THEORY AND MODELS OF LAMBDA CALCULUS: UNTYPED AND TYPED

Session 0: *Some History, Some People*

Dana S. Scott
University Professor Emeritus
Carnegie Mellon University

Jeremy G. Siek
Associate Professor of Computer Science
Indiana University, Bloomington

Leap Workshop
LambdaConf 2018
Boulder, Colorado

The Key Questions

Is it possible to have a consistent
type-free theory of functions,
where no difference is made between
operators and ***arguments*** ?

✻ ✻ ✻

And if so, what use is it?

And how would types be appropriate?

But, first, where does this all come from?

Lambda-calculus and Combinators in the 20th Century

By Felice Cardone and J. Roger Hindley.
In: "Handbook of the History of Logic, Vol. 5."
Dov M. Gabbay and John Woods, editors,
Elsevier Publishing Co., 2009, pp. 723–817.

The formal systems that are nowadays called lambda-calculus and combinatory logic were both invented in the 1920s, and their aim was to describe the most basic properties of function-abstraction, application, and substitution in a very general setting.

In lambda-calculus the concept of abstraction was taken as primitive, but in combinatory logic it was defined in terms of certain primitive operators called basic combinators. Today they are used extensively in higher-order logic and computing.

Seen in outline, the history splits into three main periods:
first, several years of intensive and very fruitful study in the 1920s and ’30s;
next, a middle period of nearly 30 years of relative quiet; ***then*** in the late 1960s an upsurge of activity stimulated by developments in higher-order function theory, by connections with programming languages, and by new technical discoveries.

Moses Iljitsch Schönfinkel

Born: 9 September 1886, Dniepropetrovsk, Ukraine

Died: ~ 1942, Moscow

A student of Hilbert's in Göttingen, he presented a report in December 1920 to the Mathematical Society in Göttingen on a new type of formal logic based on the concept of a generalized function whose argument is also a function.

Moses Schönfinkel, “Ueber die Bausteine der mathematischen Logik”, *Mathematische Annalen*, vol. 92 (1924), pp. 305–316.

An English translation appears as
“On the building blocks of mathematical logic.”
In: “From Frege to Gödel”, Jean van Heijenoort (editor),
Harvard University Press, 1967, pp. 355–366.

Haskell Brooks Curry

Born: 12 September 1900, Millis, MA

Died: 1 September 1982, State College, PA

Undergraduate at Harvard; Doctorate from Göttingen in 1930 for a thesis under Hilbert. **Thesis:** “Grundlagen der kombinatorischen Logik.” He taught at Harvard, Princeton, then for 35 years at Pennsylvania State University; during WW II he did research in applied physics at Johns Hopkins. His theory of combinators proved to be equivalent to λ-calculus, and he interacted closely with Church and his students. In 1966, after retirement from Penn State, he held a chair of mathematics at Amsterdam for four years. Curry's main work was in mathematical logic and the theory of formal systems and in logical calculi using inferential rules.

Books: “Combinatory Logic” (vol. 1, 1958, and vol. 2, 1972) and “Foundations of Mathematical Logic” (1963).

Alonzo Church

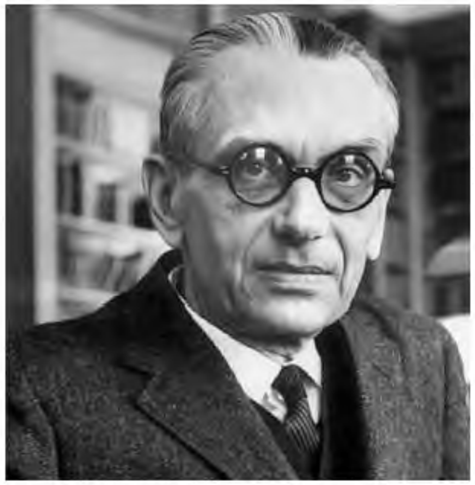
Born: 14 June 1903, Washington, DC

Died: 11 August 1995, Hudson, OH

B.S. Princeton 1924, Ph.D. 1927 under Veblen. **Thesis:** “Alternatives to Zermelo's Assumption.” He spent a year at Harvard, then half a year at Göttingen and half a year at Amsterdam where he worked with Brouwer. He returned to the USA becoming professor of mathematics at Princeton in 1929, a post he held until 1967 when he became professor of mathematics and philosophy at UCLA. He had 31 doctoral students. He created the λ-calculus in the 1930s but perhaps is best remembered for Church's Theorem (1936): *There is no decision procedure for the full predicate calculus.* A founder of the *Journal of Symbolic Logic* (1936) he remained editor of the section on reviews until 1979.

