λ-Calculus: Then & Now

Dana S. Scott

University Professor Emeritus
Carnegie Mellon University

Visiting Scholar
University of California, Berkeley

dana.scott@cs.cmu.edu

Taken from talks prepared for:

TURING CENTENNIAL CELEBRATION
Princeton University, May 10-12, 2012

ACM TURING CENTENARY CELEBRATION
San Francisco, June 15-16, 2012

Symbols of Princeton

Traditional

From the Graduate Alumni
(to encourage ecology)

The **λ-calculus** was begun by A. Church at Princeton, but it has been **recycled** every decade after the 1930s in **new** and **useful** ways.

For a General History: **F. Cardone and J.R. Hindley**.
"Lambda-Calculus and Combinators in the 20th Century."
In: Volume 5, pp. 723-818, of **Handbook of the History of Logic**,
Dov M. Gabbay and John Woods eds.,
North-Holland/Elsevier Science, 2009

What is the λ-Calculus?

The calculus gives rules for the ***explicit definition*** of functions; however, this ***type-free*** version also permits ***recursion*** and ***self-replication.***

α-conversion

$$\lambda X.[\ldots X\ldots] = \lambda Y.[\ldots Y\ldots]$$

β-conversion

$$(\lambda X.[\ldots X\ldots])(T) = [\ldots T\ldots]$$

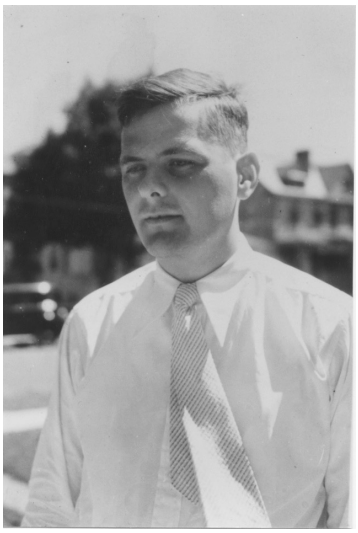
η-conversion

$$\lambda X.F(X) = F$$

The names of the rules are due to H.B. Curry. The last rule fails in many interpretations, and special efforts are needed to make it valid.

Church's original system (1932) also had rules for ***logic***, but that was the system his students Kleene & Rosser proved ***inconsistent*** (1936). Church also proposed his calculus as giving a ***definition of computability***. But K. Gödel objected and later preferred Turing's definition.

Church vs. Turing



Alonzo Church
Born: 14 June 1903 in Washington, D.C., USA.
Died: 11 Aug 1995 in Hudson, Ohio, USA.
Ph.D.: Princeton University, 1927, USA

Alan Turing
Born: 23 June 1912, Maida Vale, London, UK.
Died: 7 June 1954, Wilmslow, Cheshire, UK.
Ph.D.: Princeton University, 1938, USA.

The work of Church and Turing in 1936 was done independently.

Alonzo Church, *"An Unsolvable Problem in Elementary Number Theory,"* American J. of Mathematics, vol. 5 (1936), pp. 345-363.

Alonzo Church, *"A Note on the Entscheidungsproblem,"* J. of Symbolic Logic, vol. 1 (1936) pp. 40-41. Correction: *ibid*, pp. 101-102.

Alan Turing,*"On Computable Numbers with an Application to the Entscheidungsproblem,"* Proc. of the London Math. Soc., vol. 42 (1936), pp. 230-267. Correction: vol. 43 (1937), pp. 544-546.

Alan Turing,*"Computability and λ -definability,"* J. Symbolic Logic, vol. 2 (1937), pp. 153-163.

Three Other Pioneers

Haskell Brooks Curry

Born: 12 Sept 1900 in Millis, MA, USA.
Died: 1 Sept 1982 in State College, PA, USA.
Ph.D.: Göttingen Universität, 1930, Germany.
Thesis: Grundlagen der kombinatorischen Logik

Stephen Cole Kleene

Born: 5 Jan 1909 in Hartford, CN, USA.
Died: 25 Jan 1994 in Madison, WI, USA.
Ph.D.: Princeton University, 1934, USA.
Thesis: A Theory of Positive Integers in Formal Logic

J. Barkley Rosser

Born: 6 Dec 1907 in Jacksonville, FL, USA.
Died: 5 Sept 1989 in Madison, WI, USA.
Ph.D.: Princeton University, 1934, USA.
Thesis: A Mathematical Logic without Variables

It seems, sadly, that Alan Turing never had a chance to meet these people or Kurt Gödel.

