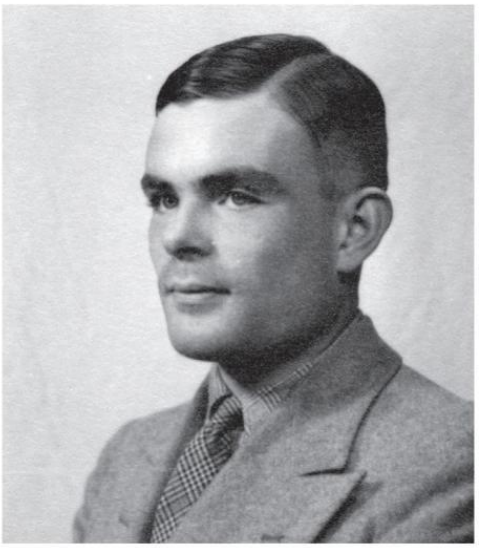
Alan Turing's Systems of Logic



Alan Turing's Systems of Logic

THE PRINCETON THESIS

Edited and introduced by Andrew W. Appel

PRINCETON UNIVERSITY PRESS
PRINCETON AND OXFORD

Published by Princeton University Press,
41 William Street, Princeton, New Jersey 08540
In the United Kingdom: Princeton University Press, 6 Oxford Street,
Woodstock, Oxfordshire OX20 1TW
press.princeton.edu

Second printing, and first paperback printing, 2014

Cloth ISBN 978-0-691-15574-6
Paper ISBN 978-0-691-16473-1
Library of Congress Control Number: 2012931772
British Library Cataloging-in-Publication Data is available

Permission to publish a facsimile reproduction of Alan Turing's Princeton dissertation, "Systems of Logic Based on Ordinals," has been granted by the Princeton University Archives. Department of Rare Books and Special Collection. Princeton University Library

Frontispiece images are reproduced with permission from the Princeton University Archives. Department of Rare Books and Special Collections. Princeton University Library

Title page image of Alan Turing is reproduced by kind permission of the Provost and Fellows, King's College, Cambridge

Solomon Feferman's "Turing's Thesis," originally published in the *Notices of the AMS*, vol. 53, no. 10, is reprinted with permission

This book has been composed in Minion Pro
Printed on acid-free paper. ∞
Printed in the United States of America
10 9 8 7 6 5 4 3 2

Contents

Preface

Alan M. Turing, after his great result in 1936 discovering a universal model of computation and proving his incompleteness theorem, came to Princeton in 1936–38 and earned a PhD in mathematics. Before 1936 there were no universal computers. By 1955 there was not only a theory of computation, but there were real universal ("von Neumann") computers in Philadelphia, Cambridge (Massachusetts), Princeton, Cambridge (England), and Manchester. The new field of computer science had a remarkably short gestation.

The great engineers who built the first computers are well known: Konrad Zuse (Z3, Berlin, 1941); Tommy Flowers (Colossus, Bletchley Park, 1943); Howard Aiken (Mark I, Harvard, 1944); Prosper Eckert and John Mauchley (ENIAC, University of Pennsylvania, 1946).

But computer science is not just the construction of hardware. Who were the creators of the intellectual revolution underlying the theory of computers and computation?

Turing is very well known as a founder and pioneer of this discipline. In 1936 at the age of twenty-four he discovered the universal model of computation now known as the Turing machine; in 1938 he developed the notion of "oracle relativization"; in 1939–45 he was a principal figure in breaking the German Enigma ciphers using computational devices (though not "Turing machines"); in 1948 he invented the LU-decomposition method in numerical computation; in 1950 he foresaw the field of artificial intelligence and made

remarkably accurate predictions about the future of computing and computers. And, of course, he famously committed suicide in 1954 after prosecution and persecution for practicing homosexuality in England.

But as significant as Turing is for the foundation of computer science, he was not the only scholar whose work in the 1930s led to the birth of this field.

In Fine Hall,[1] home in the 1930s of the Princeton Mathematics Department and the newly established Institute for Advanced Study, were mathematicians whose students would form a significant part of the new fields of computer science and operations research.

This volume presents the manuscript of Alan Turing's PhD thesis. It is accompanied by two introductory essays that explore both the work and the context of Turing's stay in Princeton. My essay elucidates the significance of Turing's work (and that of his adviser, Alonzo Church) for the field of computer science; Solomon Feferman's essay describes its significance for mathematics. Feferman also explains how to relate some of Turing's 1938 terminology to more current usage in the field. But on the whole, the notation and terminology in this field have been fairly stable: "Systems of Logic Based on Ordinals" is still readable as a mathematical and philosophical work.

