometimes new devices for *computability* and calculation. In the 1930's for the first time the goal was the refutation of Hilbert' two programs, a finite consistency proof for Peano arithmetic, and the *Entscheidungsproblem* (decision problem). For the latter, two main discoverers of computability, Alonzo Church and Alan Turing, wanted to give formal definitions of a computable function so that they could diagonalize over all computable functions and produce an *incomputable* (unsolvable) problem. The specific models of computable functions produced by 1936, Turing a-machines, λ-definable functions, and recursive functions, would all have deep applications to the design and programming of computing machines, but not until after 1940. Meanwhile, the researchers spent the remainder of the 1930's investigating more of the new world of incomputability they had created by diagonal arguments, just as Georg Cantor had spent the last quarter of the nineteenth century exploring the world of uncountable sets which he had created by the diagonal method. In §1 and §2 we consider this historical development from 1931 to 1939 and we introduce quotes from Gödel to show convincingly that he believed "the correct definition of mechanical computability was established beyond any doubt by Turing" and only by Turing.

1.3 Computing Relative to an Oracle or Database

In 1936 Turing's a-machines and Church's use of Gödel's recursive functions solved an immediate problem by producing a definition of a computable function, with which one could diagonalize and produce undecidable problems in mathematics. The Turing a-machine is a good model for offline computing such as a calculator or batch processor where the process is initially given a procedure and an input and continues with no further outside information.

However, many modern computer processes are *online processes* in that they consult an external data base of other resource during the computation process. For example, a laptop computer might consult the World Wide Web via an ethernet or wireless connection while it is computing. These processes are sometimes called *online* or *interactive* or *database* processes depending on the way they are set up.

Turing spent 1936–1938 at Princeton writing a Ph.D. thesis under Church on ordinal logics. A tiny and obscure part of his paper [1939, §4] included

a description of an *oracle machine* (o-machine) roughly a Turing a-machine which could interrogate an *"oracle" (external database)* during the computation. The one page description was very sketchy and Turing never developed it further.

Emil Post [1944, §11] considerably expanded and developed relative computability and Turing functionals. These concepts were not well understood when Post began, but in Post [1943], [1944], [1948] and Kleene-Post [1954] they emerged into their modern state. These are summarized in the *Post-Turing Thesis* of §6.2. This remarkable role by Post has been underemphasized in the literature but is discussed here in §5 and §6.

The theory of relative computability developed by Turing and Post and the o-machines provide a precise mathematical framework for database or online computing just Turing a-machines provide one for offline computing processes such as batch processing. Oracle computing processes are those most often used in theoretical research and also in real world computing where a laptop computer may communicate with a database like the World Wide Web. Often the local computer is called the "client" and the remote device the "server."

In §4–§6 we study Turing's oracle machines (o-machines) and Post's development of them into relative computability. It is surprising that so much attention has been paid to the Church-Turing Thesis 3.2 over the last seventy years and so little to the Post-Turing Thesis 6.1 on relative reducibility, particularly in view of the importance of relative computability (Turing o-machines) in comparison with plain computability (Turing a-machines) in both theoretical and practical areas.

1.4 Continuous Functions and Calculus

In §7 we show that any Turing functional on Cantor space 2^ω is an effectively continuous partial map. Conversely, for any continuous functional there is a Turing functional relative to some real parameter which defines it, thereby linking computability with analysis. This makes Turing functionals the analogue in computability of the continuous functions in analysis as we explain. In contrast, Turing a-machines viewed as functionals on Cantor space produce only a constant functional. Surprisingly, there appears to be more emphasis in calculus books on continuous functionals than there is in introductory computability books on computably continuous functionals and relative computability.

2 Origins of Computability and Incomputability

Mathematicians have studied algorithms and computation since ancient times, but the modern study of computability and incomputability began around 1900. David Hilbert was deeply interested in the foundations of mathematics. Hilbert [1899] gave an axiomatization of geometry and showed [1900] that the question of the consistency of geometry reduced to that for the real-number system, and that in turn, to arithmetic by results of Dedekind (at least in a second order system). Hilbert [1904] proposed proving the consistency of arithmetic by what emerged [1928] as his *finitist program*. He proposed using the finiteness of mathematical proofs in order to establish that contradictions could not be derived. This tended to reduce proofs to manipulation of finite strings of symbols devoid of intuitive meaning which stimulated the development of mechanical processes to accomplish this. Hilbert's second major program concerned the *Entscheidungsproblem* (decision problem). Kleene [1987b, p. 46] wrote, "The *Entscheidungsproblem* for various formal systems had been posed by Schröder [1895], Löwenheim [1915], and Hilbert [1918]." The decision problem for first order logic emerged in the early 1920's in lectures by Hilbert and was stated in Hilbert and Ackermann [1928]. It was to give a decision procedure (*Entscheidungsverfahren*) "that allows one to decide the validity of the sentence." Hilbert characterized this as the fundamental problem of mathematical logic. Davis [1965, p. 108] wrote, "This was because it seemed clear to Hilbert that with the solution of this problem, the *Entscheidungsproblem*, it should be possible, at least in principle, to settle all mathematical questions in a purely mechanical manner." Von Neumann (1927) doubted that such a procedure existed but had no idea how to prove it.

2.1 Gödel's Incompleteness Theorem

Hilbert retired in 1930 and was asked to give a special address in the fall of 1930 in Königsberg, the city of his birth. Hilbert spoke on natural science and logic, the importance of mathematics in science, and the importance of logic in mathematics. He asserted that there are no unsolvable problems and stressed,

> *Wir müssen wissen.* (We must know.)
> *Wir werden wissen.* (We will know.)

At a mathematical conference preceding Hilbert's address, a quiet, obscure young man, Kurt Gödel, only a year a beyond his Ph.D., announced a result

which would forever change the foundations of mathematics. He formalized the liar paradox, "This statement is false," to prove roughly that for any effectively axiomatized, consistent extension T of number theory (Peano arithmetic) there is a sentence σ which asserts its own unprovability in T. John von Neumann in the audience immediately understood the importance of Gödel's Incompleteness Theorem. He was at the conference representing Hilbert's proof theory program, and recognized that Hilbert's program was over. In the next few weeks von Neuman realized that by arithmetizing the proof of Gödel's first theorem, one could prove an even better one, that no such formal system T could prove its own consistency. A few weeks later he brought his proof to Gödel who thanked him and informed him politely that he had already submitted the Second Incompleteness Theorem for publication. Gödel's Incompleteness Theorem [1931] not only refuted Hilbert's program on proving consistency, but it also had a profound effect on refuting Hilbert's second theme of the *Entscheidungsproblem*. Gödel had successfully challenged the Hilbert's first proposal. This made it easier to challenge Hilbert on the second topic of the decision problem. Both refutations used diagonal arguments. Of course, diagonal arguments had been known since Cantor's work, but Gödel showed how to arithmetize the syntactic elements of a formal system and diagonalize within that system. Crucial elements in computability theory, such as the Turing universal machine, the Kleene μ-recursive functions, or the self reference in the Kleene's Recursion Theorem, all depend upon giving code numbers to computations and elements within a computation, and in calling algorithms by their code numbers (Gödel numbers). These ideas spring from Gödel's [1931] incompleteness proof.

2.2 Alonzo Church

By 1931 computability was a young man's game. Hilbert had retired and no longer had much influence on the field. As the importance of Gödel's Incompleteness Theorem began to sink in, and researchers began concentrating on the *Entscheidungsproblem,* the major figures were all young. Alonzo Church (born 1903), Kurt Gödel (b. 1906), and Stephen C. Kleene (b. 1909) were all under thirty. Turing (b. 1912), perhaps the most influential of all on computability theory, was not even twenty. Only Emil Post (b. 1897) was over thirty, and he was not yet thirty-five. These young men were about to leave Hilbert's ideas behind and open the path of computability for the next two thirds of the twentieth century, which would solve the theoretical problems and would show the way toward practical computing devices.

After completing his Ph.D. at Princeton in 1927, Alonzo Church spent one year at Harvard and one year at Göttingen and Amsterdam. He returned to Princeton as an Assistant Professor of Mathematics in 1929. In 1931 Church's first student, Stephen C. Kleene, arrived at Princeton. Church had begun to develop a formal system now called the λ-calculus. In 1931 Church knew only that the successor function was λ-definable. Kleene began proving that certain well-known functions were λ-definable. By the time Kleene received his Ph.D. in 1934 he had shown that all the usual number theoretic functions were λ-definable. On the basis of this evidence and his own intuition, Church proposed to Gödel around March, 1934 the first version of his thesis on functions which are *effectively calculable,* the term in the 1930's for a function which is computable in the informal sense. (See Davis [1965, p. 8–9].)

Definition 2.1. Church's Thesis (First Version) [1934]. A function if effectively calculable if and only if it is λ-definable.

When Kleene first heard the thesis, he tried to refute it by a diagonal argument but since the λ-definable functions were only partial functions, the diagonal was one of the rows. Instead of a contradiction, Kleene had proved a beautiful new theorem, the Kleene Recursion Theorem whose proof is a diagonal argument which fails (see Soare [1987, p. 36]). Although Kleene was convinced by Church's first thesis, Gödel was not. Gödel rejected Church's first thesis as "thoroughly unsatisfactory."

2.3 Herbrand-Gödel Recursive Functions

From 1933 to 1939 Gödel spent time both in Vienna pursuing his academic career and at Fine Hall in Princeton which housed both the Princeton University faculty in mathematics and the Institute for Advanced Study, of which he was a member. Gödel spent the first part of 1934 at Princeton. The primitive recursive functions which he had used in his 1931 paper did not constitute all computable functions. He expanded on a concept of Herbrand and brought closer to the modern form. At Princeton in the spring of 1934 Gödel lectured on the Herbrand-Gödel recursive functions which came to be known as the *general recursive functions* to distinguish them from the primitive recursive functions which at that time were called "recursive functions." Soon the prefix "primitive" was added to the latter and the prefix "general" was generally dropped from the former. Gödel's definition gave a remarkably succinct system whose simple rules reflected the way a mathematician would informally calculate a function using recursion.

Church and Kleene attended Gödel's lectures on recursive functions. Rosser and Kleene took notes which appeared as Gödel [1934]. After seeing Gödel's lectures, Church and Kleene changed their formalism (especially for Church's Thesis) from "λ-definable" to "Herbrand-Gödel general recursive." Kleene [1981] wrote,

> "I myself, perhaps unduly influenced by rather chilly receptions from audiences around 1933–35 to disquisitions on λ-definability, chose, after general recursiveness had appeared, to put my work in that format. ..."

Nevertheless, λ-definability is a precise calculating system and has close connections to modern computing, such as functional programming.

2.4 Stalemate at Princeton Over Church's Thesis

Church reformulated his thesis, with Herbrand-Gödel recursive functions in place of λ-definable ones. This time without consulting Gödel, Church presented to the American Mathematical Society on April 19, 1935, his famous proposition described in his paper [1936].

> "In this paper a definition of *recursive function of positive integers* which is essentially Gödel's is adopted. It is maintained that the notion of an effectively calculable function of positive integers should be identified with that of a recursive function, ..."

It has been known since Kleene [1952] as *Church's Thesis* in the following form.

Definition 2.2. Church's Thesis [1936]. A function on the positive integers is effectively calculable if and only if it is recursive.

As further evidence, Church and Kleene had shown the formal equivalence of the Herbrand-Gödel recursive functions and the λ-definable functions. Kleene introduced a new equivalent definition, the μ-recursive functions, functions defined by the five schemata for primitive recursive functions, plus the least number operator μ. The μ-recursive functions had the advantage of a short standard mathematical definition, but the disadvantage that any function not primitive recursive could be calculated only by a tedious arithmetization as in Gödel's Incompleteness Theorem.

2.5 Gödel's Thoughts on Church's Thesis

In spite of this evidence, Gödel still did not accept Church's Thesis by the beginning of 1936. Gödel had become the most prominent figure in mathematical logic. It was his approval that Church wanted most. Church had solved the *Entscheidungsproblem* only if his characterization of effectively calculable functions was accurate. Gödel had considered the question of characterizing the calculable functions in [1934] when he wrote,

> "[Primitive] recursive functions have the important property that, for each given set of values for the arguments, the value of the function can be computed by a finite procedure[3]."
>
> *Footnote 3.*
> "The converse seems to be true, if, besides recursion according to scheme (V) [primitive recursion], recursions of other forms (e.g., with respect to two variables simultaneously) are admitted. This cannot be proved, since the notion of finite computation is not defined, but it serves as a heuristic principle."

The second paragraph, Gödel's footnote 3, gives crucial insight into his thinking about the computability thesis and his later pronouncements about the achievements of Turing versus others. Gödel says later that he was not sure that his system of Herbrand-Gödel recursive functions comprised all possible recursions. Second, his final sentence suggests that he may have believed such a characterization "cannot be proved," but is a "heuristic principle."

This suggests that Gödel was waiting not only for a formal definition (such as recursive functions or Turing machines which came later) but evidence that these captured the informal notion of effectively calculable (which Turing later gave, but which Gödel did not find in Church's work). Here he even suggests that such a precise mathematical characterization of the informal notion cannot be proved which makes his acceptance of Turing's later work even more impressive.

3 Turing Breaks the Stalemate

3.1 Turing Machines and Turing's Thesis

At the start of 1936 those gathered at Princeton: Gödel, Church, Kleene, Rosser, and Post nearby at City College in New York, constituted the most

distinguished and powerful group in the world investigating the notion of a computable function and Hilbert's *Entscheidungsproblem*, but they could not agree on whether recursive functions constituted all effectively calculable functions. At that moment stepped forward a twenty-two year old youth, far removed from Princeton. Well, this was not just any youth. Alan Turing had already proved the Central Limit Theorem in probability theory, not knowing it had been previously proved, and as a result Turing had been elected a Fellow of King's College, Cambridge.

The work of Hilbert and Gödel had become well-known around the world. At Cambridge University topologist M.H.A. (Max) Newman gave lectures on Hilbert's *Entscheidungsproblem* in 1935. Alan Turing attended. Turing's mother had had a typewriter which fascinated him as a boy. He designed his *automatic machine (a-machine)* as a kind of idealized typewriter with an infinite carriage over which the reading head passes with the ability to read, write, and erase one square at a time before moving to an immediately adjacent square, just like a typewriter.

Definition 3.1. Turing's Thesis [1936]. A function is intuitively computable (effectively calculable) if and only if it is computable by a Turing machine, *i.e.*, an automatic machine (a-machine), as defined in Turing [1936].

Turing showed his solution to the astonished Max Newman in April, 1936. The *Proceedings of the London Mathematical Society* was reluctant to publish Turing's paper because Church's had already been submitted to another journal on similar material. Newman persuaded them that Turing's work was sufficiently different, and they published Turing's paper in volume 42 on November 30, 1936 and December 23, 1936. There has been considerable misunderstanding in the literature about exactly when Turing's seminal paper was published. This is important because of the appearance in 1936 of related papers by Church, Kleene, and Post, and Turing's priority is important here.[1]

[1]Many papers, Kleene [1943, p. 73], [1987], [1987b], Davis [1965, p. 72], Post [1943, p. 20], Gödel Collected Works, Vol. I, p. 456, and others, mistakenly refer to this paper as "Turing [1937]," perhaps because the volume **42** is 1936-37 covering 1936 and part of 1937, or perhaps because of the two page minor correction [1937a]. Others, such as Kleene [1952], [1981], [1981b], Kleene and Post [1954, p. 407], Gandy [1980], Cutland [1980], and others, correctly refer to it as "[1936]," or sometimes "[1936-37]." The journal states that Turing's manuscript was "Received 28 May, 1936–Read 12 November, 1936." It appeared in two sections, the first section of pages 230–240 in Volume **42**, Part 3, issued on November 30, 1936, and the second section of pages 241–265 in Volume **42**, Part 4,

Turing's a-machine has compelling simplicity and logic which makes it even today the most convincing model of computability. Equally important with the Turing machine was Turing's analysis of the intuitive conception of a "function produced by a mechanical procedure." In a masterful demonstration, which Robin Gandy considered as precise as most mathematical proofs, Turing analyzed the informal nature of functions computable by a finite procedure and demonstrated that they coincide with those computable by an a-machine. Also Turing [1936, p. 243] introduced the *universal* Turing machine which has great theoretical and practical importance.

3.2 Gödel's Opinion of Turing's Work

Gödel never accepted Church's Thesis in the form above, even though it was formulated with his own general recursive functions, but Gödel and most others accepted Turing's Thesis. Gödel knew of the *extensional* evidence. Church and Kleene [1936b] had shown the formal equivalence of λ-definable functions, general recursive functions, and Kleene had proved the equivalence with μ-recursive functions based on Gödel's own arithmetization of 1931. Gödel was also well aware of Turing's proof [1937b] of the equivalence of λ-definable functions with Turing computable ones, and hence the *confluence* of all the known definitions.