The Connection to Computability

Church Numerals

$\underline{0} = \lambda F.\lambda X.X$

$\underline{n+1} = \lambda F.\lambda X.F(\underline{n}(F)(X))$

$\underline{n+m} = \lambda F.\lambda X.\underline{n}(F)(\underline{m}(F)(X))$

$\underline{n \times m} = \lambda F.\underline{n}(\underline{m}(F))$

$\underline{m^n} = \underline{n}(\underline{m})$

$\underline{n-1}$ = [*a little harder*]

Fixed-Point Combinator

$Y = \lambda F.(\lambda X.F(X(X)))(\lambda X.F(X(X)))$

$Y(F) = F(Y(F))$

Theorem. For every ***partial recursive function*** g(n), there is a ***constant λ-term*** **G** such that

$\mathbf{G}(\underline{n}) = \underline{g(n)}$, for all n.

Kleene and **Turing** independently proved this in different ways.

Some λ-Definitions

$$\mathbf{pair} = \lambda X.\lambda Y.\lambda F.F(X)(Y)$$

$$\mathbf{fst} = \lambda P.P(\lambda X.\lambda Y.X)$$

$$\mathbf{snd} = \lambda P.P(\lambda X.\lambda Y.Y)$$

$$\mathbf{succ} = \lambda N.\lambda F.\lambda X.F(N(F)(X))$$

$$\mathbf{shft} = \lambda S.\lambda P.\ \mathbf{pair}(S(\mathbf{fst}(P)))(\mathbf{fst}(P))$$

$$\mathbf{pred} = \lambda N.\ \mathbf{snd}(N(\mathbf{shft}(\mathbf{succ}))(\mathbf{pair}(\underline{0})(\underline{0})))$$

Kleene's "trick" here is to introduce **pairs** as a **data structure**, and then apply iteration to get a **sequence** of pairs.

$$\mathbf{test} = \lambda N.\lambda U.\lambda V.\mathbf{snd}(N(\mathbf{shft}(\lambda X.X))(\mathbf{pair}(V)(U)))$$

$$\mathbf{mult} = \lambda N.\lambda M.\lambda F.N(M(F))$$

$$\mathbf{fact} = \lambda N.\mathbf{test}(N)(\underline{1})(\mathbf{mult}(N)(\mathbf{fact}(\mathbf{pred}(N))))$$

$$\mathbf{fact} = \mathbf{Y}(\lambda F.\lambda N.\mathbf{test}(N)(\underline{1})(\mathbf{mult}(N)(F(\mathbf{pred}(N)))))$$

The factorial function must be the most **overdefined** function in the history of mankind!

Church-Turing Thesis

accepted with the help of Kleene after Turing explained his machines.

Effectively computable functions of natural numbers can be identified with those definable by:

- **λ-calculus**
- **Herbrand-Gödel equations**
- **Partial-recursive schemata**
- **Turing-Post machine programs**

If Gödel had stayed in Princeton, and

If Church and Kleene had argued better for data structures in the λ-calculus,

Then surely Gödel would have accepted λ-calculus as a foundation much earlier.

Note that Kleene proved the equivalence with Herbrand-Gödel computability **before** Turing's work.

Kleene's Complaint

I myself, perhaps unduly influenced by rather chilly receptions from audiences around **1933-35** to disquisitions on λ-definability, chose, after ***general recursiveness*** had appeared, to put my work in that format. I did later publish one paper **1962** on λ-definability in higher recursion theory.

I thought general recursiveness came the closest to ***traditional mathematics***. It spoke in a language familiar to mathematicians, extending the theory of ***special recursiveness***, which derived from formulations of Dedekind and Peano in the mainstream of mathematics.

I cannot complain about my audiences after **1935**, although whether the improvement came from switching I do not know. In retrospect, I now feel it was too bad I did not keep active in λ-definability as well. So I am glad that interest in λ-definability has revived, as illustrated by Dana Scott's **1963** communication.

