

Alonzo Church

Alex Cho and Sam Coskey

June 2002

Introduction

In response to the second of the 23 problems which Hilbert asked of the mathematical community in 1900, Kurt Gödel showed that every consistent mathematical system contains propositions which are undecidable, that is, which can neither be proved true nor proved false within that system.

Hilbert later posed the following question: given a proposition expressed in some closed mathematical system, is there an algorithm which will determine whether there exists a proof of that proposition from the axioms of the system? This is a question which may be asked of any formal system, and it came to be known as the Entscheidungsproblem, or “decision” problem, for that system.

The result that Church showed, and which we describe below, definitively and negatively answered this question. He showed the existence of several problems for which no algorithm could exist. As is the way with such proofs, he uses a clever diagonal argument. The setting in which Church presents his proofs is his own model of computation—the lambda calculus.

Turing completed his famous work, which ran along parallel lines, just a few weeks after Church published his paper. Turing’s conclusions regarding the Entscheidungsproblem were the same as Church’s. One might be curious, then, why Turing’s model of computation has been preserved to this day. In fact the accepted model of computation, and the model for the modern computer, is the machine of Turing. Church’s model is only applied in certain very high level programming languages such as Lisp, Scheme, and Haskell.

The clearest reason for Turing’s success is the simplicity of his model. It is a very intuitive notion that any algorithm may be expressed as a program for a Turing machine; it is not so intuitive that any algorithm may be expressed in terms of mathematical constructs like recursive functions.

In this paper, we will investigate Church’s work, starting with the birth of the lambda calculus as a system of formal logic, and ending with his famous results on the Entscheidungsproblem.

The Lambda Calculus

Church's first paper on what came to be known as the lambda calculus appeared in the *Annals of Mathematics* in 1932 [1]. In it, he outlined a new system for the expression of statements of first order logic, and showed some simple uses of his theory to prove statements about the system itself.

In this section, we outline the premises under which Church created his system, and the details of the system itself, including the axioms and rules by which manipulations may be carried out. We relate Church's lambda calculus to the more modern form of logic used today.

A New Formal System of Logic

Church first motivates his system with a discussion of the two classes of variables found in mathematical formulas. A *free* variable is one which does not fall within the scope of a quantifier or qualifier. A *bound* variable is just the opposite, that is, one whose mode of being has been declared [4]. For example, in the equation

$$(\exists n \in \mathbb{N})(x^n + y^n = z^n) \tag{1}$$

the variable n appears as a bound variable, because it is existentially quantified, whereas as x , y , and z do not. The importance of the bound variable can be seen by analyzing a statement such as

$$x^2 \geq 0 \tag{2}$$

In this expression, x is free, and unexplained. And in fact, while this statement is an iron law for any real x , it is patently false if x is allowed to take on complex values (indeed, it does not necessarily make sense).

Church makes it clear that in his system of logic, it should not be possible to prove from axioms any statement which contains a free variable. A free variable should only appear if it represents some abbreviation. Thus, the variables in Church's theory carry no semantic meaning in themselves. For example, in (1), the letter n can be substituted with any other letter without changing the meaning of the statement; it is not clear whether the letter y may be substituted, for its role in the equation is ambiguous. The letter y could even have been meant to represent some physical constant, or the number π .

It was well known at the time of Church that there are many difficulties associated with the creation of any closed, formal system of logic. These are the contradictions, such as that encountered by Frege, which can stem from even the most reasonable set of assumptions. Rather than avoid all of the contradictions by convoluting his notation and definitions, as did Russell and Whitehead with *Principia Mathematica*, Church chose the increasingly common practice of discarding the law

of the excluded middle. That is, to Church, any statement from which a contradiction could be derived was not necessarily false, but merely invalid. Thus, any statement written in the lambda calculus could be either true, false, or simply undefined.

Church provides as an example of such a statement a proposition which is known as Russell's paradox. (Russell's paradox, as it is typically discussed, postulates the existence of a set S of all sets which are not members of themselves. This leads to the contradiction that S must, and yet cannot, be a member of itself.) Church expresses an analog of this paradox in his own logical notation, in terms of a propositional function, which we shall denote R , with the property that from $R(R)$ one may derive $\neg R(R)$ and vice versa. Church addresses this contradiction by merely stating that the function R is not defined on the input R . There are other common paradoxes associated with propositional logic that Church does not address, for it was unknown at the time what might become of his calculus.