However, Gödel was interested in the *intensional* analysis of finite procedure as given by Turing [1936]. He had not accepted the argument of confluence as sufficient to justify Church's Thesis. Gödel clearly expresses his opinion in his three volume collected works, Gödel [1986], Gödel [1990], and Gödel [1995]. Let us examine there what Gödel has to say. In the following article Gödel considers all these as equivalent formal definitions. The question was whether they captured the *informal* concept of a function specified by a *finite procedure*. The best source for articles of Gödel is the three volume Collected Works, which we have listed by year of publication: Volume I, Gödel [1986]; Volume II, Gödel [1990], and Volume III, Gödel [1995].

---

issued December 23, 1936. No part of Turing's paper appeared in 1937, but the two page minor correction [1937a] did. Determining the correct date of publication of Turing's work is important to place it chronologically in comparison with Church [1936], Post [1936], and Kleene [1936].

3.2.1 Gödel [193?] Notes in Nachlass [1935]

This article has an introductory note by Martin Davis in Gödel [1995, p. 156]. Davis wrote, "This article is taken from handwritten notes in English, evidently for a lecture, found in the *Nachlass* in a spiral notebook." In the Nachlass printed in Gödel [1995, p. 166] Gödel wrote,

> "When I first published my paper about undecidable propositions the result could not be pronounced in this generality, because for the notions of mechanical procedure and of formal system no mathematically satisfactory definition had been given at that time. ...
>
> The essential point is to define what a procedure is."

To formally capture this crucial informal concept, Gödel, who was giving an introductory lecture, began with a definition of the primitive recursive functions (which he quickly proved inadequate by a diagonal argument) and then his own Herbrand-Gödel recursive functions on p. 167. (Gödel gave the Herbrand-Gödel recursive function definition rather than Turing machines because he knew they were equivalent. He intended his talk to be as elementary as possible for a general audience, which he knew would be more familiar with recursion.) Gödel continued on p. 168,

> "That this really is the correct definition of mechanical computability was established beyond any doubt by Turing."

The word "this" evidently refers to the recursive functions. Gödel knew that Turing had never proved anything about recursive functions. What did he mean? Gödel knew that by work of Turing, Church, and Kleene, the formal classes of Turing computable functions, recursive functions, and λ-definable functions all coincided. Gödel was asserting that it was Turing who had demonstrated that these formal classes captured the informal notion of a procedure. It was Turing's proof [1936] and the formal equivalences which had elevated Herbrand-Gödel recursive functions to a correct characteristic of effectively calculable functions, not that the Herbrand-Gödel recursive functions had elevated Turing computable functions. Indeed Gödel had seen Church's Thesis 2.2 expressed in terms of Herbrand-Gödel recursive functions, and had rejected it in 1935 and 1936 because he was not sure his own definition had captured the informal concept of procedure.

Gödel had begun with the recursive function as more easily explained to a general audience, but having introduced Turing, Gödel now went forward with Turing's work.

> "But Turing has shown more. He has shown that the computable functions defined in this way are exactly those for which you can construct a machine with finite number of parts which will do the following thing. If you write down any number $n_1, \ldots n_r$, on a slip of paper and put the slip into the machine and turn the crank then after finite number of turns the machine will stop and the value of the function for the argument $n_1, \ldots n_r$ will be printed on the paper."

3.2.2 Princeton Bicentennial [1946]

To fully understand this article one should be familiar with Gödel's *Über die Länge von Beweisen*" ("On the length of proofs) [1936a] in Volume I [1986, p. 396–398]. Gödel discussed what it means for a function to be computable in a formal system S and remarked that given a sequence of formal systems S_i, S_{i+1} ...it is possible that passing from one formal system S_i to one of higher order S_{i+1} not only allows us to prove certain propositions which were not provable before, but also makes it possible to shorten by an extraordinary amount proofs already available.

Now for Gödel's Princeton Bicentennial address [1946] refer to the Collected Works Volume II, Gödel [1990, p. 150]. Gödel muses on the remarkable fact of the absoluteness of computability, that it is not necessary to distinguish orders (different formal systems). Once we have a sufficiently strong system (such as Peano arithmetic) we can prove anything about computable functions which could be proved in a more powerful system. Once again, Gödel identifies those formal systems known to be equivalent, general recursiveness, Turing computability, and others, as a single formal system.

> "Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness (or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemogical notion, *i.e.*, one not depending on the formalism chosen.[2] In all other cases treated previously, such as demonstrability or definability, one has been able to define them only relative to a given language, and for each individual language it is clear that the one thus obtained is not the one looked for. For the concept

[2]... "A function of integers is computable in any formal system containing arithmetic if and only if it is computable in arithmetic" ....

> of computability, however, although it is merely a special kind of demonstrability or decidability, the situation is different. By a kind of miracle it is not necessary to distinguish orders, and the diagonal procedure does not lead outside the defined notion."

Gödel stated,

> "... *one* has for the first time succeeded in giving an absolute definition of an interesting epistemogical notion ..."

Who is the *"one"* who has linked the informal notion of procedure or effectively calculable function to one of the formal definitions. This becomes irrefutably clear in §3.2.5.

3.2.3 The Flaw in Church's Thesis

Gödel himself was the first to provide one of the formalisms later recognized as a definition of computability, the general recursive functions. However, Gödel himself never claimed to have made this link. Church claimed it in his announcement of Church's Thesis in 1935 and 1936, but Gödel did not accept it then and gave no evidence later of believing that Church had done this. Modern scholars found weaknesses in Church's attempted proof [1936] that the recursive functions constituted all effectively calculable functions.

If the basic steps are stepwise recursive, then it follows easily by the Kleene Normal Form Theorem which Kleene had proved and communicated to Gödel before November, 1935 (see Kleene [1987b], p. 57]), that the entire process is recursive. The fatal weakness in Church's argument was the core assumption that the atomic steps were stepwise recursive, something he did not justify. Gandy [1988, p. 79] and especially Sieg [1994, pp. 80, 87] in their excellent analyses brought out this weakness in Church's argument. Sieg [p. 80] wrote, "...this core does not provide a convincing analysis: steps taken in a calculus must be of a restricted character and they are assumed, for example by Church, without argument to be recursive." Sieg [p. 78] wrote, "It is precisely here that we encounter the major stumbling block for Church's analysis, and that stumbling block was quite clearly seen by Church," who wrote that without this assumption it is difficult to see how the notion of a system of logic can be given any exact meaning at all. It is exactly this stumbling block which Turing overcame by a totally new approach.

3.2.4 Gödel on Church's Thesis

Gödel may not have found errors in Church's demonstration, but he never gave any hint that he thought Church had been the first to show that the recursive functions coincided with the effectively calculable ones. On the contrary, Gödel said,

> "As for previous equivalent definitions of computability, which, however, are much less suitable for our purpose, [*i.e.*, verifying the Computability Thesis], see A. Church *1936*, pp. 256–358." *– Gödel, Princeton Bicentennial, [1946, p. 84], and Gödel Collected Works, Vol. I, pp 150–153.*

3.2.5 Gödel's Letter to Kreisel [1968]

-Gödel: letter to Kreisel of May 1, 1968 [Sieg, 1994, p. 88].

> " But I was completely convinced only by Turing's paper."

3.2.6 Gibbs Lecture [1951]

Gödel, Collected Works Volume III, [Gödel, 1995, p.304–305].
Martin Davis in his introduction wrote,

> "On 26 December 1951, at a meeting of the American Mathematical Society at Brown University, Gödel delivered the twenty-fifth Josiah Willard Gibbs Lecture, "Some basic theorems on the foundations of mathematics and their implications." ...

It is probable as Wang suggests ([1987, pages 117–18]), that the lecture was the main project Gödel worked on in the fall of 1951. ... "

Gödel [1951] in his Gibbs lecture wrote,

> "Research in the foundations of mathematics during the past few decades has produced some results of interest, not only in themselves, but also with regard to their implications for the traditional philosophical problems about the nature of mathematics. The results themselves, I believe, are fairly widely known, but nevertheless I think it will be useful to present them in outline once again, especially in view of the fact that due to the work

> of various mathematicians, they have taken on a much more satisfactory form than they had had originally. The greatest improvement was made possible through the precise definition of the concept of finite procedure, [" ... equivalent to the concept of a 'computable function of integers' ..."] which plays a decisive role in these results. There are several different ways of arriving at such a definition, which, however, all lead to exactly the same concept. The most satisfactory way, in my opinion, is that of reducing the concept of a finite procedure to that of a machine with a finite number of parts, as has been done by the British mathematician Turing."

In this one paragraph,

1. Gödel stressed the importance of the results to mathematics and philosophy.
2. Gödel gave full credit to Turing and his "machine with a finite number of parts" for capturing the concept of finite procedure.
3. Gödel never mentions Church or Gödel's own definition of recursive functions.

This is one of the most convincing and explicit demonstrations of Gödel's opinion of Turing's work.

3.2.7 Godel's Postscriptum 3 June, 1964 to Gödel [1934]

-Godel's Postscriptum 3 June, 1964 to Gödel [1934], see "The Undecidable," M. Davis, [1965, p. 71] and Gödel Collected Works, Volume I [1986, p. 369–370].

> " In consequence of later advances, in particular of the fact that, due to A. M. Turing's work, a precise and unquestionably adequate definition of the general concept of formal system can now be given, the existence of undecidable arithmetical propositions and the non-demonstrability of the consistency of a system in the same system can now be proved rigorously for *every* consistent formal system containing a certain amount of finitary number theory."

Turing's work gives an analysis of the concept of "mechanical procedure" (alias "algorithm" or "computation procedure" or "finite combinatorial procedure"). This concept is shown to be equivalent with that of a "Turing machine."

3.3 Hao Wang Reports on Gödel

Hao Wang was a very close friend and professional colleague of Gödel, whom he called "G" in the following passage. Wang [1987, p. 96] wrote about Gödel's opinion of Turing's work.

> "Over the years G habitually credited A.M. Turing's paper of 1936 as the definitive work in capturing the intuitive concept [of computability], and did not mention Church or E. Post in this connection. He must have felt that Turing was the only one who gave persuasive arguments to show the adequacy of the precise concept ... In particular, he had probably been aware of the arguments offered by Church for his "thesis" and decided that they were inadequate. It is clear that G and Turing (1912–1954) had great admiration for each other, ... "

3.4 Kleene's Remarks About Turing

> "Turing's computability is intrinsically persuasive" but "λ-definability is not intrinsically persuasive" and "general recursiveness scarcely so (its author Gödel being at the time not at all persuaded)."
>
> *-Stephen Cole Kleene [1981b, p. 49]*

> "Turing's machine concept arises from a direct effort to analyze computation procedures as we know them intuitively into elementary operations. Turing argued that repetitions of his elementary operations would suffice for any possible computation.
>
> " For this reason, Turing computability suggests the thesis more immediately than the other equivalent notions and so we choose it for our exposition."
>
> *-Stephen Cole Kleene, second book [1967, p. 233]*

3.5 Church's Remarks About Turing

> Computability by a Turing machine, " has the advantage of making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately—*i.e.*, without the necessity of proving preliminary theorems."
>
> *-Alonzo Church [1937], Review of Turing [1936]*

In modern times it is sometimes stated as follows, recognizing that Church [1935], [1936] got it first, but that Turing [1936] got it right, in the opinion of Gödel and many modern scholars.

Definition 3.2. Church-Turing Thesis. A function is intuitively computable if and only if it is computable by a Turing machine, or equivalently if it is specified by a recursive function.

We strongly believe that it should not be called any of the three (Church's Thesis, Turing's Thesis, or the Church-Turing Thesis) but rather should be called the *"Computability Thesis"* as we argue in §13 and §14, just as the calculus is named for neither of its discoverers, Newton and Leibniz.

4 Oracle Machines and Relative Computability

After introducing definitions of computable functions: Turing a-machines, recursive functions, and λ-definable functions; the originators continued during 1936–1939 to explore *incomputable* phenomena, rather than computable applications of these devices, which came only a decade or more later. Church and Kleene [1936] as well as Church [1938] and Kleene [1938] studied computable well-orderings and defined recursive ordinals which were later used to extend the jump operation to the arithmetic hierarchy and beyond to the hyperarithmetic hierarchy up to the first nonrecursive ordinal ω_1^{CK}.

Turing spent 1936–1938 at Princeton working on a Ph.D. with Church. His thesis was completed in 1938 and published in Turing [1939]. Church and other mathematicians had found Gödel's Incompleteness Theorem unsettling. By Gödel's proof an effective extension T_1 of Peano arithmetic cannot prove its own consistency con_{T_1}. However, we can add the arithmetical statement con_{T_1} to T_1 to get a strictly stronger theory T_2. Continuing, we can get an increasing hierarchy of theories $\{T_\alpha\}_{\alpha\in S}$ over a set S of ordinals. Turing's Ph.D. thesis [1939] concerned such an increasing array of undecidable theories.

4.1 Turing's Oracle Machines

In one of the most important and most obscure parts of all of computability theory, Turing wrote in his ordinal logics paper [1939, §4] a short statement about oracle machines.

> "Let us suppose we are supplied with some unspecified means of solving number-theoretic problems; a kind of oracle as it were. ... this oracle ... cannot be a machine.
>
> With the help of the oracle we could form a new kind of machine (call them *o*-machines), having as one of its fundamental processes that of solving a given number-theoretic problem."

This is virtually all Turing said of oracle machines. His description was only a page long and half of that was devoted to the unsolvability of related problems such as whether an *o*-machine will output an infinite number of 0's or not.

In 1939 Turing left this topic never to return. It mostly lay dormant for five years until it was developed in a beautiful form by Post [1944], [1948], and other papers as we shall explain in §5 and §6. Before doing so, we complete this section §4.1 with a modern treatment of oracle machines and Turing functionals including some of the more important properties even though these were mostly discovered much later, even after Post. More modern properties of Turing functionals, such as effective continuity on Cantor space, will be developed in §7.6.

4.2 Modern Definitions of Oracle Machines

There are several equivalent ways that a Turing machine with oracle may be defined. We prefer the definition in Soare's book [1987, p. 46] of a machine with a head which reads the work tape and oracle tape simultaneously, but many other formulations produce the same class of functionals.

Definition 4.1. A *Turing oracle machine (o-machine)* is a Turing machine with an extra "read only" tape, called the *oracle tape*, upon which is written the characteristic function of some set A (called the *oracle*), and whose symbols cannot be printed over. The old tape is called the *work tape* and operates just as before. The reading head moves along both tapes simultaneously. As before, Q is a finite set of states, $S_1 = \{B, 0, 1\}$ is the oracle tape alphabet, $S_2 = \{B, 1\}$ is the work tape alphabet, and $\{R, L\}$ the set of

head moving operations right and left. A *Turing oracle program* $\widetilde{P}_e$ is now simply a partial map,

$$\delta : Q \times S_1 \times S_2 \longrightarrow Q \times S_2 \times \{R, L\},$$

where $\delta(q, a, b) = (p, c, X)$ indicates that the machine in state q reading symbol a on the oracle tape and symbol b on the work tape passes to state p, prints "c" over "b" on the work tape, and moves one space right (left) on both tapes if $X = R$ ($X = L$). The other details are just as previously in Soare [1987]. The Turing oracle program $\widetilde{P}_e$ takes some oracle A and defines a partial A-computable functional $\Phi_e^A(x) = y$.

Notation 4.2. (i) We let lower case Greek letters φ, ψ represent partial functions from ω to ω and lower case Latin letters f, g, and h represent total functions.

(ii) We let upper case Greek letters represent partial *functionals* from 2^ω to 2^ω. If $A \subseteq \omega$ then $\Psi^A(x)$ may be defined for some or all x. If $B \subseteq \omega$ we write $\Psi^A = B$ if $\Psi^A(x) = B(x)$ for all $x \in \omega$.

(iii) As in Soare [1987] we use $\{P_e\}_{e\in\omega}$ for an effective listing of Turing programs for Turing a-machines and let φ_e be the partial computable function defined by P_e. We let $\{\widetilde{P}_e\}_{e\in\omega}$ be an effective listing of oracle Turing programs, and let Φ_e be the computable partial functional defined by $\widetilde{P}_e$. If Φ_e^A is total and computes B we say B is *computable in* A and write $B \leq_{\mathrm{T}} A$. We refer to φ_e as a partial computable *function* ω to ω because its input and output are integers. On the other hand, Φ_e^A is called a partial computable *functional* because it takes a set A to a set B and is viewed as a map on Cantor Space 2^ω.

(iv) Since Rogers book [1967], researchers have used $\varphi_e(x)$ or $\{e\}(x)$ for the partial computable function with program P_e. Since Lachlan about 1970, researchers have used $\Phi_e^A(x)$ for the Turing functional with oracle program $\widetilde{P}_e$ and have used $\varphi_e^A(x)$ for the *use function,* the maximum element of A examined during the computation. Lachlan also used matched pairs Ψ, ψ; Γ, γ and so forth for partial computable functionals and their use functions in many papers, and this is the general usage today. There is no confusion between the notation $\varphi_e(x)$ as a partial computable function and $\varphi_e^A(x)$ as a use function for $\Phi_e^A(x)$ because the former $\varphi_e(x)$ will never have an exponent A and the latter use function $\varphi_e^A(x)$ always will.

