REVIEW

Formalism and intuition in computability

BY ROBERT IRVING SOARE*

Department of Mathematics, University of Chicago, 5734 University Avenue, Chicago, IL 60637-1546, USA

The model of recursive functions in 1934–1936 was a deductive formal system. In 1936, Turing and in 1944, Post introduced more intuitive models of Turing machines and generational systems. When they both died prematurely in 1954, their informal approach was replaced again by the very formal Kleene T-predicate for another decade. By 1965, researchers could no longer read the papers. A second wave of intuition arose with the book by Rogers and Lachlan’s revealing papers. A third wave of intuition has arisen from 1996 to the present with a return to the original meaning of computability in the sense of Turing and Gödel, and a return of ‘recursive’ to its original meaning of ‘inductive’ and the founding of Computability in Europe by Cooper and others.

Keywords: Turing machines; computability theory; oracle machine; Turing degree; Post; relative computability

1. Overview of formalism and intuition

Since 1931 there have been two main trends in computability, the formalist approach and the informal approach stressing intuition before a formal proof. The formalist approach is characterized by recursive functions and the Kleene T-predicate and the 1936 normal form theorem. It influenced the subject for over 30 years. The informal approach of intuition was initiated by Turing 1936 and 1939, and by Post 1944. It came not continuously but in three waves described below. We now give an overview of the main points very briefly, and we specify places where they will be covered in the main body below.

(a) The formalism of recursive functions

To avoid contradictions, David Hilbert emphasized the Hilbert formalist program for deductive systems. One had only formal strings of symbols. Deductions were made by formal rules of inference, and intuition played no part in a formal proof. In 1934, Gödel defined the Herbrand–Gödel (HG) recursive functions (§3f). Kleene then presented a normal form theorem $T(e, x, y)$ in §3j for indexing the HG-recursive functions and enumerating them in §3k. Even with

One contribution of 18 to a Theme Issue ‘The foundations of computation, physics and mentality: the Turing legacy’.

*soare@uchicago.edu
the indexing, the recursive functions remained a formal deductive system with initial functions for axioms and the two rules \( R1 \) and \( R2 \) as formal rules for generating new equations. Gödel remained unconvinced of Church’s thesis that the effectively calculable (intuitively computable) functions coincided with his own HG-recursive functions (§3).

\[(b) \text{The first wave of intuition: 1936–1944}\]


In addition, Post [4] rebuked the ‘forbidding, diverse and alien formalisms’ of computability and recursion in the 1930s. He gave an extraordinary affirmation of the need for intuition in the subject. Post wrote

Yet the real mathematics involved must lie in the informal development. For in every instance the informal ‘proof’ was first obtained; and once gotten, transforming it into the formal proof turned out to be a routine chore.

[4, p. 284]

This statement (expanded in §5) was the first and strongest statement of the need for intuition in computability arguments. It has become the hallmark of intuition ever since.

\[(c) \text{The second wave of intuition: Rogers and Lachlan}\]

Kleene’s 1952 book [6] was the first book on computability theory. It covered recursive functions and Turing machines, but the emphasis was on HG recursive functions and the Kleene \( \mu \)-recursive functions. The style was quite formal. Hartley Rogers’ 1967 book [7] was a significant advance in intuition and understanding of computability theory (§8).

The Lachlan papers from 1970–1975 [8–10] also had a dramatic effect on computability theory because they revealed the secrets on how to prove difficult theorems. Together, Rogers and Lachlan formed the great divide in computability theory. Before 1966, Kleene and his recursive formalism prevailed. After 1967, researchers used the ideas of Rogers and Lachlan and the Post–Turing normal form of §5b in place of the Kleene normal form. The new proofs became intuitive. Many new researchers joined the field after 1965 and brought a fresh approach.

\[(d) \text{The third wave of intuition: 1996–2012}\]

By 1990, some researchers were using the term ‘computability’ to describe the subject, but they still used Kleene’s term ‘recursive’ to mean ‘computable’ as well as the traditional meaning of ‘inductive’. In 1996, Soare [11] studied the history of computability. He noted that the founders of the two frameworks, Gödel of recursive functions, and Turing of Turing machines, never used the term ‘recursive’ to mean computable. Soare urged that in the future the term ‘recursive’ should mean only ‘inductive’ and should never mean ‘computable’.

Phil. Trans. R. Soc. A (2012)
More generally, Soare [11] urged that the subject be expanded from ‘recursive
function theory’ to the broader field of ‘computability theory’ including
interactions with other areas relating to computability.

This broader expansion was carried out by Barry Cooper and his colleagues,
first with their new organization \textit{Computability in Europe}, and second with their
organization of the Turing Centennial events in June 2012. It became apparent
that there was a worldwide interest and interaction with computability. Cooper
said that these could not have been achieved under the name \textit{Recursion in Europe}.

\section{\textit{\lambda}-definable functions}

(a) \textit{The background of Alonzo Church}

Alonzo Church had studied at Princeton under Oswald Veblen, and Veblen had
asked Church to explain Hilbert’s papers to him. After completing his PhD in
1927, Church spent 1 year at Harvard in 1927–1928 and 1 year split between
the University of Göttingen and the University of Amsterdam. By that time,
Hilbert’s \textit{Entscheidungsproblem} (decision problem) for first-order logic had been
discussed in lectures by Hilbert and was described in Hilbert & Ackermann [12,
pp. 72–73]. It was to give a decision procedure \textit{[Entscheidungsverfahren]} ‘that
allows one to decide the validity \ldots of a given logical expression by a finite
number of operations’. Hilbert characterized this as the fundamental problem
of mathematical logic. Church returned to Princeton as Assistant Professor of
mathematics in 1929.

Within the next 2 years, there were two important events that influenced his
development. First, Gödel’s famous incompleteness theorem [13] showed that
Hilbert’s first program of exhibiting a finite consistency proof of mathematics
was impossible. This indirectly suggested that Hilbert’s other program of the
\textit{Entscheidungsproblem} might also be impossible. However, to prove this, one had
to first characterize all computable functions.

(b) \textit{\lambda}-definable functions and Church’s first thesis

Secondly, Church acquired extraordinary PhD students, Stephen Cole Kleene
and John Barkley Rosser, in 1931. Church had been studying a class of effectively
calculable functions called \textit{\lambda}-definable functions. By 1933, Kleene had shown that
a large class of number-theoretic functions were \textit{\lambda}-definable. On the strength
of this evidence, Church proposed privately to Gödel, around March 1934,
\textit{Church’s first thesis} that the notion of ‘effectively calculable’ be identified with
‘\textit{\lambda}-definable’, a suggestion that Gödel rejected as ‘thoroughly unsatisfactory’.

\section{Recursive functions}

(a) \textit{The concept of recursion}

The term \textit{recursion} refers to a function \(f\) defined by induction. We first define
\(f(0)\) and then define \(f(x + 1)\) in terms of previously defined functions using as
inputs \(x\) and \(f(x)\). For example, the factorial function \(f(x) = x!\) is defined by the

\textit{Phil. Trans. R. Soc. A} (2012)
recursion schemes
\[ f(0) = 1 \quad \text{and} \quad f(x + 1) = (x + 1) \cdot f(x), \quad (3.1) \]
where we assume that multiplication has been previously defined. Dedekind [14] showed that functions could be uniquely defined by recursion. The concept of recursion gradually developed during the early 1900s particularly in the 1923 paper by Skolem [15], the 1926 paper by Hilbert [16], and especially the 1931 paper by Gödel [13].