Books: “The Calculi of Lambda Conversion,” 1941.
“Introduction to Mathematical Logic”, 1956.

Kurt Friedrich Gödel



Born: 28 April 1906, Brünn,
Austria-Hungary
Died: 14 January 1978, Princeton

Gödel published his incompleteness theorems in 1931 when he was 25 years old, one year after finishing his doctorate at the University of Vienna. The ***first*** incompleteness theorem states that for any self-consistent recursive axiomatic system powerful enough to describe the arithmetic of the natural numbers (for example Peano arithmetic), there are true propositions about the naturals that cannot be proved from the axioms. The ***second*** shows that such a system cannot prove its own consistency. He also showed that neither the ***Axiom of Choice*** nor the ***Continuum Hypothesis*** can be disproved from the accepted axioms of set theory, assuming these axioms are consistent.

Stephen Cole Kleene

Born: 5 January 1909, Hartford, CN

Died: 25 January 1994, Madison, WI

First degree, Amherst College; Ph.D., Princeton 1934, under Church. **Thesis:** "A Theory of Positive Integers in Formal Logic." He taught at Princeton until he joined the University of Wisconsin at Madison in 1935, becoming full professor in 1948. He remained on the staff there until he retired in 1979. His research was on the theory of algorithms and recursive functions, developing the field of ***Recursion Theory*** along with Church, Gödel, Turing and others. He also contributed to ***Mathematical Intuitionism*** founded by Brouwer. His long-standing work on recursion theory helped to provide the foundations of theoretical computer science.

Books: "Introduction to Metamathematics" (1952) and "Mathematical Logic" (1967).

J. Barkley Rosser

Born: 6 December 1907,
Jacksonville, FL
Died: 5 September 1989, Madison

B.S.1929 and M.S. 1931, University of Florida. Ph.D. Princeton 1934 under Church. **Thesis:** "A mathematical logic without variables." He taught at Princeton, Harvard, and Cornell and spent the latter part of his career at the University of Wisconsin, continuing to lecture well into his late 70s. He served as president of the *Association for Symbolic Logic* and the *Society of Industrial and Applied Mathematics*; was a member of the space vehicle panel for the Apollo project; and helped develop the Polaris missile. His areas of expertise include symbolic logic, ballistics, rocket development, and numerical analysis.

Books: "Logic for Mathematicians" (1953) and
"Simplified Independence Proofs" (1969).

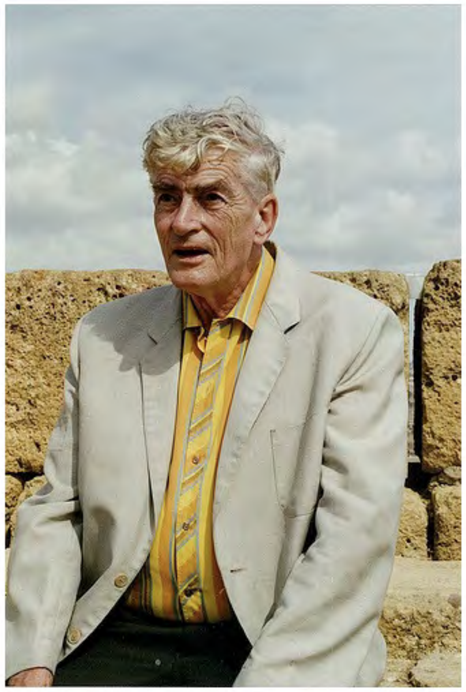
Alan Mathison Turing

Born: 23 June 1912, London, England

Died: 7 June 1954, Wilmslow, Cheshire, England

B.A. from King's College, Cambridge, 1934, and Ph.D. from Princeton, 1938, under Church. **Thesis:** "Systems of Logic Based on Ordinals." An English mathematician, logician, and cryptographer, he was awarded an OBE, FRS, and is often considered to be the father of modern computer science. He provided an influential formalization of the concept of the algorithm *via* computation with a *Turing Machine*, formulating the now widely accepted "Turing" version of the Church-Turing thesis: Any practical computing model has either the equivalent or a subset of the capabilities of a Turing machine. Using earlier work of Kleene, he proved Turing computability equivalent to Church's λ-definability. Later, his *Turing Test* made a significant – and characteristically provocative – contribution to the debate regarding *Artificial Intelligence*: Whether it will ever be possible to say that a machine is conscious and can think.