4.2.1 The Graph of a Partial Computable Function

Definition 4.3. Given a partial computable (p.c) function φ_e define the *graph* of φ_e as follows.

$$(1) \quad g_e \;=\; \text{graph}(\varphi_e) \;=_{\text{dfn}}\; \{\, \langle x, y\rangle \;:\; \varphi_e(x) = y \,\}$$

Note that if φ_e is a partial computable (p.c.) function then graph(φ_e) is a computable enumerable (c.e.) set. Likewise, given any (c.e.) set W_e we can find a single-valued c.e. subset $V_e \subseteq W_e$ which is the graph of a p.c. function. The the notions of a Turing program to compute a p.c. function ψ and a description of its graph are interchangeable and equally powerful in describing ψ.

4.2.2 The Graph of an Oracle Computable Functional

Definition 4.4. For an oracle machine program $\widetilde{P}_e$ we likewise define the *oracle graph* of the corresponding computable functional Φ_e but now taking into consideration the finite strings read by the oracle head during the computation.

$$(2) \qquad G_e \;=_{\text{dfn}}\; \text{graph}(\Phi_e) \;=_{\text{dfn}}\; \{\, \langle \sigma, x, y\rangle \;:\; \Phi_e^{\sigma}(x) = y \,\}$$

where σ ranges over $2^{<\omega}$,

Here $\Phi_e^{\sigma}(x) = y$ denotes that the oracle program $\widetilde{P}_e$ with oracle σ on its oracle tape, and x on its input tape, eventually halts and outputs y, and does not read more of the oracle tape than σ during the computation. The crucial property of the oracle graph G_e and the one which makes a Turing functional Φ_e independent of any particular machine representation is the following.

4.3 The Oracle Graph Theorem

Theorem 4.5. Oracle Graph Theorem. *If $\widetilde{P}_e$ is an oracle Turing program defining a Turing functional Φ_e, then the graph G_e defined in* (2) *is a computably enumerable (c.e.) set.*

Proof. From the definitions G_e is Σ_1^0 and therefore c.e. □

The converse holds for a c.e. set V which satisfies a singlevaluedness condition (3) and a continuity condition (4).

Theorem 4.6. *Let $V \subset 2^{<\omega} \times \omega \times \omega$ be a computably enumerable set which satisfies the following two conditions. Then there is a Turing functional Φ_e such that $G_e = V$.*

(3) $\langle \sigma, x, y \rangle \in V \quad \Longrightarrow \quad \neg(\exists \tau \supseteq \sigma)(\exists z)[\ z \neq y \quad \& \quad \langle \tau, x, z \rangle \in V\ ].$

(4) $\langle \sigma, x, y \rangle \in V \quad \Longrightarrow \quad (\forall \tau \supset \sigma)[\ \langle \tau, x, y \rangle \in V\ ].$

Proof. Given V satisfying (3) and (4) define a partial computable functional $\Psi^\sigma(x)$ as follows. If $\langle \sigma, x, y \rangle$ appears in V define $\Psi^\sigma(x) = y$. This defines an effective reduction Ψ. There must be a Turing oracle machine Φ_e such that $\Phi_e = \Psi$. □

Theorem 4.7. *Furthermore, given any c.e. set $U \subseteq 2^{<\omega} \times \omega \times \omega$ there is a c.e. subset $V \subseteq U$ satisfying* (3) *and* (4) *and having the same domain,* i.e.*,*

$$\{x : (\exists \sigma)(\exists y)[\ \langle \sigma, x, y \rangle \in U\ ]\} \quad = \quad \{x : (\exists \sigma)(\exists y)[\ \langle \sigma, x, y \rangle \in V\ ]\}.$$

Proof. Similar to single valuedness theorem in Soare [1987, Chapter II]. □

4.4 Equivalent Definitions of Relative Computability

There are several different formal definitions of relative computability. This includes an oracle machine with a single reading head reading the work tape and oracle tape, or two independent reading heads, or other variations. In addition, several authors define relative computability from oracle A by adding the characteristic function of A either to the Herbrand-Gödel general recursive function definition or to the Kleene μ-recursive function definition. Each of these formal definitions produces a c.e. graph G_e and these definitions are all equivalent.

Furthermore, any Turing a-machine can clearly be simulated by a Turing o-machine as we note in the following theorem. Therefore, in presenting the subject we can bypass a-machines altogether, present o-machines, and then draw a-machines as special cases. This reinforces the claim that it is the o-machine, not the a-machine, which is the central concept of the subject.

Theorem 4.8. *If P_e is a Turing program for a Turing a-machine, then there is a Turing oracle program $\widetilde{P}_i$ which on input x and any oracle A produces the same output y.*

Proof. Let P_e be a Turing program to compute φ_e. Now P_e consists of a finite partial map which can be identified with a set of 5-tuples,

$$\delta : Q \times S_2 \longrightarrow\ Q \times S_2 \times \{R, L\},$$

where Q is a finite set of states, $S_2 = \{B, 1\}$ is the work tape alphabet, and $\{R, L\}$ the set of head moving operations right and left. Define an oracle program $\widetilde{P}_i$ as follows with transition function

$$\widetilde{\delta} : Q \times S_1 \times S_2 \longrightarrow Q \times S_2 \times \{R, L\},$$

for $S_1 = \{B, 0, 1\}$ the oracle tape alphabet as follows. For each line in P_e of the form $\delta(q, a) = (p, b, X)$ for p, $q \in Q$ and a, $b \in S_2$, we add to oracle program $\widetilde{P}_i$ a line $\widetilde{\delta}(q, c, a) = (p, b, X)$ for both $c = 0$ and $c = 1$. Hence, $\widetilde{P}_i$ has exactly the same effect on input x as P_e regardless of the oracle A. $\square$

4.4.1 Notation for Functions and Functionals

The standard notation is that given above.

Remark 4.9. Note that the oracle Turing machine Φ_e is a finite object represented by an oracle program $\widetilde{P}_e$ or an oracle graph G_e and has no oracle associated with it, but it can use any oracle A which may be attached. This is analogous to a laptop computer with no active connection to a database which may later be connected to the World Wide Web.

Recently, some researchers have unfortunately used Φ_e to denote φ_e. This is unwise because it blurs the distinction of types in which φ_e operates on integers and Φ_e on sets. Furthermore, sometimes we would like to write Φ_e alone without its exponent A to identify it with $\widetilde{P}_e$ or its oracle graph G_e as a finite object, like a laptop computer whose link with the web has temporarily been removed. Doing so under the new convention leads to confusion with φ_e which is given by a different type of program. The functional Φ_e is defined by a program $\widetilde{P}_e$ which is a finite set of 6-tuples operating on sets, while φ_e is defined by P_e a finite set of 5-tuples operating on integers. Furthermore, there is no justification for the necessity of Φ_e to denote φ_e since the current notation φ_e is quite satisfactory. We recommend against using Φ_e to denote φ_e the partial computable function.

5 Emil Post Expands Turing's Ideas

The spirit of Turing's work was taken up by the American mathematician Emil Post, who had been appointed to a faculty position at City College of New York in 1932.

5.1 Post's Work in the 1930's

Post [1936] independently of Turing (but not independently of the work by Church and Kleene in Princeton) had defined a *"finite combinatory process"* which closely resembles a Turing machine. From this it is often and erroneously written (Kleene [1987b, p. 56], and [1981, p. 61]) that Post's contribution here was "essentially the same" as Turing's, but in fact it was much less. Post did not attempt to prove that his formalism coincided with any other formalism, such as general recursiveness, but merely expressed the expectation that this would turn out to be true, while Turing [1937b] proved the Turing computable functions equivalent to the λ-definable ones. Post gave no hint of a universal Turing machine. Most important, Post gave no analysis, as did Turing, of why the intuitively computable functions are computable in his formal system. Post offers only as a "working hypothesis" that his contemplated "wider and wider formulations" are "logically reducible to formulation 1." Lastly, Post, of course, did not prove the unsolvability of the *Entscheidungsproblem* because at the time Post was not aware of Turing's [1936], and Post believed that Church [1936] had settled the *Entscheidungsproblem*. Furthermore, Post wrote [1936] that Church's identification of effective calculability and recursiveness was working hypothesis which is in "need of continual verification." This irritated Church who criticized it in his review [1937b] of Post [1936].

Post's contributions during the 1930's were original and insightful, corresponding in spirit to Turing's more than to Church's, but they were not as influential as those of Church and Turing. It was only during the next phase from 1940 to 1954 that Post's remarkable influence was fully felt.

5.2 Post Steps Into Turing's Place During 1940–1954

As Turing left the subject of pure computability theory in 1939, his mantle fell on the shoulders of Post. This was the mantle of clarity and intuitive exposition, the mantle of exploring the most basic objects such as computably enumerable sets, and most of all, the mantle of relative computability and Turing reducibility. During the next decade and a half from 1940 until his death in 1954, Post played an extraordinary role in shaping the subject.

Post [1941] and [1943] introduced a *second* and unrelated formalism called a *production* system and (in a restricted form) a *normal* system, which he explained again in [1944]. Post's (normal) canonical system is a *generational* system, rather than a *computational* system as in general recursive functions or Turing computable functions, because it gives an al-

gorithm for generating (listing) a set of integers rather than computing a function. This led Post to concentrate on *effectively enumerable sets* rather than computable functions. Post, like Church and Turing, gave a thesis [1943, p. 201], but stated it in terms of generated sets and production systems, which asserted that "any generated set is a normal set." That is, any effectively enumerable set in the intuitive sense could be produced as a normal set is his formal system. Although he had used other terminology earlier, by the 1940's Post had adopted the Kleene-Church terminology of "recursively enumerable set" for the formal equivalent of Post's effectively enumerable set.

Definition 5.1. [Post's Thesis, 1943, 1944]. A nonempty set is effectively enumerable (listable in the intuitive sense) iff it is recursively enumerable (the range of a recursive function) or equivalently iff it is generated by a (normal) production system.

Post showed that every recursively enumerable set (one formally generated by a recursive function) is a normal set (one derived in his normal canonical system) and conversely. Therefore, normal sets are formally equivalent to recursively enumerable sets. Since recursively enumerable sets are equidefinable with partial computable functions, this definition of normal set gives a new formal definition of computability which is formally equivalent to the definitions of Church or Turing. (Equidefinable here means that from the definition of a partial computable function we can derive a c.e. set as its range and from the definition of a c.e. set one can find a single valued c.e. subset which is the graph of a partial computable function.) Post's Thesis is equivalent to Turing's Thesis.

Post used the terms "effectively enumerable set" and "generated set" almost interchangeably, particularly for sets of positive integers. Post [1944, p. 285], like Church [1936], defined a set of positive integers to be *recursively enumerable* if it is the range of a recursive function and then stated, "The corresponding intuitive concept is that of an *effectively enumerable* set of positive integers." (This is Church's [1936] terminology also). Post [1944, p. 286], explained his informal concept of a "generated set" of positive integers this way,

> "Suffice it to say that each element of the set is at some time written down, and earmarked as belonging to the set, as a result of predetermined effective processes. It is understood that once an element is placed in the set, it stays there."

Post then [1944, p. 286], restated Post's Thesis 5.1 in the succinct form,

"every generated set of positive integers is recursively enumerable."

He remarked that "this may be resolved into the two statements: every generated set is effectively enumerable, every effectively enumerable set of positive integers is recursively enumerable." Post continued, "their converses are immediately seen to be true." Post's concentration on *c.e. sets* rather than partial computable *functions* may be even more fundamental than the thesis of Church and Turing characterizing computable functions because Sacks [1990] has remarked that often in higher computability theory it is more convenient to take the notion of a generalized c.e. set as basic and to derive generalized computable functions as those whose graphs are generalized computably enumerable.

5.3 Post's Problem on Incomplete C.E. Sets

Post's most influential achievement during this period was the extraordinarily clear and intuitive paper, *Recursively enumerable sets of positive integers and their decision problems*, [1944]. Here Post introduced the terms *degree of unsolvability* and the concept that one set has *lower degree of unsolvability* than another. Post later expanded on these definitions in [1948].

Post's paper [1944] revealed with intuition and great appeal the significance of the computably enumerable sets and the significance of Gödel's Incompleteness Theorem. Post called Gödel's diagonal set,

$$K = \{e : e \in W_e\}$$

the *complete set* because every c.e. set W_e is computable in K ($W_e \leq_{\mathrm{T}} K$). Moreover, Post felt that the creative property of K revealed the inherent creativeness of the mathematical process.

5.4 Post Began With Strong Reducibilities

Post posed his famous "Post's problem" of whether there exists a computably enumerable (c.e.) set W which is not computable but which cannot compute Gödel's diagonal set K, *i.e.*, such that $\emptyset <_{\mathrm{T}} W <_{\mathrm{T}} K$. In 1944 researchers did not understand Turing reducibility, even as little as presented above in §4. Post himself was struggling to understand it, and did not explicitly discuss it until the very end of his paper, and even then only in general terms.

Post's contributions from 1943 to 1954 concerning relative computability are remarkable. First, Post resurrected in [1944] the concept of oracle machines which had been buried in Turing's [1939] paper and which other researchers had apparently ignored for five years. Second, Post defined a sequence of strong reducibilities to better understand the concept of a set B being reducible to a set A.

Along with these strong reducibilities, Post defined families of c.e. sets with thin complements, simple, hyper-simple, hyper-hypersimple, in an attempt to find an incomplete set for these reducibilities. These concepts have pervaded the literature and proved useful and interesting, but they did not lead to a solution of Post's problem. Post was able to exhibit incomplete incomputable c.e. sets for several of these stronger reducibilities but not for Turing reducibility. Post's Problem stimulated a great deal of research in the field and had considerable influence.

Slowly Post's understanding deepened of the general case of one set B being reducible (Turing reducible) to another set A. Post steadily continued gaining a deeper and deeper understanding from 1943 to 1954 until he had brought it to full development. Our modern understanding of relative computability and Turing functionals is due more to Post and his patient, persistent efforts over a decade and a half than it is due to the brief remark by Turing in 1939.

6 Post Highlights Turing Computability

When Post wrote his famous paper [1944], Turing's notion of relative computability from an oracle discussed in §4.1 had been mostly ignored for five years. It was only at the end of Post's paper [1944] in the last section, §11 *General (Turing) Reducibility,* that Post defined and named for the first time "Turing Reducibility," denoted $B \leq_{\mathrm{T}} A$, and began to discuss it in intuitive terms. Post's four and a half page discussion there is the most revealing introduction to effective reducibility of one set from another. In the same crisp, intuitive style as in the rest of the paper, Post described the manner in which the decision problem for one set S_1 could be reduced to that of a second set S_2. Post wrote it for a c.e. set S_2 in studying Post's problem, but the analysis holds for any set S_2.

6.1 Post Articulates Turing Reducibility

Post wrote in [1944, §11],

> "Now suppose instead, says Turing [1939] in effect, this situation obtains with the following modification. That at certain times the otherwise machine determined process raises the question is a certain positive integer in a given recursively enumerable set S_2 of positive integers, and that the machine is so constructed that were the correct answer to this question supplied on every occasion that arises, the process would automatically continue to its eventual correct conclusion. We could then say that the machine effectively reduces the decision problem of S_1 to that of S_2. Intuitively, this would correspond to the most general concept of reducibility of S_1 to S_2. For the very concept of the decision problem of S_2 merely involves the answering for an arbitrarily given single positive integer m of the question is m in S_2; and in a finite time but a finite number of such questions can be asked. A corresponding formulation of "Turing reducibility" should then be the same degree of generality for effective reducibility as say general recursive function is for effective calculability."

6.2 The Post-Turing Thesis

Post's statement may be restated in succinct modern terms and incorporates the statement implicit in Turing [1939, §4] in the following extension of Turing's first Thesis 3.1 and Post's first Thesis 5.1.

Definition 6.1. Post-Turing Thesis, **Turing [1939 §4], Post [1944, §11].** A set B is effectively reducible to another set A iff B is Turing reducible to A by a Turing oracle machine ($B \leq_{\mathrm{T}} A$).

Turing's brief introduction of oracles did not state this as a formal thesis, but it is largely implied by his presentation. Post makes it explicit and claims that this is the formal equivalent of the intuitive notion of *effectively reducible,* a step as significant as the Church-Turing characterization of "calculable." If we identify a Turing reduction Φ_e with its graph G_e both informally and formally then the Post-Turing Thesis is equivalent to Post's Thesis 5.1 (because G_e is c.e.), which is equivalent to Turing's Thesis 3.1.

However, there has been little analysis (along the lines of the extensive analysis of the Church-Turing Thesis 3.2 for unrelativized computations) of what constitutes a relative computation of B from A. This is surprising because the Post-Turing Thesis was stated clearly in Post [1944]. It is even more surprising because relative computability is used much more often than

ordinary computability in the theory of computability, applications of computability to other areas such as algebra, analysis, model theory, algorithmic complexity and many more. Also interactive or online computing in the real world is more common than batch processing or offline computing, using processes contained entirely inside the machine.