The reader might note that this idea of allowing propositions to be undefined bears resemblance to the ideology of intuitionism—Brouwer also abandoned the so-called law of the excluded middle. However, Church is quick to defend against potential criticism that his ideas are not new. While Brouwer had rejected the notion that $\neg\neg P$ is necessarily the same as P , Church accepted it. And while Brouwer had accepted the notion that a statement whose assumption leads to a contradiction is necessarily false, Church rejected it. But, the lambda calculus does require, as did Brouwer, that any proof of the existence of some value must construct the actual value—it is not enough to show that the negation of the existence of such a value leads to a contradiction.

Expressions

Aside from an arbitrary (countable) number of letters representing variables and propositional functions, formulas expressed in the lambda calculus use a very restricted symbol set. We will explain the most commonly used of these symbols, and their abbreviations (which Church adopts for convenience). In this discussion, F and G will represent propositional functions of one variable.

If M is any formula, then $\{F\}(M)$ denotes the value of F for the value M of its argument. It should be noted that any function must have arity exactly 1 (that is, it must take exactly one argument). Functions of two variables may be simulated using functions of functions. That is, if H depends on two variables, then it should actually be represented $\{\{H\}(x)\}(y)$, where $H(x)$ produces a function of one variable. This should be familiar to anyone who has programmed in the language Haskell, which uses what is known as function Currying to simulate functions of more than one variable. This whole discussion is actually somewhat extraneous, for Church allows that a such a function may be abbreviated in the natural way, $\{H\}(x, y)$.

Furthermore, as a general rule, the notation $\{F\}(x)$ is used when F is represented by more than

one symbol (in other words, if F is given by a literal formula), and just $F(x)$ or Fx is used in case that F is just one symbol.

The expression $\Pi(F, G)$ is the proposition that G is satisfied by all inputs x which satisfy F . The expression $\Sigma(F)$ is the proposition that F is satisfiable, in other words, that there exists some x such that $F(x)$ is true. On the other hand, the symbol ι represents a function of one input such that $\iota(F)$ results in a value x which satisfies F .

Let P and Q represent arbitrary propositions. Then the symbol $\&$ is a function of two inputs such that $\&(P, Q)$ denotes the proposition P and Q . This may be abbreviated $P \cdot Q$ or just PQ . The symbol \neg is a function of one input such that $\{\neg\}(P)$ or simply $\neg P$ denotes *not* P .

The most important symbol, and the source of the power of the lambda calculus is, not surprisingly, the λ . A lambda expression, $\lambda x[M]$, denotes a function mapping an input N to the expression resulting from substituting N for each occurrence of x in M . In other words, $\lambda x[M]$ is the function that gives the result $M(x)$ for each input x . Thus, given a formula, M , and a variable in that formula, x , the lambda creates a function of x defined by M . It is for this reason that the λ is sometimes referred to as the abstraction operator. For example, if $F = \lambda x[\&(\neg x, y)]$, then F is a function of x such that $F(N)$ yields $\&(\neg N, y)$.

Given these definitions, we may define a *well-formed* formula recursively as follows. The base cases are that any single variable is well-formed, and any one of the above discussed symbols is well-formed. The recursive step is that any combination of the above symbols and variables is well-formed. More precisely, if M and N are well-formed, then so are $\lambda x[M]$, $\neg M$, $\&(M, N)$, etc.

The symbols of the lambda calculus may seem few in number, but in fact all of the commonly used logical operations may be expressed in terms of them. To see this, we will show how simple implication may be expressed in terms of lambdas. What Frege would have written $P \supset Q$ may be written as

$$\{\lambda\mu\nu[\neg\&(\mu, \neg\nu)]\}(P, Q) \tag{3}$$

In this expression, $\lambda\mu\nu[]$ is short-hand to represent a function of two inputs. We leave it to the reader to verify that this is in fact equivalent to Frege's implication. Church also shows how to express logical *or* (which he writes \vee), logical equivalence (which he writes $==$), and the existential and universal quantifiers (which he writes $E(x)$ and $'x$, respectively).

