Alonzo Church and Lambda Calculus

Charles Hu

***Abstract*— Alonzo Church was a mathematician and philosopher known for his work on formal logic and computability theory. Chief among his achievements was his development of lambda caclulus, which paved the way for early computer science by providing a means to effectively decide the computability of an algorithm.**

I. INTRODUCTION

Alonzo Church was an American mathematician and philosopher renowned for his contributions to logic and early theoretical computer science. He is often considered one of the founders of the field of computer science for his substantial work in establishing the foundations of computability theory. In particular, his work on lambda calculus and the Church-Turing thesis paved the way for the determination and modeling of effective computability and decidability of algorithms.

II. LIFE

A. Early Life

Alonzo Church was born on June 14, 1903 in Washington D.C. to Samuel Robbins Church and Mildred Hannah Letterman Parker. He went to Princeton University for his undergraduate degree in mathematics and graduated in 1924. He then stayed there for his doctorate under the supervision of Oswald Veblen [10]. During this time, he met and married Mary Julia Kuczinski. Together, they had three children: Alonzo Jr., Mary Ann, and Mildred.

Church received his doctorate three years later in 1927 for his dissertation on the axiom of choice in Zermelo-Fraenkel set theory [1]. Following this, he worked at a number of different research institutions until he finally returned to Princeton in 1929 as an Assistant Professor of Mathematics [10].

B. Career

Church was promoted to Associate Professor in 1939 and later would gain full professorship in 1947. In 1961, Church became a Professor of Mathematics and Philosophy, a position he held until he left Princeton in 1967. He then moved to the University of California at Los Angeles where he worked as a Kent Professor of Philosophy and Professor of Mathematics until his full retirement in 1990 [10]. Church spent the last years of his life in Hudson, Ohio, where he died on August 11, 1995 [13].

During his tenure, Church would advise 31 doctoral students, the most well known of which being Alan Turing and Stephen Kleene. Church would also pioneer the formalization of logic with his proposal of lambda calculus, which led to the development of Church's theorem and the Church-Turing thesis.

III. LAMBDA CALCULUS

A. Development

Lambda calculus (also written as λ-calculus) is a set of notations and conventions devised to formalize the description of functions and their applications [8]. It was first proposed by Church through a set of two papers published in 1932 [2] and 1933 [3] in an attempt to provide a consistent, nonparadoxical system of abstraction for the representation of formal logic; however, a number of idiosyncrasies existed in lambda calculus that persisted until 1936 [12]. That year, Church finally developed the first and simplest fully logically consistent version of lambda calculus [5], now known as untyped lambda calculus. Church and others would continue to work on expanding the formalisms of lambda calculus to create versions of what would later be known as typed lambda calculus, which encorporates a system of assignable types for functions.

B. Definition

The most basic unit of lambda calculus is the variable, which is denoted as any arbitrary symbol (alphabetical symbols will be used here). Variables serve as nonspecific identifiers, and multiple variables can be combined to form an expression. Thus, for variables x, y, and z, we can create the expression xyz.

Functions are components which allow for the application of expressions and are necessary for expression resolution. A function typically is structured as follows:

$$\lambda < variable > . < expression > \tag{1}$$

The subsection before the period is known as the head, and the subsection after the period is known as the body. A function operates by binding a variable in the head and resolving instances of that variable in the body via application with another expression that follows the function. Variables that are bound by a lambda are referred to as bound variables, while variables not bound by a lambda are referred to as free variables.

Applications are the resolution mechanism in lambda caclulus and operate via substitution. An application will always consist of a function followed by another expression (from now on referred to as the applied expression as opposed to the function expression in the function body). To resolve an application, every instance of the bound variable that can also be found in the function expression must be

substituted by an applied expression. For instance, consider the following application and its resolution:

$$(\lambda x.xyz)(abc) = abcyz \tag{2}$$

These components can be recursively defined [11] as follows:

$$\begin{aligned} < expression > := & < variable > \\ & | < function > \\ & | < application > \\ & | < expression >< expression > \end{aligned} \tag{3}$$

$$< function > := \lambda < variable > . < expression > \tag{4}$$

$$< application > := < function >< expression > \tag{5}$$

Note that common shorthand writing includes parentheses and simplified nested function heads. These are primarily for clarity and do not change the operation precedence, which is left associative. This is summarized in the following:

$$(x) \equiv x \tag{6}$$

$$xyz \equiv (xy)z \tag{7}$$

$$\lambda x.\lambda y.xyz \equiv \lambda x.(\lambda y.xyz) \equiv \lambda xy.xyz \tag{8}$$

C. Application

Lambda caclulus is commonly used in mathematics and in computer science to formally describe the computation of algorithms. It also sees usage in fields such as linguistics for syntactic structure mapping [9].

A common example of an application of lambda calculus is the construction of basic natural number addition, also known as an addition algorithm. This can be developed by viewing addition as the repetition of the succession of natural numbers (i.e., $x + y$ is equivalent to incrementing y by 1 x times).

To begin, natural numbers can be defined as the incrementation of the number 0 by 1. This is formalized [11] as our base number 0 and a successor function as follows:

$$0 := \lambda ab.b \tag{9}$$

$$successor := \lambda xyz.y(xyz) \tag{10}$$

So the number 1 can be formalized as a succession upon 0 as such:

$$(\lambda xyz.y(xyz))(\lambda ab.b) \tag{11}$$

$$= \lambda yz.y((\lambda ab.b)yz) \tag{12}$$

$$= \lambda yz.y((\lambda b.b)z) \tag{13}$$

$$= \lambda yz.y(z) \equiv \lambda ab.a(b) \equiv 1 \tag{14}$$

Note that since variables are arbitrary, they can be substituted ad hoc for consistency and clarity. Here, variables y and z are replaced with a and b in the final result for visual consistency across the developed definitions.

