Church's Thesis

Guram Bezhanishvili*†

Introduction

In this project we will learn about both primitive recursive and general recursive functions. We will also learn about Turing computable functions, and will discuss why the class of general recursive functions coincides with the class of Turing computable functions. We will introduce the effectively calculable functions, and the ideas behind Alonzo Church's (1903–1995) proposal to identify the class of effectively calculable functions with the class of general recursive functions, known as "Church's thesis." We will analyze Kurt Gödel's (1906–1978) initial rejection of Church's thesis, together with the work of Alan Turing (1912–1954) that finally convinced Gödel of the validity of Church's thesis. We will learn much of this by studying and working with primary historical sources by Gödel, Stephen Cole Kleene (1909–1994), and Turing.

We begin by asking the following question: What does it mean for a function f to be *effectively calculable*? Obviously if we can find an algorithm to calculate f, then f is effectively calculable. For example, the famous Euclidean algorithm tells us that the binary function producing the greatest common divisor of two integers is effectively calculable. But what if we can not find an algorithm that calculates f? The reason could be that there is no algorithm calculating f; or it could be that f is effectively calculable but we were not successful in finding an algorithm. Thus, it is evident that we need better means to identify effectively calculable functions.

The problem of identifying the effectively calculable functions (of natural numbers) was at the center stage of mathematical research in the twenties and thirties of the twentieth century. In the early thirties at Princeton, Church and his two gifted students Kleene and John Barkley Rosser (1907–1989) were developing the theory of λ-*definable* functions. Church proposed to identify the effectively calculable functions with the λ-definable functions. Here is Kleene's description of these events, taken from page 59 of [12]:

> The concept of λ-definability existed full-fledged by the fall of 1933 and was circulating among the logicians at Princeton. Church had been speculating, and finally definitely proposed, that the λ-definable functions are all the effectively calculable functions—what he published in [2], and which I in [11] Chapter XII (or almost in [10]) called "Church's thesis".
>
> When Church proposed this thesis, I sat down to disprove it by diagonalizing out of the class of the λ-definable functions. But, quickly realizing that the diagonalization cannot be done effectively, I became overnight a supporter of the thesis.

Though Kleene became an "overnight" supporter of the thesis, it was a different story with Gödel. Gödel arrived at Princeton in the fall of 1933. Church proposed his thesis to Gödel early in 1934, but, according to a November 29, 1935, letter from Church to Kleene, Gödel regarded

*Department of Mathematical Sciences, New Mexico State University, Las Cruces, NM 88003; gbezhani@nmsu.edu.
†With thanks to Joel Lucero-Bryan.

it "as thoroughly unsatisfactory." Instead, in his lectures during the spring of 1934 at Princeton [6], Gödel generalized the notion of *primitive recursive* functions, which was introduced by him in his epoch-making paper on undecidable propositions [5].[1] He did this by modifying a suggestion made by Jacques Herbrand (1908–1931) in a 1931 letter, to obtain the notion of general recursive functions (also known as the Herbrand-Gödel general recursive functions). Below we give an excerpt from the abovementioned letter from Church to Kleene (taken from [4]) that gives an account of his discussion of effective calculability with Gödel:

> In regard to Gödel and the notions of recursiveness and effective calculability, the history is the following. In discussion [sic] with him the notion of lambda-definability, it developed that there was no good definition of effective calculability. My proposal that lambda-definability be taken as a definition of it he regarded as thoroughly unsatisfactory. I replied that if he would propose any definition of effective calculability which seemed even partially satisfactory I would undertake to prove that it was included in lambda-definability. His only idea at the time was that it might be possible, in terms of effective calculability as an undefined notion, to state a set of axioms which would embody the generally accepted properties of this notion, and to do something on that basis. Evidently it occurred to him later that Herbrand's definition of recursiveness, which has no regard to effective calculability, could be modified in the direction of effective calculability, and he made this proposal in his lectures. At that time he did specifically raise the question of the connection between recursiveness in this new sense and effective calculability, but said he did not think that the two ideas could be satisfactorily identified "except heuristically."

It appears that Gödel's rejection of λ-definability as a possible "definition" of effective calculability was the main reason behind Church's announcement of his thesis in terms of general recursive functions [1]. Church made his announcement at a meeting of the American Mathematical Society in New York City on April 19, 1935. Below is an excerpt from his abstract:

> Following a suggestion of Herbrand, but modifying it in an important respect, Gödel has proposed (in a set of lectures at Princeton, N. J., 1934) a definition of the term *recursive function*, in a very general sense. In this paper a definition of *recursive function of positive integers* which is essentially Gödel's is adopted. And it is maintained that the notion of an effectively calculable function of positive integers should be identified with that of a recursive function, since other plausible definitions of effective calculability turn out to yield notions which are either equivalent to or weaker than recursiveness.

Note that in the abstract Church relegated λ-definability to "other plausible definitions of effective calculability" that were "either equivalent to or weaker than recursiveness," which indicates that, at the time, Church was not yet certain whether λ-definability was equivalent to general recursiveness. Kleene filled in this gap in [8] by showing that these two notions were indeed equivalent. Thus, in the full version of his paper [2], Church was already fully aware that the two notions of general recursiveness and λ-definability coincide.

