The Interplay Between Computability and Incomputability Draft 619.tex

Robert I. Soare*

January 7, 2008

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,klee Alan Turing, Kurt Gödel, Emil Post, computability, incomputability, undecidability, Church-Turing Thesis (CTT), Post-Church Second Thesis on relative computability, computable approximations, Limit Lemma, effectively continuous functions, computability in analysis, strong reducibilities.

Since the time of the Babylonians mathematicians have been exploring the notions of algorithm and computable function and devices for carrying out calculations. However, Church *1936* and Turing *1936* were more interested in investigating *incomputable* problems. They gave the present definitions of computable functions (λ-definable functions, recursive functions, Turing machines, called a-machines) primarily to diagonalize over them and produce incomputable (unsolvable) problems in ordinary mathematics in order to refute Hilbert's *Entscheidungsproblem* (decision problem).

Turing *1939* introduced the more powerful notion of an *oracle* machine (o-machine) and *relative computability* of one set B from another set A to help understand the information content of unsolvable problems. The o-machine also gives the notion of an effectively continuous function, the analogue of a continuous function in analysis.

The Turing o-machine, not the Turing a-machine, is the fundamental notion of computability theory. It is a *computability* notion, not an incomputability notion. It links the fundamental notions of computability theory and analysis because any continuous function in analysis is a Turing functional relative to a real parameter $X \subseteq \omega$. In a calculus or analysis book, continuous functions are presented almost immediately, but in many computability books, Turing functionals are not presented until the second half of the book, if at all.

We give an historical development of these notions and an explanation of why there has been so much emphasis on Turing a-machines and the Church-Turing Thesis 3.2 , and so little on Turing o-machines and on the *Post-Turing Second Thesis 6.1.* We also explore how the twin themes of computability and incomputability reinforced and stimulated the development of one another.

1 Calculus, Continuity, and Computability

Open any book on calculus or introductory analysis. The definition of a continuous function will be usually presented in the first couple of chapters. Now open an introductory book on computability theory. The corresponding definition of an effectively continuous function, one given by an Turing oracle machine or by recursive functions relativized to an oracle $A \subseteq \omega$, rarely occurs in the very beginning of the book, and sometimes does not occur at all. The book usually starts with the definition of Turing's automatic machine (a-machine), now called a *Turing machine*, and its formal equivalence with recursive functions and other formalisms demonstrated by Turing, Church, and Kleene.

1.1 When to Introduce Relative Computability?

It is clear that many of the leading authors of introductory textbooks and references on computability introduce relative computability and effective continuity relatively late in their books. For example, Kleene's book *1952, Introduction to Metamathematics,* 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 Odifreddi, *Classical Recursion Theory* Vol. I *1989* and Vol. II *1999* introduces 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 function. 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* or Soare *1987*.

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.

My own previous book, Soare *1987,* introduces oracle machines and relative computability on page 47 of Chapter III, but not in Chapter I. Because of the increasing importance of Turing reducibility and effective continuity as the key concept of the subject, I considered introducing it in Chapter 1 of my new book *Computability Theory and Applications [CTA]*, and then drawing the ordinary Turing machines as a special case. I was dissuaded by my graduate students who convinced me that beginning students need time to

become familiar with the basic notions of Turing machines and computably enumerable sets before learning of oracle Turing machines. Nevertheless, I have introduced oracle machines and relative computability as soon as feasible in Chapter 3 of the new book *CTA,* and I have made it the central notion of the book.

1.2 Between Computability and Relative Computability?

If most computability books begin with ordinary Turing machines in Chapter 1 and do not cover oracle machines and relative computability until late in the book, what do they cover in between? Many books study numerous properties of Turing machines and computable functions. There is fine, but one must remember that, viewed as a function F from 2^ω to 2^ω if F is defined by an a-machine, then F is merely a constant function on reals (subsets of ω) because any oracle A produces the same output B from F because F, being an a-machine does not consult any oracle. Constant functions are important in calculus, but no one would spend weeks on them.

Many books follow the example of Post *1944* and study strong reducibilities such as many-one reducible ($A \leq_{\rm m} B$) or tt-reducibility ($A \leq_{\rm tt} B$). This is analogous to having a calculus book spend many weeks on specific examples of continuous or differentiable functions, such as polynomials, trigonometric functions, and rational functions, before even defining a continuous or differentiable function. Both strong reducibilities in computability and polynomials in calculus are excellent examples and are very useful, but they cannot take the place of the general case of continuity or relative computability (effective continuity), fundamental concepts of the subjects.

1.3 The Development of Relative Computability

There are historical reasons for the emphasis on ordinary Turing computability and the Church-Turing Thesis and a lack of emphasis on Turing oracle machines, relative computability, and Turing's Second Thesis on relative computability. The computability investigations in the 1930's were strongly influenced by Hilbert's *Entscheidungsproblem* stated in *1928*. The main object for Church and Turing in the early 1930's was to give a precise definition of computable so that one could diagonalize against the computable functions and produce an undecidable problem in mathematics.