7 Continuous and Total Functionals

7.1 Representations of Open and Closed Sets

Definition 7.1. (i) Using ordinal notation identify the ordinal 2 with the set of smaller ordinals $\{0, 1\}$. Identify the sets $A \subseteq \omega$ with their characteristic functions $f : \omega \to \{0, 1\}$ and represent the set of such f as 2^ω.

(ii) Let $2^{<\omega}$ denote the set of finite strings of $0'$s and 1's, *i.e.*, finite initial segments of functions $f \in 2^\omega$. Let σ and τ represent finite strings in $2^{<\omega}$ and $\sigma \prec \tau$ or $\sigma \prec f$ denote that σ is an initial segment of τ or f.

(iii) Recall the definition of a finite set D_y with canonical index y where $D_y = \{x_1 < x_2, \ldots < x_k\}$ and $y = 2^{x_1} + 2^{x_2} \ldots = 2^{x_k}$. Define the string σ_y with canonical index y by $\sigma_y(z) = 1$ if $z \in D_y$ and $\sigma_y(z) = 0$ otherwise.

(iv) *Cantor space* is 2^ω with the following topology (class of open sets). For every $\sigma \in 2^{<\omega}$ the *basic open (clopen) set*

$$\mathcal{N}_\sigma \;=\; \{\, f \;:\; f \in 2^\omega \quad \& \quad \sigma \prec f \,\}.$$

The sets $\mathcal{N}_\sigma$ are called *clopen* because they are both closed and open. The *open* sets of Cantor space are the countable unions of basic open sets.

(iv) Set $A \subseteq 2^{<\omega}$ is an *open representation* of the open set $\mathcal{N}_A = \bigcup_{\sigma \in A} \mathcal{N}_\sigma$. We may assume A is closed *upwards*, *i.e.*, $\sigma \in A$ and $\sigma \prec \tau$ implies $\tau \in A$.

(v) A set $\mathcal{C}$ is (topologically) *closed* if its complement $\mathcal{N}_A$ is open, *i.e.*, $\mathcal{C} = \overline{\mathcal{N}}_A = (2^\omega - \mathcal{N}_A)$. In this case $T =_{\text{dfn}} 2^{<\omega} - A$ is a *closed representation* for $\mathcal{C}$. Now T is closed *downwards* (because A is closed upwards). Hence, we shall see that T forms a *tree* as in Definition 7.2 (i).

7.2 Notation for Trees

Definition 7.2. (i) A *tree* $T \subseteq 2^{<\omega}$ is a set closed under initial segments, *i.e.*, $\sigma \in T$ and $\tau \prec \sigma$ imply $\tau \in T$. Fix any tree T.

(ii) The set of *infinite paths* through T is

(5) $\quad [\,T\,] = \{\, f : (\forall n)\,[\, f\restriction n \in T\,]\,\}.$

Note that $[\,T\,]$ is always a closed set. If $\mathcal{C} \subseteq 2^\omega$ is any closed set then by Theorem 7.3 there is a (nonunique) tree $T \subseteq 2^{<\omega}$ such that $\mathcal{C} = [\,T\,]$, which is called a *closed representation* for $\mathcal{C}$.

(iii) For $\sigma \in T$ define the subtree T_σ of nodes $\tau \in T$ *comparable with* σ,

(6) $\quad T_\sigma = \{\, \tau \;:\; \sigma \preceq \tau \quad \text{or} \quad \tau \prec \sigma \,\}.$

(iv) Define the subtree of *extendible nodes* $\sigma \in T$.

(7) $\quad T^{\rm ext} \;=\; \{\, \sigma \in T \;:\; (\exists f \succ \sigma)[\, f \in [\,T\,]\,]\,\}$

Note that if the tree T is computable then $T^{\rm ext}$ is co-c.e. Usually for a given tree T there are many other trees T' such that $[\,T\,] = [\,T'\,]$, *i.e.*, many different representations for the same closed set $[\,T\,]$. The closed sets are closed under finite union and countable intersection since the open sets have the dual properties in Definition 7.1 (iii), closure under finite intersection and countable union. The clopen sets are both open and closed, so any countable union of them is open and any countable intersection is closed.

Theorem 7.3. *If T is a tree, then $[\,T\,]$ is a closed set, and for every closed set $\mathcal{C}$ there is a tree T such that $\mathcal{C} = [\,T\,]$.*

Proof. ($\Longrightarrow$). Given tree T let $A = 2^{<\omega} - T$. Then $\mathcal{N}_A$ is open. Therefore, $[\,T\,] = 2^\omega - \mathcal{N}_A$ and $[\,T\,]$ is closed.

($\Longleftarrow$). Given any closed $\mathcal{C}$ with complement $\mathcal{N}_A$ define T by putting σ in T if $(\forall \tau \preceq \sigma)[\,\tau \notin A\,]$. Then T is closed downward and $[\,T\,] = \mathcal{C}$. □

Theorem 7.4. [Compactness]. *The following very easy and well-known properties hold for Cantor Space 2^ω. The term* "compactness" *refers to any of them, but particularly to (iv). These properties can be proved from one another but here we give direct proofs of each.*

(i) **König's Lemma**. *If $T \subseteq 2^{<\omega}$ is an infinite tree, then $[\,T\,] \neq \emptyset$.*

(ii) *If $T_0 \supseteq T_1 \ldots$ is a decreasing sequence of trees with $[\,T_n\,] \neq \emptyset$ for every n, and intersection $T_\omega = \cap_{n\in\omega} T_n$, then $[\,T_\omega\,] \neq \emptyset$.*

(iii) *If $\{\mathcal{C}_i\}_{i\in\omega}$ is a countable family of closed sets such that $\cap_{i\in F} \mathcal{C}_i \neq \emptyset$ for every* finite *set $F \subseteq \omega$, then $\cap_{i\in\omega} \mathcal{C}_i \neq \emptyset$ also.*

(iv) **Finite subcover**. *Any open cover $\{\mathcal{N}_\sigma\}_{\sigma\in A} \supseteq 2^\omega$ has a finite open subcover $\{\mathcal{N}_\sigma\}_{\sigma\in F} \supseteq 2^\omega$ for some finite subset $F \subseteq A$.*

7.3 Dense Open Subsets of Cantor Space

Definition 7.5. Let $\mathcal{S}$ be Cantor space 2^ω or Baire space ω^ω.

(i) A set $\mathcal{A} \subseteq \mathcal{S}$ is *dense* if $(\forall \sigma)\,(\exists f \succ \sigma)\,[\, f \in \mathcal{A}\,]$.

(ii) A set $\mathcal{A} \subseteq \mathcal{S}$ is *dense open* if

(8) $\quad (\forall \tau)(\exists \sigma \succeq \tau)(\forall f \succ \sigma)\,[\, f \in \mathcal{A}\,]$.

(iii) Let $T \subseteq 2^{<\omega}$ be a tree. A point $f \in [\,T\,]$ is *isolated* in $[\,T\,]$ if

(9) $\quad (\exists \sigma)[\quad [\,T_\sigma\,] \;=\; \{\, f\,\} \quad]$.

We say that σ *isolates* f because $\mathcal{N}_\sigma \cap [\,T\,] = \{\, f\,\}$. If f is not isolated, then f is a *limit point.*

(iv) A space $\mathcal{S}$ is *separable* if it has a countable base of open sets. (Both Cantor space and Baire space are separable.)

(v) A class $\mathcal{B} \subseteq \mathcal{S}$ is G_δ, *i.e.*, boldface $\mathbf{\Pi^0_2}$, if $\mathcal{B} = \cap_i \mathcal{A}_i$ a countable intersection of open sets $\mathcal{A}_i$.

After open and closed sets, much attention is paid in point set topology to G_δ sets (see Oxtoby[1971]). If the open sets $\mathcal{A}_i$ are also *dense open*, then they have special significance. Banach-Mazur games can be used for finding a point $f \in \cap_i \mathcal{A}_i$ where the the sets $\mathcal{A}_i$ are dense and open. This is the paradigm for the *finite extension* constructions in computability theory and oracle constructions as Kleene and Post, especially the Finite Extension Paradigm of Kleene and Post, which we use to construct sets and degrees meeting an countable sequence of "requirements." Meeting a given requirement R_i amounts to meeting the corresponding dense open set $\mathcal{A}_i$.

7.4 Effectively Open and Closed Sets

Definition 7.6. (i) If $A \subseteq 2^{<\omega}$ is c.e. and $\mathcal{A} = \mathcal{N}_A$, then $\mathcal{A}$ is *effectively (computably) open.*

(ii) If $\mathcal{C} = 2^\omega - \mathcal{N}_A$ for A c.e., or equivalently if $\mathcal{C} = [\,T\,]$ for a computable tree $T \subseteq 2^{<\omega}$, then $\mathcal{C}$ is *effectively (computably) closed*.

(iii) $\mathcal{C} \subseteq 2^\omega$ is a Π^0_1 *class* if there is a computable relation $R(x)$ such that

(10) $\quad \mathcal{C} = \{f : (\forall x)\, R(f(x))\}$.

We call this *lightface* Π^0_1 because it is defined in (10) by a universal quantifier outside of a *computable* relation $R(x)$.

(iv) A set $\mathcal{C} \subseteq 2^\omega$ is *boldface* $\mathbf{\Pi_1}$ if there is some set $S \subseteq \omega$ such that

$$\mathcal{C} = \{f : (\forall x)\, R^S(f(x))\},$$

and we say $\mathcal{C}$ is in Π^S_1. Here R^S denotes a relation computable in the set S which we call the *parameter* determining R^S. A set $\mathcal{A} \subset 2^\omega$ is *boldface* $\mathbf{\Sigma_1}$ or Σ^S_1 if its complement $2^\omega - \mathcal{A}$ is boldface $\mathbf{\Pi_1}$ or Π^S_1.

Theorem 7.7. *The open (closed) sets of* 2^ω *are exactly the* boldface $\mathbf{\Sigma_1}$ ($\mathbf{\Pi_1}$) *sets and any boldface set is lightface in some parameter* A*.*

Proof. If $\mathcal{A}$ is open, then $\mathcal{A} = \mathcal{N}_A$ for some countable set A. Now A determines exactly which σ contribute in the countable union $\mathcal{A} = \cup_{\sigma \in A} \mathcal{N}_\sigma$. Hence, if we fix A as a parameter, then the definition and properties become *lightface* Σ^A_1, *i.e.*, effectively open *relative* to the oracle A. (But this entire section is about working relative to an oracle.) □

We often convert an open set $\mathcal{A} = \mathcal{N}_A$ into the realm of computability theory as follows. We: (1) usually fix the parameter A and do a construction which is computable *relative* to the parameter A; (2) often replace the open set $\mathcal{N}_A$ by its complement the closed set $\mathcal{C} = \overline{\mathcal{N}}_A$; and (3) replace the closed set by $\mathcal{C} = [\,T^A\,]$ for a A-computable tree T. By fixing the parameter A we can apply all the methods of this chapter on A-computable constructions, such as the Recursion Theorem, construction of an A-c.e. set B, and so forth.

Theorem 7.8. *A class* $\mathcal{C}$ *is a* Π^0_1 *class iff* $\mathcal{C}$ *is effectively closed,* i.e.*,* $\mathcal{C} = [\,T\,]$ *for some computable tree* T*.*

Proof. ($\Longrightarrow$). Let $\mathcal{C} = \{\ f\ :\ (\forall x)\, R(f(x))\ \}$ for R computable. Define a computable tree $T = \{\ \sigma : (\forall \tau \subseteq \sigma)[R(\tau)]\ \}$. Then $[\,T\,] = \mathcal{C}$.

($\Longleftarrow$). Let $\mathcal{C} = [\,T\,]$ for T a computable tree. Define $R(\sigma)$ iff $\sigma \in T$. Then $\{f : (\forall x)\, R(f(x))\} = [\,T\,]$. □

7.5 Continuous Functions on Cantor Space

In elementary calculus courses a continuous function is usually defined with δ and ϵ concepts or with limits. In advanced analysis or topology courses

the more general definition is presented that a function F is continuous iff the image of every basic open set is open. We state continuity now for functionals on the Cantor space 2^ω . For arbitrary topological spaces an open set is an arbitrary union of basic open sets, but for Cantor space we may take countable unions.

Definition 7.9. (i) A functional Ψ on Cantor space 2^ω is *continuous* if for every $\tau \in 2^{<\omega}$ there is a countable set X_τ such that

$$\Psi^{-1}(\mathcal{N}_\tau) \quad = \quad \cup \{ \mathcal{N}_\sigma \ : \ \sigma \in X_\tau \}. \tag{11}$$

By identifying string σ_y with code number y defined in Definition 7.1 (iii) we can think of X_τ as a subset of ω.

(ii) A functional Ψ on Cantor space 2^ω is *total* if $\Psi^A(x)$ is defined for *every* $A \subseteq \omega$ and $x \in \omega$, *i.e.*, if $(\forall A)(\exists B)(\forall x)[\ \Psi^A(x) = B(x)\ ]$.

Theorem 7.10. *Let Ψ be a continuous functional on Cantor Space 2^ω. Then Ψ is total iff for each set X_τ of* (11) *there is a finite subset $\widehat{X}_\tau \subseteq X_\tau$ such that,*

$$2^\omega \quad \subseteq \quad \cup \{ \mathcal{N}_\sigma \ : \ \sigma \in \widehat{X}_\tau \}. \tag{12}$$

Proof. ($\Longleftarrow$). Given such $\widehat{X}_\tau$ for every τ we see that Ψ is total by (12).

($\Longrightarrow$). Assume Ψ is total. Then for every τ there is a countable set X_τ which covers 2^ω in the sense of (12). Now the Compactness Theorem asserts that any open cover of Cantor space 2^ω has a finite subcover. Therefore, we can replace every set X_τ by a finite subset $\widehat{X}_\tau$. □

Note that this involves only compactness and has no computable content, although it is usually presented in its effective analogue as the theorem that Φ_e is total iff it is a truth-table reduction.

7.6 Effectively Continuous Functionals

A *Turing* functional Φ_e is not only continuous but *effectively* continuous in the following sense.

Theorem 7.11. *For Φ_e define for every $\tau \in 2^{<\omega}$ the set of strings*

$$X^e_\tau \quad = \{ \sigma \ : \ (\exists s)(\forall x < |\tau|)\, [\ \Phi^\sigma_{e,s}(x)\!\downarrow \ = \ \tau(x)\ ] \}. \tag{13}$$

Then X^e_τ is c.e. and uniformly so in the sense that there is a computable function $h(e,\tau)$ such that $W_{h(e,\tau)} = X^e_\tau$.

Proof. Use the Oracle Graph G_e. Identify string σ with $\mathcal{N}_\sigma$. Now the set of strings $\{ X^e_\tau \ : \ \tau \in 2^\omega \}$ witnesses that the functional Φ_e is continuous. $\square$

Definition 7.12. Since X^e_τ is not only countable but also computably enumerable uniformly in e the functional Φ_e is *effectively continuous, i.e.*, that X^e_τ is a *computably enumerable* set of strings.

7.7 Continuous Functions are Relatively Computable

We showed that any Turing functional is continuous, indeed *effectively* continuous. Now we prove that any continuous functional on 2^ω is a Turing functional relative to some real parameter $X \subseteq \omega$ and therefore is *effectively* continuous *relative to* X.

Theorem 7.13. *Suppose Ψ is a continuous functional on 2^ω. Then Ψ is a Turing functional relative to some real parameter $X \subseteq \omega$.*

Proof. Since Ψ is continuous, the inverse image of every basic open set $\mathcal{N}_\tau$, $\tau \in 2^{<\omega}$, is open and therefore is a countable union of basic open sets. Hence, (identifying strings σ with their code numbers as integers) there is a set $X_\tau \subseteq \omega$ such that,

$$\Psi^{-1}\,(\,\mathcal{N}_\tau\,) \quad = \quad \cup\,\{\,\mathcal{N}_\sigma \ : \ \sigma \in X_\tau\,\}.$$

Therefore, the set $X = \oplus\{X_\tau : \tau \in 2^{<\omega}\}$ provides a complete oracle for calculating $\Psi : 2^\omega \to 2^\omega$. $\square$

8 Bounded Reducibilities

8.1 A Matrix M_x for Bounded Reducibilities

A *bounded reducibility* is a Turing reducibility $\Phi^A_e(x)$ with a computable function $h(x)$ which bounds the use function, *i.e.*, $\varphi^A_e(x) \leq h(x)$. Given $h(x)$ imagine a matrix M_x whose rows are all the strings σ of length $h(x)$. The action on x is entirely determined by the action of $\Phi^\sigma_e(x)$ for these rows $\sigma \in M_x$. For example, if $B \leq_{\rm m} A$ via a computable function $f(x)$ then this reduction is bounded by $h(x) = f(x)$ and $x \in B$ iff $\sigma(f(x)) = 1$ where $\sigma \prec A$, *i.e.*, where σ is an initial segment of A. In Definition 7.1 (iii) we defined σ_y to be the string with canonical index y derived from D_y. This gives a method for indexing the rows $\sigma_y \in M_x$.