(b) Primitive recursive functions

Up until the early 1930s, the term ‘recursive function’ meant what we now call a primitive recursive function to distinguish it from the HG general recursive function defined in §3f. Gödel [13] used primitive recursive functions in the proof of his famous incompleteness theorem and called them simply by the German term ‘rekursiv’. The main property of recursion is the primitive recursion scheme (V) below, which yields an inductive definition of \( f(n + 1) \) using the preceding value \( f(n) \) and previously defined functions \( g \) and \( h \). In his influential 1952 book Kleene [6, p. 219] put the primitive recursive functions in the following succinct form, which has become standard.

**Definition 3.1.** The class of primitive recursive functions is the least class \( C \) of functions closed under the following schemes (I)–(V).

(I) The successor function \( f(x) = x + 1 \) is in \( C \).

(II) The constant functions \( f(x_1, \ldots, x_n) = m \), are in \( C \), \( 0 \leq m, n \).

(III) The identity functions \( f(x_1, x_2, \ldots, x_n) = x_i, 1 \leq i \leq n \), are in \( C \).

(IV) (Composition) If \( g_1, g_2, \ldots, g_m, h \in C \), then
\[
f(\bar{x}) = h(g_1(\bar{x}), \ldots, g_m(\bar{x}))
\]
is in \( C \), where \( g_1, \ldots, g_m \) are functions of \( n \) variables, \( \bar{x} = (x_1, \ldots, x_n) \), and \( h \) is a function of \( m \) variables.

(V) (Primitive recursion) If \( g, h \in C \), and \( n \geq 1 \), then \( f \in C \), where
\[
f(0, \bar{x}) = g(\bar{x}) \quad \text{and} \quad f(x + 1, \bar{x}) = h(x, f(x, \bar{x}), \bar{x}),
\]
where \( \bar{x} = (x_2, \ldots, x_n) \) are the \( n - 1 \) variables treated as parameters assuming that \( g \) and \( h \) are functions of \( n - 1 \) and \( n + 1 \) variables, respectively, and \( f \) is a function of \( n \) variables. (In the case \( n = 1 \), a 0-ary function is a constant function that is in \( C \) by scheme (II).)

Therefore, a function \( f \) is primitive recursive if and only if (iff) there is a derivation, namely a sequence \( f_1, f_2, \ldots, f_k = f \) such that each \( f_i, i \leq k \), is either an initial function (i.e. is obtained by schemes (I), (II) or (III)), or \( f_i \) is obtained from \( \{f_j : j < i\} \), by an application of scheme (IV) or (V).

(c) Non-primitive recursive functions

First, primitive recursive functions are total, and can be effectively listed. Therefore, the diagonal argument in §3d produces a calculable function not primitive recursive. Secondly, the primitive recursive functions do not even include all possible recursions. Scheme (V) allowed recursion on only one variable. The
1928 Ackermann article [17] generalized exponential is defined by simultaneous
induction on two variables, but it is not primitive recursive because it dominates
every primitive recursive function.

(d) The obstacle of diagonalization and partial functions

We begin with functions on \( \omega \), and therefore we can identify all objects
with code numbers (Gödel numbers) as we shall define later. To characterize
the effectively calculable functions, we would like to produce a list \( C = \{f_n\}_{n \in \omega} \)
of functions with three properties: (i) the list \( \{f_n\}_{n \in \omega} \) includes exactly the
algorithmically computable functions, (ii) there is a uniformly effective listing
of them, namely an algorithmically computable function \( g(n, x) = f_n(x) \), and
(iii) every \( f_n \) is a total function, i.e. is defined on all \( x \in \omega \). However, these
three conditions are contradictory because we can define the diagonal function
\( h(x) = g(x, x) + 1 \) that is algorithmically computable because \( g \) is, but \( h \notin C \)
because \( h(x) \neq f_x(x) \).

Hence, we must abandon one of the three conditions, but we do not abandon
(i) or (ii), both of which are crucial to the entire theory. Hence, we give up
(iii). If we are dealing with partial computable functions, then (iii) is no longer
an obstacle because in the definition of \( h \), we may no longer have that \( g(x, x) \) is
defined and hence cannot argue that \( h(x) \neq g(x, x) \). It is more natural to consider
partial computable functions anyway because we will effectively list all algorithms
in some formalism, and certain algorithms may be naturally defined only on
some but not all arguments. All the formal classes that we consider produce
algorithmic functions that are partial and are called partial computable
functions. Those partial functions in the class that are total are called total computable
functions or simply computable functions.

(e) Gödel 1934 [5] lectures on recursive functions

From 1933 to 1939, Gödel spent time both in Vienna, Austria and at Fine Hall
in Princeton, which housed both the Princeton University faculty in mathematics
and the Institute for Advanced Study, of which he was a member. Gödel spent
the first part of 1934 at Princeton. The primitive recursive functions that he had
used [5] did not constitute all computable functions. He expanded on a concept
of Herbrand and brought it closer to the modern form. At Princeton in the spring
of 1934, Gödel lectured on the HG recursive functions, which came to be known
as the general recursive functions.

(f) Herbrand–Gödel recursive functions emerge

A remarkably succinct characterization of all functions defined by recursion,
including partial ones, was achieved in 1934 by Gödel [5]. He had used the
primitive recursive functions in the proof of his 1931 incompleteness theorem [13],
but he realized they did not constitute all effectively calculable functions. In
the spring of 1934, Gödel gave a series of lectures at Princeton in which he
introduced what he called the general recursive functions to distinguish them
from the primitive recursive functions which up to that time had been called
‘recursive functions’. Gödel refined a suggestion of Herbrand. Hence, these are
called (HG) general recursive functions or simply recursive functions.
We avoid the formal definition (see Kleene’s 1952 book [6]), but informally sketch the idea. Consider the equations that define the factorial function, \( f(0) = 1 \) and \( f(x + 1) = (x + 1) \cdot f(x) \). Roughly, let the formal language \( L \) consist of non-logical symbols: a unary function symbol \( S \) for successor, and the constant symbol 0 for 0. Let numeral \( k \) denote the term \( S^k(0) \). In addition, \( L \) has a variety of function letters, one of which \( F \) is called the principal function letter corresponding to the informal function \( f \) being defined. We write the system of equations \( \mathbb{E}_F \) to define \( F \),

\[
\begin{align*}
F(0) &= 1 \\
F(x + 1) &= G((x + 1), F(x)).
\end{align*}
\]

(3.2)

Here, we assume that a previously specified system of equations \( \mathbb{E}_G \) defines multiplication \( g(x, y) = x \cdot y \). The rules for deriving new equations from those in (3.2) are the following.

R1 Substitution. In an equation, replace every occurrence of \( x \) by term \( d \).

R2 Replacement. Assume that \( H(c) = d \) has been derived. We may replace \( H(c) \) by numeral \( d \) in the right-hand side of an equation.

**Definition 3.2 (Gödel 1934 [5]).** A (partial) function \( f \) on the integers is (HG) general recursive (usually abbreviated recursive) if there is a finite system of equations \( \mathbb{E} \) with principal function letter \( F \) such that \( f(n) = m \) iff we can derive \( F(n) = m \) from \( \mathbb{E} \) using the rules R1 and R2.

It is easy to see how to derive any value of the factorial function \( f(x) = y \) as an equation \( F(x) = y \), and indeed to show that all the primitive recursive functions are (HG) general recursive. The calculations are natural in that they closely resemble those a mathematician would make with pencil and paper calculating the same values. The main point is that these two simple rules give a formal characterization that captures the notion of *all recursions*, even for partial functions.