Addition can then be viewed as applying the successor function on a given number by some number of times. So 1 + 2 is equivalent to applying the successor function one time to the number 2 as follows:

$$1 + 2 \tag{15}$$

$$\equiv (\lambda ab.a(b))(\lambda xyz.y(xyz))(\lambda cd.c(c(d))) \tag{16}$$

$$= (\lambda b.(\lambda xyz.y(xyz))(b))(\lambda cd.c(c(d))) \tag{17}$$

$$= (\lambda xyz.y(xyz))(\lambda cd.c(c(d))) \tag{18}$$

$$= (\lambda yz.y((\lambda cd.c(c(d)))yz)) \tag{19}$$

$$= (\lambda yz.y((\lambda d.y(y(d)))z)) \tag{20}$$

$$= \lambda yz.y(y(y(z))) \equiv 3 \tag{21}$$

D. Impact

Lambda calculus is significant in that it provides a robust and powerful framework which allows for the easy expression of algorithmic functions. Leveraging the descriptive capabilities of lambda calculus, Church provably qualified the intuitive notion that algorithms existed and were calculable through the construction of the formal definition of an algorithm and its function via λ-definablity. He posited that an algorithm was effectively calculable if and only if it could be constructed using lambda calculus and recursively defined [5]. An example of this can be seen with the previously described addition algorithm (recall how it was recursively defined from base integer 0 and leveraged λ-defined functions to perform the algorithm of addition).

Church's graduate student, Alan Turing, would later verify Church's findings independently through his development of Turing machines to solve the issue of computable numbers and the Entscheidungsproblem [6]. Turing himself would note the equivalency of his Turing machines and Church's lambda calculus for effective computability in a follow-up paper [7]. Church's λ-definability, combined with Turing's Turing machines and Gödel's definition for general recursive functions, serves as the basis of what is now known as the Church-Turing thesis, which provides a general foundational understanding of the computability of a function given its nature and describability by the aforementioned theoretical frameworks.

IV. CONCLUSION

Church's work on lambda calculus revolutionized mathematical logic by providing an expressive framework to formally define algorithms. Using lambda calculus, Church was able to answer the Entscheidungsproblem by proving that first-order logic is not decidable [4]. He would also go on to use lambda calculus to provide a proof for effective computability of an algorithm [5], which would be later combined with Turing's proof for the effective computability for Turing machines to create the Church-Turing thesis. These achievements helped found early theoretical computer

science by providing a means to write potential computer algorithms and verify their effective calculability and limitations.

REFERENCES

[1] A. Church, "Alternatives to Zermelo's assumption," Transactions of the American Mathematical Society, vol. 29, no. 1, pp. 178-208, 1927, doi: https://doi.org/10.1090/S0002-9947-1927-1501383-1.

[2] A. Church, "A Set of Postulates for the Foundation of Logic," The Annals of Mathematics, vol. 33, no. 2, p. 346, Apr. 1932, doi: https://doi.org/10.2307/1968337.

[3] A. Church, "A Set of Postulates For the Foundation of Logic," The Annals of Mathematics, vol. 34, no. 4, p. 839, Oct. 1933, doi: https://doi.org/10.2307/1968702.

[4] A. Church, "A note on the Entscheidungsproblem," Journal of Symbolic Logic, vol. 1, no. 1, pp. 40-41, Mar. 1936, doi: https://doi.org/10.2307/2269326.

[5] A. Church, "An Unsolvable Problem of Elementary Number Theory," American Journal of Mathematics, vol. 58, no. 2, p. 345, Apr. 1936, doi: https://doi.org/10.2307/2371045.

[6] A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," Proceedings of the London Mathematical Society, vol. s2-42, no. 1, pp. 230-265, 1937, doi: https://doi.org/10.1112/plms/s2-42.1.230.

[7] A. M. Turing, "Computability and λ-definability," Journal of Symbolic Logic, vol. 2, no. 4, pp. 153-163, 1937. doi:10.2307/2268280.

[8] H. Deutsch and O. Marshall, "Alonzo Church," Stanford Encyclopedia of Philosophy, 2023. https://plato.stanford.edu/entries/church/supplementD.html

[9] H. Deutsch and O. Marshall, "The λ-Calculus and Type Theory," Stanford Encyclopedia of Philosophy, 2023. https://plato.stanford.edu/entries/church/supplementD.html

[10] J. O'Connor and E. Robertson , "Alonzo Church - Biography," Maths History, Nov. 2004. https://mathshistory.st-andrews.ac.uk/Biographies/Church/

[11] R. Rojas, "A Tutorial Introduction to the Lambda Calculus," arXiv.org, Mar. 27, 2015. https://arxiv.org/abs/1503.09060

[12] S. Steinert-Threlkeld, "Lambda Calculi," Internet Encyclopedia of Philosophy. https://iep.utm.edu/lambda-calculi/

[13] The Editors of Encyclopaedia Britannica, "Alonzo Church," Britannica, Aug. 07, 2023. https://www.britannica.com/biography/Alonzo-Church