Even a careful reader might not immediately see the need for such a complicated expression for implication when we already have the symbol $\Pi(F, G)$. Church distinguishes between this elemental form and the more natural $'x[F(x) \supset G(x)]$. The first expression reads " G is satisfied by all inputs which satisfy F ," whereas the second reads "for all x , if $F(x)$ is true then $G(x)$ is true." Precisely, the difference is that the first expression is valid even if F is undefined for some inputs, whereas the second is undefined if F is undefined for some x .

Postulates

We now are able to parse any expression written in the lambda calculus. Next, we must have rules by which we can manipulate them, and derive equivalent expressions from others. This, as in any logical system, is done by applying a pre-defined set of rules of procedure. In the lambda calculus, manipulations are carried out by substitution, which is defined by the following three rules.

The first rule allows one to replace any part of the expression M which depends on a single argument x , by another expression N which depends on another single variable y , where N is the result of substituting y for x throughout M . For example, $\lambda x[\&(F(x), G(x))]$ can be replaced by $\lambda y[\&(F(y), G(y))]$. In the modern context, such a substitution is known as an α -conversion. Essentially, this rule formalizes the notion that the name of a bound variable is irrelevant.

The second rule states that one may replace any part of a lambda expression by an expression obtained by evaluating it on the argument, provided that the argument does not contain any bound variables of the original expression as free variables. For example, $\{\lambda x[\neg(\&(F(x), G(x)))]\}(\neg y)$ can be replaced by $\neg(\&(F(\neg y), G(\neg y)))$. Not surprisingly, this substitution is known as a β -conversion.

The third rule is exactly the opposite operation of the second rule. It states that any expression may be substituted by a lambda expression in a certain way. As an example, an expression $F(y)$ might be replaced by the equivalent lambda expression $\{\lambda x[F(x)]\}(y)$.

Given these three rules for substitution, a formula A is said to be *convertible* to another formula B if B is obtainable from A by finite sequence of the three substitution operations.

Church gives an additional pair of rules, an application of which is called a *step* in a proof. The first of these rules is confusing and rarely used. The second rule states that if, in the course of the proof, one has derived $\Pi(F, G)$ and $F(M)$, then one may derive $G(M)$. This is an intuitive consequence of the definition of Π , for it gives that G is true on any inputs for which F is true.

Of course, no system of logic is complete without a set of axioms from which the theory of that system may be derived, using the rules of conversion. Church gives a list of 37 statements which he defines to be the postulates of the lambda calculus. (As a side note, Church later modified this list in [2], to avoid certain contradictions which he noticed in between the publication of the two papers. One of the errors in his original list allowed him to derive the R proposition of Russell's paradox!) We will give a few examples of the postulates laid out in [1].

The first postulate is something like a reflexive property, and it reads

$$\Sigma(\phi) \supset_{\phi} \Pi(\phi, \phi) \tag{4}$$

The symbol \supset_{ϕ} we have not yet explained—it just means that the implication holds for all ϕ for which the left hand side is defined. Thus, this postulate says roughly that if a function ϕ is satisfied

by some value, then ϕ is satisfied by any input which satisfies ϕ . Most of the axioms are similarly basic; they tie together the meaning of the various symbols.

A somewhat more complicated, yet very important postulate is the 13th, which states that

$$\Sigma(\phi) \supset_{\phi} [[\phi(x) \supset_x \psi(x)] \supset_{\psi} \Pi(\phi, \psi)] \quad (5)$$

The meaning of this statement goes back to the aforementioned subtle difference between the Π symbol and implication. Just $\phi(x) \supset_x \psi(x)$ is not enough to derive $\Pi(\phi, \psi)$; one also must know that ϕ is satisfiable in the first place.

One final, simpler example of one of Church's postulates is the 14th, which reads

$$p \supset_p [q \supset_q \&(p, q)] \quad (6)$$

In a more modern context, this rule is known as conjunction, and it roughly states that if p and q are both true, then the logical product p and q is true.

An Unsolvable Problem

In 1936, Church published his most important paper [3]. In this paper, he used the ideas and symbols of the lambda calculus which he had previously developed in a much more general context than that of first order logic. In particular, he used his symbols to address the question of Hilbert.

In this section, we discuss in some detail the structure of definitions which Church built from his lambda calculus. We then explain the important theorems which stem from these definitions. We will not spend too much time on the details of the proofs of these theorems, for they are moderately technical.