Robin Oliver Gandy

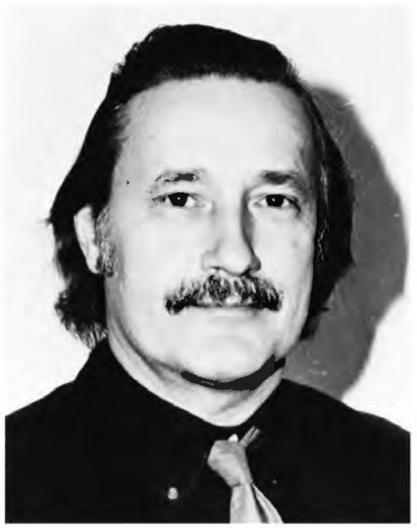


Born: 23 September 1919, Peppard, Oxon., UK.
Died: 20 November 1995, Oxford, UK.

Ph.D., Kings College Cambridge, 1953, under Turing. **Thesis:** "On axiomatic systems in Mathematics and theories in Physics." He later became a key contributor to the development of *Recursive Function Theory*.

He held positions at the University of Leicester, the University of Leeds, and the University of Manchester, was a visiting associate professor at Stanford University from 1966 to 1967, and at University of California, Los Angeles in 1968. In 1969, he moved to Wolfson College, Oxford, where he became *Reader in Mathematical Logic* until his retirement in 1986. He supervised 26 Ph.D. students and has 126 descendants.

Christopher Strachey



Born: 16 November 1916, Hampstead, London
Died: 18 May 1975, Oxford, England

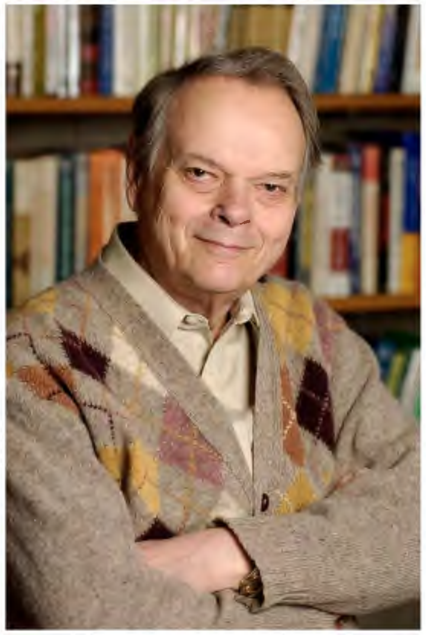
He passed the Tripos, lower second, at King's College, Cambridge in 1938. His professional experience was varied: physicist, 1938-1945; physics and mathematics school master, 1945-1949; Master, Harrow School, 1949-1952; technical officer, 1952- 1959; private consultant 1959-1966; programmer, University Mathematical Laboratory, Cambridge, 1962-1966; founder, Programming Research Group, Oxford University, 1966-1975, and finally Oxford University's first Professor of Computer Science from 1971. He was an early proposer of a form of time-sharing in 1959. He worked on the design and understanding of programming languages with Peter Landin, especially in the use of λ-calculus, and later, in collaboration with Dana Scott, the development of denotational semantics.

Corrado Böhm

Born: 17 January 1923, Milan, Italy
Died: 23 October 2017, Rome, Italy

Ph.D. at the ETH, Zurich, under Paul Bernays, where he describes for the first time a full meta-circular compiler, that is, a translation mechanism of a programming language written in that same language. He is known especially for his contributions to the theory of structured programming, constructive mathematics, combinatory logic, λ-calculus, and the semantics and implementation of functional programming languages. His most influential contribution is the so-called *Structured Program Theorem*, published in 1966 together with Giuseppe Jacopini. Regarding the λ-Calculus, he proved *Böhm's Theorem*, an important separation theorem between normal forms. Together with Alessandro Berarducci, he demonstrated an isomorphism between the strictly-positive algebraic data types and polymorphic λ-terms, otherwise known as *Böhm–Berarducci encoding*. He was a Professor Emeritus, University of Rome "La Sapienza".