Andrew W. Appel
Princeton, New Jersey

1 Fine Hall was built in 1930, named for the mathematician Henry Burchard Fine. During the 1930s it housed the Mathematics Department of Princeton University and the mathematicians (e.g., Gödel and von Neumann) and physicists (e.g., Einstein) of the Institute for Advanced Study. In 1939, the Institute moved to its own campus about a mile away from Princeton University's central campus. In 1969, the University's Mathematics Department moved to the new Fine Hall on the other side of Washington Road. The old building was renamed Jones Hall, in honor of its original donors, and now houses the departments of East Asian Studies and Near Eastern Studies.

OSWALD VEBLEN, chairman of the Princeton University Mathematics Department and first professor at the Institute for Advanced Study. His students include Alonzo Church (PhD 1927), and his PhD descendants through Philip Franklin (Princeton PhD 1921) via Alan Perlis (Turing Award 1966) include David Parnas, Zohar Manna, Kai Li, Jeannette Wing, and 500 other computer scientists. Veblen has more than 8000 PhD descendants overall. He helped oversee the development of the pioneering ENIAC digital computer in the 1940s.

(Photographer unknown, from the Shelby White and Leon Levy Archives Center, Institute for Advanced Study, Princeton, NJ, USA.)

ALONZO CHURCH, professor of mathematics, whose students include Alan Turing, Leon Henkin, Stephen Kleene, Martin Davis, Michael Rabin (Turing Award 1976), Dana Scott (Turing Award 1976), and Barkley Rosser, and whose PhD descendants include 3000 other mathematicians and computer scientists, among them Robert Constable, Edmund Clarke (Turing Award 2007), Allen Emerson (Turing Award 2007), and Les Valiant (Turing Award 2010).

(Photo from the Alonzo Church Papers. Department of Rare Books and Special Collection. Princeton University Library.)

JOHN VON NEUMANN, at Princeton University from 1930 and professor at the Institute for Advanced Study from 1933, had only a few students (including the pioneer in parallel computer architecture Donald Gillies), but also had an enormous influence on the development of physics, mathematics, logic, economics, and computer science. In 1931 he was the first to recognize the significance of Gödel's work, and toward 1950 he brought Turing's ideas of program-as-data to the engineering of the first stored-program computers. Stored-program computers are called "von Neumann machines," and essentially all computers today are von Neumann machines.

(Photographer unknown, from the Shelby White and Leon Levy Archives Center, Institute for Advanced Study, Princeton, NJ, USA.)

The Birth of Computer Science at Princeton in the 1930s

ANDREW W. APPEL

The "Turing machine" is the standard model for a simple yet universal computing device, and Alan Turing's 1936 paper "On computable numbers . . . " (written while he was a fellow at Cambridge University) is the standard citation for the proof that some functions are not computable. But earlier in the same decade, Kurt Gödel at the Institute for Advanced Study in Princeton had developed the theory of recursive functions; Alonzo Church at Princeton University had developed the lambda-calculus as a model of computation; Church (1936) had just published his result that some functions are not expressible as recursive functions; and he had stated what we know as Church's Thesis: that the recursive functions characterize exactly the *effectively calculable* functions. In hindsight, the first demonstration that some functions are not computable was Church's.

It was only natural that the mathematician M. H. A. Newman (whose lectures on logic Turing had attended) should suggest that Turing come to Princeton to work with Church. Some of the greatest logicians in the world, thinking about the issues that in later decades became the foundation of computer science, were in Princeton's (old) Fine Hall in the 1930s: Gödel, Church, Stephen Kleene, Barkley Rosser, John von Neumann, and others. In fact, it is

hard to imagine a more appropriate place for Turing to have pursued graduate study. After publishing his great result on computability, Turing spent two years (1938–38) at Princeton, writing his PhD thesis on "ordinal logics" with Church as his adviser.

If Turing was not the first to define a universal model of computable functions, why is the Turing machine the standard model? These three models—Gödel's recursive functions, Church's λ-calculus, and Turing's machine—were all proved equivalent in expressive power by Kleene (1936) and Turing (1937). But Turing's model is, most clearly of the three, a *machine,* with simple enough parts that one could imagine building it. As Solomon Feferman explains in his introduction to Turing's PhD thesis later in this volume, even Gödel was not convinced that either λ-calculus or his own model (recursive functions) was a sufficiently general representation of "computation" until he saw Turing's proof that unified recursive functions with Turing machines. That is, Church proved, and Turing independently re-proved, that some functions are not computable, but Turing's result was much more convincing about the *definition* of "computable."