Were the truth to be known, Kleene **translated** much of what he had done in λ-calculus into working with integers. Indeed, the **application operation** {e}(n) defines a **partial combinatory algebra** with many properties similar to the work of Curry and Rosser.

Does λ-Calculus have Models?

Yes! There ***is*** a calculus for ***enumeration operators!***
First we need some simple definitions on integers and sets of integers:

$$(n,m) = 2^n(2m+1)$$

$$\mathbf{set}(0) = \varnothing$$

$$\mathbf{set}((n,m)) = \mathbf{set}(n) \cup \{m\}$$

$$X^* = \{ n \mid \mathbf{set}(n) \subseteq X \}$$

Application

$$F(X) = \{ m \mid \exists n \in X^*.(n,m) \in F \}$$

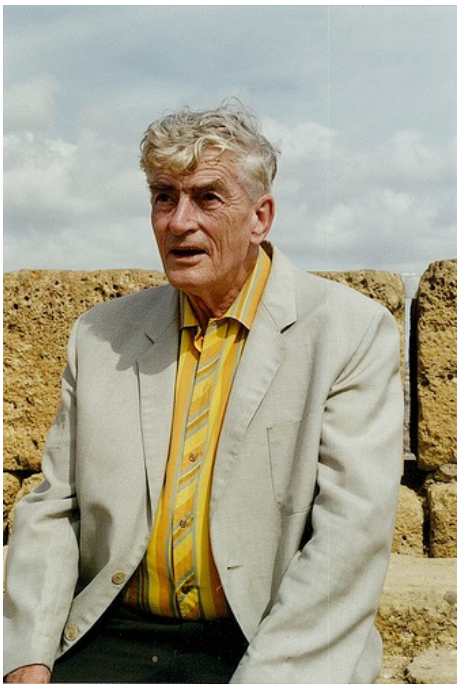
Abstraction

$$\lambda X.[\ldots X \ldots] =$$

$$\{0\} \cup \{ (n,m) \mid m \in [\ldots \mathbf{set}(n) \ldots] \}$$

Every set of integers can be used as an enumeration operator. The operator is computable if the set is r.e. Many compound contexts do define enumeration operators.

Turing's Only Student



Robin Oliver Gandy

Born: 23 September 1919, Peppard, Oxon., UK.
Died: 20 November 1995, Oxford, UK.
Ph.D.: Cambridge, 1953.
Thesis: *On axiomatic systems in Mathematics and theories in Physics.*
Supervisor: Alan Turing.
Reader: Oxford University, Wolfson College, 1969-1986.
Students: 26 and 126 descendants.

Another pioneer, **Gandy**, later became a key contributor to the development of **Recursive Function Theory**.

It is interesting to note that both the teams of

Myhill and **Shepherdson**

and, later,

Friedberg and **Rogers**

defined enumeration operators without seeing they had **models** for the λ-calculus.

What is the Entscheidungsproblem?

To determine whether a formula of the ***first-order*** predicate calculus is ***provable*** or not.

Church's Solution

Theorem. Only a finite number of axioms are needed to define a ***non-recursive*** set of integers.

R.M.Robinson's Arithmetic

(1) $\forall x \forall y [x = y \iff Sx = Sy]$

(2) $\forall x [x = 0 \iff \neg \exists y.\ x = Sy]$

(3) $\forall x \forall y [(x + 0) = x \ \& \ (x + Sy) = S(x + y)]$

(4) $\forall x \forall y [(x \times 0) = 0 \ \& \ (x \times Sy) = ((x \times y) + x)]$

After the solution of Hilbert's 10th Problem, the applicability of this theory became even easier.

Turing's Solution

Theorem. Only a finite number of axioms are needed to define the ***Universal Turing Machine***.

Minskyizing the UTS

Starting with **Claude Shannon** in 1956, many people — often in competition with **Marvin Minsky** — proposed **very small UTMs** (but their operation requires extensive coding of **patterns**). But, **axiomatically**, they do not require as many axioms as Turing did.

Post-Markov's Solution

The basic idea of Post (1943) was that a **logistic system** is simply a set of rules specifying how to **change** one string of symbols (**antecedent**) into another string of symbols (**consequent**). This leads to:

The Word Problem for Semigroups

(1) $\forall x \forall y [x\ 1 = x = 1\ x]$

(2) $\forall x\ \forall y\ \forall z [x\ (y\ z) = (x\ y)\ z]$

Problem: Determine the provability of

$A_0 = B_0 \ \&\ A_1 = B_1 \ \&\ \ldots\ \&\ A_{n-1} = B_{n-1} \Longrightarrow A_n = B_n$.