1.4 Turing Introduces Relative Computability

Turing's description *1939* of an oracle machine was a tiny and obscure part of his paper. He might have gone further in later papers, but the Second World War began that same year and he immediately entered Bletchley Park, the British cryptographic facility for the duration of the war. During this time he helped build actual computers partly based on his earlier theoretical ideas. When he emerged in 1946 he moved to the University of Manchester to build computers and never returned to questions of oracle machines and relative computability, but spent his time instead on the topics covered in the papers, Turing *1948* through *1954.*

1.5 Post Develops Relative Computability

From 1939, when Turing left the subject, until his death in 1954, Emil Post continued where Turing had stopped and brought the notion of relative computability to the forefront. He also played a key role in our understanding of Turing reducibility as an effectively continuous function on Cantor space or Baire space, and the use of this property to classify the degree of unsolvability of sets. His introduction of finite forcing in Kleene-Post *1954* gave us the most important tool for solving Post's Problem. Post did more than anyone else to reveal the intuition and significance behind Gödel's results, to classify the information content of computably enumerable sets, and to understand not only Turing reducibility but several stronger reducibilities as well.

1.6 Relative Computability in Real World Computing

We also explore why relative computability and oracle programs play a much larger role in real world computing than the original Turing machine models or programs without input/output (i/o) devices.

2 Origins of Computability and Incomputability

Mathematicians have studied algorithms and computation since the time of the Babylonians. Kleene *1988,* page 19 wrote, "The recognition of algorithms goes back at least to Euclid (c. 330 B.C.)" with Euclid's famous greatest common divisor algorithm. The name "algorithm" comes from the name of the ninth century Arabian mathematician Al-Khowarizmi who gave us algebra as well as algorithms.

Along with the development of theoretical mathematical algorithms there developed an interest in actual calculating machines. In 1642 the French mathematician and scientist, Blaise Pascal, invented an adding machine which may be the first digital calculator. In 1671 the German mathematician and philosopher, Gottfried Wilhelm Leibniz, co-inventor with Newton of the calculus, invented a machine that performed multiplication. Leibniz's machine, the *stepped reckoner* could not add and multiply, and also divide, and extract square roots. His stepped gear wheel still appears in a few twentieth century devices. Leibniz' main contribution was the demonstration of the superiority of the binary over the decimal representation for mechanical computers. Around 1834 Babbage invented the idea of an "Analytic Engine," which could be programmed to perform long and tedious calculations.

In each of these instances algorithms or mechanical devices were used to extend our knowledge of *computability* in mathematics. This changed in the twientieth century. By the 1930's the emphasis was on understanding *incomputability*. As the twentieth century opened, Georg Cantor had introduced set theory in order to solve a problem in trigonometric series. Paradoxes with his naive set theory soon demonstrated that unrestricted set formation (comprehension) led to contradictions. Mathematicians scrutinized definite axioms for their mathematics and searched for a proof of consistency of the axioms. The geometer and topologist Oswald Veblen, later the thesis adviser of Alonzo Church at Princeton, and the first professor in the Institute for Advanced Study, completed his Ph.D. in 1903 at the University of Chicago under E.H. Moore with a dissertation called, "A System of Axioms for Geometry." In 1927 Church completed under Veblen at Princeton his thesis, "Alternatives to Zermelo's Assumption."

David Hilbert, one of the two leading mathematicians in 1900 along with Poincaré, 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 became known by *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 of the early twientieth century was *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 formally defined in Hilbert and Ackermann *1928*. It was to give a decision procedure (*Entscheidungsverfahren*) "that allows one to decide the validity of the sentence." (Here "valid" means "true in the standard structure," not the modern sense of valid as true in all structures.) Hilbert characterized this as the fundamental problem of mathematical logic. Davis *1965,* page 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 an 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 extension T of number theory (Peano arithmetic) there is a sentence σ which asserts its own unprovability. If T proves σ then T is inconsistent; if not then T is incomplete. Few in the audience understood the importance of Gödel's Incompleteness Theorem. One who understood at once was John von Neumann who was at the conference representing Hilbert's proof theory program, and recognized that Hilbert's program was over. In the next couple 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. He brought his proof to Gödel who thanked him and informed him politely that Gödel had aleady 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 previously sacrosanct Hilbert. 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 Incomputability and Undecidability

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 behind Hilbert's ideas 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.

2.3 Alonzo Church

After completing his Ph.D. at Princeton in 1927, 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 but 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 in 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. On the strength of this evidence, Church proposed to Gödel around March, 1934 (see Davis *1965,* p. 8–9) that the notion of "effectively calculable" be identified with "λ-definable."