Turing's "On computable numbers" convinced Gödel, and the rest of the world, in part because of the *philosophical* effort he put into that paper, as well as the mathematical effort. Turing described a process of computation as a human endeavor, or as a mechanical endeavor, in such a way that no matter which of these endeavors was dearest to the reader's heart, the result would come out the same: the Turing machine would express it. In contrast, it was not at all obvious that the Herbrand-Gödel recursive functions or the λ-calculus really constitutes the essence of "computation." We know that they do only because of the proof of equivalence with Turing machines.

The real computers of the 1940s and 1950s, like those of today, were never actually *Turing machines* with a finite control and an unbounded tape. But the electronic computers that *were* built, on both sides of the Atlantic, by von Neumann and others, were heavily (and explicitly) influenced by Turing's ideas, so that from the very beginning the field of computer science has often referred to computers in general as Turing machines—especially when considering their expressive power as universal computation devices.

What became of the other two models—recursive functions and λ-calculus? Most mathematicians working in computability theory use the theory of recursive functions; computer scientists working in computational complexity theory use both Turing machines and recursive functions. Turing himself used λ-calculus in his own PhD thesis, but, as Feferman explains,

> One reason that the reception of Turing's [PhD thesis] may have been so limited is that (no doubt at Church's behest) it was formulated in terms of the λ-calculus, which makes expressions for the ordinals and formal systems very hard to understand. He could instead have followed Kleene, who wrote in his retrospective history: "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. I cannot complain about my audiences after 1935."

For Feferman and Kleene, and for other mathematicians working in the field known as "recursive function theory," the particular *implementations* of functions (as described in λ-calculus) are rarely useful, and it is usually sufficient (and simpler) to talk more abstractly about the *existence* of implementations, that is, about definability and about enumerations of computable functions. Soare (1996) points out that the very name of the field (in mathematics) "recursive function theory" was invented by Kleene; Soare suggested "computability theory" as a more descriptive name for the field, and pointed out that Turing and Gödel used "computable" in preference to "recursive." Of course, Soare is both a mathematician and a computer scientist, and it is my impression that many of the latter used the term "computable" more frequently than "recursive" for decades before 1996, influenced (for example) by Martin Davis (PhD 1950 under Church).

So there were several models of computation, all known (by the end of the 1930s) to be equivalent: recursive functions, λ-calculus, Turing machines, and in fact others; for a few decades, mathematicians studied what can be represented as recursive functions, while the computer scientists studied what can be calculated by Turing machines.

unions of logical systems at limit ordinal notations. His main result was that one can thereby overcome incompleteness for an important class of arithmetical statements (though not for all).

It is clear that Turing regards the formalization of mathematics as a desirable goal. He excuses himself at one point (on pp. 9–10 of the manuscript):

> There is another point to be made clear in connection with the point of view we are adopting. It is intended that all proofs that are given [in this thesis] should be regarded no more critically than proofs in classical analysis. The subject matter, roughly speaking, is constructive systems of logic, but as the purpose is directed towards choosing a particular constructive system of logic for practical use; an attempt at this stage to put our theorems into constructive form would be putting the cart before the horse.

Here it is clear that Turing is a logician and not just a great mathematician; few mathematicians believe that it would be a useful purpose to choose a constructive system of logic for *practical* use, and no ordinary mathematician would excuse himself for being no more rigorous than a mathematician.

Just as one of the strengths of Turing's great 1936 paper was its philosophical component—in which he explains the motivation for the Turing machine as a model of computation—here in the PhD thesis he is also motivated by philosophical concerns, as in section 9 (p. 60 of the manuscript):

> We might hope to obtain some intellectually satisfying system of logical inference (for the proof of number theoretic theorems) with some ordinal logic. Gödel's theorem shows that such a system cannot be wholly mechanical, but with a complete ordinal logic we should be able to confine the non-mechanical steps entirely to verifications that particular formulae are ordinal formulae.

Turing greatly expands on these philosophical motivations in section 11 of the thesis. His program, then, is this: We wish to reason in some logic, so that our proofs can be mechanically checked (for example, by a Turing machine). Thus we don't need to trust our students and journal-referees to check our proofs. But no (sufficiently expressive) logic can be complete, as Gödel

showed. If we are using a given logic, sometimes we may want to reason about statements unprovable in that logic. Turing's proposal is to use an ordinal logic sufficiently high in the hierarchy; checking proofs in that logic will be completely mechanical, but the one "intuitive" step remains of verifying ordinal formulas.

