Ordinals and Typed Lambda Calculus

Numeral Systems in the Lambda Calculus

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2018-2

Church Encoding

Booleans

We encoding the Booleans.

$$\mathsf{true} \coloneqq \lambda\, x\, y.\, x, \text{ where } \mathsf{true}\, M\, N =_\beta M.$$

$$\mathsf{false} \coloneqq \lambda\, x\, y.\, y, \text{ where } \mathsf{false}\, M\, N =_\beta N.$$

Church Encoding

Definition

For $n \in \mathbb{N}$ and $F, M \in \Delta$, we define

$$F^0\, M \coloneqq M,$$
$$F^{n+1}\, M \coloneqq F\,(F^n\, M).$$

Church Encoding

Definition

The **Church numerals**, denoted by c_n with $n \in \mathbb{N}$, encode the natural numbers.

One definition:

$$\mathsf{c}_n \coloneqq \lambda\, f\, x.\, f^n\, x,$$

Other definition:

$$\begin{aligned} \mathsf{c}_0 &\coloneqq \lambda\, f\, x.\, x, \\ \mathsf{succ} &\coloneqq \lambda\, n\, f\, x.\, f\, (n\, f\, x), \\ \mathsf{c}_{n+1} &\coloneqq \mathsf{succ}\, \mathsf{c}_n. \end{aligned}$$

Church Encoding

Example

β-nf's for some numerals.

$$
\begin{aligned}
\mathsf{c}_0 &=_\beta \lambda f\, x.\, x, \\
\mathsf{c}_1 &=_\beta \lambda f\, x.\, f\, x, \\
\mathsf{c}_2 &=_\beta \lambda f\, x.\, f\, (f\, x), \\
\mathsf{c}_3 &=_\beta \lambda f\, x.\, f\, (f\, (f\, x)).
\end{aligned}
$$

Whiteboard.

Church Encoding

Addition, multiplication and exponentiation

Encoding for the arithmetic operations.

$$\mathsf{add} := \lambda\, m\, n\, f\, x.\, m\, f\, (n\, f\, x), \text{ where} \qquad \mathsf{add}\ \mathsf{c}_m\ \mathsf{c}_n =_\beta \mathsf{c}_{m+n}.$$

$$\begin{aligned}\mathsf{mult} &:= \lambda\, m\, n\, f\, x.\, m\, (n\, f)\, x \\ &\equiv \lambda\, m\, n\, f.\, m\, (n\, f), \text{ where}\end{aligned} \qquad \mathsf{mult}\ \mathsf{c}_m\ \mathsf{c}_n =_\beta \mathsf{c}_{m\times n}.$$

$$\mathsf{exp} := \lambda\, m\, n\, f\, x.\, (n\, m)\, f\, x, \text{ where} \qquad \mathsf{exp}\ \mathsf{c}_m\ \mathsf{c}_n =_\beta \mathsf{c}_{m^n}.$$

Church Encoding

Testing for zero

A combinator for testing for zero is defined by

$$\mathsf{isZero} \coloneqq \lambda n.\, n\,(\lambda x.\mathsf{false})\,\mathsf{true},$$

where

$$\begin{aligned} \mathsf{isZero}\,\mathsf{c}_0 &=_\beta \mathsf{true}, \\ \mathsf{isZero}\,\mathsf{c}_{n+1} &=_\beta \mathsf{false}. \end{aligned}$$

Church Encoding

Predecessor

A predecessor combinator is defined by

$$\mathsf{pred} := \lambda\, n\, f\, x.\, n\, (\lambda\, g\, h.\, h\, (g\, f))\, (\lambda u.x)\, (\lambda u.u),$$

where

$$\begin{aligned} \mathsf{pred}\, \mathsf{c}_0 &=_\beta \mathsf{c}_0, \\ \mathsf{pred}\, \mathsf{c}_{n+1} &=_\beta \mathsf{c}_n. \end{aligned}$$

Church Encoding

Recursion

The factorial function is an example of recursive functions on natural numbers following the schemata

$$\begin{aligned} f &: \mathbb{N} \to A \\ f\,0 &= a \\ f\,(\mathrm{S}\,n) &= g\,n\,(f\,n) \end{aligned}$$

where A is a set, $a \in A$, $g : \mathbb{N} \to A \to A$ and S is the successor function.

Remark

A reader familiarised with recursion theory will identify the primitive recursion schemata.