We begin in §8.2 with the most general bounded reducibility called *bounded Turing reducibility ($B \leq_{\rm bT} A$)* where we are given only the computable bound $h(x)$, *i.e.*, only the matrix M_x. In §8.3 we study the more

restrictive case of *truth-table reducibility ($B \leq_{\rm tt} A$)* where $\Phi_e^\sigma(x)$ converges for every x and every row $\sigma \in M_x$. If $\Phi_e^\sigma(x)$ diverges for even *one* x and $\sigma \in M_x$ then the reduction is a *partial* bT-reduction, but if it converges for every x and every $\sigma \in M_x$ then it gives a genuine truth-table as in beginning logic courses as defined in Theorem 8.3 (iii). This convergence on all strings $\sigma_y \in M_x$ is the key defining property for a tt-reduction as expressed in (14).

8.2 Bounded Turing Reducibility

Definition 8.1. (i) A set B is *bounded Turing reducible (bT-reducible)* to a set A ($B \leq_{bT} A$) if there is a Turing reduction $\Phi_e^A = B$ and a computable function $h(x)$ such that the use $\varphi_e^A(x) \leq h(x)$.

(ii) A set B is *identity bounded Turing reducible* to A ($B \leq_{ibT} A$) if $B \leq_{bT} A$ with $h(x) = x$.

The bT and ibT reductions occur naturally in several parts of the subject. For example, often we are given a noncomputable c.e. set A and we construct a simple set $B \leq_{\rm T} A$ by simple permitting where an element x is allowed to enter B at stage some stage only if some $y \leq x$ has just entered A. When $A\!\upharpoonright\! x$ has settled then $B\!\upharpoonright\! x$ has settled also, and hence $B \leq_{\rm ibT} A$.

More recently, *ibT* has occurred in applications of computability to differential geometry in Soare [2004] and Csima-Soare [ta], as described later, and ibT reducibility has also been used in applications to algorithmic randomness and Kolmogorov complexity. Barmpalias and Lewis [ta] have shown the nondensity of the *ibT*-degrees of c.e. sets.

8.3 Truth-Table Reductions

Definition 8.2. A set B is *truth-table reducible* to set a A ($B \leq_{\rm tt} A$) if $B \leq_{\rm bT} A$ via $\Phi_e^A = B$ and $h(x)$ as in Definition 8.1, and also

$$(14) \quad (\forall x)(\forall \sigma)[\ \ |\sigma| \ = \ h(x) \qquad \Longrightarrow \qquad \Phi_e^\sigma(x)\!\downarrow \ \ ].$$

Theorem 8.3. [Truth-Table Theorem, Nerode]. *The following definitions of tt-reducible are equivalent.*

(i) $B \leq_{\rm tt} A$ *as defined in Definition 8.2 and* (14).

(ii) $\Phi_e^A = B$ *for some total* Φ_e^X *that is* $(\forall X)(\exists Y)(\forall x)[\ \Phi_e^X(x) = Y(x)\ ]$.

(iii) $B \leq_{\rm tt} A$ *via* Φ_e^A *and* $h(x)$ *as in Definition 8.2. In addition, there is a computable function* $g(x)$ *such that*

$$D_{g(x)} = \{\ y\ :\ |\sigma_y| = h(x) \quad \& \quad \Phi_e^{\sigma_y}(x) = 1\ \}.$$

(iv) There exist computable functions g *and* h *such that, for all* x*,*

$$x \in B \iff (\exists z \in D_{g(x)}[\ A\restriction(h(x)+1)\ =\ D_z\ ].$$

Proof. The implications (i) $\implies$ (ii), (iii) $\implies$ (i), and (iii) $\iff$ (iv) are obvious. It remains to prove (ii) $\implies$ (iii).

(ii) $\implies$ (iii). Uniformly in x we can effectively enumerate the set

$$U_x = \{\ \sigma\ :\ \Phi_e^\sigma(x)\downarrow\ \}.$$

Since Φ_e is total apply the Compactness Theorem 7.4 (iv) to get a finite subset $V_x \subseteq U_x$ such that $\cup\{\,\mathcal{N}_\sigma\ :\ \sigma \in V_x\,\} \supseteq 2^\omega$. Define $h(x)$ and $g(x)$ by

$$h(x) = \max\{\ |\sigma|\ :\ \sigma \in V_x\ \}.$$

$$D_{g(x)} = \{\ y\ :\ |\sigma_y| = h(x) \quad \& \quad \Phi_e^{\sigma_y}(x) = 1\ \}.$$

□

Remark 8.4. Post [1944, §6] introduced the first definition of $B \leq_{\rm tt} A$. For every integer n he required a effective procedure to produce integers k, and $m_1, m_2 \ldots m_k$. He then drew a diagram of a matrix with columns $m_1, \ldots, m_k$ and 2^k rows with individual entries of $+$ for $x \in A$ and $-$ for $x \notin A$. Letting $h(x)$ be the maximum of the $\{m_i\}_{i\leq k}$, filling in all columns less than $h(x)$, and replacing $+$ by 1 and $-$ by 0 we have exactly the matrix M_x described in §8.1. Next Post drew a vertical line to the right of this matrix and added an extra column such that an entry v_z in this column following row R_z indicated that if A satisfies row R_z then $x \in B$ iff $v_z = +$.

Post's extra column converted the matrix M_x into a *total* reduction because A had to extend exactly one of the rows R_z and then the value for $B(x)$ was completely specified by v_z given in advance. Therefore, Post's original definition [1944] of $B \leq_{\rm tt} A$ is virtually identical in intuition and description to our Definition 8.2 above where we could attach to matrix M_x an extra column with the value $\Phi_e^\sigma(x)$ on row σ. In both cases the crucial point is that $\Phi_e^\sigma(x)$ is defined for *every* string of length $h(x)$ before examining the oracle A. Property (iv) of $B \leq_{\rm tt} A$ has been used in the

past *e.g.*, Soare [1987, p. 83] and is short and slick but is does not give the necessary elegance and intuition for beginning students.

Turing reducibility was not well understood in 1944 and an understanding of it in its modern state emerged only gradually during the 1940's and 1950's. Post's tt-reducibility in 1944 was understood at once. Therefore, in 1959 Friedberg and Rogers introduced wtt-reducibility ($B \leq_{\rm wtt} A$) as a *weakening* of tt-reducibility which was already a *strengthening* of $\leq_{\rm T}$. Therefore, the concept of *wtt* has distance two from the central concept $\leq_{\rm T}$. The concept of bounded Turing reducibility (bT), which has the same definition as wtt, goes immediately to the main concept of bounding the queried information by a computable function $h(x)$. Thus, $\leq_{\rm bT}$ has distance one from the main reducibility of $\leq_{\rm T}$. The concept of tt-reducibility is tied to the *totality* of the reduction Φ_e^A which is exactly what bT-reducibility lacks. Also *bT* is more recognizable than *wtt* whose meaning cannot be guessed from its name.

8.4 Difference of c.e. sets, n-c.e., and ω-c.e. sets

The Limit Lemma characterized sets $A \leq_{\rm T} \emptyset'$ as those such that $A = \lim_s A_s$ for a computable sequence $\{A_s\}_{s\in\omega}$. Now consider special cases based upon how many times the approximation changes on x. These notions are more general than c.e. sets A but not the most general case of $A \leq_{\rm T} \emptyset'$.

Definition 8.5. (i) A set D is the *difference of c.e. sets* (*d.c.e.*) if $D = A - B$ where A and B are c.e. sets.

(ii) The set A is *omega-c.e.* (written ω-*c.e.*) if there is a computable sequence $\{A_s\}_{s\in\omega}$ with $A_0 = \emptyset$ and a computable function $g(x)$ which bounds the number of changes in the approximation $\{A_s\}_{s\in\omega}$ in the following sense,

$$(15) \qquad A = \lim_s A_s \quad \& \quad |\{ s : A_s(x) \neq A_{s+1}(x) \}| \leq g(x).$$

(iii) For $n \in \omega$ the set A is n-*c.e.* if $g(x) \leq n$.

For example, the only 0-c.e. set is $\emptyset$, the 1-c.e. sets are the usual c.e. sets, and the 2-c.e. sets are the d.c.e. sets. The next theorem gives an elegant characterization of ω-c.e. sets.

Theorem 8.6. *The following are equivalent.*

(i) $A \leq_{\rm bT} \emptyset'$.

(ii) A *is* ω-*c.e.*

(iii) $A \leq_{\rm tt} \emptyset'$.

Proof. Clearly, (iii) implies (i).

(i) $\implies$ (ii). Suppose $\Phi_e^{\emptyset'} = A$ with computable bound $g(x) \geq \varphi_e^{\emptyset'}(x)$. Let $A_s(x) = \Phi_e^{\emptyset'}(x)[\,s\,]$ where we may speed up the computation so the latter is always defined. Now $A_{s+1}(x) \neq A_s(x)$ only if some element $z \leq g(x)$ enters $\emptyset'$ which can happen at most $g(x)+1$ times, once for each $z \in [0, g(x)]$.

(ii) $\implies$ (iii). Assume $A = \lim_s A_s$ satisfies (15) via $g(x)$. Define the c.e. *change set* C by

$$C = \{ \quad \langle k, x\rangle \ : \ 1 < k \leq \ | \ \{s : A_s(x) \neq A_{s+1}(x)\} \ | \quad \}.$$

(Intuitively, whenever $A_s(x) \neq A_{s+1}(x)$ we put into C the least element of the form $\langle k, x\rangle$, *i.e.*, on row $\omega^{[x]}$, not already in C.) Therefore, $|C^{[x]}|$ is the number of changes on x during the approximation, and $|C^{[x]}| \leq g(x)$ by hypothesis (ii). Furthermore, $A(x) = 1$ iff $|C^{[x]}|$ is odd, because the approximation changes between 0 and 1 starting with 0.

First build a truth-table to demonstrate that $A \leq_{\rm tt} C$. For a given x the truth-table has rows of width $g(x)$ as in Definition 8.2 (ii). For each $k \leq g(x)$ construct a row beginning with k many $1's$ followed by all $0's$. Now to tt-compute from C whether $x \in A$ compute $k = |C^x|$. Next find the row with k many 1's. Now $A(x) = 1$ iff k is odd. Hence, $A \leq_{\rm tt} C$. However, C is c.e. Hence, $C \leq_{\rm m} 0'$ and therefore $A \leq_{\rm tt} 0'$. □

This result shows the difference between T-reductions and tt-reductions. First, the assumption on $h(x)$ ensures that $A \leq_{\rm bT} D$. Second, the approximation always begins with $A_s(x) = 0$ for $s = 0$ and changes between 0 and 1 because all values are in $\{0, 1\}$ rather than in ω. Therefore, we can make up a row in the truth-table which gives value for $A(x)$ based only on the number of changes. This ensures that $A \leq_{\rm tt} D$. This case is unusual since most reductions we consider only produce $A \leq_{\rm T} B$ for some set B and sometimes produce $A \leq_{\rm bT} B$ as in permitting arguments.

9 Online Computing

The original implementations of computing devices were generally offline devices such as calculators or batch processing devices. However, in recent years the implementations have been increasingly online computing devices which can access or interact with some external database or other device. The Turing o-machine is a better model to study them because the Turing a-machine lacks this online capacity.

Definition 9.1. (i) An *online* or *interactive* computing process is one which interacts with its environment, for example a computer communicating with an external data base such at the World Wide Web.

(ii) An *offline* computing process is one which begins with a program and input data, and proceeds internally, not interacting with any external device. This includes a calculator, and *batch processing* devices where a user handed a deck of punched IBM cards to an operator, who fed them to the computer and produced paper output later.

There are many descriptions in the computing literature about online and interactive processes. In [Goldin-Smolka-Wegner] a chapter by Yuri Gurevich, *Interactive Algorithms 2005* is described,

> "In this chapter, Gurevich asserts that computer science is largely about algorithms, and broadens the notion of algorithms to include interaction by allowing intra-step interaction of an algorithm with its environment."

About the chapter, *A Theory of Interactive Computation* by Jan van Leeuwen and Jiri Wiedermann, the book states,

> "This chapter asks what a computational theory of interactive, evolving programs should look like. The authors point out that a theory of interactive computation must necessarily lead beyond the classical, finitary models of computation. A simple model of interactive computing is presented consisting of one component and an environment, interacting using single streams of input and output signals."

It appears that the Turing o-machine is a good theoretical model to analyze an interactive process because there is usually a fixed algorithm or procedure at the core, which by Turing's thesis we can identify with a Turing a-machine, and there is a mechanism for the process to communicate with its environment, which when coded into integers may be regarded as a Turing type oracle. Under the Post-Turing Thesis 6.1 these real world online or interactive processes can be described by a Turing oracle machine.

In real world computing the oracle may be called a *database* or an environment. A laptop obtaining data from the World Wide Web is a good example. In the real world the database would not be literally infinite but may appear so (like the web) and is usually too large to be conveniently downloaded into the local computer. Sometimes the local computer is called the "client" and the remote device the "server."

9.1 Turing Machines and Online Processes

We could continue analyzing to what extent Turing oracle machines can model modern online processes, but let us now examine the reverse direction, that o-machines are online while a-machines are not. The following points appear self-evident.

- Turing oracle machines are online. An o-machine has fixed program but has the capacity to interact with its environment and receive new data during its computation.
- The original Turing a-machines are not online. A Turing a-machine, even a universal machine, begins with a fixed program and fixed input and proceeds without further outside input until (if ever) it halts.
- A large number and rapidly increasing number of computing processes in the real world are online or interactive. See the authors in [Goldin-Smolka-Wegner] for only a few.
- A large number of books presenting an introduction to computability mention Turing oracle machines and relative computability very late in the book or not at all.
- A large number of books with articles on Turing and the Church-Turing Thesis do not mention Turing oracle machines or relative computability at all, *e.g.*, Christof Teuscher [2004].

Discussing *only* Turing a-machines in modern texts, or *only* the Church-Turing Thesis, and not the Post-Turing Thesis on oracle computers, is like discussing only batch processing machines of the 1950's long after the emergence of online computing.

9.2 Trial and Error Computing

We expect Turing a-machines and o-machines to be absolutely correct. However, there are many computing processes in the real world which give a sequence of approximations to the final answer. Turing considered machines which make mistakes. In his talk to the London Mathematical Society, February 20, 1947, quoted in Hodges, p. 360–361, Turing said,

> "I would say that fair play must be given to the machine. Instead of it sometimes giving no answer we could arrange that it gives

> occasional wrong answers. But the human mathematician would likewise make blunders when trying out new techniques. It is easy for us to regard these blunders as not counting and give him another chance, but the machine would probably be allowed no mercy. In other words if a machine is expected to be infallible, it cannot also be intelligent. There are several theorems which say exactly that. But these theorems say nothing about how much intelligence may be displayed if a machine makes no pretence at infallibility.

Hillary Putnam [1965] described *trial and error* predicates as ones for which there is a computable function which arrives at the correct answer after a finite number of mistakes. In modern terminology this is called a *limit computable* function as described in Soare [1987] or Soare [CTA] Chapter 3. This is a model for many processes in the real world which allow finitely many mistakes but gradually move closer to the correct answer.

9.3 The Limit Lemma

There are several different approaches for computing with finite errors.

Definition 9.2. A computable sequence of computable sets $\{B_s\}_{s\in\omega}$ is a *computable (Δ_0) approximating sequence* for a set B if $B = \lim_s B_s$. If there is such a sequence we say that B is *limit computable*,

Lemma 9.3. [Limit Lemma, Shoenfield, 1959]. *The following are equivalent.*

(i) $B \leq_{\mathrm{T}} A$ *for some c.e. "*oracle set*" A.*

(ii) B *is limit computable.*

(iii) $B \in \Delta_2$*, that is,* B *is is both two quantifier forms.*

Proof. (See Soare [1987] or Soare [CTA] Lemma 3.3.6.) □

Now in the real world imagine an example of a computing process with error such as: a robot learning a maze; a financial trader receiving information from around the world updated every second; a meteorologist predicting the weather a week from now given constantly updated weather conditions today. In our idealized model we assume that the individual makes finitely many errors during the process $\{B_t\}_{t\in T}$ for time periods $t \in T$ but eventually gets the correct answer. (In practice, the final answer may not be

exactly correct in these examples, but is presumably more accurate than the first approximation B_0. The sequences of improving approximations, even if not exact, are usually useful in financial trading, meteorology, and other approximations in real time.)

9.4 Two Models for Computing With Error

9.4.1 The Limit Computable Model

In the limit computable or approximation model we have a sequence of Turing programs $\{P_t : t \in T\}$ so that P_t computes function g_t at time $t \in T$. There is not necessarily any connection between different programs and we may have to compute all over again with a new program as we pass from time t to $t + 1$.