Definitions

Church defines a *reduction* of A to be any formula B such that B is the result of exactly one application of the second substitution rule to A . A formula is said to be in *normal form* if it is well-formed and does not contain any un-applied function. (In other words, if it does not contain some expression of the form $\{\lambda x[M]\}(N)$.) A formula B is a normal form of A if B is in normal form, and A is convertible into B using a finite sequence of reductions. Church states without proof the following three facts about normal forms: (1) a normal form is unique up to the naming of variables, (2) no further reduction of a normal form is possible, and (3) if a formula has a normal form, then every well-formed part of it has a normal form.

Along with the rules and notions of reduction and normal form of a formula, Church introduced a way of representing a natural number n in terms of a “numeral” \mathbf{n} , expressed in the language of lambdas. The symbol \mathbf{n} is short-hand for $\lambda xy[x^n(y)]$, so that a generic application $\{\mathbf{n}\}(F, N)$, represents the formula $F(\dots(F(FN))\dots)$, in other words, the application of a function F to N repeated n times. We should also note that $\mathbf{3}$ can be used for any representation of application of a formula three times.

As an example of working with these so-called numerals, consider how one would add two numbers in the lambda calculus. We propose that the addition function may be written as a function of two numerals as follows.

$$+(\mathbf{m}, \mathbf{n}) =_{\text{def}} \lambda \mathbf{m} \mathbf{n} [\lambda ab[\mathbf{m}(a, \mathbf{n}(a, b))]] \quad (7)$$

While it may not be immediately obvious that this will add the two numerals \mathbf{m} and \mathbf{n} , we demonstrate the operation with the addition of the numerals $\mathbf{1}$ and $\mathbf{2}$.

$$\begin{aligned} +(\mathbf{1}, \mathbf{2}) &= \{\lambda \mathbf{m} \mathbf{n} [\lambda ab[\mathbf{m}(a, \mathbf{n}(a, b))]]\}(\mathbf{1}, \mathbf{2}) \\ &\text{conv } \lambda ab[\mathbf{1}(a, \mathbf{2}(a, b))] \\ &= \lambda ab[\{\lambda cd[c(d)]\}(a, a(a(b)))] \\ &\text{conv } \lambda ab[a(a(a(b)))] \\ &= \mathbf{3} \end{aligned}$$

Similarly, the function for multiplication may be expressed as

$$\times(\mathbf{m}, \mathbf{n}) =_{\text{def}} \lambda \mathbf{m} \mathbf{n} [\mathbf{m}(\mathbf{n})] \quad (8)$$

In the preceding formula, \mathbf{n} is curried, that is, substituted for the first argument of \mathbf{m} . We leave it to the reader to verify that this does in fact result in the numeral $\mathbf{m} \times \mathbf{n}$.

We now introduce the term *λ -definability* of a function of natural number arguments. A function F is λ -definable if there exists a formula in the lambda calculus, \mathbf{F} , such that whenever $F(n_1, \dots, n_k) = r$, for natural numbers n_i and r , we have that $\{\mathbf{F}\}(\mathbf{n}_1, \dots, \mathbf{n}_k)$ is convertible into \mathbf{r} . By applying a finite number of substitution operations, one may convert any λ -definable formula to a normal form. Once we have a normal form, we can calculate its value very easily.

Church next gives a preliminary definition of *effective calculability* of a function in terms of λ -definability. If a function is λ -definable, “the process of reduction of formulas to normal form provides an algorithm for the effective calculation of particular values of the function” [3]. This means that the conversion of a formula into its normal form merely represents the process of evaluating the function for a particular input. In this way, Church takes the effective calculation of a function as his definition of an algorithm.

We have already seen how to express arithmetic in the lambda calculus. Church next shows us how to express formulas in the lambda calculus in the language of arithmetic. To do this, Church essentially adopts the formulation that Gödel used to represent propositions in *Principia Mathematica*. That is, Church assigns to each symbol (λ , $\{$, $\}$, etc.) of the lambda calculus a small prime number, and to each variable of the formula successive prime numbers. It is then possible to convert any sequence of n characters representing a formula into a corresponding sequence of primes, $(p_1, p_2, p_3, p_4, \dots, p_n)$. One may thus represent any formula uniquely by the integer $2^{p_1} 3^{p_2} 5^{p_3} 7^{p_4} \dots q^{p_n}$, where q represents the n^{th} prime.