Kleene's theorem that identified general recursive and λ-definable functions, together with Kleene's famous Normal Form Theorem[2] were beginning to convince Gödel of the validity of Church's thesis. However, it wasn't until the work of Turing that he finally accepted Church's thesis.

[1] It has to be noted that what we now call "primitive recursive" functions Gödel simply called "recursive." The term "primitive recursive" was introduced by Kleene in [7].

[2] The Normal Form Theorem appeared in [7] and considerably simplified the notion of general recursive functions.

Turing's famous paper [15] appeared in 1936 (a correction to it was published in 1937). Turing introduced what we now call *Turing machines*, and defined a function to be *computable* if it can be computed on a Turing machine. His work was entirely independent of the related research being done in Princeton. According to [12], page 61:

> Turing learned of the work at Princeton on λ-definability and general recursiveness just as he was ready to send off his manuscript, to which he then added an appendix outlining a proof of the equivalence of his computability to λ-definability. In [16] he gave a proof of the equivalence in detail.

Thus, Turing introduced his notion of computability in 1936–1937 and, using some of the results of Kleene, showed that the three notions of Turing computable, general recursive, and λ-definable functions coincide.

On page 72 of Gödel's "postscriptum" to his 1934 lecture notes which he prepared in 1964 for [3], Gödel states:

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

Thus, Gödel made it clear that, in his view, Turing's work was of fundamental importance in establishing the validity of Church's thesis. In particular, it influenced Gödel to accept it.

Our account of effective calculability would be incomplete if we did not mention that around the same time and independently of Turing, but not of the work in Princeton, Emil Leon Post (1897–1954) formulated yet another equivalent version of computability [13]. However, his work was less detailed than Turing's.

Lastly we mention that *partial* recursive functions were introduced by Kleene in [9]. In [11] he also generalized the notion of Turing computable functions to partial functions and showed that a partial function is Turing computable if, and only if, it is partial recursive. The importance of his work is underlined in footnote 20 of [4] quoted below:

> It is difficult for those who have learned about recursive functions via a treatment that emphasized partial functions from the outset to realize just how important Kleene's contribution was. Thus Rogers' excellent and influential treatise [14], p. 12, contains an historical account which gives the impression that the subject had been formulated in terms of partial functions from the beginning.

To summarize, in the mid-thirties there were several versions proposed to formalize the intuitive concept of effective calculability. These were λ-definability (Church), general recursiveness (Gödel), Turing computability (Turing), and Post computability (Post). All these concepts were seen to be equivalent to each other, thus producing evidence for general acceptance of Church's thesis.

This project will only scratch the surface of the subject. Instead of working with partial functions, we will restrict our attention to total functions. We will learn about primitive recursive and general recursive functions from the work of Gödel and Kleene. We will also learn about Turing machines, Turing computable sequences (of natural numbers), and Turing computable real numbers (in the interval $[0, 1]$) from the work of Turing and Kleene. We will examine Turing's and Kleene's definitions of computable functions (of natural numbers), and show that every general recursive function is computable on a Turing machine. The fact that every Turing computable function is general recursive requires the relatively advanced technique of Gödel numbers and will not be

addressed in this project. Instead we refer the interested reader to [5], [6] or [11] for the definition of Gödel numbers, and [16] or [11] for the proof that every Turing computable function is general recursive.

The four sources by Gödel, Turing, and Kleene that will be used in the project are [6], [15], [10], and [11]. Edited versions of the first three with forewords have been reprinted in [3].

Part one. Primitive recursive functions

Note: For the reading in part I we will use excerpts of Gödel's 1934 lecture notes [6] reprinted in Davis [3] and edited by Gödel himself. We decided to choose the reprinted version over the original since its definition of rule (1) is more convenient for our purposes.

1.(a) Read carefully the following excerpt of Gödel's 1934 lecture notes [6] reprinted in Davis [3].