Suppose the financial trader in Chicago receives data every second $t \in T$ about currency prices in London, Milan, New York and Tokyo. The configuration at his trading desk may be described using the Limit Lemma by a computable function where g_t is the computable characteristic function of B_t, the configuration of his computation at the end of time t. The computable function g_t gives an algorithm to compute the condition B_t at time t but it gives no relationship between B_t and B_{t+1}. It will not be possible for the trader to write a new program every second. How will the trader write a program to uniformly compute the index g_t for $t \in T$?

9.4.2 The Online Model

By the Limit Lemma there is a c.e. set A (or even a Δ^0_2 set) and oracle machine Φ_e such that $B = \Phi^A_e$. Now the trader can program the algorithm Φ_e into his laptop once and for all at the start of the trading day. Every second $t \in T$ he receives from New York and abroad the latest quotes A_t which enter directly into his computer by an internet connection. He does not (and cannot) change the program Φ_e every second. His algorithm simply receives the "oracle" information A_t from the internet as it is continually updated, and computes the approximation $B_t(x) = \Phi^{A_t}_e(x)$. His program then executes a trade when the algorithm determines that conditions are favorable. It is difficult to see how this trader could have carried out his business using a batch processing, Turing a-machine model, instead of an online model.

10 Three Displacements in Computability Theory

The dictionary defines "displacement" to be the moving of something from its rightful place or position, often when replaced by something else. There are three important issues in computability which have at one time been displaced from their correct or proper positions as evaluated by historical and scientific criteria. These issues are at the very heart of the subject and they define how we think about computability. We now examine these three issues one by one in the next three sections. These items may have been displaced accidentally or without conscious thought or decision, simply acting from the exigencies of the situation at the time, but are not consistent with a careful scientific analysis later.

When computabiliy theory originated in the 1930's it was a very small field attempting to consolidate its ideas. Furthermore, three of its leaders, Gödel, Turing, and Church effectively left the field after 1940 and had little direct influence thereafter on its development, although Church supervised the Ph.D. theses of many prominent researchers in the field. After 1940 the field was developed and promulgated primarily by Stephen C. Kleene with some additional influence by Emil Post as described earlier. In dedicating his book *Degrees of Unsolvability* [1971] Shoenfield recognized Kleene's overwhelming influence,

> "Dedicated to S.C. Kleene who made recursive function theory into a theory."

This is entirely accurate and without Kleene's leadership we would not have the field as we know it today. However, a number of changes took place after 1940, perhaps by accident, that were not consistent with the original development of computability in the 1930's and would not have been approved of by Gödel and Turing.

11 "Computable" versus "Recursive"

Starting in 1936, Church and Kleene used the term "recursive" to mean "computable" even though Turing and Gödel later objected. Kleene later introduced the term "recursive function theory" for the subject although Gödel disagreed (see below). This was the first time in history that the term "recursive" which had meant roughly "inductive" acquired the additional meaning "computable" of "calculable." After 1996 the term the term 'recursive" was again used only to mean "inductive" not "computable" or

"calculable." From 1931 to 1934 Church and Kleene had used the λ-definable functions as the formal equivalent of effectively calculable functions, and Church had first proposed his thesis privately to Gödel in that form.

11.1 Church Defends Church's Thesis with "Recursive"

Recall *Church's Thesis 2.1 (First Version) [1934]* "A function is effectively calculable if and only if it is λ-definable." When Gödel strongly rejected to this thesis Church turned instead to Gödel's own Herbrand-Gödel general recursive functions as a formalism and proposed in [1935] and [1936] the well-known form of his thesis, *Church's Thesis 2.2 [1936]* A function on the positive integers is effectively calculable if and only if it is recursive.

Church and Kleene knew almost immediately and published by 1936 the proof of the formal equivalence of recursive functions with λ-definable functions. After seeing Gödel's lectures in 1934 Church and Kleene dropped the λ-definable functions and adopted the recursive functions. This was not because of the inadequacy of λ-definable functions in comparison to the recursive functions. Indeed Church seems to have preferred the λ-definable functions and caused Turing to write his thesis [1939] in that formalism.

Church was very eager for mathematicians to accept his thesis and he know that the recursive functions were more familiar to a mathematical audience than λ-definable ones. Church and Kleene used the second version of Church's Thesis above, phrased in terms of recursive functions primarily for *public relations* as Kleene [1981] explained, "I myself, perhaps unduly influenced by rather chilly receptions from audiences around 1933–35 to disquisitions on λ-definability, chose, after general recursiveness had appeared, to put my work in that format. ..." Church and Kleene were simply doing what most scientists do, arrange the work in a framework which will be understandable and appealing to as large a scientific audience as possible. Ironically, this is exactly what caused the change in 1996 from "recursive" back to "computable" because in 1996 the term "computable" was much better understood by a general audience than "recursive." The irony is that the term "computable" was there all along and was preferred by Turing and Gödel.

11.2 Church and Kleene Define "Recursive" as "Computable"

By 1936 Kleene and Church had begun thinking of the word "recursive" to mean "computable." Church had seen his first thesis rejected by Gödel and was heavily invested in the acceptance of his 1936 thesis in terms of

recursive functions. Without the acceptance of this thesis Church had no unsolvable problem. Church wrote in [1936, p. 96] printed in Davis [1965] that a *"recursively enumerable set"* is one which is the range of a recursive function. This is apparently the first appearance of the term "recursively enumerable" in the literature and the first appearance of "recursively" as an adverb meaning "effectively" or "computably."

In the same year Kleene [1936, p. 238] cited in Davis [1965] [p. 238] mentioned a *"recursive enumeration"* and noted that there is no recursive enumeration of Herbrand-Gödel systems of equations which gives only the systems which define the (total) recursive functions. By a "recursive enumeration" Kleene states that he means "a recursive sequence (*i.e.*, the successive values of a recursive function of one variable)." Post [1944], under the influence of Church and Kleene, adopted this terminology of "recursive" and "recursively enumerable" over his own terminology [1943], [1944] of "effectively generated set," "normal set," "generated set." Thereafter, it was firmly established.

11.3 Gödel Rejects "Recursive Function Theory"

Neither Turing nor Gödel ever used the word "recursive" to mean "computable." Gödel *never* used the term "recursive function theory" to name the subject; when others did Gödel reacted sharply negatively, as related by Martin Davis.

> In a discussion with Gödel at the Institute for Advanced Study in Princeton about 1952–54, Martin Davis casually used the term "recursive function theory" as it was used then. Davis related, "To my surprise, Gödel reacted sharply, saying that the term in question should be used with reference to the kind of work Rosza Peter did."

(See Peter's work on recursions in [1934] and[1951].) By 1990 the situation had become very difficult. Most people access to a personal computer on their desks and the terms of computing were familiar to the general population but "recursive" was limited to very small number who mainly associated it with a first year programming course or a definition by induction on mathematics and almost never with computability. So few people understood the meaning of "recursive" that by 1990 I had to begin my papers with,

> "Let f be a recursive function (that is, a computable function),"

as if I were writing in Chinese and translating back into English.

11.4 The Ambiguity in the Term "Recursive"

The traditional meaning of "recursive" as "inductive" led to ambiguity. Kleene often wrote of calculation and algorithms dating back to the Babylonians, the Greeks and other early civilizations. However, Kleene [1981b, p. 44] wrote,

> "I think we can say that recursive function theory was born there ninety-two years ago with Dedekind's Theorem 126 ('Satz der Definition durch Induktion') that functions can be defined by primitive recursion."

Did he mean that recursion and inductive definition began with Dedekind or that computability and algorithms began there? The latter would contradict his several other statements, such as Kleene [1988, p. 19] where he wrote, "The recognition of algorithms goes back at least to Euclid (c. 330 B.C.)." When one uses a term like "recursive" to also mean "computable" or "algorithmic" as Kleene did, then one is never sure whether a particular instance means "calculable" or "inductive" and our language has become indistinct. Returning "recursive" to its original meaning of "inductive" has made its use much clearer. We do not need another word to mean "computable." We already have one.

11.5 Changing "Recursive" Back to "Inductive"

By 1996 the confusion had become intolerable. I wrote an article on *Computability and Recursion* for the *Bulletin of Symbolic Logic* [1996] on the history and scientific reasons for why we should use "computable" and not "recursive" to mean "calculable." "Recursive" should mean "inductive" as is had for Dedekind and Hilbert. At first few were willing to make such a dramatic change, overturning a sixty year old tradition of Kleene, and the words "computability theory" and "computably enumerable (c.e.) set" did not come tripping from the lips. However, in a few months more people were convinced by the undeniable logic of the situation. Three years later at the A.M.S. conference in Boulder, Colorado referenced in Soare [2000], most researchers, especially those under forty years old, had adopted the new terminology and conventions. Changing back from from "recursive" to "computable" during 1996–1999 has had a number of advantages.

Historical Accuracy. The founders of the two key models of computabilty, Turing and Gödel, had never used "recursive" to mean computable and indeed had objected when it was so used. The object of

everyone was to formally capture the informal concept of "effectively calculable" which Turing machines did to Gödel's satisfaction, while at first Gödel's own model of Herbrand-Gödel recursive functions did not. The object was never to understand the notion of recursions or inductive definitions. In 1935 Church adopted Gödel's recursive functions as a definition of effectively calculable before seeing Turing machines As Kleene relates, Church and Kleene did this as a matter of public relations to relate to a concept mathematicians could understand, not in an attempt to better understand the nature of recursion and inductive definition.

Scientific Accuracy. The words "calculate" and "compute" are very close in the dictionary, the former being a bit more general. The word "recursive" means a procedure characterized by recurrence or repetition from the Latin verb "recurrere," to run back. The word has nothing to do with "calculate" or "compute." The general public understands the first two words in this context. To the extent that they have any idea about "recursive" they understand it in this context. For example, a first year programming course speaks of definition by iteration versus definition by recursion.

Name Recognition. Suppose a student is scanning a catalogue for a course to take and sees the course title "Recursive Function Theory" versus "Computability Theory." Which will give him more information about the content of the course? Should a fresh Ph.D. apply for jobs under the general area of his work as the first or second? Should a professor write an abstract for his lecture at another university under the first title or second?

Names do matter. They mattered to Church and Kleene in 1936 when they changed from the term "λ-definable function" to "recursive function" to achieve greater name recognition among mathematicians and to make Church's Thesis more convincing before the appearance of Turing. Names matter today as we try to relate our specialty of computability to the world of computers and algorithmic procedures all around us, a world partially created by Turing.

12 Renaming it the "Computability Thesis?"

12.1 Kleene Called it "Thesis I" in [1943]

In the 1930's both Church and Turing thought they were giving definitions of an effectively calculable function, not putting forth a "thesis." These were not even called "theses" at all until Kleene [1943, p. 60] referred to Church's "definition" as "Thesis I."

12.2 Kleene Named it "Church's thesis" in [1952]

Later in his very influential book [1952] it is fascinating to see how Kleene's thinking and terminology progressed from "Thesis I" to "Church's Thesis" and not to "Turing's Thesis" or the "Church-Turing Thesis." Kleene [1952, p. 300] took up where Kleene [1943] had left off.

> "This heuristic evidence and other considerations led Church 1936 to propose the following thesis.
>
> *Thesis I. Every effectively calculable function (effectively decidable predicate) is general recursive.*"

This is identical with what we previously called Church's Thesis 2.2. Of course, Kleene was aware of other similar "theses" advanced nearly simultaneously and he continued [1952, p.300],

> "This thesis is also implicit in the conception of a computing machine formulated by Turing 1936–7 and Post 1936."

Next Kleene begins a subtle shift of terminology from "Thesis I" to "Church's thesis." Apparently he did not feel it necessary to include Turing's name when he used the term "Church's thesis." Kleene wrote [1952, p.317] began a new section §62 and called it "Church's thesis" instead of "Thesis I" as he had been doing. Kleene wrote,

> " §62. **Church's thesis**. One of the main objectives of this and the next chapter is to present the evidence for Church's thesis (Thesis I) §60."

12.3 Kleene Dropped "Thesis I" for "Church's thesis"

As Kleene progressed through [1952, §62, pp. 318–319] he dropped any reference to "Thesis I" and used only "Church's thesis" with no mention of Turing or Post in the thesis. He wrote [1952, p. 318–319],

> " Church's thesis, by supplying a precise delimitation of all effectively calculable functions, . . . "
>
> "While we cannot prove Church's thesis, since its role is to delimit precisely an hitherto vaguely conceived totality, we require evidence that it cannot conflict with the intuitive notion which it is supposed to complete; . . . "
>
> "The converse of Church's thesis, *i.e.*, that every general recursive function φ is effectively calculable, we take to be already confirmed by the intuitive notion (cf. §60)."

12.4 Evidence for the Computability Thesis

In one of the most familiar parts of the book, Kleene summarized the evidence for "Church's thesis" in [1952, §62, p. 319]. These arguments have been cited in hundreds of computability books and papers for the last half century. After some heuristic evidence in (A), Kleene presented perhaps the most powerful evidence, the "(B) Equivalence of diverse formulations:" such as the (Herbrand-Gödel) general recursive functions, λ-definable functions, Turing computable functions, and systems of Post [1943] and [1946] of canonical and normal systems. Ironically, to clinch the evidence for Church's thesis Kleene began a new part (C) where he appealed to Turing's work and wrote,

> "(C) Turing's Concept of a Computing Machine."
>
> Turing's computable functions [1936–7] are those which can be computed by a machine of a kind which is designed, according to his analysis, to reproduce all sorts of operations which a human computer could perform, working according to preassigned instructions. Turing's notion is thus the result of a direct attempt to formulate mathematically the notion of effective calculability, while other notions arose differently and were afterwards identified with effective calculability. Turing's formulation hence constitutes an independent statement of Church's thesis."

We see here the amalgamation of different meanings into a single term (just as for "recursive" above). Kleene here was using the term "Church's thesis" to include Turing's Thesis and Turing's justification. Turing's work was to establish the connection between effectively calculable functions and Turing computable functions in Turing's Thesis 3.1. As an *intensional* claim it had nothing to do with the recursive functions of Church's Thesis 2.2. It was only the equivalence of Turing computable functions first with λ-definable functions and hence with recursive functions which links them *extensionally* but not intensionally.

Remark 12.1. [The Computability Thesis Emerges]. *It is exactly here that Kleene introduces (without explicit mention) another convention and term which has lasted until today. Kleene knows that the various formal definitions all coincide extensionally, and he regards them interchangeably, regardless of their intensional or historical import. From now on when Kleene uses the term "Church's thesis" he means the following "Computability Thesis." Kleene omits Turing's name from the thesis even though in part (C) above he gave Turing credit as the only one who made a "direct attempt to formulate mathematically the notion of effective calculability."*

Definition 12.2. [The Computability Thesis]. A function on the positive integers is effectively calculable if and only if it is recursive or Turing computable, or defined by any of the other formalisms.

Note that the Computability Thesis is essentially the same as what we have called the Church-Turing Thesis 3.2. It does not refer to one single man or to one single formalism. The Computability Thesis, used extensively in the subject, is usually listed under the name "Church's Thesis" as in the title of the new book [Olszewski, 2007] with no mention of Turing. Many people today use "Church's Thesis" in this way.

12.5 Who First Demonstrated the Computability Thesis?

Demonstrating something like the Computability Thesis requires two steps.

The Formalism. A formal mathematical definition for effectively calculable functions. Church [1936] proposed Gödel's general recursive functions [1934]. Turing invented the new formalism of Turing machines because this was closest to his idea of a mechanical process and because it lent itself to proving that any effectively calculable function lay in this class.

The Demonstration. The second and perhaps more important step is to prove that the informal notion of a person calculating a function can be simulated by the formal model. This step is not a purely mathematical one but it needs to be as convincing and logical as possible. We refer to this step as a "demonstration" rather than a formal "proof."

Although it is not a formal proof, this second step is so crucial that it has been referred to as a "theorem" by Gandy and others. Gandy [1988, p. 82] observed, "Turing's analysis does much more than provide an argument for 'Turing's Thesis,' it proves a theorem."[3] Furthermore, as Gandy [1988, pp. 83–84] pointed out, "Turing's analysis makes no reference whatsoever to calculating machines. Turing machines appear as a result, a codification, of his analysis of calculations by humans." *Turing's Thesis* [1936, §9] stated in Definition 3.1 is that every intuitively computable (effectively calculable) function is computable by a Turing machine.

In contrast, Church used the Herbrand-Gödel general recursive functions as his formal model but even their inventor, Gödel, was not convinced as we have seen in §2 and §3. No modern book uses the H-G general recursive formalism to define the effectively calculable functions.