John Reynolds



Born: 1 June 1935, Glen Ellyn, IL
Died: 28 April 2013, Pittsburgh, PA

B.S. Purdue University 1956; Ph.D. Theoretical Physics, Harvard University, 1961. **Thesis:** "Surface Properties of Nuclear Matter." He did fundamental and far-sighted research on programming languages: in the areas of semantics, specifications, language design, logics and proof methodology. He is also noted for the co-invention of polymorphic λ-calculus and separation logic. He was Professor of Information Science at Syracuse University, 1970-1986. From then until his death he was Professor of Computer Science at Carnegie Mellon University.

Books: "The Craft of Programming", 1981, and "Theories of Programming Languages", 1998.

Nicolaas Govert de Bruijn

Born: 9 July 1918, The Hague
Died: 17 February 2012, Nuenen

He received an MA in Mathematics at Leiden,1941, and a PhD from Vrije Universiteit Amsterdam, 1943. He was Professor of Mathematics, University of Amsterdam,1952-1960; then Professor of Mathematics, Technical University Eindhoven, 1960-1984. Made a member of the Royal Netherlands Academy of Arts and Sciences, 1957, and Knighted with the Order of the Netherlands Lion. His computer-based type system *Automath* for automated proof checking dates from 1967, and it has been influential for developing constructive logic, dependent type theory, and logical frameworks.

Nicolaas Govert de Bruijn. "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem." Indagationes Mathematicae, vol. 75 (1972), pp. 381-392.

David Michael Ritchie Park

Born: 12 September 1935, Stockton-on-Tees, UK
Died: 29 September 1990, Auvergne, France.

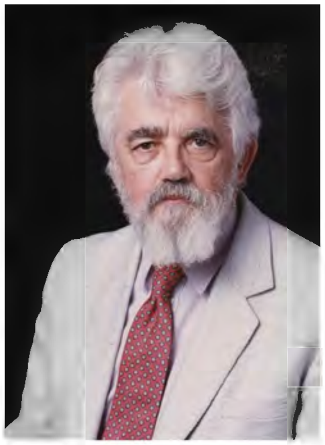
A pioneer in computer science, he was considered to be the senior theoretical computer scientist in Britain. He had worked on the first implementation of LISP and became an authority on the topics of fairness, program schemas and bisimulation in concurrent computing. After an undergraduate degree at Oxford he went to MIT and obtained a PhD in model theory under Hartley Rogers. In the late 50s John McCarthy developed his theory of computation and his novel list-processing language, LISP. Park was one of the authors of the report ***LISP I***. His later work on program schemas and fixed-point theory, made important contributions to the early theory of computer science. He returned to the UK in 1964, first to the Mathematical Laboratory in Cambridge, then to Christopher Strachey's new Programming Research Group at Oxford. Moving to Warwick in 1968, David was one of the earliest members of the Computer Science Department and put much of his professional work into helping to build it up to its present position.

Arthur John Robin Gorell Milner

Born: 13 January 1934, Yealmpton, Plymouth, UK
Died: 20 March 2010, Cambridge, UK

Born into a military family, awarded a scholarship to Eton College in 1947, he subsequently served in the Royal Engineers. Graduating from King's College, Cambridge, in 1957, he worked first as a school teacher then as a programmer at Ferranti. He entered academia at City University, London, then Swansea University, then Stanford University, and from 1973 at the University of Edinburgh as co-founder of the ***Laboratory for Foundations of Computer Science*** (LFCS). Back as professor at Cambridge in 1995, he headed the ***Computer Laboratory.*** From 2009, he had an advanced research fellowship and a part-time chair at Edinburgh. His three major contributions to computer science were: (i) LCF, one of the first tools for automated theorem proving; (ii) the programming language ML, the first language with polymorphic type inference and type-safe exception handling; and (iii) a theoretical framework for analyzing concurrent systems. He was a Fellow of the Royal Society and a Fellow of the British Computer Society, and he received the ACM Turing Award in 1991.

John McCarthy



Born: 4 September 1927,
Boston, MA
Died: 24 October 2011,
Stanford, CA

McCarthy graduated from Belmont High School two years early and was accepted into Caltech in 1944. Having taught himself college math, he was able to skip two years. Suspended for failure to attend PE, he then served in the Army and was readmitted for a Mathematics B.S. in 1948. A lecture by John von Neumann inspired his future endeavors. Initially doing graduate studies at Caltech, he moved to Princeton and received a Ph.D. in Mathematics in 1951 under Lefschetz. After short-term appointments at Princeton and Stanford, he became an assistant professor at Dartmouth in 1955, where, with others, he organized the first AI summer conference in 1956. McCarthy moved to MIT as a research fellow in the autumn of 1956 where LISP was developed. In 1962, McCarthy became a full professor at Stanford, where he remained until his retirement in 2000.