Unfortunately, it is not at all clear that verifying ordinal formulas is in any way "intuitive." Feferman (1988, sec. 6) estimates that "the demand on 'intuition' in recognizing 'which formulae are ordinal formulae' is somewhat greater than Turing suggests." Feferman concludes his essay included in this volume with a mention of his and Kreisel's subsequent approaches to this problem, between 1958 and 1970.

Turing, in the thesis, recognizes significant problems with his ordinal logics, which can be summarized by his statement (manuscript, p. 73) that "with almost any reasonable notation for ordinals, completeness is incompatible with invariance" (and see also Feferman's essay).

But the PhD thesis contains, almost as an aside, an enormously influential mathematical insight. Turing invented the notion of oracles, in which one kind of computer consults from time to time, in an explicitly axiomatized way, a more powerful kind. Oracle computations are now an important part of the tool kit of both mathematicians and computer scientists working in computability theory and computational complexity theory (see Feferman 1992; Soare 2009). This method may actually be the most significant result in Turing's PhD thesis.

So the thesis exhibits Turing as logician. Alonzo Church also continued to be a logician, as in 1940 he published "A Formulation of the Simple Theory of Types," setting out the system now known as higher-order logic. As a practical means of actually doing mechanized reasoning, Turing's 1938 result was not nearly as influential as Church's higher-order logic.

In many other fields of engineering, such as the construction of bridges, chemical processes, or photonic circuits, the applicable mathematics is from analysis or quantum mechanics (see Wigner 1960, "The Unreasonable Effectiveness of Mathematics in the Natural Sciences"). But software does not (principally) rely on continuous or quantum artifacts of the natural world,

where that kind of math works so well. Instead, software follows the discrete logic of bits, and it obeys axioms specified by the engineers who designed the instruction-set architecture of the computer, and by those who specified the semantics of the programming languages. Thus the applicable mathematics is, in fact, logic (see Halpern et al. 2001, "On the Unusual Effectiveness of Logic in Computer Science").

It might seem that the Boolean algebra of bits is simpler than real analysis, but the problem is that software systems are so complex that the reasoning is difficult. Thus in the twenty-first century many computer scientists do mechanized formal reasoning, and the most significant application domain for mechanized proof is in the verification of computer software itself. Software is large and complex, and for at least some software it is very desirable that it conform to a given formal specification. The theorems and proofs are too large for us to reliably build and maintain by hand, so we mechanize.

Mechanized proof comes in two flavors; the first flavor is fully automated. *Automated theorem proving* is the use of computer programs to find proofs automatically. *Automatic static analysis* is the use of computer programs to calculate behavioral properties of other computer programs, sometimes by calling upon automated theorem provers as subroutines to decide the validity of logical propositions. Do not be frightened by Turing's result that this problem is uncomputable; his result is simply that no automated procedure can decide the provability of *every* mathematical proposition, and no automated procedure can test nontrivial properties of *every* other program.[3] We do not need to prove *every* theorem or analyze *every* program; it will suffice to automatically prove many useful theorems, or analyze useful programs. Some automated provers work in undecidable logics, and (therefore) sometimes fail to find the proof. In those cases, the user is expected to simplify or reformulate the theorem as necessary, or provide hints. We would not ask Fermat to reformulate his Last Theorem for the convenience of Wiles; but when the theorem is "This horrible program meets its specification," we might well rewrite the program to make it more easily reasoned about. Other automated provers work in decidable logics—for example, Presburger arithmetic or Boolean satisfiability.

3 Actually, this generalization of Turing's 1936 result about halting is known as Rice's theorem (1953).

Do not be frightened by Cook's result (1971) that satisfiability is NP-complete; that result is simply that no (known) automated procedure can solve *every* instance in polynomial time. In practice, SAT-solvers are now a big industry; they are quite effective in solving the actual cases that come up in theorem-proving applications. (Of course, SAT-solvers are not so effective in solving problems that arise from *deliberately* intractable problems, such as cryptography.) The extension of SAT-solvers to SMT (satisfiability modulo theories) is also now a big academic and commercial industry. Many of these solvers use variants of the Davis-Putnam algorithm for resolution theorem proving, discovered in 1960 by Martin Davis (PhD 1950 under Church) and Hilary Putnam (PhD UCLA 1951; in 1960 a colleague of Church's at Princeton).