> The function $\phi(x_1, \ldots, x_n)$ shall be *compound* with respect to $\psi(x_1, \ldots, x_m)$ and $\chi_i(x_1, \ldots, x_n)$ $(i = 1, \ldots, m)$ if, for all natural numbers $x_1, \ldots, x_n$,
>
> $$(1) \quad \phi(x_1, \ldots, x_n) = \psi(\chi_1(x_1, \ldots, x_n), \ldots, \chi_m(x_1, \ldots, x_n)).$$
>
> $\phi(x_1, \ldots, x_n)$ shall be said to be *recursive* with respect to $\psi(x_1, \ldots, x_{n-1})$ and $\chi(x_1, \ldots, x_{n+1})$ if, for all natural numbers $k, x_2, \ldots, x_n$,
>
> $$(2) \quad \begin{array}{l} \phi(0, x_2, \ldots, x_n) = \psi(x_2, \ldots, x_n) \\ \phi(k+1, x_2, \ldots, x_n) = \chi(k, \phi(k, x_2, \ldots, x_n), x_2, \ldots, x_n). \end{array}$$
>
> In both (1) and (2), we allow the omission of each of the variables in any (or all) of its occurrences on the right side (e.g. $\phi(x, y) = \psi(\chi_1(x), \chi_2(x, y))$ is permitted under (1))[3]. We define the class of *recursive* functions to be the totality of functions which can be generated by substitution, according to the scheme (1), and recursion, according to the scheme (2), from the successor function $x + 1$, constant functions $f(x_1, \ldots, x_n) = c$, and identity functions $U_j^n(x_1, \ldots, x_n) = x_j$ $(1 \leq j \leq n)$. In other words, a function ϕ shall be recursive if there exists a finite sequence of functions $\phi_1, \ldots, \phi_n$ which terminates with ϕ such that each function of the sequence is either the successor function $x+1$ or a constant function $f(x_1, \ldots, x_n) = c$, or an identity function $U_j^n(x_1, \ldots, x_n) = x_j$, or is compound with respect to preceding functions, or is recursive with respect to preceding functions.

1.(b) Rewrite rule (1) for $n = 1$ and $m = 2$. Also rewrite rule (2) for $n = 1$. Explain in your own words what the rules express.

1.(c) Give a definition of primitive recursive functions in your own words.

1.(d) On page 44 of [3] Gödel states:

> The functions $x + y$, xy, x^y and $x!$ are clearly [primitive] recursive.

Give an argument for why this is so. Hint: Review your answer to 1.(c) and make sure that you have a good grasp of Gödel's definition of primitive recursive functions.

1.(e) Is every function of natural numbers primitive recursive? Hint: Use a cardinality argument.

1.(f) Read carefully the excerpts from Church's 1935 letter to Kleene and Church's 1935 abstract appearing above. Also read carefully the following excerpt of Gödel's 1934 lecture notes [6] reprinted in Davis [3].

[3] [This sentence could have been] omitted, since the removal of any of the occurrences of variables on the right may be effected by means of the function U_j^n. This footnote occurs in the original source.

> Recursive functions have the important property that, for each given set of values of the arguments, the value of the function can be computed by a finite procedure[4].

Do you see any connection between Gödel's writing and Church's thesis? Explain your answer.

Part two. General recursive functions

2.(a) Read carefully the following excerpt of section 1 of Kleene [10].

> We consider the following schemata as operations for the definition of a function ϕ from given functions appearing in the right members of the equations (c is any constant natural number):
>
> (I) $\quad \phi(x) = x'$,
> (II) $\quad \phi(x_1, \ldots, x_n) = c$,
> (III) $\quad \phi(x_1, \ldots, x_n) = x_i$,
> (IV) $\quad \phi(x_1, \ldots, x_n) = \theta(\chi(x_1, \ldots, x_n), \ldots, \chi_m(x_1, \ldots, x_n))$,
> (Va) $\quad \begin{cases} \phi(0) = c \\ \phi(y') = \chi(y, \phi(y)), \end{cases}$
> (Vb) $\quad \begin{cases} \phi(0, x_1, \ldots, x_n) = \psi(x_1, \ldots, x_n) \\ \phi(y', x_1, \ldots, x_n) = \chi(y, \phi(y, x_1, \ldots, x_n), x_1, \ldots, x_n), \end{cases}$
>
> Schema (I) introduces the successor function, Schema (II) the constant functions, and Schema (III) the identity functions. Schema (IV) is the schema of definition by substitution, and Schema (V) the schema of primitive recursion. Together we may call them (and more generally, schemata reducible to a series of applications of them) the *primitive recursive* schemata.
>
> A function ϕ that can be defined from given functions $\psi_1, \ldots, \psi_k$ by a series of applications of these schemata we call *primitive recursive* in the given functions; and in particular, a function ϕ definable ab initio[5] by these means, *primitive recursive*.

Is your answer to 1.(c) equivalent to Kleene's definition of primitive recursive functions? Explain why.

2.(b) Recall the definition of *total* and *partial* functions. Discuss briefly the difference between the two. Read carefully the following definition of the μ-operator taken from the beginning of section 3 of Kleene [10].

> Consider the operator: μy (the least y such that). If this operator is applied to a predicate $R(x_1, \ldots, x_n, y)$ of the $n+1$ variables $x_1, \ldots, x_n, y$, and if this predicate satisfies the condition
>
> $$(2) \qquad (\forall x_1) \cdots (\forall x_n)(\exists y) R(x_1, \ldots, x_n, y),$$
>
> we obtain a function $\mu y R(x_1, \ldots, x_n, y)$ of the remaining n free variables $x_1, \ldots, x_n$.
>
> Thence we have a new schema,
>
> $$(\mathrm{VI}_1) \qquad \phi(x_1, \ldots, x_n) = \mu y[\rho(x_1, \ldots, x_n, y) = 0],$$

[4] The converse seems to be true, if, besides recursions according to the scheme (2), 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. This footnote occurs in the original source.