Herbrand had written Gödel a letter in 1931 describing systems of equations that uniquely define a (partial) function. Gödel [13] made two restrictions on this definition to make it *effective*, first that the left-hand sides of the functional equations be in standard form with \( F \) being the outermost symbol, and second that, for each set of natural numbers \( n_1, \ldots, n_j \), there exists a unique \( m \) such that \( F(n_1, \ldots, n_j) = m \) is a derived equation [18]. Kleene [1, 6, 19, 20] introduced variants of Gödel’s two rules that give an equivalent formulation of the HG definition.

In the initial definitions and advances for computability during 1931–1937, researchers considered only total computable functions, as is common in other branches of mathematics. It was Kleene [21] who first proposed considering partial computable functions and this helped resolve some difficulties (see Davis [22, p. 17]).

\((g)\) Gödel remained unconvinced

By 1934, Kleene had shown that a large class of number-theoretic functions were \( \lambda \)-definable. On the strength of this evidence, Church informally proposed to Gödel around March 1934 that the notion of ‘effectively calculable’ be identified with ‘\( \lambda \)-definable’, a suggestion that Gödel rejected as ‘thoroughly unsatisfactory’, according to Martin Davis’s account. After hearing Gödel’s lectures in 1934
on the general recursive functions, Church changed the formal definition from ‘\(\lambda\)-definable’ to ‘recursive’, his abbreviation for HG general recursive, and Church presented on 19 April 1935, to the American Mathematical Society (AMS), his famous proposition [23] known (since Kleene [6]) as Church’s thesis, which asserts that the effectively calculable functions should be identified with the recursive functions.

Gödel, however, remained unconvinced of the validity of Church’s thesis from the time Church first introduced it to Gödel in 1934 through to its publication in Church 1936 [23] and beyond. This is all the more significant, first, because Gödel had originated the formalism of the general recursive functions, the one upon which Church based his thesis, and the one that captured the notion of all recursions. Secondly, much of the evidence for Church’s thesis rested on the coincidence of these formal classes (general recursive functions, \(\mu\)-recursive functions and \(\lambda\)-definable functions), and this was based largely on Kleene’s use in scheme (VI) of arithmetization, the method that Gödel himself had introduced so dramatically in his incompleteness theorem [13]. Until seeing Turing’s 1936 paper [2], Gödel was ‘not at all persuaded’.

In addition, Church’s demonstration of his thesis had some flaws as discussed in Seig’s 1994 paper [18], while Turing’s 1936 demonstration [2] of Turing’s thesis did not. Nowadays, we simply refer to the combined thesis as the Church–Turing thesis.

\((h)\) Church and Kleene embrace recursive functions

Church and Kleene attended Gödel’s lectures on recursive functions. Rosser and Kleene took notes, which appeared as Gödel [5]. After seeing Gödel’s lectures, Church and Kleene changed their formalism (especially for Church’s thesis) from ‘\(\lambda\)-definable’ to ‘Herbrand–Gödel general recursive’, partly for reasons of public relations. In his retrospective paper [24, p. 62], Kleene wrote

I myself, perhaps unduly influenced by rather chilly receptions from audiences around 1933–1935 to disquisitions on \(\lambda\)-definability, chose, after general recursiveness had appeared, to put my work in that format…

[24]

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

\((i)\) Church’s second thesis

Church reformulated his thesis, with HG recursive functions in place of \(\lambda\)-definable ones. This time without consulting Gödel, Church presented to the AMS on 19 April 1935, a second thesis, his famous proposition described in his paper [23] and known since Kleene [6] as Church’s thesis. Church wrote

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

[25, p. 333]
Church’s thesis [23]. A function on the positive integers is effectively calculable if and only if it is recursive.

(j) Kleene normal form and T-predicate

Kleene defined a refinement of the HG recursive functions that played an important role in the subject for several decades.

Theorem 3.3 (Normal form theorem [1]).

(i) There is a primitive recursive predicate (relation) \( T(e, x, y) \) which holds if \( y \) is the Gödel number (code number) of a computation according to the HG system of equations with Gödel number \( e \) on input \( x \).
(ii) There is a primitive recursive function \( U(y) \) such that if \( T(e, x, y) \), then \( U(y) \) is the output of the final configuration of computations coded by \( y \).

Proof. The proof is immediate from previous results. Use the HG recursive functions. Use the Gödel primitive recursive coding method from his incompleteness theorem in Gödel [13]. Since each configuration is primitive recursively coded, simply decode the output coded in the final configuration. ■

(k) Kleene’s \( \mu \)-recursive functions

Kleene [1] adapted Gödel’s method of arithmetization of syntax [13] to give scheme (VI), which, together with primitive recursive schemes (I)–(V), gives an alternative and useful characterization of the general recursive functions.

Definition 3.4. The class \( C \) of \( \mu \)-recursive (partial) functions is the least class obtained by closing under schemes (I)–(V) for the primitive recursive functions and the following scheme (VI) [1].

(VI) (Unbounded search) If \( \theta(\bar{x}, y) \in C \) is a partial function, and
\[
\psi(\bar{x}) = (\mu y)[\theta(\bar{x}, y) \downarrow = 1 \text{ and } (\forall z < y) \theta(\bar{x}, z) \downarrow \neq 1],
\]
then \( \psi \) is in \( C \). (Here, \( \psi(\bar{x}) \) diverges if there is no such \( y \). Hence, \( \psi \) may be non-total.)

The \( \mu \)-recursive functions give a compact, mathematically appealing definition and they are convenient to prove the following Kleene enumeration theorem. Church [23] and Kleene [19] proved that the classes of general recursive functions and \( \lambda \)-definable functions are the same as shown in Kleene’s 1967 book [26, p. 232]. Kleene also proved the equivalence with the \( \mu \)-recursive functions.

Church [23] and Kleene [19] proved that the classes of general recursive functions and \( \lambda \)-definable functions are the same [26, p. 232]. Kleene also proved the equivalence with the \( \mu \)-recursive functions.

Theorem 3.5 (Enumeration theorem [1]). There is a \( \mu \)-recursive enumeration of all partial \( \mu \)-recursive functions,
\[
\psi_e(x) = U(\mu yT(e, x, y)).
\]
Proof. The Kleene \(T\)-predicate in the normal form theorem 3.3 is primitive recursive. In the definition of \(\psi_e\), there is one application of scheme (VI). This makes \(\psi_e\) \(\mu\) recursive. ■

Kleene [1] adapted Gödel’s method of arithmetization of syntax [13] to give the following scheme (VI), which, together with primitive recursive schemes (I)–(V), gives an alternative and useful characterization of the general recursive functions.

Definition 3.6. The class \(C\) of \(\mu\)-recursive (partial) functions is the least class obtained by closing under schemes (I)–(V) for the primitive recursive functions and the following scheme (VI) [1].

\[
(VI) \text{(Unbounded search)} \quad \text{If } \theta(\bar{x}, y) \in \mathcal{C} \text{ is a partial function, and}
\]

\[
\psi(\bar{x}) = (\mu y)[\theta(\bar{x}, y) \downarrow = 1 \text{ and } (\forall z < y)[\theta(\bar{x}, z) \downarrow \neq 1]], \tag{3.4}
\]

\[
\text{then } \psi \text{ is in } \mathcal{C}. \text{ (Here, } \psi(\bar{x}) \text{ diverges if there is no such } y. \text{ Hence, } \psi \text{ may be non-total.)}
\]

To see that partial algorithmic functions are closed under scheme (VI), fix \(\bar{x}\) and a partial function \(\theta(\bar{x}, y)\) for which we have an algorithm. Now compute in order \(\theta(\bar{x}, y)\) for \(y = 0, 1, \ldots\), and do not proceed to \(y + 1\) until (if ever) the computation for \(y\) converges. If there is a first \(y\) with \(\theta(\bar{x}, y) \downarrow = 1\), then output \(y\). Otherwise, continue forever.