A considerably more serious objection is that there was a flaw in Church's demonstration that every effectively calculable function is general recursive. The flaw in Church's argument [1936, §7] for his thesis was this. Church began by defining an "effectively calculable" function to be one for which "there exists an algorithm for the calculation of its values." Church analyzed the informal notion of the calculation of a value $f(n) = m$ according to a step-by-step approach (so called by Gandy [1988, p. 77]) from two points of view, first by an application of an algorithm, and second as the derivation in some formal system, because as he pointed out, Gödel had shown that the steps in his formal system P were primitive recursive. Following Davis [1958, p. 64] or Shoenfield [1967, pp. 120–121] it is reasonable to suppose that the calculation of f proceeds by writing expressions on a sheet of paper, and that the expressions have been given code numbers, c_0, c_1, $\ldots c_n$. Define $\langle c_0, c_1, \ldots c_n\rangle = p_0^{c_0} \cdot p_1^{c_1} \ldots p_n^{c_n}$. We say that the calculation is *stepwise recursive* if there is a partial recursive function ψ such that $\psi(\langle c_0, \ldots, c_i\rangle) = c_{i+1}$ for all i, $0 \leq i < n$.

[3] Gandy actually wrote "Church's thesis" not "Turing's thesis" as written here, but surely Gandy meant Turing's Thesis, *i.e.*, the computability thesis, at least intensionally, because Turing did not prove anything in [1936] or anywhere else about general recursive functions.

If the basic steps are stepwise recursive, then it follows easily by the Kleene Normal Form Theorem which Kleene had proved and communicated to Gödel before November, 1935 (see Kleene [1987b, p. 57]), that the entire process is μ-recursive. The fatal weakness in Church's argument was the core assumption that the atomic steps were stepwise recursive, something he did not justify. Gandy [1988, p. 79] and especially Sieg [1994, pp. 80, 87], in their excellent analyses, brought out this weakness in Church's argument. Sieg [p. 80] wrote, "...this core does not provide a convincing analysis: steps taken in a calculus must be of a restricted character and they are assumed, for example by Church, without argument to be recursive." Sieg [p. 78] wrote, "It is precisely here that we encounter the major stumbling block for Church's analysis, and that stumbling block was quite clearly seen by Church," who wrote that without this assumption it is difficult to see how the notion of a system of logic can be given any exact meaning at all. It is exactly this stumbling block which Turing overcame by a totally new approach.

12.6 The Computability Thesis and the Calculus

Why all the fuss over names? Why not simply use the term "Church's Thesis" invented by Kleene [1952], and let it refer to the "Computability Thesis?" This is in fact what is widely (and incorrectly) done.

If we had to attach a single name to "the calculus" every time we mentioned it, whose name should it be? Isaac Newton began working on a form of the calculus in 1666 but did not publish it until much later. Gottfried Leibniz began work on the calculus in 1674 and published his account of the differential calculus in 1684 and the integral calculus in 1686. Newton did not publish it until 1687 although most believe he had been working on it before Leibniz began in 1674. There was a great controversy about priority up until Leibniz's death in 1716. The British Royal Society handed down a verdict in 1715 crediting Isaac Newton with the discovery of the calculus, and stating that Leibniz was guilty of plagarism (although these charges were later proved false). Newton was more established than Leibniz and had vigorous supporters. Newton and his followers campaigned vigorously for his position. The Wiki encylopedia wrote,

> "Despite this ruling of the Royal Society, mathematics throughout the eighteenth century was typified by an elaboration of the differential and integral calculus in which mathematicians generally discarded Newton's fluxional calculus in favor of the new methods presented by Leibniz."

12.7 Founders of Computability and the Calculus

There is a parallel between the development of the Calculus and the demonstration of the Computability Thesis.

1. Church and Turing both worked independently on it and came up with different models.

2. Turing began slightly later than Church. Leibnitz began later than Newton. Both Leibniz and Turing worked at a distance and independently of Newton and Church, respectively, unaware of the work by another researcher.

3. Turing's model of Turing machines is overwhelmingly more appealing and popular that Church's model of the (Herbrand-Gödel) general recursive functions, a model which is rarely presented in any books and never used for actual calculations in any courses. The general recursive functions are used only in an historical discussion. (This refers to the Herbrand-Gödel general recursive functions. The Kleene μ-recursive functions have other uses but are were not mentioned in the original Church's Thesis 2.2.)

4. For the calculus, despite the Royal Society ruling, "mathematicians generally discarded Newton's fluxional calculus in favor of the new methods presented by Liebniz."

5. In computer science, the Turing machines (and other calculating machines like register machines) dominate. The general recursive functions are never seen. Kleene's μ-recursive functions are sometimes used in courses and mistakenly called *recursive functions* but to prove that an effectively calculable function is μ-recursive requires a tedious arithmetization as in Gödel's incompleteness theorem [1931] and is virtually never done.

6. Newton was in the position of power with the Royal Society which not only affirmed his claim but denied the claim by his rival Leibniz. Church was the senior figure in computability (after Gödel). Church's claim to the Computability Thesis was affirmed by his former Ph.D. student Stephen Kleene [1952]. Kleene occasionally mentioned Turing's Thesis and repeatedly used the Turing machine model and Turing's demonstration of its success to demonstrate the Computability Thesis. However, Kleene deliberately and overwhelmingly established

> the phrase "Church's thesis" to stand for the "Computability Thesis, " at a time when nobody called it a thesis and when the field was about to rapidly expand in the 1950's and 1960's and would turn to Kleene for direction as the last representative of the original computability researchers of the 1930's and the most prominent and influential of all computability theorists in the 1950's. The Royal Society was not successful in excluding Leibniz from the calculus, but Kleene was certainly successful in excluding Turing's name from the Computability Thesis.

Virtually all the papers and books including Rogers [1967] and Soare [1987] and many others followed Kleene's lead. Unlike the calculus, the participants, Church, Turing, and their followers, have given credit to the others. The problem is simply that Kleene has not given Turing credit in his naming of the Computability Thesis. Kleene could have called it "the Computability Thesis" analogously with "the calculus." We never refer to "the Newton calculus" or the "Leibniz calculus." Why do we need to give a person's name to the Computability Thesis? Kleene never denied credit to Turing and in many places such as his books [1952] and [1967] he gives Turing credit for the most intuitive presentation of computability. Kleene just does not include Turing in the name he chose.

Remark 12.3. *Kleene thought and wrote with* **tokens,** *words that are given arbitrary and nonstandard meaning by the author: "recursive" means "computable," "Church's Thesis" means the "Computability Thesis." The arbitrary and sometimes misleading use of words has diminished our communications among ourselves and with other scientific and scholarly colleagues.*

It is ambiguous to use "recursive" with both meanings, inductive and computable, as we have seen. Second, it is simply wrong to use "Church's Thesis" to refer to a proposition first demonstrated by Turing and never successfully demonstrated by Church. It would also be wrong to refer to "the Newton Calculus" without mentioning Leibniz. Today we refer to "the calculus" without any founder's name. Why not call it simply, "the Computability Thesis" and not "Church's Thesis," or the "Church-Turing Thesis?" Which of the three terms is more understandable to an outsider who has never heard about the subject?

13 Turing a-machines versus o-machines?

13.1 Turing, Post, and Kleene on Relative Computability

In §4 we have seen how Turing [1939, §4] briefly introduced an oracle machine (o-machine) and how and in §5–§7 how Post developed relative computability through the influential Kleene-Post [1954] paper. Kleene [1952, p. 314] took up the theme of relative computabilty of a function from an oracle set. Analogous to his version of Thesis I considered above, Kleene defined Thesis I* to be the corresponding relative computability thesis which we have called Post-Turing Thesis 6.1. He recommended this thesis but did not give a separate justification. Kleene [1952, p. 314] wrote, "The evidence for Thesis I will also apply to Thesis I*," and on [1952, p. 319] he wrote,

> "We now summarize the evidence for Church's Thesis (and Thesis I*, end §61) under three main headings (A)–(C), and one other (D) which may be included under (A)."

13.2 Relative Computability Unifies Incomputability

The field of computability theory deals mostly with *incomputable* not computable objects. The objects considered in degrees of unsolvability, in computable model theory, in differential geometry as in Csima-Soare [2006] or Soare [2004] all deal with incomputable objects. However, we should not call the subject "incomputability theory" because the underlying theme is the notion of *relative computability* as absolute because of the Oracle Graph Theorem 4.5, and because it relates and unifies the myriad incomputable objects.

13.3 The Key Concept of the Subject

The notion of an oracle machine and relative computability is the single most important in the subject.

1. A Turing a-machine can easily be simulated by a Turing o-machine and the latter is scarcely more complicated to explain.

2. Most of the objects considered in computability theory and applications to algebra, model theory, geometry, analysis and other fields are incomputable not computable and relative computability unifies them.

3. Many if not most computing processes in the real world are online or interactive processes, better modelled by an o-machine than an a-machine.

4. A relative computability process Φ_e^A corresponds to a continuous functional on Cantor space analogous to continuous functions in analysis. A function on Cantor space given by an a-machine is merely a constant function.

13.4 When to Introduce Relative Computability

In view of the importance of relative computability, and online computing in both theoretical results and real world computing it is surprising how many computability books introduce oracle machines and Turing functionals so late in the book or not at all. For example, Kleene's book [1952] was the first real book on computability theory and the principal reference for at least fifteen years until Rogers [1967] appeared. Kleene introduced relative computability in Chapter 11 on page 266 by adding the characteristic function of the oracle set A to the Herbrand-Gödel general recursive functions. Rogers [1967] took the Turing machine approach and immediately defined computability using regular Turing machines (a-machines). Rogers quickly became the most readable textbook on computability and remains a popular reference. Rogers introduced relative computability only in Chapter 9 on page 128 using Turing's original definition [1939] of an ordinary Turing machine with the additional capacity to consult an oracle A occasionally during the computation. In another popular introduction, *Computability*, Cutland [1980] introduces relative computability relatively late on page 167. Boolos and Jeffrey in *Computability and Logic* [1974] do not discuss it at all. The more recent and very extensive books by Odifreddi, *Classical Recursion Theory* Vol. I [1989] and Vol. II [1999] introduce relative computability only on page 175 by adding the characteristic function of oracle set A to the Kleene μ-recursive functions. Lerman [1983] defines relative computability from an oracle by adding the characteristic function of the oracle to the Kleene μ-recursive functions. This occurs on page 11 but Lerman is assuming that the reader has already mastered a first course in computability using a text such as Rogers [1967]. Cooper's new book [2004] introduces oracle Turing machines on page 139.

Kleene's second and more introductory book, *Mathematical Logic* [1967, p. 267] has a brief discussion of reducing one predicate to another and on degrees of unsolvability. The only genuine introduction to computability I

found which introduces relative computability immediately is Martin Davis *Computability and Unsolvability* [1958], which defines it on page 20 of Chapter 1 using oracle Turing machines. In the former book Soare [1987] and new book [CTA] Turing a-machines come in Chapter I and Turing o-machines at the beginning of Chapter III after which the book is based on Turing reducibility. I thought of moving o-machines to Chapter I and doing it all at once, but my students persuaded me that people need time to absorb the concepts. Nevertheless, I believe that one should introduce o-machines and relative computability as soon as possible.

14 Conclusions

Conclusion 14.1. *We should use the term "recursive" to mean "defined inductively" not "calculable" or "computable." The subject is called "Computability Theory" not "Recursive Function Theory" or "Recursion Theory."*

Church and Kleene [1936] introduced the term "recursive" to mean "computable" primarily for public relations reasons as Kleene [1981] explained (see §2.3). Over the next few decades Kleene reinforced and promulgated this convention, but by the 1990's it had become much more useful for communication and more accurate scientifically and historically to remove the meaning of "computable" from the term "recursive," particularly since Turing and Gödel had both rejected this usage. This change has largely been accomplished since the papers on computability and recursion in Soare [1996] and Soare [1999]. See these papers and the discussion in §11. This emphasis on computability (rather than recursion) and its relation to incomputability has been developed in many recent books and papers such as Cooper [2004].

Conclusion 14.2. *It is most accurate and informative to refer to the central thesis of the subject as the "Computability Thesis" not "Church's Thesis," "Turing's Thesis," or the "Church-Turing Thesis."*

It is misleading to refer to it as "Church's Thesis" as many people do because Church never demonstrated the thesis (at least to the satisfaction of Gödel and modern scholars like Gandy [1988] and Sieg [1994]), but Turing did demonstrate his thesis in a manner convincing to essentially everyone. Church gave a formal model (the Herbrand-Gödel general recursive functions) which was not convincing even to its author, while Turing invented a new model, the Turing a-machine which everyone, including Church and Kleene, agreed was the most convincing of the thesis.

Neither of Turing nor Gödel thought of this as a thesis. The term "Church's thesis" was started arbitrarily by Kleene alone in [1952]. We use the term "the calculus" without the name of either founder, Newton or Leibniz, attached. Why not replace the name in computability by a descriptive and informative term like "Computability Thesis?" See the discussion in §12.

Conclusion 14.3. *The subject is primarily about incomputable objects not computable ones, and has been since the 1930's. The single most important concept is that of relative computability to relate incomputable objects.*

See §13 and §4–§7.

Conclusion 14.4. *For pedagogical reasons with beginning students it is reasonable to first present Turing a-machines and ordinary computability. However, any introductory computability book should then present as soon as possible Turing oracle machines (o-machines) and relative computability. Parallels should be drawn with offline and online computing in the real world.*

See §13.4.

References

[1] [Ambos-Spies and Fejer, ta] K. Ambos-Spies, and P. Fejer, *Degrees of unsolvability* Handbook of Logic, volume 9, to appear.

[2] [Boolos and Jeffrey, 1974] G. Boolos and R. Jeffrey, *Computability and Logic*, Cambridge Univ. Press, Cambridge, Engl., 1974.

[3] [Church, 1935] A. Church, An unsolvable problem of elementary number theory, Preliminary Report (abstract), *Bull. Amer. Math. Soc.* **41** (1935), 332-333.

[4] [Church, 1936] A. Church, An unsolvable problem of elementary number theory, *American J. of Math.*, **58** (1936), 345-363.

[5] [Church, 1936b] A. Church, A note on the Entscheidungsproblem, *J. Symbolic Logic*, **1** (1936), 40–41. Correction 101–102.

[6] [Church, 1937] A. Church, Review of Turing *1936*, *J. Symbolic Logic* **2(1)** (1937), 42–43.

[7] [Church, 1937b] A. Church, Review of Post *1936*, *J. Symbolic Logic* **2(1)** (1937), 43.

[8] [Church, 1938] A. Church, The constructive second number class, *Bull. A.M.S.* **44** (1938), 224–232.

[9] [Church and Kleene, 1936] A. Church and S. C. Kleene, Formal definitions in the theory of ordinal numbers, *Fund. Math.* **28** (1936) 11–21.

[10] [Cooper, 2004] S.B. Cooper, *Computablility Theory*, Chapman & Hall/CRC Mathematics, London, New York, 2004.

[11] [Cooper, 2004b] S.B. Cooper, The incomparable Alan Turing, Lecture at Manchester University, 5 June, 2004, published electronically, http://www.bcs.org/server.php?show=ConWebDoc.17130

[12] [Cooper-Lowe-Sorbi, 2007] S.B. Cooper, B. Löwe, A. Sorbi, (eds.), Computation and Logic in the Real World, Proceedings of the Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18–23, 2007, Lecture Notes in Computer Science, No. 4497, S.B. Cooper, B. Löwe, Andrea Sorbi (Eds.), (Springer-Verlag, Berlin, Heidelberg, 2007.

[13] [Cooper-Lowe-Sorbi, 2008] S.B. Cooper, B. Löwe, A. Sorbi, (eds.), New computational paradigms: changing conceptions of what is computable, Springer-Verlag, 2008.

[14] [Cooper-Odifreddi, 2003] S.B. Cooper and P. Odifreddi, Incomputability in nature. In: S.B. Cooper and S.S. Goncharov (Eds.) Computability and Models, Perspectives East and West, Kluwer Academic/Plenum, Dordrecht, (2003) 137–160.

B. Löwe, A. Sorbi, (eds.), New computational paradigms: changing conceptions of what is computable, Springer-Verlag, 2008.

[15] [Copeland-Posy-Shagrir, ta] Jack Copeland, Carl Posy, and Oron Shagrir, *Computability: Gödel, Church, Turing, and Beyond,* MIT Press, to appear.

[16] [Csima-Soare, 2006] B. F. Csima and R.I. Soare, Computability Results Used in Differential Geometry, *J. Symbolic Logic*, vol. 71 (2006), pp. 1394–1410.

[17] [Cutland, 1980] Nigel Cutland, *Computability: An introduction to recursive function theory,* Cambridge Univ. Press, Cambridge, Engl., 1980, reprinted 1983.

[18] [Davis, 1958] M. Davis, *Computability and Unsolvability,* Mc-Graw-Hill, New York, 1958; reprinted in 1982 by Dover Publications.

[19] [Davis, 1965] M. Davis, (ed.), *The Undecidable. Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions*, Raven Press, Hewlett, New York, 1965.

[20] [Davis, 1982] M. Davis, Why Gödel did not have Church's Thesis, *Information and Control* **54** (1982), 3–24.

[21] [Davis, 1988] M. Davis, Mathematical logic and the origin of modern computers, In: Herken, *1988*, 149–174.

[22] [Davis, 2000] M. Davis, The universal computer: The road from Leibniz to Turing, W.W. Norton & Co., New York, London, 2000. (The same book was also published by Norton in 2000 under the title "Engine of Logic.")