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 the diagonal was one of the rows. Instead of a contradiction, Kleene had proved a beautiful new theorem, the Kleene Recursion Theorem, described in Soare *1987,* p. 36, whose proof is a diagonal argument which fails. However, Gödel rejected as Church's first thesis as "thoroughly unsatisfactory."

Gödel had come to Princeton from Europe by 1934. He knew that the primitive recursive functions which he has used in his 1931 paper did not constitute all computable functions. He expanded on a concept of Herbrand, modifying it to be more effective, and at Princeton in 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. (Strictly speaking, since 1934 "recursive function" has meant "Herbrand-Gödel recursive," not the μ-recursive functions of Kleene, although the two are extensionally equivalent.)

2.4 Herbrand-Gödel Recursive Functions

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.

2.5 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 should be identified with that of a recursive function, ..."

It has been known since Kleene *1952* as *Church's Thesis* in this 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 class, 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.

2.6 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 researcher 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 (his footnote 3), gives crucial insight into Gödel's 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."

3 Turing Breaks the Stalemate

At the start of 1936 those gathered at Princeton, Gödel, Church, Kleene, Rosser, and Post nearby, constituted the most distinguished and powerful group in the world investigating the notion of a computable function and Hilbert's *Entscheidungsproblem*. At that moment stepped forward a twenty-two year old youth, far removed from Princeton. Well, not just any youth. Alan Turing had already proved the Central Limit Theorem in probability theory, not knowing it had been previously proved, as described in Zabell *1995*. As a result Turing had been elected a Fellow of King's College, Cambridge.

3.1 Turing's Machines and Turing's Thesis

The work of Hilbert and Gödel had become well-known around the world. At Cambridge University topologist Professor 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 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), defined in Turing *1936*.

Turing showed his solution to the astonished Professor 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 on similar material. Newman persuaded them that Turing's work was sufficiently different, and they published his paper in volume 42 of the journal[1] on November 30, 1936 and December 23, 1936.

Turing's machine has compelling simplicity and logic which makes it even today the most convincing model of computability. qually important with this 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. Turing asserted the following thesis.

3.2 Gödel's Opinion of Turing's Work

Gödel's reaction was swift and emphatic. He never accepted Church's Thesis, but he accepted Turing's Thesis immediately. Gödel was interested in the *intensional* analysis of finite procedure. He never believed the arguments and confluence evidence which Church presented to justify his Thesis. On the other hand Gödel accepted immediately not only Turing machines, but more importantly, the analysis Turing gave of a finite procedure. The fact that Turing machines were later proved *extensionally* equivalent to general recursive functions did not convince Gödel of the intrinsic merit of the other definitions.

"That this really is the correct definition of mechanical com-

[1]Many papers, Kleene [*1943*, p. 73], *1987*, *1987b*, Davis [*1965*, p. 72], Post [*1943*, p. 200], 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 *1937*. 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, issued December 23, 1936. No part of Turing's paper appeared in 1937, but the two page minor correction *1937* 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*.

putability was established beyond any doubt by Turing."
-*Gödel* 193? *Notes in Nachlass [*1935*]*

" But I was completely convinced only by Turing's paper."
-*Gödel: letter to Kreisel of May 1, 1968 [Sieg, 1994, p. 88].*

" ...one [Turing] has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, *i.e.*, one not depending on the formalism chosen."
-*Gödel, Princeton Bicentennial, [1946, p. 84].*

" ...For the concept 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: [1946, p. 84], Princeton Bicentennial*

"The greatest improvement was made possible through the precise definition of the concept of finite procedure, ... This concept, ... is equivalent to the concept of a 'computable function of integers' ... The most satisfactory way, in my opinion, is that of reducing the concept of finite procedure to that of a machine with a finite number of parts, as has been done by the British mathematician Turing."
—-*Gödel [1951, pp. 304–305], Gibbs lecture*

" ...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."
-*Godel's Postscriptum to Gödel [1934], see Davis, [1965].*

3.3 Kleene Said 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]

" 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]

3.4 Church Said 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]

3.5 Naming the Church-Turing Thesis

Neither Church nor Turing referred to their hypotheses as a "thesis" but saw them as definitions. The definitions of Church and Turing were not even called "theses" at all until Kleene [*1943*, p. 60] referred to Church's "definition" as "Thesis I."

In his highly influential paper *1944* Post did not use the terms "Church's Thesis" or "Turing's Thesis" at all. By then Post had accepted Church's definition and terminology of recursive function as the formal equivalent of effectively calculable. Post simply wrote,

> "The importance of the technical concept recursive function derives from the overwhelming evidence that it is coextensive with the intuitive concept of effectively calculable function."