Perhaps the most important definition that Church makes is that of a recursive function. An elementary equation is one of the form $A == B$, where A and B are composed of the symbol 1, a successor function S , and other functional and numerical variables. A recursive function f is some function of natural numbers for which there is a set of elementary equations E , called the recursion equations of f , which satisfy certain properties. The key fact about this generating set of elementary equations is that its properties provide a method by which f may be evaluated. We omit further details on recursive functions.

Given some property of natural numbers, one may define the corresponding propositional function as one whose value is 2 when the property is true, and 1 when the property is false. For example, if the desired property is odd-ness of a number n , then the corresponding propositional function may be written in modern terms as $f(n) = 2(\frac{n}{2} - \lfloor \frac{n}{2} \rfloor) + 1$, where $\lfloor \rfloor$ represents the greatest integer function. In principal, one could write down a set of recursion equations for $f(x)$, but it is very confusing and we will not do so here.

Church gives a relatively large body of functions which are recursive. First of all, the propositional function which tells whether its input is the Gödel representation of a well-formed formula of lambda calculus is recursive. An important consequence of this fact is that there is an algorithm to enumerate all of the well-formed formulas. Similarly, the propositional function which tells whether its input is in normal form is recursive.

As another example, the function whose value is n when the Gödel representation of the formula \mathbf{n} is input, and whose value is some fixed integer k otherwise, is recursive. Not surprisingly, the function whose behavior is exactly the same except that it outputs $n + 1$ on input \mathbf{n} is also recursive.

Church states the result (due both to himself and to his student, Kleene) that every recursive function (of natural numbers) is λ -definable, and also that every λ -definable function is recursive. This allows him to argue that it is reasonable to take recursiveness as his definition of effective calculability. Hence, the ideas of effective calculability, λ -definability, and recursiveness are taken to be equivalent. This claim is the basis of what is known as Church's Thesis; anyone who accepts this principle must also accept that Church's work is important.

For any system of logic, a function or formula is effectively calculable means that either there exists an algorithm that can generate all the possible derivations from it by the axioms and rules of procedures from the system, or if the function is evaluated to some value, then the every single evaluation process must be justified or provable within the system or both. Therefore, either of λ -definability or recursiveness of a function implies that the function is effectively calculable in this sense.

Main Theorem

The fundamental result of this 1936 paper is the following. There is no recursive propositional function which tells, given a formula, C , whether or not C has a normal form. Since Church has taken recursiveness as the defining property of an algorithm, this theorem provides an example of a problem for which no algorithm exists!

The proof of this theorem is quite technical, but fortunately Church writes it twice: once in symbols, and once in words. The first proof relies on functions which operate on the Gödel number of certain propositions. It is quite explicit, and quite difficult to follow. We will summarize the crucial points of the more abstract proof here.

Suppose that there exists an algorithm which decides whether a formula C has a normal form. Then, first of all, we can use this algorithm as a subroutine to help us decide whether a formula is convertible into a numeral (as defined in the previous section) in the following way. Given a formula F , we can use the hypothetical algorithm to determine whether F has a normal form. If it does, then we can enumerate every formula to which F is convertible by a sequence of substitutions. Since F has a unique normal form, we must eventually find it in this enumeration. We can then easily check whether the unique normal form is a numeral, and if so, which numeral it is.

Next, we construct a complete list A_1, A_2, \dots of formulas which have a normal form. This can be done, for we have already seen that the set of well-formed formulas is enumerable, and for each one we can use our hypothetical algorithm to check if it has a normal form. It is helpful to consider the table below. Our list of formulas which have a normal appears down the left column, and the list of numerals appears along the top row. The cell on the i^{th} row and j^{th} column shows the normal form of $A_i(j)$. Some of the $A_i(j)$ reduce down to numerals, and others reduce to other

normal formulas. Illustrative values are filled in.