The \(\mu\)-recursive functions give a compact, mathematically appealing definition and they are convenient to prove the Kleene normal form theorem. However, proving that a non-primitive recursive function (such as the Ackermann function) is \(\mu\)-recursive requires first using the Gödel [13] arithmetization method and then applying scheme (VI). Furthermore, arithmetization is very tedious. In contrast, Turing programs can be decomposed into submodules and often give a more perspicious demonstration that a function is computable. Church [23] and Kleene [19] proved that the classes of general recursive functions and \(\lambda\)-definable functions are the same [26, p. 232]. Kleene also proved the equivalence with the \(\mu\)-recursive functions.

(I) A critique of recursive functions

The recursive functions have the advantage of being in the framework of recursion, a familiar concept. Church [23] and Kleene [19] proved that the classes of general recursive functions and \(\lambda\)-definable functions are the same [26, p. 232]. Kleene also proved the equivalence with the \(\mu\)-recursive functions. Turing proved equivalence of the Turing computable functions with the \(\lambda\)-definable ones, thereby giving extensional equivalence of all three classes.

However, the Kleene normal form is formal and cumbersome in practice, particularly when extended to the relatively computable case. The Turing–Post normal form for Turing computable functions is much easier conceptually and much easier to use. The Kleene normal form and the use of recursive functions in proofs and introductory books for students mostly died out by the end of the 1960s.

Phil. Trans. R. Soc. A (2012)
4. Turing’s automatic machine (a-machine)

(a) Stalemate at Princeton

By the beginning of 1936, Church [23] and Kleene [19] had been accepted for publication. Church and Kleene had proved the equivalence of HG recursive functions with the \( \lambda \)-definable functions. In spite of this ‘concurrence’, Gödel still did not accept Church’s thesis. Gödel had become the most prominent figure in mathematical logic. It was his approval that Church wanted most. Church had solved the Entscheidungsproblem only if his characterization of effectively calculable functions was accurate.

(b) Gödel on effectively calculable functions

In 1934, Gödel [5] had considered the question of characterizing the calculable functions 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.

Gödel maintained that the first sentence of footnote 3 was not an assertion of Church’s thesis. Gödel always used Turing’s work as the definition of effectively calculable functions rather than Gödel’s own recursive functions. Gödel said later that he was not sure that his system of HG recursive functions comprised all possible recursions.

The second sentence of footnote 3 gives crucial insight into Gödel’s thinking about computability. Furthermore, his final sentence suggests that he may have believed such a characterization ‘cannot be proved’, but is a ‘heuristic principle’. This suggests that Gödel was waiting not only for a formal definition, but evidence that it captured the informal notion of effectively calculable. Gödel later found this in Turing’s work, but not in Church’s.

Here, Gödel even suggests that such a precise mathematical characterization of the informal notion cannot be proved, which makes his later acceptance of Turing’s work even more impressive.

(c) The flaw in Church’s thesis

If the basic steps are stepwise recursive, then it follows easily by the Kleene normal form theorem, which Kleene had proved and communicated to Gödel before November 1935 [27, p. 57], that the entire process is recursive. The fatal weakness in Church’s argument was the core assumption that the atomic steps were stepwise recursive, something he did not justify. Gandy’s 1988 paper [28, p. 79] in Herken’s 1988 book [29] and especially Sieg’s 1994 paper [18, pp. 80–87], in their excellent analyses, brought out this weakness in Church’s argument. Sieg wrote,
... this core does not provide a convincing analysis: steps taken in a calculus must be of a restricted character and they are assumed, for example by Church, without argument to be recursive.

[Sieg wrote][18, p. 80]

It is precisely here that we encounter the major stumbling block for Church’s analysis, and that stumbling block was quite clearly seen by Church,

[Sieg wrote][18, p. 78]

who wrote that without this assumption it is difficult to see how the notion of a system of logic can be given any exact meaning at all. It is exactly this stumbling block that Turing overcame by a totally new approach.

(d) Turing breaks the stalemate at Princeton

As 1936 began, there was a stalemate at Princeton about computability. Church [23] and Kleene [1] accepted Church’s thesis but Gödel, the most important figure, did not. The stalemate was broken by a youth a continent away. Alan Turing had already proved the central limit theorem in probability theory (not knowing it had been previously proved), and as a result, Turing had been elected a Fellow of King’s College, Cambridge.

The work of Hilbert and Gödel had already attracted interest at Cambridge University where topologist Professor M.H.A. (Max) Newman gave lectures on Hilbert’s Entscheidungsproblem in 1935. Alan Turing attended. Turing returned to Newman in April 1936 with a solution.

(e) The typewriter model

Turing’s mother had had a typewriter, which fascinated him as a boy, according to Turing’s biographer, Hodges [30]. Turing 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. Most significant was Turing’s analysis of the intuitive conception of a ‘function produced by a mechanical procedure’. The Turing machine can be pictured as in figure 1.

---

Figure 1. Turing machine.

---

Phil. Trans. R. Soc. A (2012)
Thesis 4.1 (Turing’s 1936 thesis [2]). A function on the integers is computable by a finite procedure iff it is computable by a Turing $a$-machine.

Church [23] had tried to give a similar informal argument for Church’s thesis, but that argument was not convincing to Gödel or to modern scholars. No modern textbook uses the notion of recursive function as the definition of an intuitively computable function, but nearly all use Turing machines. In 1936, Turing’s was the only convincing demonstration of any thesis that a formal definition captured the informal notion of computable. Gödel’s reaction was extraordinary. He never accepted Church’s thesis, but he accepted Turing’s thesis at once.\(^1\)

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

Gödel Notes in Nachlass [31]

But I was completely convinced only by Turing’s paper.

Gödel: letter to Kreisel of 1 May 1968 [18, 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 [31, p. 84].

\((f)\) Turing’s proof of Turing’s thesis

Then Turing 1936 [2, §9] demonstrated that any effectively calculable function is Turing computable. In this masterful demonstration, which Robin Gandy considered as precise as most mathematical proofs, Turing analysed the informal nature of functions computable by a finite procedure and demonstrated that they coincide with those computable by an $a$-machine. Gandy observed

Turing’s analysis does much more than provide an argument for [for the thesis] [Turing’s Thesis], it proves a theorem.

[28, p. 82]

Gandy pointed out

Turing’s analysis makes no reference whatsoever to calculating machines. Turing machines appear as a result, a codification, of his analysis of calculations by humans.

[28, pp. 83–84]

Wittgenstein remarked about Turing machines, ‘These machines are humans who calculate’. For a more recent analysis of the Church–Turing thesis, See Kleene [32].

\(^1\)Gödel was interested in the intensional analysis of a finite procedure. He never believed the confluence evidence that 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.
5. The first wave of intuition: Turing and Post

Of course, Turing machines were much more intuitive than recursive functions, and Turing’s demonstration of Turing’s thesis was much more convincing, but this is not all.