[5] From the beginning.

for the definition of a function ϕ from a given function ρ which satisfies the condition

$$(3) \qquad (\forall x_1)\cdots(\forall x_n)(\exists y)[\rho(x_1,\ldots,x_n,y)=0].$$

Give an example of a function obtainable by the application of the μ-operator that is not total. What does condition (3) guarantee about the function obtained by using schema (VI_1)?

2.(c) Read carefully excerpts of theorem III and the corollary to it taken from page 51 of Kleene [10].

> THEOREM III. The class of general recursive functions is closed under applications of Schemata (I)–(VI) with (3) holding for applications of (VI).
>
> COROLLARY. Every function obtainable by applications of Schemata (I)–(VI) with (3) holding for applications of (VI) is general recursive.

Based on the corollary, give a definition of general recursive functions. Explain whether or not every primitive recursive function is general recursive.

2.(d) Is every function of natural numbers general recursive? Explain your answer.

2.(e) (Extra Credit) Give a reasonable argument for why the class of general recursive functions is *strictly* larger than the class of primitive recursive functions.

Part three. Turing machines

3.(a) Read carefully the following excerpt of section 1 of Turing [15].

> We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \ldots, q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible bahaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the m-configuration may be changed. Some of the symbols written down will form the sequence of figures which is the decimal of the real number which is being computed. The others are just rough notes to "assist the memory". It will only be these rough notes which will be liable to erasure.

Also read carefully the following excerpt of section 2 of Turing [15].

Automatic machines.

If at each stage the motion of a machine (in the sense of §1) is *completely* determined by the configuration, we shall call the machine an "automatic machine" (or a-machine).

For some purposes we might use machines (choice machines or c-machines) whose motion is only partially determined by the configuration (hence the use of the word "possible" in §1). When such a machine reaches one of these ambiguous configurations, it cannot go on until some arbitrary choice has been made by an external operator. This would be the case if we were using machines to deal with axiomatic systems. In this paper I deal only with automatic machines, and will therefore often omit the prefix a-.

Computing machines.

If an a-machine prints two kinds of symbols, of which the first kind (called figures) consists entirely of 0 and 1 (the others being called symbols of the second kind), then the machine will be called a computing machine. If the machine is supplied with a blank tape and set in motion, starting from the correct initial m-configuration, the subsequence of the symbols printed by it which are of the first kind will be called the *sequence computed by the machine*. The real number whose expression as a binary decimal is obtained by prefacing this sequence by a decimal point is called the *number computed by the machine*.

At any stage of the motion of the machine, the number of the scanned square, the complete sequence of all symbols on the tape, and the m-configuration will be said to describe the *complete configuration* at that stage. The changes of the machine and tape between successive complete configurations will be called the *moves* of the machine.

Circular and circle-free machines.

If a computing machine never writes down more than a finite number of symbols of the first kind, it will be called *circular*. Otherwise it is said to be *circle-free*.

A machine will be circular if it reaches a configuration from which there is no possible move, or if it goes on moving, and possibly printing symbols of the second kind, but cannot print any more symbols of the first kind...

Computable sequences and numbers.

A sequence is said to be computable if it can be computed by a circle-free machine. A number is computable if it differs by an integer from the number computed by a circle-free machine.

Note: The machines that Turing designs will be called Turing machines.

3.(b) What are the primary components of a Turing machine? Describe m-configurations, configurations, and complete configurations of a Turing machine, and their differences.

3.(c) Formulate in your own words Turing's definition of a computing machine, a computable sequence (of natural numbers), and a computable real number (in the interval $[0, 1]$).

3.(d) Read carefully the following excerpt of section 3 of Turing [15] where it is shown that the sequence 010101... is computable.

A machine can be constructed to compute the sequence $010101\ldots$. The machine is to have the four configurations "$\mathfrak{b}$", "$\mathfrak{c}$", "$\mathfrak{f}$", "$\mathfrak{e}$" and is capable of printing "0" and "1". The behaviour of the machine is described in the following table in which "R" means "the machine moves so that it scans the square immediately on the right of the one it was

scanning previously". Similarly for "L". "E" means "the scanned symbol is erased" and "P" stands for "prints". This table (and all succeeding tables of the same kind) is to be understood to mean that for a configuration described in the first two columns the operations in the third column are carried out successively, and the machine then goes over into the m-configuration described in the last column. When the second column is left blank, it is understood that the behaviour of the third and fourth columns applies for any symbol and for no symbol. The machine starts in the m-configuration $\mathfrak{b}$ with a blank tape.

Configuration		*Behaviour*	
m-*config.*	*symbol*	*operations*	*final* m-*config.*
$\mathfrak{b}$	None	$P0, R$	$\mathfrak{c}$
$\mathfrak{c}$	None	R	$\mathfrak{e}$
$\mathfrak{e}$	None	$P1, R$	$\mathfrak{f}$
$\mathfrak{f}$	None	R	$\mathfrak{b}$

If (contrary to the description in §1) we allow the letters L, R to appear more than once in the operations column we can simplify the table considerably.

m-*config.*	*symbol*	*operations*	*final* m-*config.*
$\mathfrak{b}$	None	$P0$	$\mathfrak{b}$
	0	$R, R, P1$	$\mathfrak{b}$
	1	$R, R, P0$	$\mathfrak{b}$

What can you conclude about the real number $\frac{1}{3}$? Design a Turing machine that computes the real number $\frac{1}{7}$. Give an argument for why the machine you just designed does what it is supposed to do.