Schönfinkel-Curry's Solution

Schönfinkel in 1924 and then Curry in 1929, both at Göttingen, began the study of **combinators**, which were quickly connected with Church's **λ-calculus** of 1932. From them — with hindsight — we get:

Another Undecidable Theory

(1) $\forall x\, \forall y\, [\, \mathbf{K}(x)(y) = x\,]$

(2) $\forall x\, \forall y\, \forall z\, [\, \mathbf{S}(x)(y)(z) = x(z)(y(z))\,]$

(3) $\neg\ \mathbf{K} = \mathbf{S}$

Problem: Determine the provability of $T = \underline{\mathbf{0}}$.

The only problem with this theory is that you either need **models** or something like the

Church-Rosser Theorem

to know it is **consistent**. A weaker theory of **deterministic reduction** can be given a fairly short axiomatization and then be proved consistent by much simpler means.

McCarthy, LISP, & λ-Calculus

LISP History according to McCarthy's memory in 1978. Presented at the ACM SIGPLAN History of Programming Languages Conference, June 1-3, 1978. It was pubished in **History of Programming Languages**, edited by Richard Wexelblat, Academic Press 1981. ***Two quotations:***

I spent the summer of 1958 at the IBM Information Research Department at the invitation of Nathaniel Rochester and chose differentiating algebraic expressions as a sample problem. It led to the following innovations beyond the FORTRAN List Processing Language:

• • • •

(c) To use functions as arguments, one needs a notation for functions, and it seemed natural to use the λ-notation of Church (1941). I didn't understand the rest of his book, so I wasn't tempted to try to implement his more general mechanism for defining functions. Church used higher-order functionals instead of using conditional expressions. Conditional expressions are much more readily implemented on computers.

• • • •

Logical completeness required that the notation used to express functions used as functional arguments be extended to provide for recursive functions, and the LABEL notation was invented by Nathaniel Rochester for that purpose. D. M. R. Park pointed out that LABEL was logically unnecessary since the result could be achieved using only λ — by a construction analogous to Church's Y-operator, albeit in a more complicated way.

Other key McCarthy publications:

Recursive Functions of Symbolic Expressions and their Computation by Machine (Part I). The original paper on LISP from **CACM**, April 1960. Part II, which never appeared, was to have had some Lisp programs for algebraic computation.

A Basis for a Mathematical Theory of Computation, first given in 1961, was published by North-Holland in 1963 in **Computer Programming and Formal Systems**, edited by P. Braffort and D. Hirschberg.

Towards a Mathematical Science of Computation, IFIPS 1962 extends the results of the previous paper. Perhaps the first mention and use of ***abstract syntax***.

Correctness of a Compiler for Arithmetic Expressions with James Painter. May have been the first proof of ***correctness of a compiler***. Abstract syntax and Lisp-style recursive definitions kept the paper short.

An HTML site concerning Lisp history can be found at:

http://www8.informatik.uni-erlangen.de/html/lisp-enter.html

A Closing Thought from Robert Harper

For me, I think it is important to stress the ***overwhelming influence*** of the λ-calculus among all other models of computation:

- It codifies not only computation, but also the basic principles of ***human reason*** (natural deduction).
- Moreover, it was ***born fully formed***, and is directly and immediately relevant to this day, rather than something that collects dust on the shelf.

Admittedly Turing's model had the advantage of being ***explicitly psychologically motivated***, but on the other hand Church focused on one of the greatest achievements of the human mind, ***the concept of a variable*** (= reasoning under hypotheses). Church saw that this was central, and time has born out the significance of his insight.

By contrast, no one cares one bit about the ***details*** of a Turing Machine; for, it fails to address the central issue of ***modularity*** (logical consequence), which is so important in programming and reasoning. And it does not extend to ***higher-order computation*** in anything like a natural or smooth way.

λ CONQUERS ALL!

Perhaps my good friend and colleague has spoken a little too strongly here, as Turing Machines have had many applications, say in Complexity Theory. But the study of Programming Languages does not seem to need them today.