(a) Post stresses intuition

In one of the strongest statements of the philosophy of intuition in computability and a rejection of the formalist approach in 1994, Post wrote

Recent developments of symbolic logic have considerable importance for mathematics both with respect to its philosophy and practice. That mathematicians generally are oblivious to the importance of this work of Gödel, Church, Turing, Kleene, Rosser and others as it affects the subject of their own interest is in part due to the forbidding, diverse and alien formalisms in which this work is embodied. Yet, without such formalism, this pioneering work would lose most of its cogency. But apart from the question of importance, these formalisms bring to mathematics a new and precise mathematical concept, that of the general recursive function of Herbrand-Gödel-Kleene, or its proved equivalents in the developments of Church and Turing. It is the purpose of this lecture to demonstrate by example that this concept admits of development into a mathematical theory much as the group concept has been developed into a theory of groups. Moreover, that stripped of its formalism, such a theory admits of an intuitive development which can be followed, if not indeed pursued, by a mathematician, layman though he be in this formal field. It is this intuitive development of a very limited portion of a sub-theory of the hoped for general theory that we present in this lecture. We must emphasize that, with a few exceptions explicitly so noted, we have obtained formal proofs of all the consequently mathematical theorems here developed informally. Yet the real mathematics involved must lie in the informal development. For in every instance the informal ‘proof’ was first obtained; and once gotten, transforming it into the formal proof turned out to be a routine chore.

[4, p. 284]

(b) The Post–Turing normal form

The Kleene normal form is mathematically correct but is cumbersome and not intuitive. It has not been extensively used for several decades. The following Post–Turing normal form for Turing computable functions is implicit in Turing [2] and Post [4], and is explicit in the books by Rogers [7], Soare [33] and Soare [34].

Definition 5.1 (Post–Turing normal form for computable functions). As in Soare’s 1987 book [33, pp. 11–12], let \{P_e\}_{e \in \omega} be an effective listing of all Turing programs. The Turing machine \(M\) with program \(P_e\) and input \(x\) proceeds in a deterministic way beginning in the starting state \(q_1\). If it ever reaches the halting state \(q_0\), it makes no more moves, and the output \(z\) is the total number of 1’s on the tape. Let \(\varphi_e(x)\) be the computable partial function defined this way.

(i) Write \(\varphi_{e,s}(x) = z\) if \(P_e\) with input \(x\) halts in \(\leq s\) steps with output \(z\). Define the Post–Turing normal form

\[
\text{PT}(e,x,s) \iff (\exists z \leq s)[\varphi_{e,s}(x) = z].
\]  

(5.1)
The Post–Turing predicate $\text{PT}(e, x, s)$ and the Kleene predicate $\text{T}(e, x, y)$ are extensionally similar. Both are computable predicates, and both assert the convergence of a computation within a certain bound $y$ or $s$. However, they are intensionally quite different. In the second, we have to search through exponentially many possible derivations of length $k$ for a given bound. When we pass from one unsuccessful $y$ to the next candidate $y + 1$, the latter is an entirely new possible derivation. In the Turing machine model, we perform a linear search. When we pass from one unsuccessful $s$ to the next candidate $s + 1$, we simply perform one more step in the existing Turing computation.

6. Turing’s oracle machine (o-machine)

(a) An extraordinary but almost incidental discovery

Perhaps, Turing’s most important invention, that of an oracle machine, appeared very briefly in Turing 1939 [3, §4]. It appeared as an aside and was unnecessary there. Turing’s oracle machine was developed by Post into Turing reducibility and other reducibilities. Turing reducibility allows us to measure the information content and complexity of structures and sets. It is the most important concept in computability theory. Today, the notion of a local machine interacting with a remote database or remote machine is central to practical computing.

After Turing’s a-machine discovery in April 1936, Max Newman at Cambridge suggested that Turing go to Princeton to study with Church. Turing completed his PhD at Princeton under Church during 1936–1938. Many mathematicians found Gödel’s incompleteness theorem unsettling. Turing’s dissertation [3] was on ordinal logics, apparently a suggestion of Church, and was an attempt to get around Gödel’s incompleteness theorem by adding new axioms. If $T_1$ is a consistent extension of Peano arithmetic, then the arithmetical sentence $\sigma_1$ asserting the unprovability of itself is independent of $T_1$, but we can form a new theory $T_2 = T_1 \cup \{\sigma_1\}$, which strictly extends $T_1$. One can continue this sequence through all the computable (constructive) ordinals.

(b) Turing’s use of oracle machines

In one of the most important and most obscure parts of all of computability theory, Turing wrote in §4 of his 1939 logics paper a short statement about oracle machines (figure 2),

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.

[3, pp. 172–173]

Turing [3, §4] introduced oracle machines for a very specific purpose. In the preceding section [3, §3], Turing had considered $\Pi_2$ predicates ($\forall\exists$-predicates over a computable matrix), and had shown that the Reimann hypothesis and other common problems were $\Pi_2$, which Turing called ‘number theoretic’. For example, the question whether a Turing a-machine computes a partial function with infinite
domain is $\Pi_2$. More precisely, let $\varphi_e$ be the partial computable function computed by the Turing program $P_e$ with Gödel number $e$ and let $W_e$ be the domain of $\varphi_e$. Now $W_e$ is a $\Sigma_1$-set. Define

$$\text{Inf} = \{ e : W_e \text{ is infinite} \}.$$ 

Now Inf is $\Pi_2$, and in fact $\Pi_2$-complete, although the latter concept arose only later. See Soare [33, theorem 3.2] that Inf is $\Pi_2$-complete, i.e. that for every $\Pi_2$ set $V$, there is a computable function $f$ such that $x \in V$ iff $f(x) \in \text{Inf}$.

Turing invented oracle machines to construct a set that was not $\Pi_2$. This could easily have been accomplished by a diagonal argument without oracle machines. Turing put the oracle Inf on the oracle tape. By the same diagonal argument as in Turing [2] for a non-computable function, he used the oracle machine to construct a non-$\Pi_2$ set. Turing wrote

"Given any one of these machines we may ask the question whether it prints an infinity of figures 0 or 1; I assert that this class of problem is not number-theoretic [i.e. $\Pi_2$]."]" [3, p. 173]

In his excellent analysis of Turing [3], Sol Feferman [35] wrote

In §4 Turing introduced a new idea that was to change the face of the general theory of computation (also known as recursion theory) but the only use he made of it was curiously inessential. His aim was to produce an arithmetical problem that is not number-theoretic in his sense, i.e. not in $\forall\exists$-form. This is trivial by a diagonalization argument, since there are only countably many effective relations $R(x, y)$ of which we can say that $\forall x\exists y R(x, y)$ holds. Turing’s way of dealing with this, instead, is through the new notion of computation relative to an oracle. . . .

He then showed that the problem of determining whether an $o$-machine terminates on any given input is an arithmetical problem not computable by any $o$-machine, and hence not solvable by the oracle itself. Turing did nothing further with the idea of $o$-machines, either in this paper or afterward.

[35, p. 1204]
For a fixed set $A \subseteq \omega$, the set of positive integers, effectively number all Turing oracle programs. Let $\Phi_e^A(x)$ denote the partial function computed by the $\sigma$-machine with Gödel number $e$ and with $A$ on the oracle tape. Define the relative halting set

$$K^A = \{ e : \Phi_e^A(e) \text{ halts} \}. \quad (6.1)$$

**Theorem 6.1 (Turing [3]).** Given $A$, the set $K^A$ is not computable in $A$.

Turing [2] showed that there is a diagonal set not computable by a Turing $a$-machine. The same proof on $\sigma$-machines relativized to $A$ establishes the theorem. His specific application is that if $A = \text{Inf}$, then $K^A$ is not $\Pi_2$. In 1939, Turing left the topic of oracle machines, never to return. It mostly lay dormant for 5 years until it was developed into a beautiful form by Post [4,36], Kleene & Post [37] and other papers.

(c) Kleene’s definition of ‘general recursive in’

Kleene gave a definition of ‘general recursive in’. Kleene wrote

“A function $\varphi$ which can be defined from given functions $\psi_1, \ldots, \psi_k$ by a series of applications of general recursive schemata we call general recursive in the given functions; and in particular, a function definable ab initio by these means we call general recursive.”