THEORY AND MODELS OF LAMBDA CALCULUS: UNTYPED AND TYPED

Session 1: *Introduction to Combinators and Lambda Calculus*

Dana S. Scott
University Professor Emeritus
Carnegie Mellon University

Jeremy G. Siek
Associate Professor of Computer Science
Indiana University, Bloomington

Leap Workshop
LambdaConf 2018
Boulder, Colorado

The Key Questions

Is it possible to have a consistent
type-free theory of functions,
where no difference is made between
operators and ***arguments*** ?

❇ ❇ ❇

And if so, what use is it?

And how would types be appropriate?

Combinatory Algebra

Constants:	**S, K, I**
Variables:	`x, y, z, ...`
Combinations:	expressions built up from constants and variables using a binary ***application operation*** `M(N)`.
Combinators:	combinations ***without*** variables.
Interpretation:	A combination `F(X)` is meant to indicate the ***evaluation*** of a function `F` at an argument `X`. A combinator `C` applied to a list of combinations, such as $C(M_0)(M_1)...(M_{(n-1)})$, is meant to give us a ***way*** of making a ***compound combination*** from a number of given expressions.

Note: When we construct models, we will add to these logical combinators additional basic arithmetic combinators.

The Combinatory Axioms

$$\exists x,y.[x \neq y]$$

$$\mathbf{K}(x)(y) = x$$

$$\mathbf{I}(x) = x$$

$$\mathbf{S}(x)(y)(z) = x(z)(y(z))$$

$$\forall z.[x(z) = y(z)] \Rightarrow x = y$$

Note: In our first model, the last axiom will hold only for certain special elements x,y.

Easy Exercises: (1) Prove: $\mathbf{S}(\mathbf{K})(\mathbf{K}) = \mathbf{I}$.

(2) Prove: $\mathbf{S} \neq \mathbf{K}$. (A better axiom?)

(3) Prove: $\mathbf{S}(\mathbf{K})(\mathbf{K}(\mathbf{I}))(x)(y) = x(y)$.

Eliminating Variables

Metatheorem: Let $\mathtt{M}$ be a given combination with all variables in the set $\{\mathtt{x}_0,\ \mathtt{x}_1,\ \mathtt{x}_2, \ldots, \mathtt{x}_{(n-1)}\}$. Then we can find a combinator $\mathtt{C}$ for which it is provable that

$$\mathtt{C}(\mathtt{x}_0)(\mathtt{x}_1)(\mathtt{x}_2)\ldots(\mathtt{x}_{(n-1)}) = \mathtt{M}.$$

Proof Idea: Given one variable $\mathtt{x}$, define a mapping $\mathtt{M} \longmapsto (\lambda \mathtt{x}.\mathtt{M})$ by structural recursion on combinations by:

$$\begin{array}{lll} (\lambda \mathtt{x}.\mathtt{M}) = \mathbf{K}(\mathtt{M}) & & \text{if } \mathtt{M} \text{ does not contain } \mathtt{x}\,; \\ (\lambda \mathtt{x}.\mathtt{M}) = \mathbf{I} & & \text{if } \mathtt{M} \text{ is the variable } \mathtt{x}\,; \\ (\lambda \mathtt{x}.\mathtt{M}) = \mathbf{S}((\lambda \mathtt{x}.\mathtt{P}))((\lambda \mathtt{x}.\mathtt{Q})) & & \text{if othrewise } \mathtt{M} = \mathtt{P}(\mathtt{Q}). \end{array}$$

Then, $\mathtt{C} = (\lambda \mathtt{x}.\mathtt{M})$ is a combination not containing $\mathtt{x}$ for which $\mathtt{C}(\mathtt{x}) = \mathtt{M}$ is provable.

Church Numerals

$$\underline{n}(f)(x) = f(f(f(\ldots f(x)\ldots)))$$

n–fold iteration

Beginning of arithmetic by Church and Rosser:

$$\underline{0} = \lambda f.\lambda x.x$$

$$\underline{n+1} = \lambda f.\lambda x.f(\underline{n}(f)(x))$$

$$\underline{n+m} = \lambda f.\lambda x.\underline{n}(f)(\underline{m}(f)(x))$$

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

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

Key problem solved in Kleene's Ph.D.:

How to define $\underline{n-1}$ and all primitive-recursive functions.