It was nearly a decade later when Kleene's first book *1952* appeared, and Kleene referred to "Church's Thesis" and "Turing's Thesis." Kleene there and in later papers used the phrase "Church's Thesis" to denote the claim that any intuitively computable function is formally computable by any one of the formalisms, recursive functions, Turing machines, or λ-definable functions. Although Kleene also initiated the phrase "Turing's Thesis" at the same place in his book *1952* as Church's Thesis, Kleene apparently did not see the need to append Turing's name to it. Therefore, after 1952 the phrase "Church's Thesis" came to denote also "Turing's Thesis" and perhaps others as well, thereby blurring all intensional distinctions.

Thirty years after the fact, Kleene wrote another book *Mathematical Logic, 1967* at a more introductory level than his first book *1952*. In §41 page 232 Kleene discusses Church's Thesis and Turing's Thesis. He notes that they are extensionally equivalent because λ-definable and Turing computable functions are mathematically equivalent by Turing *1937b* and hence the same as general recursive functions. Kleene wrote on p. 232,

> "So Turing's and Church's theses are equivalent. We shall usually refer to them both as *Church's Thesis.* or in connection with that one of its three versions which deals with 'Turing machines' as *the Church-Turing thesis.*

However, Kleene p. 233 then based his presentation of computable functions on Turing machines. He continued,

> "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 or 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."

Kleene went on during pages 233–240 to develop the entire theory of computable functions on Turing machines and he closed on p. 240.

> "For further discussion of Church's Thesis or the Church-Turing Thesis, we refer to the literature."

However, by 1967 the name "Church's Thesis" for the Church-Turing Thesis or Turing's Thesis had stuck. It had become so well established that, for example Rogers *1967,* referred to a "proof by Church's Thesis" meaning that one can write a precise but informal algorithm for an intuitively computable function, and then simply appeal to Church's Thesis for the existence of a corresponding Turing machine without explicitly describing it. In general Turing's name was not mentioned in connection with the thesis. Rogers' book was highly influential after 1967 because it was intuitive Post *1944* had been, and because it was based on Turing machines, not the heavy formalism of Kleene *1952.* It is significant that the new book *Church's Thesis after 70 years* by Olszewski and Wolenski *2007* does not mention Turing in the title, although a number of papers there have "Church-Turing Thesis" in their titles. In modern times it seems the fairest and most accurate way of stating the thesis is the following.

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

4 Turing Defines Relative Computability

In most of the previous studies of computability with Leibnitz, Pascal, or Babbage, advances in computability led to an deeper study of computability and further advances theoretical and practical, sometimes with the invention of mechanical computing devices. However, during the period 1936–39 immediately the discoverers of computability studied *incomputability*. Applications of Turing machines, the λ-calculus, and recursive functions in real world computing were to come only later after 1940.

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}.

After Turing's discovery in April, 1936, Professor Newman suggested that he go to Princeton to take his Ph.D. Turing studied under Church from 1936–1938. 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 any 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 α. Turing's thesis *1939* concerned such an increasing array of 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."

There are several equivalent ways that a Turing machine with oracle may be defined. We prefer the definition in Soare *1987,* p. 46, of a machine with a head which reads the work tape and oracle tape simultaneously. 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 $\widetilde{P}_e$ defines a partial computable functional $\Phi_e^A(x) = y$.

If $\{\varphi_e\}_{e\in\omega}$ is a list of partial computable functions produced by the ordinary Turing machines, Turing a-machine programs P_e, then we define the graph as follows.

$$\text{graph}(\varphi_e) \quad = \quad \{\langle x, y\rangle : \varphi_e(x) = y\}$$

For an oracle machine program $\widetilde{P}_e$ we likewise define the *oracle graph* of Φ_e,

$$(1) \qquad G_e \quad =_{\text{dfn}} \quad \text{graph}(\Phi_e) \ =_{\text{dfn}} \quad \{\langle \sigma, x, y\rangle : \Phi_e^\sigma(x) = y\}$$

where $\Phi_e^\sigma(x) = y$ 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 is the following.

4.2 The Oracle Graph Theorem

Theorem 4.1. Oracle Graph Theorem.

(i) If $\widetilde{P}_e$ is the oracle program defining Turing functional Φ_e then the graph G_e is a computably enumerable (c.e.) set.

(ii) For any c.e. set W_e, regarded as a set of triples, we can find a subset $G_i \subseteq W_e$ which is the oracle graph of a Turing functional Φ_i.

4.3 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 of to the Kleene μ-recursive function definition. Each of these formal definitions produces a c.e. graph G_e. Hence, by the Oracle Graph Theorem 4.1 they are all equivalent. Any Turing a-machine can clearly be simulated by a Turing o-machine.