	1	2	3	4	...
A_1	7	3	2	20	
A_2	9	2	other	6	
A_3	1	2	7	8	
A_4	5	other	31	6	
		\vdots			\ddots

(9)

Now, we define the diagonal on input n as follows:

$$D(n) = \begin{cases} 1 & \text{if the normal form of } A_n(\mathbf{n}) \text{ is not a numeral} \\ m + 1 & \text{if the normal form of } A_n(\mathbf{n}) \text{ is some numeral } \mathbf{m} \end{cases} \quad (10)$$

We have already created an algorithm to determine if $A_n(\mathbf{n})$ is convertible to a numeral, and to which one. Thus, the function D is effectively calculable, and hence is λ -definable. According to the definition of λ -definability (previous section), this means that there exists a formula, which we will call \mathcal{D} , such that the normal form of $\mathcal{D}(\mathbf{n})$ is the numeral representing $D(n)$.

Clearly, $\mathcal{D}(\mathbf{1})$ has a normal form, since $\mathcal{D}(\mathbf{1})$ by definition reduces to the numeral defined by $D(1)$ in (10). Since \mathcal{D} is itself a well-formed part of the formula $\mathcal{D}(\mathbf{1})$, the third fact about normal forms (previous section) tells us that \mathcal{D} must itself have a normal form. But, this means that \mathcal{D} must appear in the list A_i of formulas which have a normal form. In other words, \mathcal{D} appears on the left-hand column of table (9).

This is a contradiction. For, suppose for example \mathcal{D} is A_3 . Then, according to the definition of \mathcal{D} , the normal form of $\mathcal{D}(\mathbf{3})$ is the numeral **8**. But, according to table (9), the normal form of $A_3(\mathbf{3})$ is the numeral **7**. This contradicts the fact that \mathcal{D} belongs in table (9), hence our initial assumption (that an algorithm which decides whether a formula C has a normal form) cannot exist.

It is interesting to contrast the fact that this main theorem was proved in Church's paper by contradiction. For, not four years earlier, Church had abandoned the notion that a statement from which a contradiction can be derived is necessarily false in his lambda calculus.

One Final Reduction

To start, we define a term of Gödel. Loosely speaking, a system is ω -consistent if there does not exist a propositional function F of one variable such that: (1) $F(n)$ is provable for every n , and (2) the statement "for every n , $F(n)$ " is provably false within the system. It has been shown that ω -consistency is equivalent to consistency in the usual sense, that is, in the sense that there should not exist true contradictory propositions.

There are two important problems that Church investigated in [3]. The first (which we shall call problem P_1) is the problem of finding recursive propositional function which tells whether a formula has a normal form or not. We showed in the previous subsection that this problem is unsolvable. Another problem (which we shall call problem P_2) is that of finding a recursive propositional function of two formulas A and B , which tells whether or not A is convertible into B . Church showed in his paper that these two problems are equivalent, and hence that the latter is also unsolvable.

As a corollary to this fact, Church was able to end his paper by showing that the Entscheidungsproblem is unsolvable for any ω -consistent system of symbolic logic. He does so by a reduction from problem P_2 above to the Entscheidungsproblem in the following way. Given two formulas A and B , construct a computable propositional function ψ such that A is convertible into B if and only if ψ is provable.

Suppose that we had an algorithm to solve the Entscheidungsproblem for some suitably powerful ω -consistent system. In such a system, we may express the propositional function ϕ of two natural inputs a and b which tells whether a and b are the Gödel representations of formulas A and B such that A is immediately convertible to B (in one step). And if this can be done, then we may express the propositional function ψ of two natural inputs a and b which tells whether a and b are the Gödel representations of formulas A and B such that A is convertible to B (in some finite sequence of steps).

Suppose that a and b are the Gödel representations of formulas A and B . If A is convertible into B , then $\psi(a, b)$ may be proved by the sequence of conversions which yields B from A . Conversely, if A is not convertible into B , then $\psi(a, b)$ must not be provable, or else the system would be inconsistent (and hence not ω -consistent). This completes the reduction.

References

- [1] Church, Alonzo. "A Set of Postulates for the Foundation of Logic." *Annals of Mathematics*, volume 33 issue 2 (April, 1932), pp. 346-366.
- [2] Church, Alonzo. "A Set of Postulates for the Foundation of Logic." *Annals of Mathematics*, volume 34 issue 4 (October, 1933), pp. 839-864.
- [3] Church, Alonzo. "An Unsolvable problem in elementary number theory." *American Journal of Mathematics*, volume 58 issue 2 (April, 1936), pp. 345-363.
- [4] Weisstein, Eric. mathworld.com. Wolfram Reasearch.