Kleene Arithmetic

Paring by Combinators:

$$\textbf{pair} = \lambda x.\lambda y.\lambda f.f(x)(y)$$
$$\textbf{fst} = \lambda p.p(\lambda x.\lambda y.x)$$
$$\textbf{snd} = \lambda p.p(\lambda x.\lambda y.y)$$

Defining Predecessor:

$$\textbf{succ} = \lambda n.\lambda f.\lambda x.f(n(f)(x))$$
$$\textbf{shft} = \lambda s.\lambda p.\textbf{pair}(s(\textbf{fst}(p)))(\textbf{fst}(p))$$
$$\textbf{pred} = \lambda n.\textbf{snd}(n(\textbf{shft}(\textbf{succ}))(\textbf{pair}(\underline{0})(\underline{0})))$$

Why It Works:

0	**1**	**2**	**3**	**...**	**n-1**	**n**	**n+1**	**...**
0	**0**	**1**	**2**	**...**	**n-2**	**n-1**	**n**	**...**

Reviewing Recursion

Some history: ***Primitive recursive arithmetic*** was first proposed by Thoralf Skolem in 1923. Our current terminology comes from Rózsa Péter in 1934, after Ackermann had found in 1928 a computable function which was ***not*** primitive recursive, an event which prompted the need to rename what until then were simply called recursive functions.

Definition. ***Primitive recursive functions*** are those generated from ***constant*** functions, ***projection*** functions, and the ***successor*** function by ***composition*** and ***simple recursion***:

```
h(0, x) = f(x)

h(n+1, x) = g(n, x, h(n, x)),
```

where `f` and `g` are previously obtained functions.

Recursively enumerable sets (RE) are those of the form

$\{ m \mid \exists n.\, p(n) = m+1 \}$, with `p` primitive recursive.

Programming Primitive Recursion

$$\begin{aligned} h(\underline{0})(x) &= f(x) \\ h(\underline{n+1})(x) &= g(\underline{n})(x)(h(\underline{n})(x)) \end{aligned}$$

Finding a Combination:

$$\textbf{step} = \lambda x.\lambda p.\textbf{pair}(\textbf{succ}(\textbf{fst}(p)))(g(\textbf{fst}(p))(x)(\textbf{snd}(p)))$$

$$h = \lambda n.\lambda x.\textbf{snd}(n(\textbf{step}(x))(\textbf{pair}(\underline{0})(f(x))))$$

Exercises: **(1)** Why does this work?

(2) Explain how to program this recursion:

$$\begin{aligned} k(\underline{0})(x) &= f(x) \\ k(\underline{1})(x) &= g(x) \\ k(\underline{n+2})(x) &= h(\underline{n})(x)(k(\underline{n})(x))(k(\underline{n+1})(x)) \end{aligned}$$

(3) Use combinators to eliminate mention of the extra variable.

The Fixed-Point Operator

Definition: $\mathbf{Y} = \lambda f.(\lambda x.f(x(x)))(\lambda x.f(x(x)))$

Theorem: $\mathbf{Y}(f) = f(\mathbf{Y}(f))$

Exercises:

(1) Let $\mathbf{L} = \mathbf{Y}(\mathbf{K})$. Show $\mathbf{L}(\mathbf{L}) = \mathbf{L}$.

(2) Does $\mathbf{L} = \mathbf{K}$?

Definitions:

$\mathbf{test} = \lambda n.\lambda u.\lambda v.\mathbf{snd}(n(\mathbf{shft}(\lambda x.x))(\mathbf{pair}(v)(u)))$

$\mathbf{mult} = \lambda n.\lambda m.\lambda f.n(m(f))$

$\mathbf{fact} = \mathbf{Y}(\lambda f.\lambda n.\mathbf{test}(n)(1)(\mathbf{mult}(n)(f(\mathbf{pred}(n)))))$

(3) Prove: $\mathbf{fact}(\underline{n}) = \underline{n!}$.

Metatheorem: Combinatory Algebra, as a first-order theory, is essentially undecidable.

Axioms for λ-Calculus

Constants:	none
Variables:	`x, y, z, ...`
Terms:	expressions built up from variables using a binary ***application operation*** `M(N)` and a variable-binding operation of **λ**-***abstraction*** `(λx.M)`.
Substitutions:	`M[N/x]` is defined for each variable `x` by replacing all ***free*** occurrences of `x` in `M` by a copy of `N` — ***provided that*** no free variables in `N` get captured by a variable binder in `M`.