The other flavor of mechanized proof is the use of computer programs to *check* proofs automatically, and to *assist* in the bureaucratic details of their construction. These are the proof assistants. One of the earliest of these was Robin Milner's LCF (Logic for Computable Functions) system (Gordon et al. 1979). Milner was influenced by the work of Church and by that of Dana Scott (PhD 1958 under Church), Christopher Strachey (a fellow student of Turing's at Cambridge, and one of the first to program the ACE computer in 1951), and Peter Landin (a student of Strachey's). Strachey, Landin, and Milner, all British computer scientists, were important figures in the application of Church's λ-calculus and logic to the design of programming languages and formal methods for reasoning about them.

Although some proof assistants use first-order logics (i.e., logics where each quantifier ranges over elements of a particular fixed type), for the expression of mathematical ideas it is much more convenient to use higher-order logics (i.e., where the type of a quantifier can itself be a variable bound in an outer scope). One of the earliest higher-order logics is Church's "simple theory of types" (1940), but even more expressive (and, to my taste, more useful) logics have dependent types, where the type of one variable may depend on the value of another. Such logics include LF (the Logical Framework) and CoC (the Calculus of Constructions). Proof assistants such as HOL (using the simple theory of types), Twelf (using LF), and Coq (using CoC) are now routinely used to specify and prove substantial theorems about computers and computer programs.

Not only theorems about software; sometimes these proof assistants are even used to prove theorems in mathematics. Georges Gonthier (2008) used Coq to implement a proof of the four-color theorem end-to-end in "Church/Turing-style" fully formal logic. Gonthier's implementation improved on the 1976 proof by Kenneth Appel and Wolfgang Haken that relied in part on "von Neumann–style" Fortran programs to calculate reducibility and in part on "Pythagoras-style" traditional mathematics. (In 1976 the reaction of some mathematicians was to distrust those parts of Appel and Haken's proof that were calculated by computer, whereas the reaction of some computer scientists was to distrust the parts that were checked only "by hand.") In the twenty-first century, computer programs that prove mathematical theorems are expected themselves to be formalized within a mechanically checked logical system. Thomas Hales (2005) proved the Kepler conjecture about sphere packing, using computer programs written in Mathematica and C++, about which the referees were "99% certain." In order to reach 100%, Hales's current project (nearly complete) is to formalize this proof in the HOL Light proof assistant.

In Cambridge, Turing (1936) had brilliant, unprecedented ideas about the nature of computation. He was certainly not the first to build an actual computer; there was already work in progress at (for example) the University of Iowa. But when Turing came to Princeton to work with Church, in the orbit of Gödel, Kleene, and von Neumann,[4] among them they founded a field of computer science that is firmly grounded in logic. In some of Turing's other work (1950) he foresees the field (now within computer science) of artificial intelligence. But in his PhD thesis he makes it clear that he looks to a day when, in proving mathematical theorems, "the strain put on the intuition should be a minimum" (manuscript, page 83). That is, to the extent possible, every step in a proof should be mechanically checkable. We all know the Church-Turing thesis: that no realizable computer will be able to compute more functions than λ-calculus or a Turing machine. But in reading Turing's "Systems of Logic …" we can see quite clearly another kind of Church-Turing thesis, that came

4 Gödel was away from Princeton during Turing's time here, and Kleene had already finished his PhD and left; but clearly they had an enormous influence on Turing's PhD thesis. Turing worked with von Neumann during his time at Princeton, but on other kinds of mathematics than logic and computation.

Turing's Thesis

SOLOMON FEFERMAN

In the sole extended break from his life and varied career in England, Alan Turing spent the years 1936–1938 doing graduate work at Princeton University under the direction of Alonzo Church, the doyen of American logicians. Those two years sufficed for him to complete a thesis and obtain the Ph.D. The results of the thesis were published in 1939 under the title "Systems of logic based on ordinals" [23]. That was the first systematic attempt to deal with the natural idea of overcoming the Gödelian incompleteness of formal systems by iterating the adjunction of statements—such as the consistency of the system—that "ought to" have been accepted but were not derivable; in fact these kinds of iterations can be extended into the transfinite. As Turing put it beautifully in his introduction to [23]:

> The well-known theorem of Gödel (1931) shows that every system of logic is in a certain sense incomplete, but at the same time it indicates means whereby from a system L of logic a more complete system L' may be obtained. By repeating the process we get a sequence L, $L_1 = L'$, $L_2 = L'_1$ … each more complete than the preceding. A logic L_ω may then be constructed in which the provable theorems are the totality of theorems provable with the help of the logics L, L_1 , L_2 , … Proceeding in this way we can associate a system of logic with any constructive ordinal. It may