3.(e) (Extra Credit) Read carefully the following excerpt of section 3 of Turing [15] where it is shown that the sequence $00101101110111011111\ldots$ is computable.

As a slightly more difficult example we can construct a machine to compute the sequence $00101101110111011111\ldots$. The machine is to be capable of five m-configurations, viz. "$\mathfrak{o}$", "$\mathfrak{q}$", "$\mathfrak{p}$", "$\mathfrak{f}$", "$\mathfrak{b}$" and of printing "∂", "x", "0", "1". The first three symbols on the tape will be "$\partial\partial 0$"; the other figures follow on alternate squares. On the intermediate squares we never print anything but "x". These letters serve to "keep the place" for us and are erased when we have finished with them. We also arrange that in the sequence of figures on alternate squares there shall be no blanks.

Configuration		*Behaviour*	
m-config.	*symbol*	*operations*	*final m-config.*
$\mathfrak{b}$		$P\partial, R, P\partial, R, P0, R, R, P0, L, L$	$\mathfrak{o}$
$\mathfrak{o}$	1	R, Px, L, L, L	$\mathfrak{o}$
	0		$\mathfrak{q}$
$\mathfrak{q}$	Any (0 or 1)	R, R	$\mathfrak{q}$
	None	$P1, L$	$\mathfrak{p}$
$\mathfrak{p}$	x	E, R	$\mathfrak{q}$
	∂	R	$\mathfrak{f}$
	None	L, L	$\mathfrak{p}$
$\mathfrak{f}$	Any	R, R	$\mathfrak{f}$
	None	$P0, L, L$	$\mathfrak{o}$

To illustrate the working of this machine a table is given below of the first few complete configurations. These complete configurations are described by writing down the sequence of symbols which are on the tape, with the m-configuration written below the scanned symbol. The successive complete configurations are separated by colons.

$$
\begin{array}{llllllllllllll}
 & : \partial\partial 0 & 0 : \partial\partial 0 & 0 : \partial\partial 0 & 0 : \partial\partial 0 & 0 & & : \partial\partial 0 & 0 & 1 & & & & \\
\mathfrak{b} & \mathfrak{o} & \mathfrak{q} & \mathfrak{q} & \mathfrak{q} & & & \mathfrak{p} & & & & & & \\
\partial\partial 0 & 0 & 1 : \partial\partial 0 & 0 & 1 : \partial\partial 0 & 0 & 1 : \partial\partial 0 & 0 & 1 : \partial\partial 0 & 0 & 1 : & & & \\
 & \mathfrak{p} & & \mathfrak{p} & & \mathfrak{f} & & & & \mathfrak{f} & & & & \\
\partial\partial 0 & 0 & 1 : \partial\partial 0 & 0 & 1 & : \partial\partial 0 & 0 & 1 & 0 : & & & & & \\
 & & \mathfrak{f} & & & \mathfrak{f} & & \mathfrak{o} & & & & & & \\
\partial\partial 0 & 0 & 1x0 : \ldots . & & & & & & & & & & & \\
 & \mathfrak{o} & & & & & & & & & & & &
\end{array}
$$

In this example Turing uses the symbols "∂" and "x", which are the symbols of the second kind. Explain the need for these symbols, and their use in this particular Turing machine. Give an argument for why the machine described above does what it is supposed to do.

Part four. Turing computable functions

4.(a) Read carefully Turing's definition of computable functions of natural numbers taken from page 254 of [15].

> If γ is a computable sequence in which 0 appears infinitely[6] often, and n is an integer, then let us define $\xi(\gamma, n)$ to be the number of figures 1 between the n-th and the $(n+1)$-th figure 0 in γ. Then $\phi(n)$ is computable if, for all n and some γ, $\phi(n) = \xi(\gamma, n)$.

Note: We will call these functions Turing computable.

4.(b) In your own words explain what it means for a function of natural numbers of one variable to be Turing computable. Extra Credit. Generalize the concept of Turing computable functions to functions of two variables. Hint: Can you code every function of two variables by a function of one variable? Generalize the concept of Turing computable functions to functions of multiple variables.

[6] If $\mathcal{M}$ computes γ, then the problem whether $\mathcal{M}$ prints 0 infinitely often is of the same character as the problem whether $\mathcal{M}$ is circle-free. This footnote occurs in the original source.

4.(c) In your own words explain Kleene's definition of a Turing machine by reading carefully the following excerpt from section 67 of Kleene [11].