Theorem 4.2. *If P_e is a Turing program for an 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 has consists of a finite partial map

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

where $S_2 = \{B, 1\}$ 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 $\delta(q, s) = (p, X)$ we add to oracle program $\widetilde{P}_i$ a line $\widetilde{\delta}(q, a, s) = (p, X)$ for both $a = 0$ and $a = 1$. Hence, $\widetilde{P}_i$ has exactly the same effect on input x as P_e regardless of the oracle A. □

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 such as general recursiveness but merely expressed the expectation that this would turn out to be true, while Turing 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*. 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 1939 to 1954 that Post's remarkable influence was fully felt.

5.2 Post Steps Into Turing's Place During 1939–1954

As Turing left the subject of pure computability theory in 1939 his mantle fell on the shoulders of Post, 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 fifteen years from 1939 to his death in 1954, Post played a key role in the development of 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, and 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 in terms of generated sets and production systems, which asserted that "any generated set is a normal 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.

He 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.

Post's Thesis is equivalent to Turing's Thesis because every nonempty recursively enumerable set is the range of a Turing computable function and conversely given any recursively enumerable set of ordered pairs, one can effectively find a single valued subset and this is the graph of some partial computable function.

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 **??** 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 generalized c.e. set as basic and to derive generalized computable functions as those whose graphs are generalized computably enumerable.

5.3 Post Problem on Incomplete C.E. Sets

Post 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 of 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

Poste 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.*, whether $\emptyset <_{\mathrm{T}} W <_{\mathrm{T}} K$. In 1944 people did not fully understand full Turing reducibility, and Post did not explicitly discuss it until the very end of his paper. He began by defining a number of strong reducibilities instead, such as m-reducibility($B \leq_{\mathrm{m}} A$, truth-table reducibility ($B \leq_{\mathrm{tt}} A$), and a succession 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 lead to an understanding of Turing reducibility or a solution of Post's problem. Post was able to exhibit incomplete 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.

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 Second 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 Thesis.

Definition 6.1. Post-Turing Second 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 Second Thesis is equivalent to Post's First Thesis **??** (because G_e is c.e.), which is equivalent to Turing's First Thesis **??**.

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 Second 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; computing in the real world where actual computers use oracle programs much more than ones contained entirely inside the machine.

7 Effectively Continuous Functions

7.1 Representations of Open Sets

It is useful to view a Turing reduction Φ_e as a continuous functional on Cantor space 2^ω.

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 this 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 $f \in 2^\omega$.

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

$$\mathcal{N}_\sigma = \{\ f : f \in 2^\omega \quad \& \quad \sigma \subset f\ \}$$

where $\sigma \subset f$ denotes that the function f extends σ. 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, so the open sets are closed under finite intersection and countable union.

(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 $\sigma \subset \tau$ implies $\tau \in A$.

(v) A set $\mathcal{C}$ is *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 $T =_{\rm dfn} 2^{<\omega} - A$ is a *closed representation* for $\mathcal{C}$. Now T is closed *downwards* (because A is closed upwards). Hence, T forms a *tree*. For a tree $[T]$ we have the closed set $[T] = \overline{\mathcal{N}}_{\overline{T}}$, the complement of the open set $\mathcal{N}_A$ for $A = \overline{T}$.

7.2 Continuous and Effectively Continuous Functions

In elementary calculus courses a continuous function is usually defined with δ and ϵ concepts or with limits. However, in more advanced analysis or topology courses the more general definition is given that a function F is continous iff the image of every basic open set is open, *i.e.*, a countable union of basic open sets. We state continuity now for functions on the Cantor space 2^ω (or Baire space ω^ω which differs from Cantor space mainly because Cantor space is compact while Baire space is not). We use the notation of Definition 7.1.

Definition 7.2. (i) A function F on Cantor space 2^ω is *continuous* iff for every $\tau \in 2^{<\omega}$ there is a countable set D such that

$$(2) \quad F^{-1}(\mathcal{N}_\tau) \quad = \quad \cup \{ \mathcal{N}_\sigma \, : \, \sigma \in D \}.$$

By identifying strings σ with their code numbers we can think of D as a subset of ω.

(ii) The function F is *effectively continuous* if the set D may be chosen to be computably enumerable. (If so, then by a slight trick D may be chosen to be computable.)

7.3 Continuity of Turing Functionals

In the last section of his *1944* paper and in *1948* Post defined the notion of *degrees* of computability (unsolvability). Two sets A and B have the same degree, $A \equiv_{\rm T} B$, if they are equicomputable, *i.e.*, $A \leq_{\rm T} B$ and $B \leq_{\rm T} A$. This did not solve Post's problem at once, but it led Post to think more deeply about the nature of Turing reducibility. Nearing the end of his life and very sick, he still produced a stack of notes on Turing reducibility and gave them to Kleene before he died. Kleene rewrote them and published them as Kleene-Post *1954*. This remarkable paper revealed for the first time the fundamental continuity of a Turing functional $\Phi_e : \, 2^\omega \to 2^\omega$, although Post had described the ideas intuitively in the last section of *1944*.