Church Encoding

A recursor for natural numbers

Let A be a set. From the previous schemata for (primitive) recursive functions, we define a recursor.*

$$\begin{aligned} \mathrm{rec} &: (\mathbb{N} \to A \to A) \to A \to \mathbb{N} \to A \\ \mathrm{rec}\, f\, a\, 0 &= a \\ \mathrm{rec}\, f\, a\, (\mathrm{S}\, n) &= f\, n\, (\mathrm{rec}\, f\, a\, n) \end{aligned}$$

*A higher-order function which allow define recursive functions.

Church Encoding

Recursor

Since rec is also a recursive function, we can define a recursor for numerals using fix.

$$\begin{aligned} \mathsf{h} &\coloneqq \lambda\, r\, f\, a\, n.\, (\mathsf{isZero}\, n)\, a\, (f\, (\mathsf{pred}\, n)\, (r\, f\, a\, (\mathsf{pred}\, n)))), \\ \mathsf{rec} &\coloneqq \mathsf{fix}\, \mathsf{h}, \end{aligned}$$

where

$$\begin{aligned} \mathsf{rec}\, f\, a\, \mathsf{c}_0 &=_\beta a, \\ \mathsf{rec}\, f\, a\, \mathsf{c}_{n+1} &=_\beta f\, \mathsf{c}_n\, (\mathsf{rec}\, f\, a\, \mathsf{c}_n). \end{aligned}$$

Church Encoding

Recursor

Since rec is also a recursive function, we can define a recursor for numerals using fix.

$$\begin{aligned} \mathsf{h} &\coloneqq \lambda r\, f\, a\, n.\, (\mathsf{isZero}\, n)\, a\, (f\, (\mathsf{pred}\, n)\, (r\, f\, a\, (\mathsf{pred}\, n)))), \\ \mathsf{rec} &\coloneqq \mathsf{fix}\, \mathsf{h}, \end{aligned}$$

where

$$\begin{aligned} \mathsf{rec}\, f\, a\, \mathsf{c}_0 &=_\beta a, \\ \mathsf{rec}\, f\, a\, \mathsf{c}_{n+1} &=_\beta f\, \mathsf{c}_n\, (\mathsf{rec}\, f\, a\, \mathsf{c}_n). \end{aligned}$$

Question

What is it necessary for defining rec? A fixed-point combinator, (implicit) 'Boolean' case analysis, a test for zero and a predecessor function.

Church Encoding

Example

A combinator for the factorial function

$$\begin{aligned} \mathrm{fac} &: \mathbb{N} \to \mathbb{N} \\ \mathrm{fac}\,0 &= 1 \\ \mathrm{fac}\,(\mathrm{S}\,n) &= \mathrm{S}\,n \times \mathrm{fac}\,n \end{aligned}$$

is defined by

$$\mathsf{fac} \coloneqq \mathsf{rec}\,(\lambda\, x\, y.\, \mathsf{mult}\,(\mathsf{succ}\, x)\, y)\,\mathsf{c}_1.$$

Church Encoding

Definition

A **number-theoretic function** is a function whose signature is

$$\mathbb{N}^k \to \mathbb{N}, \text{ with } k \in \mathbb{N}.$$

Church Encoding

Definition

Let φ be a partial number-theoretic function $\varphi : \mathbb{N}^k \to \mathbb{N}$. The function φ is **λ-definable respect to the Church encoding** iff there exists a λ-term F such that for all $n_1, \ldots, n_k \in \mathbb{N}$,

$$\varphi(n_1, \ldots, n_k) = a \Rightarrow F\, \mathsf{c}_{n_1} \ldots \mathsf{c}_{n_k} =_\beta \mathsf{c}_a,$$
$$\varphi(n_1, \ldots, n_k) \text{ does not exits} \Rightarrow F\, \mathsf{c}_{n_1} \ldots \mathsf{c}_{n_k} \text{ has no } \beta\text{-nf.}$$

*See, e.g. [Barendregt (1981) 2004, Corollary 6.4.6] and [Hindley and Seldin 2008, Theorem 4.23].

Church Encoding

Definition

Let φ be a partial number-theoretic function $\varphi : \mathbb{N}^k \to \mathbb{N}$. The function φ is **λ-definable respect to the Church encoding** iff there exists a λ-term F such that for all $n_1, \ldots, n_k \in \mathbb{N}$,

$$\varphi(n_1, \ldots, n_k) = a \Rightarrow F\, \mathsf{c}_{n_1} \ldots \mathsf{c}_{n_k} =_\beta \mathsf{c}_a,$$
$$\varphi(n_1, \ldots, n_k) \text{ does not exits} \Rightarrow F\, \mathsf{c}_{n_1} \ldots \mathsf{c}_{n_k} \text{ has no } \beta\text{-nf.}$$

Theorem

The Turing-computable functions are λ-definable respect to the Church encoding.*

*See, e.g. [Barendregt (1981) 2004, Corollary 6.4.6] and [Hindley and Seldin 2008, Theorem 4.23].