> The machine is supplied with a linear *tape*, (potentially) infinite in both directions (say to the *left* and *right*). The tape is divided into *squares*. Each square is capable of being *blank*, or of having *printed* upon it any one of a finite list $s_1, \ldots, s_j$ $(j \geq 1)$ of *symbols*, fixed for a particular machine. If we write "s_0" to stand for "blank", a given square can thus have any one of $j+1$ *conditions* $s_0, \ldots, s_j$. The tape will be so employed that in any "situation" only a finite number (≥ 0) of squares will be printed.
>
> The tape will pass through the machine so that in a given "situation" the machine *scans* just one square (the *scanned square*). The symbol on this square, or s_0 if it is blank, we call the *scanned symbol* (even though s_0 is not properly a symbol).
>
> The machine is capable of being in any one of a finite list $q_0, \ldots, q_k$ $(k \geq 1)$ of (*machine*) *states* (called by Turing "machine configurations" or "m-configurations"). We call q_0 the *passive* (or *terminal*) *state*; and $q_1, \ldots, q_k$ we call *active states*. The list $q_0, \ldots, q_k$ is fixed for a particular machine.
>
> A (*tape vs. machine*) *situation* (called by Turing "complete configuration") consists in a particular printing on the tape (i.e. which squares are printed, and each with which of the j symbols), a particular position of the tape in the machine (i.e. which square is scanned), and a particular state (i.e. which of the $k+1$ states the machine is in). If the state is active, we call the situation *active*; otherwise, *passive*.
>
> Given an active situation, the machine performs an (*atomic*) *act* (called a "move" by Turing). The act performed is determined by the scanned symbol s_a and the machine state q_c in the given situation. This pair (s_a, q_c) we call the *configuration*. (It is *active* in the present case that q_c is active; otherwise *passive*.) The act alters the three parts of the situation to produce a resulting situation, thus. First, the scanned symbol s_a is changed to s_b. (But $a = b$ is permitted, in which case the "change" is identical.) Second, the tape is shifted in the machine (or the machine shifts along the tape) so that the square scanned in the resulting situation is either one square to the left of, or the same square as, or one square to the right of, the square scanned in the given situation. Third, the machine state q_c is changed to q_d. (But $c = d$ is permitted.)
>
> No act is performed, if the given situation is passive.
>
> The machine is used in the following way. We choose some active situation in which to start the machine. We call this the *initial situation* or *input*. Our notation will be chosen so that the state in this situation (the *initial state*) is q_1. The machine then performs an atomic act. If the situation resulting from this act is active, the machine acts again. The machine continues in this manner, clicking off successive acts, as long and only as long as active situations result. If eventually a passive situation is reached, the machine is said then to *stop*. The situation in which it stops we call the *terminal situation* or *output*.
>
> The change from the initial situation to the terminal situation (when there is one) may be called the *operation* performed by the machine.
>
> To describe an atomic act, we use an expression of one of the three following forms:
>
> $$s_b L q_d, \qquad s_b C q_d, \qquad s_b R q_d.$$
>
> The "L", "C", "R", indicate that the resulting scanned square is to the left of, the same as ("center"), or to the right of, respectively, the given scanned square.

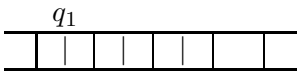
The first part of the act (i.e. the change of s_a to s_b) falls into four cases: when $a = 0$ and $b > 0$, it is "prints s_b"; when $a > 0$ and $b = 0$, "erases s_a"; when $a, b > 0$ and $a \neq b$, "erases s_a and prints s_b" or briefly "overprints s_b"; when $a = b$, "no change". We often describe this part of the act as "prints s_b" without regard to the case.

To define a particular machine, we must list the symbols $s_1, \ldots, s_j$ and the active states $q_1, \ldots, q_k$, and for each active configuration (s_a, q_c) we must specify the atomic act to be performed. These specifications may be given by displaying the descriptions of the required acts in the form of a (*machine*) *table* with k rows for the active states and $j + 1$ columns for the square conditions.

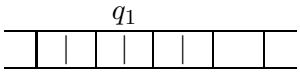
EXAMPLE I. The following table defines a machine ("Machine $\mathfrak{A}$") having only one symbol s_1 and only one active state q_1.

Name of machine	Machine state	Scanned symbol s_0	s_1
$\mathfrak{A}$	q_1	$s_1 C q_0$	$s_1 R q_1$

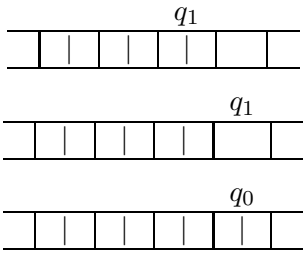
Suppose the symbol s_1 is actually a tally mark "|". Let us see what the machine does, if a tape of the following appearance is placed initially in the machine so that the square which we identify by writing the machine state q_1 over it is the scanned square. The conditions of all squares not shown will be immaterial, and will not be changed during the action.



The machine is in the state q_1, and is scanning a square on which the symbol s_1 is printed. In this configuration, the atomic act ordered by the table is $s_1 R q_1$; i.e. no change is made in the condition of the scanned square, the machine shifts right, and again assumes state q_1. The resulting situation appears as follows.



The next three acts lead successively to the following situations, in the last of which the machine stops.