The key properties used over and over are: (i) that any oracle Turing computation $\Phi_e^A(x) = y$ from oracle A depends on only a finite initial segment $\sigma \subset A$ of the oracle A denoted by $\Phi_e^\sigma(x) = y$ (because any computation converges in finitely many steps); and (iii) that if (i) holds for A via σ, then any other set $C \supset \sigma$ must also produce the same value $\Phi_e^C(x) = y$ because σ is the only part of the oracle C which the computation will examine. Post discussed some of these ideas in *1944* as mentioned in §6.1. The properties may be summarized in the following theorem.

Theorem 7.3. [Continuity of Turing Functionals].

(i) $\Phi_e^A(x) = y \implies (\exists s)\,(\exists \sigma \subset A)\,[\,\Phi_{e,s}^\sigma(x) = y\,]$;

(ii) $\Phi_{e,s}^\sigma(x) = y \implies (\forall t \geq s)\,(\forall \tau \supseteq \sigma)\,[\,\Phi_{e,t}^\tau(x) = y\,]$

(iii) $\Phi_e^\sigma(x) = y \implies (\forall A \supset \sigma)\,[\,\Phi_e^A(x) = y\,]$.

Using this understanding of Turing reductions, Kleene and Post *1954* invented an oracle construction computable in K with finite forcing to define sets $A \leq_{\mathrm{T}} K$ and $B \leq_{\mathrm{T}} K$ which are Turing incomparable ($A \mid_T B$). Although this did not immediately solve Post's problem, it provided the key ingredients for the solution soon thereafter by Friedberg *1957* and Muchnik *1956*. They took the forcing and continuity argument of Kleene and Post and replaced the oracle K by a computable approximation in which the strategy for a given requirement is restarted whenever previous action is injured by action for a higher priority requirement. This paper and all later papers in the subject made heavy use of the continuity of Turing functionals. Their paper may be looked at as an effectivization of the Baire Category Theorem and Banach-Mazur games where one finds a point in the intersection of a countable sequence of dense open sets by performing a finite extension for each dense open set.

7.4 Continuous Functions are Relatively Computable

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

Theorem 7.4. *Suppose F is a continuous functional on 2^ω. Then F is a Turing functional relative to some real parameter $X \subseteq \omega$.*

Proof. Since F 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,

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

Therefore, the set $X = \oplus\{X_\tau : \tau \in 2^{<\omega}\}$ provides a complete oracle for calculating $F : 2^\omega \to 2^\omega$. Equivalently, F has a graph G_e as defined in (1) which is X-c.e. and hence there is a Turing functional Φ_e computable in X such that $F(A) = B$ iff $\Phi_e(A) = B$. □

Let $\mathbf{c}$ be the cardinality of 2^ω, *i.e.*, the cardinality of the continuum. Now by cardinal exponentiation there are $\mathbf{c}^\mathbf{c}$ functions from 2^ω to 2^ω. However, there are only $\mathbf{c}$ many *continuous* such functions because by Theorem 7.4 each must be specified by a real parameter $X \subseteq \omega$ of which there are only $\mathbf{c}$ many. Although we can define pathological functions which are discontinuous, the ones we actually study in calculus or more advanced analysis courses are usually continous, or at least piecewise continuous. Moreover, most of them are computable. It is rare to encounter a noncomputable function if an elementary calculus course.

8 Computing in the Real World

8.1 Turing o-machines in the Real World

Turing described his original a-machines as having an infinite tape, and we usually present them this way today. Of course, all real world implementations of the Turing machine have a finite tape, but that has not prevented them from incorporating many of the features of a Turing machine. The tape need not be infinite, just large enough for the given computation.

Likewise, Turing's oracle machine was defined by him, and is generally presented now with an infinite, and indeed noncomputable, oracle. Turing *1939* wrote "a kind of oracle as it were. ...this oracle ...cannot be a machine." Presumably Turing put on this restriction because otherwise the oracle could be incorporated into the program for some new a-machine which would have no need of any oracle.

For this paper we shall identify a Turing *a-machine* with any computing device in the real world which has a finite program but no i/o-device: no floppy drive, CD drive, DVD drive, no airport wireless card, modem, no ethernet connection. All input must be entered through the keyboard. The program and data are internal, usually keyed into the computer from the keyboard, like a hand calculator.

However, in real world computing we often attach to our laptop or desktop computers an input/output (i/o) device regarded here as an "oracle" which the laptop program may consult during a computation. For example, our laptops often have a CD/DVD drive, and a wireless or ethernet connection to the world wide web, or other i/o devices. For the purposes of this paper we regard such computers with i/o devices as oracle machines where the oracle (such as the web) is too large or too inconvenient to be completely incorporated into the laptop program and is consulted by the program through an i/o device such as a CD/DVD drive or wireless or ethernet connection the web. Hence, the laptop has its internal program but derives information from the oracle during the computation.