Axioms: *(provided the substitutions are defined)*

```
    (λx.M)  =  (λy.M[y/x])
(λx.M)(N)   =  M[N/x]
(λx.f(x))   =  f
```

Metatheorem: With suitable combinator definitions the two first-order theories are logically equivalent.

THEORY AND MODELS OF LAMBDA CALCULUS: UNTYPED AND TYPED

Session 1: *Introduction to Combinators and Lambda Calculus*

Dana S. Scott
University Professor Emeritus
Carnegie Mellon University

Jeremy G. Siek
Associate Professor of Computer Science
Indiana University, Bloomington

Leap Workshop
LambdaConf 2018
Boulder, Colorado

The Key Questions

Is it possible to have a consistent
type-free theory of functions,
where no difference is made between
operators and ***arguments*** ?

❊ ❊ ❊

And if so, what use is it?

And how would types be appropriate?

Combinatory Algebra

Constants:	**S, K, I**
Variables:	`x, y, z, ...`
Combinations:	expressions built up from constants and variables using a binary ***application operation*** `M(N)`.
Combinators:	combinations ***without*** variables.

Interpretation: A combination `F(X)` is meant to indicate the ***evaluation*** of a function `F` at an argument `X`.

A combinator `C` applied to a list of combinations, such as $C(M_0)(M_1)...(M_{(n-1)})$, is meant to give us a ***way*** of making a ***compound combination*** from a number of given expressions.

Note: When we construct models, we will add to these logical combinators additional basic arithmetic combinators.

The Combinatory Axioms

$$\exists x,y.[x \neq y]$$

$$\mathbf{K}(x)(y) = x$$

$$\mathbf{I}(x) = x$$

$$\mathbf{S}(x)(y)(z) = x(z)(y(z))$$

$$\forall z.[x(z) = y(z)] \Rightarrow x = y$$

Note: In our first model, the last axiom will hold only for certain special elements x,y.

Easy Exercises: (1) Prove: $\mathbf{S}(\mathbf{K})(\mathbf{K}) = \mathbf{I}$.

(2) Prove: $\mathbf{S} \neq \mathbf{K}$. (A better axiom?)

(3) Prove: $\mathbf{S}(\mathbf{K})(\mathbf{K}(\mathbf{I}))(x)(y) = x(y)$.

Eliminating Variables

Metatheorem: Let $\mathtt{M}$ be a given combination with all variables in the set $\{\mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_{(n-1)}\}$. Then we can find a combinator $\mathtt{C}$ for which it is provable that

$$\mathtt{C(x_0)(x_1)(x_2)\ldots(x_{(n-1)})} = \mathtt{M}.$$

Proof Idea: Given one variable $\mathtt{x}$, define a mapping $\mathtt{M} \longmapsto (\lambda\mathtt{x.M})$ by structural recursion on combinations by:

$(\lambda\mathtt{x.M}) = \mathbf{K}(\mathtt{M})$	if $\mathtt{M}$ does not contain $\mathtt{x}$;
$(\lambda\mathtt{x.M}) = \mathbf{I}$	if $\mathtt{M}$ is the variable $\mathtt{x}$;
$(\lambda\mathtt{x.M}) = \mathbf{S}((\lambda\mathtt{x.P}))((\lambda\mathtt{x.Q}))$	if othrewise $\mathtt{M} = \mathtt{P(Q)}$.

Then, $\mathtt{C} = (\lambda\mathtt{x.M})$ is a combination not containing $\mathtt{x}$ for which $\mathtt{C(x)} = \mathtt{M}$ is provable.

Church Numerals

$\underline{n}(f)(x) = f(f(f(\ldots f(x)\ldots)))$

n-fold iteration

Beginning of arithmetic by Church and Rosser:

$$\begin{aligned}
\underline{0} &= \lambda f.\lambda x.x \\
\underline{n+1} &= \lambda f.\lambda x.f(\underline{n}(f)(x)) \\
\underline{n+m} &= \lambda f.\lambda x.\underline{n}(f)(\underline{m}(f)(x)) \\
\underline{n\times m} &= \lambda f.\underline{n}(\underline{m}(f)) \\
\underline{m^n} &= \underline{n}(\underline{m})
\end{aligned}$$

Key problem solved in Kleene's Ph.D.:

How to define $\underline{n-1}$ and all primitive-recursive functions.