Machine $\mathfrak{A}$ performs the following operation: It seeks the first blank square at or to the right of the scanned square, prints a | there, and stops scanning that square.

4.(d) Explain the similarities and differences of Turing's and Kleene's definitions. Which definition of Turing machines do you prefer? Explain why.

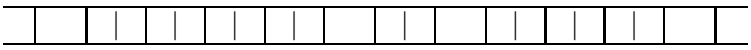
4.(e) The following is Kleene's definition of Turing computable functions of multiple variables (see [11], p. 359).

Now we define how a machine shall 'compute' a partial number-theoretic function ϕ of n variables (cf. §63). The definition for an ordinary (i.e. completely defined) number-theoretic function is obtained by omitting the reference to the possibility that $\phi(x_1, \ldots, x_n)$ may be undefined.

We begin by agreeing to represent the natural numbers $0, 1, 2, \ldots$ by the sequence of tallies $|, ||, |||, \ldots$, respectively, the tally "|" being the symbol s_1. There are $y + 1$ tallies in the representation of the natural number y.

Then to represent an m-tuple $y_1, \ldots, y_m$ ($m \geq 1$) of natural numbers on the tape, we print the corresponding numbers of tallies, leaving a single blank between each two groups of tallies and before the first and after the last.

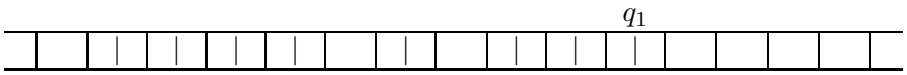
EXAMPLE 2. The triple $3, 0, 2$ is represented thus:



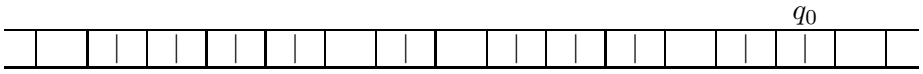
We say that (the representation of) a number y (or of any m-tuple $y_1, \ldots, y_m$) on the tape is (*scanned*) *in the standard position*, when the scanned square is the one bearing the last tally in the representation of y (or of y_m).

Now we say that a given machine $\mathfrak{A}$ *computes* a given partial function ϕ of n variables ($n \geq 1$), if the following holds for each n-tuple $x_1, \ldots, x_n$ of natural numbers. (For the case $n = 0$, cf. Remark 1 below.) Let $x_1, \ldots, x_n$ be represented on the tape, with the tape blank elsewhere, i.e. outside of the $x_1 + \cdots + x_n + 2n + 1$ squares required for the representation. Let $\mathfrak{A}$ be started scanning the representation of $x_1, \ldots, x_n$ in standard position. Then $\mathfrak{A}$ will eventually stop with the $n + 1$-tuple $x_1, \ldots, x_n, x$ represented on the tape and scanned in standard position, if and only if $\phi(x_1, \ldots, x_n)$ is defined and $\phi(x_1, \ldots, x_n) = x$. (If $\phi(x_1, \ldots, x_n)$ is undefined, $\mathfrak{A}$ may fail to stop. It may stop but without an $n + 1$-tuple $x_1, \ldots, x_n, x$ scanned in standard position.)

Example 2 (concluded). If $\phi(3, 0, 2) = 1$ and $\mathfrak{A}$ computes ϕ, then when $\mathfrak{A}$ is started in the situation



with all squares other than those shown blank, it must eventually stop in the situation



where the condition of the squares other than those shown is immaterial.

Although only one symbol s_1 or "|" is used in stating the arguments and in receiving the function value, others may be used in the progress of the computation. For each $n \geq 1$, each machine (with its first symbol s_1 serving as the tally) computes a certain partial function of n variables.

A partial function ϕ is *computable*, if there is a machine $\mathfrak{A}$ which computes it.

Remark 1 that Kleene refers to is on page 363 of Kleene [11]. Below we give an excerpt from it.

REMARK 1. In this chapter, outside the present remark and passages referring to it, we shall understand that we are dealing with functions of $n \geq 1$ variables. Since we have not provided for representing n-tuples of natural numbers on the tape for $n = 0$, we say a machine computes a function ϕ of 0 variables, if it computes the function $\phi(x)$ of 1 variable such that $\phi(x) \simeq \phi$.

State in your own words Kleene's definition of Turing computable functions of multiple variables. Kleene mentions that the representation of the tuple $x_1, \ldots, x_n$ on a tape requires $x_1 + \cdots + x_n + 2n + 1$ squares. Explain why.

4.(f) Are Turing's and Kleene's definitions of computable functions of one variable equivalent? Explain why.

4.(g) (Extra Credit) Is your generalization of Turing computability to functions of multiple variables equivalent to Kleene's definition of computable functions of multiple variables? Explain why.

Part five. Turing computable and general recursive functions

5.(a) Explain why the successor function $s(x) = x + 1$ is Turing computable.

5.(b) Explain why the constant functions $f(x_1, ..., x_n) = c$ are Turing computable.