[20, p. 44]

Kleene was apparently thinking of auxiliary functions used in some HG computation, where the functions are all recursive, such as multiplication being used in the definition of factorial. It is possible that Kleene thought of an infinite non-recursive oracle $A$ as the auxiliary function $\psi$, and the result a function $\psi$, which interrogates $A$ as an external database during the computation. If so, Kleene would surely have cited [3, §4], where this idea first appeared. There is no evidence in Kleene [20] that Kleene thought this way. The notion of ‘general recursive in’ or relative recursive does not appear again in Kleene [20].

Both Turing [3] and Kleene [20] very briefly alluded to a computation using an external set $A$, but neither developed the idea of relative computability. That was done by Post [4]. Soare [38] analyses how the Turing oracle machine also gives a model for modern interactive computing where a local computer like a laptop communicates with an external database such as the Internet.

7. Emil Post extends Turing’s work

(a) Post’s early work

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. Post [39], independently of Turing (but not independently of the work by Church and Kleene in Princeton), defined a ‘finite combinatorial process’ that closely resembles a Turing machine. From this it is often and erroneously written ([27, p. 56] and [24, p. 61]) that Post’s contribution here
was ‘essentially the same’ as Turing’s, but, in fact, it was much less. Post did not attempt to prove that his formalism coincided with any other formalism, such as general recursiveness, but merely expressed the expectation that this would turn out to be true, while Turing [40] proved the Turing computable functions equivalent to the \( \lambda \)-definable ones. Post gave no hint of a universal Turing machine. Most importantly, 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 [2]; and Post believed that Church [23] had settled the Entscheidungsproblem. Furthermore, Post [39] wrote that Church’s identification of effective calculability and recursiveness was a working hypothesis that was in ‘need of continual verification’. This irritated Church who criticized it in his review [41] of Post [39]. Post’s contributions during the 1930s were original and insightful, corresponding in spirit to Turing’s more than to Church’s, but they were not as influential as those of Church and Turing. It was only during the next phase from 1940 to 1954 that Post’s remarkable influence was fully felt.

As Turing left the subject of pure computability theory in 1939, his mantle fell on Post. Post continued the tradition of clear, intuitive proofs, of exploring the most basic objects such as computability and computably enumerable (c.e.) sets, and most of all, exploring relative computability and Turing reducibility. During the next decade and a half, from 1940 until his death in 1954, Post played an extraordinary role in shaping the subject. Post [42,43] introduced a second and unrelated formalism called a production system and (in a restricted form) a normal system, which he explained again [4]. Post’s (normal) canonical system is a generational system, rather than a computational system, as for general recursive functions or Turing computable functions, because it gives an algorithm for generating (listing) a set of integers rather than computing a function. This led Post to concentrate on effectively enumerable sets rather than computable functions.

(b) Post considers the complete set \( K \)

Post’s most influential achievement during this period was the remarkably clear and intuitive paper [4], *Recursively enumerable sets of positive integers and their decision problems*. Post defined the notion of one set to be reducible to another, and introduced the terms degree of unsolvability for the equivalence class of all sets mutually reducible to one another [36]. Post [4] revealed with intuition and great appeal the significance of the c.e. sets and the importance 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 \( K \) (\( W_e \leq_T K \)). The set \( K \) has the same degree as the halting problem of whether a Turing machine with program \( P_e \) halts on a given input \( x \). Moreover, Post felt that the creative
property of $K$ revealed the inherent creativeness of the mathematical process. Post posed his famous ‘Post’s problem’ of whether there exists a c.e. set $A$ such that $\emptyset <_T A <_T K$.

(c) Post defines relative computability

Researchers in the 1930s concentrated on a formal definition of an effectively calculable function, not a definition of a function computable in an oracle. The exception was Turing [3, §4] who defined oracle machines as in §6b. The idea lay dormant for 5 years until Post [4] studied c.e. sets and their decision problems. Post believed that along with decidable and undecidable problems, the relative reducibility (solvability) or non-reducibility of one problem to another is a crucial issue. Post not only studied the structure of the c.e. sets, those that could be effectively listed, but he also initiated a series of reducibilities of one set to another culminating in the most general reducibility, which he generously named ‘Turing’ reducibility. Post’s aim was to use these reducibilities to study the information content of one set relative to another. In his introduction Post wrote

Related to the question of solvability or unsolvability of problems is that of the reducibility or non-reducibility of one problem to another. Thus, if problem $P1$ has been reduced to problem $P2$, a solution of $P2$ immediately yields a solution of $P1$, while if $P1$ is proved to be unsolvable, $P2$ must also be unsolvable. For unsolvable problems the concept of reducibility leads to the concept of degree of unsolvability, two unsolvable problems being of the same degree of unsolvability if each is reducible to the other, one of lower degree of unsolvability than another if it is reducible to the other, but that other is not reducible to it, of incomparable degrees of unsolvability if neither is reducible to the other.

[4, p. 289]

Here, Post is not merely introducing the reducibility of one problem to another for the sake of demonstrating solvability or unsolvability. He takes it further by introducing for the first time in the subject the term ‘degree of unsolvability’ to mean that two sets code the same information, i.e. have the same information content. For 70 years since then, researchers have classified objects in algebra, model theory, complexity and computability theory according to their degree of unsolvability or information content. Post continued

A primary problem in the theory of recursively enumerable sets is the problem of determining the degrees of unsolvability of the unsolvable decision problems thereof. … Now in his paper on ordinal logics [44, §4] Turing presents as a side issue a formulation which can immediately be restated as the general formulation of the ‘recursive reducibility’ of one problem to another, and proves a result which immediately generalizes to the result that for any ‘recursively given’ unsolvable problem there is another of higher degree of unsolvability. While his theorem does not help us in our search for that lower degree of unsolvability, his formulation makes our problem precise. It remains a problem at the end of this paper.

[4, p. 289]

Post [4, §11] wrote an intuitive section on the general case of Turing reducibility, but it was not well understood at that time. Post continued to study it for a decade and introduced the concept of degree of unsolvability, now called ‘Turing degree’ [36]. Before his death in 1954, Post gave his notes to Kleene, who published the joint paper Kleene & Post [37], which laid the foundation of all the subsequent results on Turing reducibility and Turing degree.
In 1944, researchers did not understand Turing reducibility. Post himself was struggling to understand it and did not explicitly discuss it until the very end of his paper, and even then only in general terms. Post’s contributions from 1943 to 1954 concerning relative computability are remarkable. First, Post [4] resurrected the concept of oracle machines, which had been buried in Turing’s [3] paper and which other researchers had apparently ignored for 5 years. Second, Post defined a sequence of strong reductions to better understand the concept of a set $B$ being reducible to a set $A$, including $B \leq_{tt} A$, truth-table reducible, and $B \leq_m A$, $m$-reducibility.

Slowly Post’s understanding deepened of the general case of one set $B$ being reducible (Turing reducible) to another set $A$. Post continued gaining a deeper and deeper understanding from 1943 to 1954 until he had brought it to full development. Our modern understanding of relative computability and Turing functionals is due more to Post and his patient and persistent efforts over more than a decade than to anyone else.

It was only at the end of Post’s paper [4, §11] in the last section, general (Turing) reducibility, that Post defined and named for the first time ‘Turing reducibility’, which was denoted $B \leq_T A$, and began to discuss it in intuitive terms. Post’s four and a half page discussion there is the most revealing introduction to the 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 $S_2$. Post wrote

Now suppose instead, says Turing [3] 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.

Post’s statement may be restated in succinct modern terms and incorporates the statement implicit in Turing [3, §4], in the following extension of Turing’s thesis 4.1.