In the 1960's computing was done at the computer center of a university where one turned in a stack of punched IBM cards and a few hours later collected the folded paper output. By 1980 IBM was selling personal computers (PC computers). Even these had primitive i/o devices such as five and a quarter inch floppy drives and modems. These were later replaced by CD drives, DVD drives, and were eventually supplemented by by wireless card to connect to the internet and world wide web, which we may think of as a vast oracle.

An a-machine alone is simply a giant calculator, good for calculating the n^{th} place in the characteristic function of π but not useful for much else in isolation since it is too tedious to enter all the information by keyboard. Most of the programs for our laptop and desktop computers are oracle type programs, which consult or download some information from an external source and then process it with the local machine. It other words, *most of our computing in the real world is done on o-machines with access to some oracle, not on a-machines with no i/o device. Yet we still spend the majority of our time in introductory computability courses on a-machines not o-machines.* The great emphasis on ordinary computability, Turing a-machines and the Church-Turing Thesis 3.2 and the underemphasis on relative computability, Turing o-machines and the Post-Turing Second Thesis 6.1 is hard to understand in view of computing practice in the real world.

8.2 The Limit Lemma and Real World Computing

We normally place great emphasis in introductory computability courses on functions which are computable, or at least partial computable. However, many problems attacked by computers in the real world do not admit of an exact algorithmic solution but rather a sequence of computable approxima-

tions which may converge to the answer or at least partially converge to an approximation.

For example, in machine learning or artificial intelligence a robot may navigate a maze, learning from its mistakes and continually update its information about the maze. A meteorologist may be asked to predict the weather on a certain date in the future, and his updated predictions for that date may become more accurate as the date draws closer and he had more information. A trader of financial futures may make a contract for the price of a currency at the opening today, and update his bid by computer every minute during the day.

Definition 8.1. A computable sequence of computable functions $\{f_s\}_{s\in\omega}$ is a Δ_2-*approximating sequence* for a f on ω if $f = \lim_s f_s$. If there is such a sequence we say that f is *limit computable* and we call f a Δ_2^0 *function* because it can be shown that f can be represented in both $\forall\exists$ and $\exists\forall$ forms over a computable predicate.

The most useful result about limit computable functions if the following.

Lemma 8.2. [Limit Lemma] *f is limit computable iff $f \leq_{\mathrm{T}} A$ for some c.e. set A.*

Now suppose a person is using a laptop computer to trade financial instruments around the world, and his computer is updating the prices around the world every minute. He will not have time to write a new program every minute. Rather it is likely he will use a fixed program inside the laptop, and use the web or another external device to bring the data needed to update the program at least every minute. Then the type of program he is using is an oracle program, a program inside the laptop which received a sequence $\{A_t\}_{t\in\omega}$ of snapshots of external conditions at time t and such that the program acts on this apparent information $\Phi_{e,s}^{A_s}(x)$ to execute the trades. Similar examples illustrate that in the real world one is much more likely to use an implementation of a Turing o-machine rather than an a-machine in a limit approximation process, even if the limit is not exactly the correct answer but a good approximation.

9 Conclusions

We now summarize some points of this paper which may not have been stressed in other papers or books on computability.

9.1 Computability and Incomputability Interaction

In the 1930's for the first time the primary goal was to introduce *incomputable (unsolvable)* problems, not directly to explore computability. However, the devices which arose to better understand computability had a great effect on the understanding of computable functions and their later implementation on actual computers and computer programs, such as Turing machines, recursive functions, λ-definable functions, Post production systems and more. Exhilarated by the discovery of incomputable problems, researchers turned to the discovery and classification of more incomputability phenomena, such as recursive ordinals, the arithmetic hierarchy, ordinal logics and other similar items. This is roughly analogous to the period after Cantor's set theory was introduced, when researchers explored the the phonemenon of uncountable sets (also produced by a diagonal argument) and attempted to classify and compare them. The incomputability results led researchers to compare unsolvable problems and introduce relative reducibility to do so.

9.2 Turing Reducibility

The study of unsolvable problems and incomputable phenomena for the first time led to Turing's oracle machines and relative computability to determine when a set A were "more unsolvable" than another set B, *i.e.*, A contains strictly more information ($B <_{\mathrm{T}} A$). This relation was introduced by Turing *1939* §*4* but in a very brief version of about one page. Half of that page was devoted to the problem of showing that whether an oracle machine prints an infinite number of 0's or 1's is not "number theoretic." Turing might have developed these ideas further, but immediately after this paper entered Britain's Bletchley Park, the crypographic laboratory, where he spent the war.

9.3 Post Elucidates Turing Reducibility

Considering the attention to computability in mid 1930's it is surpising that this key concept of *relative computability* lay dormant for five years until Post's paper *1944*. Post's final section, §11 *General (Turing) reducibility* pages 332–336, contains a beautiful, intuitive explanation of what he called *Turing reducibility.* Post went on in *1948* and Kleene-Post *1954* to develop and apply its properties as we understand them today.

