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 $\{x_0, x_1, x_2, \ldots, x_{(n-1)}\}$. Then we can find a combinator $\mathtt{C}$ for which it is provable that

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

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

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

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

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:

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

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

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

(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)*

$$(\lambda x.M) = (\lambda y.M[y/x])$$

$$(\lambda x.M)(N) = M[N/x]$$

$$(\lambda x.f(x)) = f$$

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