Kleene Arithmetic

Paring by Combinators:

pair = λx.λy.λf.f(x)(y)
fst = λp.p(λx.λy.x)
snd = λp.p(λx.λy.y)

Defining Predecessor:

succ = λn.λf.λx.f(n(f)(x))
shft = λs.λp.**pair**(s(**fst**(p)))(**fst**(p))
pred = λn.**snd**(n(**shft**(**succ**))(**pair**($\underline{0}$)($\underline{0}$)))

Why It Works:

0	1	2	3	...	n-1	n	n+1	...
0	0	1	2	...	n-2	n-1	n	...

Reviewing Recursion

Some history: ***Primitive recursive arithmetic*** was first proposed by Thoralf Skolem in 1923. Our current terminology comes from Rózsa Péter in 1934, after Ackermann had found in 1928 a computable function which was ***not*** primitive recursive, an event which prompted the need to rename what until then were simply called recursive functions.

Definition. ***Primitive recursive functions*** are those generated from ***constant*** functions, ***projection*** functions, and the ***successor*** function by ***composition*** and ***simple recursion***:

$$h(0, x) = f(x)$$

$$h(n+1, x) = g(n, x, h(n, x)),$$

where f and g are previously obtained functions.

Recursively enumerable sets (RE) are those of the form

$\{ m \mid \exists n. p(n) = m+1 \}$, with p primitive recursive.

Programming Primitive Recursion

$$h(\underline{0})(x) = f(x)$$
$$h(\underline{n+1})(x) = g(\underline{n})(x)(h(\underline{n})(x))$$

Finding a Combination:

$$\textbf{step} = \lambda x.\lambda p.\textbf{pair}(\textbf{succ}(\textbf{fst}(p)))(g(\textbf{fst}(p))(x)(\textbf{snd}(p)))$$
$$h = \lambda n.\lambda x.\textbf{snd}(n(\textbf{step}(x))(\textbf{pair}(\underline{0})(f(x))))$$

Exercises: **(1)** **Why does this work?**
(2) **Explain how to program this recursion:**

$$k(\underline{0})(x) = f(x)$$
$$k(\underline{1})(x) = g(x)$$
$$k(\underline{n+2})(x) = h(\underline{n})(x)(k(\underline{n})(x))(k(\underline{n+1})(x))$$

(3) **Use combinators to eliminate mention of the extra variable.**

The Fixed-Point Operator

Definition: $\mathbf{Y} = \lambda f.(\lambda x.f(x(x)))(\lambda x.f(x(x)))$

Theorem: $\mathbf{Y}(f) = f(\mathbf{Y}(f))$

Exercises:

(1) Let $\mathbf{L} = \mathbf{Y}(\mathbf{K})$. Show $\mathbf{L}(\mathbf{L}) = \mathbf{L}$.

(2) Does $\mathbf{L} = \mathbf{K}$?

Definitions:

$$\textbf{test} = \lambda n.\lambda u.\lambda v.\textbf{snd}(n(\textbf{shft}(\lambda x.x))(\textbf{pair}(v)(u)))$$

$$\textbf{mult} = \lambda n.\lambda m.\lambda f.n(m(f))$$

$$\textbf{fact} = \mathbf{Y}(\lambda f.\lambda n.\textbf{test}(n)(1)(\textbf{mult}(n)(f(\textbf{pred}(n)))))$$

(3) Prove: $\textbf{fact}(\underline{n}) = \underline{n!}$.

Metatheorem: Combinatory Algebra, as a first-order theory, is essentially undecidable.

Axioms for λ-Calculus

Constants:	none
Variables:	`x, y, z, ...`
Terms:	expressions built up from variables using a binary ***application operation*** `M(N)` and a variable-binding operation of **λ**-***abstraction*** `(λx.M)`.
Substitutions:	`M[N/x]` is defined for each variable `x` by replacing all ***free*** occurrences of `x` in `M` by a copy of `N` — ***provided that*** no free variables in `N` get captured by a variable binder in `M`.

Axioms: *(provided the substitutions are defined)*

```
   (λx.M)  =  (λy.M[y/x])
(λx.M)(N)  =  M[N/x]
(λx.f(x))  =  f
```

Metatheorem: With suitable combinator definitions the two first-order theories are logically equivalent.