**Thesis 7.1 (Post–Turing thesis).** One set $B$ is effectively reducible to another set $A$ iff $B$ is Turing reducible to $A$ by a Turing oracle machine ($B \leq_T A$).

Post [36] defined two sets $A$ and $B$ to have the same ‘degree of unsolvability’, *Turing degree*, if they code the same information, i.e. $A \equiv_T B$. Post’s work is extremely important to computability, and can be found in Davis [45].

*Phil. Trans. R. Soc. A* (2012)
### Developing the Turing jump

Gödel [13] was the first to introduce the diagonal $\Pi_1$-set $D$. The complement $\bar{D}$ is computably isomorphic to the complete $\Sigma_1$-set $K$, defined in (7.1). By relativizing Post’s results to the set $K^A$ defined by Turing in (6.1), we see that $K^A$ is c.e. in $A$ and $A <_T K^A$, i.e. $K^A$ is of strictly higher Turing degree than $K$. Post [4] noted that Turing was the first to define $K^A$ in this form relative to an oracle $A$, although Turing did not develop its properties. More properties of the jump and of Turing reducibility were developed in Kleene & Post [37]. Let $A'$ denote the Turing jump $K^A$. Let $0'$ be the degree of $\emptyset'$. The jump operator is well defined on Turing degrees and gives a hierarchy in the diagram of Turing degrees (figure 3).

### 8. The second wave of intuition: Rogers and Lachlan 1967–1975

Kleene’s well-known 1952 book [6] was the first book on computability theory. It covered metamathematics, recursive functions and Turing machines, but the emphasis was on HG recursive functions and the Kleene $\mu$-recursive functions. The style was quite formal. Turing and Post, the advocates of informalism and intuition, both died prematurely 2 years later in 1954. Kleene formalism prevailed for another decade through to 1965. By 1965, the influence of formalism, the Kleene normal form and lack of interest in explaining intuition in proofs led to a completely unacceptable state where researchers could no longer read papers. The field was rescued from this excessive formalism 1965 by two events: the 1967 book by Hartley Rogers [7] and several papers by Alistair Lachlan [8–10].
Hartley Rogers’ book [7] was a significant advance in intuition and understanding of computability theory. Rogers based his book on the ideas of Post’s epochal paper [4]. Rogers wrote in his preface

Although Post’s paper concerns only one part of a larger subject, it marks an epoch in that subject, not only for the specific problems and methods that it presents, but also for the emphasis that it places on intuitive naturalness in basic concepts.

[7, p. vii]

Rogers continued about Post [4], which was the basis for Rogers’s own book,

The book presents its subject matter in semiformal terms, with an intuitive emphasis that is, hopefully, appropriate and instructive. The use of semiformal procedures in recursive function theory is analogous to the use, in other parts of mathematics, of set-theoretical arguments that have not been fully formalized. It is possible for one who possesses a good grasp of the simple, primitive ideas of our theory to do research just as it is possible for a student of elementary algebra in school to do research in the theory of natural ‘numbers’.

[7, p. vii]

9. Computability and recursion

Starting in 1936, Church and Kleene used the term ‘recursive’ to mean ‘computable’, even though Turing and Gödel later objected. Kleene later introduced the term ‘recursive function theory’ for the subject, although Gödel disagreed (see below). This was the first time in history that the term ‘recursive’, which had meant roughly ‘inductive’, acquired the additional meaning ‘computable’ or ‘calculable’. After 1996, the term ‘recursive’ was again used only to mean ‘inductive’, not ‘computable’ or ‘calculable’. From 1931 to 1934, Church and Kleene had used the $\lambda$-definable functions as the formal equivalent of effectively calculable functions, and Church had first proposed his thesis privately to Gödel in that form.

(a) Church formulated his second thesis with ‘recursive’

Church’s first version of his thesis privately to Gödel in 1934 was that a function is effectively calculable iff it is $\lambda$-definable. When Gödel strongly objected to this thesis, Church turned instead to Gödel’s own HG general recursive functions as a formalism and proposed the well-known form of his thesis that a function on the positive integers is effectively calculable iff it is recursive.

Church and Kleene knew almost immediately and published by 1936 the proof of the formal equivalence of recursive functions with $\lambda$-definable functions. After seeing Gödel’s lectures in 1934, Church and Kleene dropped the $\lambda$-definable functions and adopted the recursive functions. This was not because of the inadequacy of $\lambda$-definable functions in comparison with the recursive functions. Indeed, Church seems to have preferred the $\lambda$-definable functions and caused Turing to write his thesis [3] in that formalism.

Church was very eager for mathematicians to accept his thesis, and he knew that the recursive functions were more familiar to a mathematical audience than $\lambda$-definable ones. Church and Kleene used the second version of Church’s thesis, phrased in terms of recursive functions primarily for public relations.
Kleene [24] explained that in 1933–1935, audiences were rather chilly to lectures on \( \lambda \)-definability. Church and Kleene were simply doing what most scientists do: arrange the work in a framework that will be understandable and appealing to as large a scientific audience as possible. Ironically, this is exactly what caused the change in 1996 from ‘recursive’ back to ‘computable’ because in 1996, the term ‘computable’ was much better understood by a general audience than ‘recursive’. The irony is that the term ‘computable’ was there all along and was strongly preferred by Turing and Gödel from the beginning.

(b) Church and Kleene let ‘recursive’ mean ‘computable’

By 1936, Kleene and Church had begun thinking of the word ‘recursive’ to mean ‘computable’. Church had seen his first thesis rejected by Gödel and was heavily invested in the acceptance of his 1936 thesis in terms of recursive functions. Without the acceptance of this thesis, Church had no unsolvable problem. Church wrote [23] and printed in Davis [46] that a ‘recursively enumerable set’ is one that is the range of a recursive function. This is apparently the first appearance of the term ‘recursively enumerable’ in the literature and the first appearance of ‘recursively’ as an adverb meaning ‘effectively’ or ‘computably’.

In the same year, Kleene [1] cited in Davis [46, p. 238] mentioned a ‘recursive enumeration’ and noted that there is no recursive enumeration of HG systems of equations that gives only the systems that define the (total) recursive functions. By a ‘recursive enumeration’, Kleene states that he means ‘a recursive sequence (i.e. the successive values of a recursive function of one variable)’.

Post [43] defined a generational system rather than one for computing functions and called the intuitive concept an effectively enumerable set. Post [4, p. 284] continued with these two concepts and stressed the importance of an intuitive approach as we have seen in §5. Post wrote

That mathematicians generally are oblivious to the importance of this work of Gödel, Church, Turing, Kleene, and Rosser and others ...is in part due to the forbidding, diverse and alien formalisms in which this work is embodied.

[4, p. 284],

No formalism was more forbidding and alien than that of Kleene on the \( T \)-predicate, the papers of Kleene and the use of ‘recursive’ to mean ‘computable’. Kleene’s \( T \)-predicate \( T(e, x, y) \) meant that \( e \) was the Gödel number of an HG system of equations, and \( y \) was the Gödel number of a (halting) derivation on input \( x \) from these equations. There is no new concept of computability here. It merely combines two unintuitive ideas of Gödel and Herbrand [5,13]. Even Gödel did not find the latter a convincing definition of effectively calculable. Post [4] began with the intuitive terms, ‘effectively enumerable’, and ‘generated set’. He then adopted the prevailing terminology at the time of Kleene’s ‘recursive’ and ‘recursively enumerable’.