9.4 The Post-Turing Second Thesis

Each of Turing and Post had already proposed a "thesis" characterizing computable function, *i.e.*, Turing's Thesis 3.1 and Post's Thesis 5.1. Their contributions on Turing *1939, §4* and Post *1944, §11* may be summarized in what we call the *Post-Turing Second Thesis* that a set B is effectively reducible to A iff B is Turing reducible to A ($B \leq_{\mathrm{T}} A$). This implies the Church-Turing Thesis 3.2 by taking $A =$, but may be equivalent to it depending of the meaning of "effectively reducible." It is suprising that so much attention has been paid to the Church-Turing Thesis over the last seventy years and so little to this thesis on relative reducibility.

9.5 The Central Importance of Turing Recucibility

Although originally introduced for *unsolvability* this relation $B \leq_{\mathrm{T}} A$ is Σ^0_3 as a relation of A and B and is the fundamental *computability* notion of the subject. The graph G_e of any Turing reduction Φ_e is a computably enumerable set, which demonstrates that relative computations are as effective as Turing a-machines computations, given the oracle A.

Both in theoretical computability theory, its applications to algebra, analysis, complexity theory, algorithmic randomness, and many other areas, and finally in real world computing, the notions of oracle programs and relative computability are used much more simple computable functions. It is to find a key theorem on theoretical computability or its applications which deals only with computable functions on computable objects. The richness of the subject and it applications depends comparing incomputable objects to one another. This is ordinarily done by Turing reducibility or one of the stronger under it.

9.6 Computing in the Real World

Most implementations in the real world resemble a kind of oracle machine with input/output devices rather than the simple a-machine modes with no input/output device. For example, many applications of limit computable functions in the real world are best viewed as fixed internal programs which have access to a series of oracles which are very rapidly updated over time. Turing functionals are effectively continuous and indeed any continuous function on Cantor space may be viewed as an effectively continuous function relative to some real parameter. This links the fundamental notions of computability and analysis.

Furthermore, the Turing relation $\{\langle B, A\rangle : B \leq_{\mathrm{T}} A\}$ encompasses many applications in the real world besides just "computing" such as information retrieval, data base, data mining, and more. We may have a laptop with a DVD drive or wireless connection to the web and we may want to view a movie, or extract information from a large data base without computing on it. Such an algorithm requires that we know where to search on oracle A for the data and how to extract it. This relationship of information retrieval will play an increasingly important role in the future real world compared to mere computing.

References

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

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

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

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

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

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

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

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

[9] [Crossley, 1967] J.N. Crossley (ed.), Sets, Models, and Recursion Theory, Proceedings of the Summer School in Mathematical Logic and Logic Colloquium, Leicester, England, 1965, North-Holland, Amsterdam, 1967.

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

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

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

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

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

[15] [Davis, *2000*] M. Davis, The universal computer: The road from Leibniz to Turing, W.W. Norton & Co., New York, London, 2000.

[16] [Epstein and Carnielli, *1989*] Epstein and Carnielli, *Computability*, 1989.

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

[18] [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.

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

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

[21] [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.

[22] [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* [12, 39–74]).

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

[24] [Gödel, *1946*] K. Gödel, Remarks before the Princeton bicentennial conference of problems in mathematics, 1946. Reprinted in: Davis *1965* [12], pp. 84–88.

[25] [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.)

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

[27] [Gödel, *1986*] K. Gödel, *Collected works Volume I: Publications 1929–36*, S. Feferman et. al., editors, Oxford Univ. Press, Oxford, 1986.

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

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

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

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

[32] [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.

[33] [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).

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

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

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

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

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

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

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

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

[42] [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.

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

[44] [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.

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

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

[47] [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.

[48] [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.

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

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

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

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

[53] [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.

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

[55] [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.

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

[57] [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.

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

[59] [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).

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

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

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

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

[64] [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.

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

[66] [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.

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

[68] [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.)

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

[70] [Rogers, 1959] H. Rogers, Jr., Computing degrees of unsolvability, Math. Ann. **138** (1959), 125–140.

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

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

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

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

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

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

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

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

[79] [Soare, 2000] 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.

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

[81] [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, *1937*] A correction, ibid. **43** (1937), 544–546.

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

[83] [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.

[84] [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.)

[85] [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.

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

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

[87] [Turing, *1953*] A. M. Turing, Solvable and unsolvable problems, *Science News* **31** (1954), 7–23.

[88] [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.

[89] [Zabell, *1995*] S. L. Zabell, Alan Turing and the Central Limit Theorem, *American Mathematical Monthly* **102** No. 6 (Jun.-Jul. 1995), 483–494.

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