[23] [Davis, 2004] M. Davis, The myth of hypercomputation. In: Teutscher, C. (Ed), Alan Turing: Life and Legacy of a Great Thinker, Springer-Verlag, Berlin, Heidelberg, New York, (2004) 195–211.

[24] [Dawson, 1997] J. W. Dawson, Logical dilemmas: The life and work of Kurt Gödel, A.K. Peters Press, Cambridge, 1997.

[25] [Epstein and Carnielli, 1989] Epstein and Carnielli, *Computability, Computable Functions, Logic, and the Foundations of Mathematics*, 1989, Second Printing Wadsworth Thomson Learning, 2000.

[26] [Friedberg, 1957] R. M. Friedberg, Two recursively enumerable sets of incomparable degrees of unsolvability, *Proc. Natl. Acad. Sci. USA* **43** (1957), 236–238.

[27] [Friedberg-Rogers, 1959] R. M. Friedberg and H. Rogers, Jr. Reducibility and completeness for sets of integers, Z. Math. Logik Grundlag. Math. **5** (1959), 117–125.

[28] [Gandy, 1980] R. Gandy, Church's thesis and principles for mechanisms, In: *The Kleene Symposium*, North-Holland, (1980), 123–148.

[29] [Gandy, 1988] R. Gandy, The confluence of ideas in 1936, In: Herken, 55–111.

[30] [Gödel, 1931] K. Gödel, Über formal unentscheidbare sätze der Principia Mathematica und verwandter systeme. I, *Monatsch. Math. Phys.* **38** (1931) 173-178. (English trans. in Davis [1965, 4–38], and in van Heijenoort, 1967, 592–616.)

[31] [Gödel, 1934] K. Gödel, On undecidable propositions of formal mathematical systems, Notes by S. C. Kleene and J. B. Rosser on lectures at the Institute for Advanced Study, Princeton, New Jersey, 1934, 30 pp. (Reprinted in Davis [1965, p. 39–74].)

[32] [Gödel, 193?] K. Gödel, Undecidable diophantine propositions, In: Gödel [1995, 156–175].

[33] [Gödel, 1946] K. Gödel, Remarks before the Princeton bicentennial conference of problems in mathematics, 1946. Reprinted in: Davis [1965, p. 84–88].

[34] [Gödel, 1951] K. Gödel, Some basic theorems on the foundations of mathematics and their implications, In: Gödel [1995, 304–323]. (This was the Gibbs Lecture delivered by Gödel on December 26, 1951 to the Amer. Math. Soc.)

[35] [Gödel, 1964] K. Gödel, Postscriptum to Gödel [1931], written in 1946, printed in Davis [1965, 71–73].

[36] [Gödel, 1972] K. Gödel, Some remarks on the undecidability results, (written in 1972); In: Gödel [1990, p. 305–306].

[37] [Gödel, 1986] K. Gödel, *Collected works Volume I: Publications 1929–1936*, S. Feferman et. al., editors, Oxford Univ. Press, Oxford, 1986.

[38] [Gödel, 1990] K. Gödel, *Collected works Volume II: Publications 1938–1974*, S. Feferman et. al., editors, Oxford Univ. Press, Oxford, 1990.

[39] [Gödel, 1995] K. Gödel, *Collected works Volume III: Unpublished essays and lectures*, S. Feferman et. al., editors, Oxford Univ. Press, Oxford, 1995.

[40] [Goldin-Smolka-Wegner] D. Goldin, S. Smolka, P. Wegner, Interactive Computation: The new Paradigm, Springer-Verlag, 2006.

[41] [Herken, 1988] R. Herken (ed.), *The Universal Turing Machine: A Half-Century Survey*, Oxford Univ. Press, 1988.

[42] [Hilbert, 1899] D. Hilbert, *Grundlagen der Geometrie,* 7th ed., Tuebner-Verlag, Leipzig, Berlin, 1930.

[43] [Hilbert, 1904] D. Hilbert, Über die Grundlagen der Logik und der Arithmetik, In: *Verhandlungen des Dritten Internationalen Mathematiker-Kongresses in Heidelberg vom 8. bis 13. August 1904*, pp. 174–185, Teubner, Leipzig, 1905. Reprinted in van Heijenoort [1967, 129–138].

[44] [Hilbert, 1926] D. Hilbert, Über das Unendliche, *Mathematische Annalen* **95** (1926), 161–190. (English trans. in van Heijenoort [1967, 367–392].)

[45] [Hilbert, 1927] D. Hilbert, Die Grundlagen der Mathematik, *Abhandlungen aus dem mathematischen Seminar der Hamburgischen Universität* **6** (1928), 65–85. Reprinted in van Heijenoort [1967, 464–479].

[46] [Hilbert and Ackermann, 1928] D. Hilbert and W. Ackermann, *Grundzüge der theoretischen Logik*, Springer, Berlin, 1928 (English translation of 1938 edition, Chelsea, New York, 1950).

[47] [Hilbert and Bernays, 1934] D. Hilbert and P. Bernays, Grundlagen der Mathematik I (1934), II (1939), Second ed., I (1968), II (1970), Springer, Berlin.

[48] [Hodges, 1983] A. Hodges, *Alan Turing: The Enigma*, Burnett Books and Hutchinson, London, and Simon and Schuster, New York, 1983.

[49] [Hodges, 2004] A. Hodges, Alan Turing: the logical and physical basis of computing, Lecture at Manchester University, 5 June, 2004, published electronically, http://www.bcs.org/ewics.

[50] [Kleene, 1936] S. C. Kleene, General recursive functions of natural numbers, *Math. Ann.* **112** (1936), 727–742.

[51] [Kleene, 1936b] S. C. Kleene, λ-definability and recursiveness, *Duke Math. J.* **2** (1936), 340–353.

[52] [Kleene, 1936c] S. C. Kleene, A note on recursive functions, *Bull. A.M.S.* **42** (1936), 544–546.

[53] [Kleene, 1938] S. C. Kleene, On notation for ordinal numbers, *J. Symbolic Logic*, **3** (1938), 150–155.

[54] [Kleene, 1943] S. C. Kleene, Recursive predicates and quantifiers, *Trans. A.M.S.* **53** (1943), 41–73.

[55] [Kleene, 1944] S. C. Kleene, On the forms of the predicates in the theory of constructive ordinals, *Amer. J. Math.* **66** (1944), 41–58.

[56] [Kleene, 1952] S. C. Kleene, *Introduction to Metamathematics*, Van Nostrand, New York (1952). Ninth reprint 1988, Walters-Noordhoff Publishing Co., Groningën and North-Holland, Amsterdam.

[57] [Kleene, 1955] S. C. Kleene, Arithmetical predicates and function quantifiers, *Trans. A.M.S.* **79** (1955), 312–340.

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

[59] [Kleene, 1955c] S. C. Kleene, Hierarchies of number-theoretical predicates, *Bull. A.M.S.* **61** (1955), 193–213.

[60] [Kleene, 1959] S. C. Kleene, Recursive functionals and quantifiers of finite type I, *Trans. A.M.S.* **91** (1959), 1–52.

[61] [Kleene, 1962] S. C. Kleene, Turing-machine computable functionals of finite types I, Logic, methodology, and philosophy of science: Proceedings of the 1960 international congress, Stanford University Press, 1962, 38–45.

[62] [Kleene, 1962b] S. C. Kleene, Turing-machine computable functionals of finite types II, *Proc. of the London Math. Soc.* **12** (1962), no. 3, 245–258.

[63] [Kleene, 1963] S. C. Kleene, Recursive functionals and quantifiers of finite type II, *Trans. A.M.S.* **108** (1963), 106–142.

[64] [Kleene, 1967] S. C. Kleene, *Mathematical Logic*, John Wiley and Sons, Inc., New York, London, Sydney, 1967.

[65] [Kleene, 1981] S. C. Kleene, Origins of recursive function theory, *Annals of the History of Computing*, **3** (1981), 52–67.

[66] [Kleene, 1981b] S. C. Kleene, The theory of recursive functions, approaching its centennial, *Bull. A.M.S. (n.s.)* **5**, (1981), 43–61.

[67] [Kleene, 1981c] S. C. Kleene, Algorithms in various contexts, *Proc. Sympos. Algorithms in Modern Mathematics and Computer Science* (dedicated to Al-Khowarizimi) (Urgench, Khorezm Region, Uzbek, SSSR, 1979), Springer-Verlag, Berlin, Heidelberg and New York, 1981.

[68] [Kleene, 1987] S. C. Kleene, Reflections on Church's Thesis, *Notre Dame Journal of Formal Logic,* **28** (1987), 490–498.

[69] [Kleene, 1987b] S. C. Kleene, Gödel's impression on students of logic in the 1930's, In: P. Weingartner and L. Schmetterer (eds.), *Gödel Remembered*, Bibliopolis, Naples, 1987, 49–64.

[70] [Kleene, 1988] S. C. Kleene, Turing's analysis of computability, and major applications of it, In: Herken *1988*, 17–54.

[71] [Kleene-Post, 1954] S. C. Kleene and E. L. Post, The upper semilattice of degrees of recursive unsolvability, *Ann. of Math.* **59** (1954), 379–407.

[72] [Lerman, 1983] M. Lerman, *Degrees of Unsolvability: Local and Global Theory*, Springer-Verlag, Heidelberg New York Tokyo, 1983.

[73] [Muchnik, 1956] A. A. Muchnik, On the unsolvability of the problem of reducibility in the theory of algorithms, *Doklady Akademii Nauk SSR* **108** (1956), 194–197, (Russian).

[74] [Odifreddi, 1989] P. Odifreddi, *Classical Recursion Theory*, North-Holland, Amsterdam, Volume I 1989, Volume II 1999.

[75] [Olszewski, 2007] A. Olszewski and J. Wolenski (Eds.), Church's Thesis After 70 years, Ontos Verlag, 2007. (551 pages, hardbound.) (ISBN-13: 978-3938793091.)

[76] [Peter, 1934] R. Péter, Über den Zussammenhang der verschiedenen Begriffe der rekursiven Funktion, *Mathematische Annalen* **110** (1934), 612–632.

[77] [Peter, 1951] R. Péter, *Rekursive Funktionen*, Akadémaiai Kiadó (Akademische Verlag), Budapest, 1951, 206 pp. *Recursive Functions*, Third revised edition, Academic Press, New York, 1967, 300 pp.

[78] [Post, 1936] E. L. Post, Finite combinatory processes–formulation, *J. Symbolic Logic* **1** (1936) 103–105. Reprinted in Davis [1965, 288–291].

[79] [Post, 1941] E. L. Post, Absolutely unsolvable problems and relatively undecidable propositions: Account of an anticipation. (Submitted for publication in 1941.) Printed in Davis [1965, 340–433].

[80] [Post, 1943] E. L. Post, Formal reductions of the general combinatorial decision problem, *Amer. J. Math.* **65** (1943), 197–215.

[81] [Post, 1944] E. L. Post, Recursively enumerable sets of positive integers and their decision problems, *Bull. Amer. Math. Soc.* **50** (1944), 284–316. (Reprinted in Davis [1965, 304–337].)

[82] [Post, 1946] E. L. Post, Note on a conjecture of Skolem, J. Symbolic Logic *11* (1946), 73–74.

[83] [Post, 1947] E. L. Post, Recursive unsolvability of a problem of Thue, *J. Symbolic Logic* **12** (1947), 1–11. (Reprinted in Davis [1965, 292–303].)

[84] [Post, 1948] E. L. Post, Degrees of recursive unsolvability: preliminary report (abstract), *Bull. Amer. Math. Soc.* **54** (1948), 641–642.

[85] [Putnam, 1956] H. Putnam, Trial and error predicates and the solution to a problem of Mostowski, *J. Symbolic Logic* **30**, 49–57.

[86] [Rogers, 1967] H. Rogers, Jr., Theory of Recursive Functions and Effective Computability, McGraw-Hill, New York, 1967.

[87] [Sacks, 1990] G. E. Sacks, *Higher Recursion Theory*, Springer-Verlag, Heidelberg New York, 1990.

[88] [Shoenfield, 1967] J. R. Shoenfield, *Mathematical Logic*, Addison-Wesley, Reading, Mass. (1967), 344 pp.

[89] [Shoenfield, 1971] J. R. Shoenfield, *Degrees of Unsolvability*, North-Holland, Amsterdam, London, New York, 1971.

[90] [Shoenfield, 1991] J. R. Shoenfield, Recursion Theory, *Lecture Notes in Logic*, Springer-Verlag, Heidelberg New York, 1991.

[91] [Shoenfield, 1995] J. R. Shoenfield, The mathematical work of S. C. Kleene, *Bull. A.S.L* **1** (1995), 8–43.

[92] [Sieg, 1994] W. Sieg, Mechanical procedures and mathematical experience, In: A. George (ed.), *Mathematics and Mind*, Oxford Univ. Press, 1994.

[93] [Soare, 1987] R. I. Soare, *Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets*, Springer-Verlag, Heidelberg, 1987.

[94] [Soare, 1996] R. I. Soare, Computability and recursion, *Bulletin of Symbolic Logic* **2** (1996), 284–321.

[95] [Soare, 1999] R. I. Soare, The history and concept of computability, In: *Handbook of Computability Theory*, ed. E. Griffor, North-Holland, Amsterdam, 1999, 3–36.

[96] [Soare, 2000] R. I. Soare, Extensions, Automorphisms, and Definability, in: P. Cholak, S. Lempp, M. Lerman, and R. Shore, (eds.) Computability Theory and its Applications: Current Trends and Open Problems, American Mathematical Society, Contemporary Math. #257, American Mathematical Society, Providence, RI, 2000. pps. 279–307.

[97] [Soare, 2004] R. I. Soare, Computability theory and differential geometry, *Bull. Symb. Logic*, Vol. 10 (2004), 457–486.

[98] [Soare, 2007] R. I. Soare, Computability and Incomputability, Computation and Logic in the Real World, in: Proceedings of the Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18–23, 2007, Lecture Notes in Computer Science, No. 4497, S.B. Cooper, B. Löwe, Andrea Sorbi (Eds.), (Springer-Verlag, Berlin, Heidelberg, 2007.

[99] [Soare, CTA] R. I. Soare, *Computability Theory and Applications,* Springer-Verlag, Heidelberg, to appear.

[100] [Soare, ta] R. I. Soare, Turing-Post Oracle Computability and Realigning Computability Theory, in: Jack Copeland, Carl Posy, and Oron Shagrir (eds.), *Computability: Gödel, Church, Turing, and Beyond,* MIT Press, to appear.

[101] [Teuscher, 2004] C. Teuscher (ed.), Alan Turing: Life and legacy of a great thinker, Springer-Verlag, 2004.

[102] [Turing, 1936] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.* ser. 2 **42** (Parts 3 and 4) (1936) 230–265; [Turing, 1937a] A correction, ibid. **43** (1937), 544–546.

[103] [Turing, 1937b] A. M. Turing, Computability and λ-definability, *J. Symbolic Logic*, **2** (1937), 153–163.

[104] [Turing, 1939] A. M. Turing, Systems of logic based on ordinals, *Proc. London Math. Soc.* **45** Part 3 (1939), 161–228; reprinted in Davis [1965, 154–222].

[105] [Turing, 1948] A. M. Turing, Intelligent machinery, In: *Machine Intelligence* **5**, 3–23. (Written in September, 1947 and submitted to the National Physical Laboratory in 1948.)

[106] [Turing, 1949] A. M. Turing, Text of a lecture by Turing on June 24, 1949, In: F. L. Morris and C. B. Jones, "An early program proof by Alan Turing," Annals of the History of Computing **6** (1984), 139-143.

[107] [Turing, 1950] A. M. Turing, Computing machinery and intelligence, *Mind* **59** (1950) 433–460.

[108] [Turing, 1950b] A. M. Turing, The word problem in semi-groups with cancellation, *Ann. of Math.* **52** (1950), 491–505.

[109] [Turing, 1954] A. M. Turing, Solvable and unsolvable problems, *Science News* **31** (1954), 7–23.

[110] [Turing, 1986] A. M. Turing, Lecture to the London Mathematical Society on 20 February 1947, In: B. E. Carpenter and R. W. Doran, eds., *A. M. Turing's ACE Report of 1946 and Other Papers*, Cambridge Univ. Press, 1986, 106–124.

[111] [Wang, 1974] H. Wang, *From Mathematics to Philosophy*, Routledge & Kegan Paul, London, 1974.

[112] [Wang, 1981] H. Wang, Some facts about Kurt Gödel, *J. Symbolic Logic* **46** (1981) 653–659.

[113] [Wang, 1987] H. Wang, Reflections on Kurt Gödel, MIT Press, Cambridge, MA.

[114] [Wang, 1993] H. Wang, On physicalism and algorithmism: can machines think?, *Philosophia Mathematica, 3rd series* **1** (1993), 97–138.

Department of Mathematics
University of Chicago
Chicago, Illinois 60637-1546
soare@uchicago.edu
URL: http://www.people.cs.uchicago.edu/∼soare/