(c) Gödel rejects ‘recursive function theory’

Gödel never used the word ‘recursive’ to mean ‘computable’. Gödel never used the term ‘recursive function theory’ to name the subject; when others did, Gödel reacted sharply negatively, as related by Martin Davis,
In a discussion with Gödel at the Institute for Advanced Study in Princeton in the fall of 1952, Martin Davis casually used the term ‘recursive function theory’ as it was used then. Davis related, ‘To my surprise, Gödel reacted sharply, saying that the term in question should be used with reference to the kind of work Rosza Peter did’.

(M. Davis 2012, personal communication)

(See Peter’s work on recursions [47,48].) At his 1949 lecture on verifying program correctness, Turing used the term ‘induction variable’, to which Prof. Hartree objected that the term should be ‘recursive variable’ to distinguish it from the sense of mathematical induction. Turing [49] rejected the suggestion. By 1990, the situation had become very difficult. Most people had access to a personal computer on their desks and the terms of computing were familiar to the general population, but ‘recursive’ was limited to a very small number who mainly associated it with a first year programming course or a definition by induction in mathematics and almost never with computability. So few people understood the meaning of ‘recursive’ that by 1990, I had to begin my papers with

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

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

\[(d) \text{ The ambiguity in the term ‘recursive’} \]

The traditional meaning of ‘recursive’ as ‘inductive’ led to ambiguity. Kleene often wrote of calculation and algorithms dating back to the Babylonians, the Greeks and early civilizations. However, Kleene wrote

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

[24, p. 44]

Did he mean that recursion and inductive definition began with Dedekind or that computability and algorithms began there? The latter would contradict his several other statements, such as in Kleene [50, p. 19] where he wrote, ‘The recognition of algorithms goes back at least to Euclid (ca 330 BC)’. When one uses a term like ‘recursive’ to also mean ‘computable’ or ‘algorithmic’ as Kleene did, then one is never sure whether a particular instance means ‘calculable’ or ‘inductive’ and our language has become indistinct. Returning ‘recursive’ to its original meaning of ‘inductive’ has made its use much clearer. We do not need another word to mean ‘computable’. We already have one.

\[(e) \text{ Changing ‘recursive’ back to ‘inductive’} \]

By 1996, the confusion had become intolerable. Soare [11] wrote an article, *Computability and recursion*, for the *Bulletin of Symbolic Logic* on the history and scientific reasons for why we should use ‘computable’ and not ‘recursive’ to mean ‘calculable’. ‘Recursive’ should mean ‘inductive’ as it had for Dedekind and Hilbert. At first, few were willing to make such a dramatic change, overturning a 60 year old tradition of Kleene, and the words ‘computability theory’ and ‘c.e.
set’ did not come easily. However, in a few months more, people were convinced by the undeniable logic of the situation. Three years later at the AMS conference in Boulder, CO, USA, referenced in Soare in 2000 [51], most researchers had adopted the new terminology and conventions. Changing back from ‘recursive’ to ‘computable’ after 1996 has had a number of advantages.

(i) **Historical accuracy**

The founders of the two key models of computability, Turing and Gödel, had never used ‘recursive’ to mean computable and indeed had objected when it was so used. The object of everyone was to formally capture the informal concept of ‘effectively calculable’, which Turing machines did to Gödel’s satisfaction, while at first, Gödel’s own model of HG recursive functions did not. The object was never to understand the notion of recursions or inductive definitions. In 1935, Church adopted Gödel’s recursive functions as a definition of effectively calculable before seeing Turing machines. As Kleene relates, Church and Kleene did this mostly as a matter of public relations to relate to a concept mathematicians could understand, not in an attempt to better understand the nature of recursion and inductive definition.

(ii) **Scientific accuracy**

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

(iii) **Name recognition**

Suppose a student is scanning a catalogue for a course to take and sees the course title ‘recursive function theory’ versus ‘computability theory’. Which of these will give him more information about the content of the course? Should a fresh PhD apply for jobs under the general area of his work as the first or second? Should a professor write an abstract for his lecture at another university under the first title or second?

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

10. **The consequences of computable**

By 1990, many people were using ‘computability’ in their lectures or papers to better inform the audience, but they all still used the terms ‘recursive function’ and ‘recursively enumerable (r.e.) set’ with their original meaning e.g. Davis [52].
In his 1996 paper Soare [11] challenged the 60 year old tradition of Church and Kleene. He recommended using ‘computable’ as Turing and Gödel had done and as we do today. But in a stronger break with tradition, he introduced new phrases like ‘c.e. set’ and ‘computability theory’, and advocated no longer using ‘recursive’ to mean computable but only ‘inductive’, its original meaning. So long as people used ‘recursive’ to mean ‘computable’, they could not escape the hegemony of that tradition. The new phrases did not come easily to researchers conditioned by 60 years of the old tradition, but they have since been accepted. With each use of the terms, scholars were reminded that the subject is about computability and not recursion (induction). As this seeped into the minds of scholars, more connections with other fields arose. Soare’s paper [53] raised the rhetorical question ‘Why Turing and not Church?’ Church got it first and got it right (at least extensionally). Soare first gave a mathematical answer and then an artistic one, ‘Why Michelangelo and not Donatello?’. Soare [54] covers a number of issues about interactive computing.

The primary accomplishment of Soare [11] was to make that change. The suggestions were largely adopted 3 years later by the time of the AMS meeting in Boulder in 1999, which was entitled ‘Computability theory and its applications’, a title that would have been unthinkable in 1995. Likewise, the Elsevier Handbook of recursion theory, planned in the mid 1990s, became by 1999 Handbook of computability theory, and the lead article was Soare’s invited paper, The history of computability theory.

Soon there arose many more connections between computability theory and outside disciplines and organizations. In addition, Cooper [55] wrote a book in the new computability style. Soare’s new book [34] gives an introduction to computability in the new style, and has been used in electronic preprint form since 2005 by students and researchers around the world. Barry Cooper and his colleagues also founded Computability in Europe, an organization with a yearly meeting and several hundred scholars in diverse fields attending, a journal and the sponsorship of books on computability. Barry Cooper stated that the founding of a broadly based organization in the theory of computability, such as Computability in Europe, could not have arisen under the name ‘recursion in Europe’. Names do matter. Philosopher Charles S. Peirce described the importance of language for science this way,

> the woof and warp of all thought and all research is symbols, and the life of thought and science is the life inherent in symbols; so that it is wrong to say that a good language is important to good thought, merely; for it is of the essence of it.

[56, p. 129]

### 11. Determining the date of Turing’s paper

There has been confusion about the date of publication of Turing’s main paper. This is crucial in evaluating Turing’s contributions because in 1936 Church, Kleene and Post all published papers closely related to Turing’s work, although Turing did not discover their work until later. If Turing is relegated to a 1937 publication date (which is false), then it severely diminishes his impact. It would appear that he merely copied existing work or at least discovered it later.
Turing submitted his monumental paper to his adviser, M. H. A. Max Newman, on April 15, 1936. When they became aware of the work in the USA by Church, Kleene and Post, they decided it was sufficiently different that Turing’s paper should be submitted to the *Journal of the London Mathematical Society*. Turing’s paper was received there on May 28, 1936, published in volume 42, parts 3 and 4 in 1936 which include 6 issues including Turing’s paper.

One can physically hold the December 1936 issue containing Turing’s paper. 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 main paper appeared in 1937. Turing’s two page correction did appear in 1937 volume 43.

This work was partially supported by a grant from the Simons Foundation 204186 to Robert Irving Soare. The grant is *Computability theory and applications*, funded by the Simons Foundation Program for Mathematics and the Physical Sciences.

References

14. Dedekind, R. 1888 *Was sind und was sollen die Zahlen?* Braunschweig, Germany.
Review. Formalism and intuition