5.(c) Explain why the identity functions $U_j^n(x_1, ..., x_n) = x_j$ are Turing computable.

5.(d) Explain why whenever $\psi(x_1, \ldots, x_m)$ and $\chi_1(x_1, \ldots, x_n), \ldots, \chi_m(x_1, \ldots, x_n)$ are Turing computable, then $\phi(x_1, \ldots, x_n)$ defined by

$$\phi(x_1, \ldots, x_n) = \psi(\chi_1(x_1, \ldots, x_n), \ldots, \chi_m(x_1, \ldots, x_n))$$

is also Turing computable. In other words, explain why the schema of substitution preserves Turing computability.

5.(e) Explain why whenever $\psi(x_1, \ldots, x_{n-1})$ and $\chi(x_1, \ldots, x_{n+1})$ are Turing computable, then $\phi(x_1, \ldots, x_n)$ defined by

$$\phi(0, x_2, \ldots, x_n) = \psi(x_2, \ldots, x_n)$$

and

$$\phi(k+1, x_2, \ldots, x_n) = \chi(k, \phi(k, x_2, \ldots, x_n), x_2, \ldots, x_n)$$

is also Turing computable. In other words, explain why the schema of primitive recursion preserves Turing computability.

5. (f) Explain why whenever $\rho(x_1, \ldots, x_n, y)$ is Turing computable and

$$(\forall x_1) \ldots (\forall x_n)(\exists y)[\rho(x_1, \ldots, x_n, y) = 0],$$

then $\phi(x_1, \ldots, x_n)$ defined by

$$\phi(x_1, \ldots, x_n) = \mu y[\rho(x_1, \ldots, x_n, y) = 0]$$

is also Turing computable. In other words, explain why Kleene's μ-operator preserves Turing computability.

5.(g) Using exercises 5.(a)–5.(f) make a deduction concerning primitive recursive and general recursive functions. How is your conclusion related to Church's thesis?

Notes to the Instructor

This project is designed for an advanced undergraduate or graduate course in mathematical logic. It is also well suited for an advanced undergraduate or graduate course in computability theory. It could be assigned as a three to four week project on primite recursive, general recursive, and Turing computable functions. If there is any extra time available, the instructor may wish to elaborate on why every Turing computable function is general recursive.

References

[1] Church, A., "An Unsolvable Problem of Elementary Number Theory, Preliminary Report (abstract)," *Bull. Amer. Math. Soc.*, **41** (1935), 332–333.

[2] Church, A., "An Unsolvable Problem of Elementary Number Theory," *Amer. Journal Math.*, **58** (1936), 345–363. This paper with a short foreword by Davis was reprinted on pages 88–107 of [3].

[3] Davis, M., *The Undecidable. Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions*, Martin Davis (editor), Raven Press, Hewlett, N.Y., 1965.

[4] Davis, M., "Why Gödel Didn't Have Church's Thesis," *Inform. and Control*, **54** (1982), no. 1-2, 3–24.

[5] Gödel, K., "Über Formal Unentscheidbare Sätze der Principia Mathematica und Verwandter Systeme I," *Monatsh. Math. Phys.*, **38** (1931), 173–198. The English translation of this paper by Mendelson with foreword by Davis was reprinted on pages 4–38 of [3].

[6] Gödel, K., *On Undecidable Propositions of Formal Mathematical Systems*, Mimeographed lecture notes by S. C. Kleene and J. B. Rosser, Institute for Advanced Study, Princeton, N.J., 1934. This lecture note with foreword by Davis and postscriptum by Gödel were reprinted in [3], pages 39–74.

[7] Kleene, S. C., "General Recursive Functions of Natural Numbers," *Math. Ann.*, **112** (1936), 727–742. This paper with a short foreword by Davis was reprinted on pages 236–253 of [3].

[8] Kleene, S. C., "λ-Definability and Recursiveness," *Duke Math. Journal*, **2** (1936), 340–353.

[9] Kleene, S. C., "On Notation for Ordinal Numbers," *Journal of Symbolic Logic*, **3** (1938), 150–155.

[10] Kleene, S. C., "Recursive Predicates and Quantifiers," *Trans. Amer. Math. Soc.*, **53** (1943), 41–73. This paper with foreword by Davis was reprinted on pages 254–287 of [3].

[11] Kleene, S. C., *Introduction to Metamathematics*, D. Van Nostrand Co., Inc., New York, 1952.

[12] Kleene, S. C., "Origins of Recursive Function Theory," *Ann. Hist. Comput.*, **3** (1981), no. 1, 52–67.

[13] Post, E. L., "Finite Combinatory Processes, Formulation I," *Journal of Symbolic Logic*, **1** (1936), 103–105. This paper with a short foreword by Davis was reprinted on pages 288–291 of [3].

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

[15] Turing, A. M., "On Computable Numbers with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society* **42** (1936), 230–265. A correction, **43** (1937), 544–546. This paper with a short foreword by Davis was reprinted on pages 115–154 of [3].

[16] Turing, A. M., "Computability and λ-Definability," *Journal of Symbolic Logic*, **2** (1937), 153–163.