Numeral Systems

Definition

A **numeral system** is a sequence of combinators

$$\mathsf{d} = \mathsf{d}_0, \mathsf{d}_1, \ldots$$

and combinators true_d, $\mathsf{false}_\mathsf{d}$, succ_d and $\mathsf{isZero}_\mathsf{d}$ such that for all $n \in \mathbb{N}$ [Barendregt (1981) 2004, Definition 6.4.1],

$$\begin{aligned} \mathsf{succ}_\mathsf{d}\, \mathsf{d}_n &=_\beta \mathsf{d}_{n+1}, \\ \mathsf{isZero}_\mathsf{d}\, \mathsf{d}_0 &=_\beta \mathsf{true}_\mathsf{d}, \\ \mathsf{isZero}_\mathsf{d}\, \mathsf{d}_{n+1} &=_\beta \mathsf{false}_\mathsf{d}. \end{aligned}$$

Numeral Systems

Definition

A **numeral system** is a sequence of combinators

$$\mathsf{d} = \mathsf{d}_0, \mathsf{d}_1, \ldots$$

and combinators true_d, $\mathsf{false}_\mathsf{d}$, succ_d and $\mathsf{isZero}_\mathsf{d}$ such that for all $n \in \mathbb{N}$ [Barendregt (1981) 2004, Definition 6.4.1],

$$\begin{aligned} \mathsf{succ}_\mathsf{d}\, \mathsf{d}_n &=_\beta \mathsf{d}_{n+1}, \\ \mathsf{isZero}_\mathsf{d}\, \mathsf{d}_0 &=_\beta \mathsf{true}_\mathsf{d}, \\ \mathsf{isZero}_\mathsf{d}\, \mathsf{d}_{n+1} &=_\beta \mathsf{false}_\mathsf{d}. \end{aligned}$$

Example

The Church numerals $\mathsf{c} = \mathsf{c}_0, \mathsf{c}_1, \ldots$ and the combinators true, false, succ and isZero are a numeral system.

Numeral Systems

Notation

We shall denote by $\mathtt{d}$ a numeral system $\mathtt{d}_0$, $\mathtt{d}_1$, ..., $\mathtt{true}_\mathtt{d}$, $\mathtt{false}_\mathtt{d}$, $\mathtt{succ}_\mathtt{d}$ and $\mathtt{isZero}_\mathtt{d}$.

Numeral Systems

Notation

We shall denote by d a numeral system d_0, d_1, ..., true_d, $\mathsf{false}_\mathsf{d}$, succ_d and $\mathsf{isZero}_\mathsf{d}$.

Lambda definable functions on a numeral system

Let d be a numeral system and let φ be a partial number-theoretic function $\varphi : \mathbb{N}^k \to \mathbb{N}$. The function φ is **$\boldsymbol{\lambda}$-definable respect to the numeral system** d iff there exists a λ-term F such that for all $n_1, \ldots, n_k \in \mathbb{N}$,

$$\varphi(n_1, \ldots, n_k) = a \Rightarrow F\,\mathsf{d}_{n_1} \ldots \mathsf{d}_{n_k} =_\beta \mathsf{d}_a,$$
$$\varphi(n_1, \ldots, n_k) \text{ does not exits} \Rightarrow F\,\mathsf{d}_{n_1} \ldots \mathsf{d}_{n_k} \text{ has no } \beta\text{-nf}.$$

Numeral Systems

Definition

A numeral system d is **adequate** iff all the Turing-computable functions are λ-definable with respect to d [Barendregt (1981) 2004, Definition 6.4.2ii].

Numeral Systems

Definition

A numeral system d is **adequate** iff all the Turing-computable functions are λ-definable with respect to d [Barendregt (1981) 2004, Definition 6.4.2ii].

Example

The Church numeral system is adequate.

Numeral Systems

Theorem

A numeral system d is adequate iff there exists a combinator $\mathsf{pred_d}$ such that for all $n \in \mathbb{N}$ [Barendregt (1981) 2004, Theorem 6.4.3],

$$\mathsf{pred_d}\ \mathsf{d}_{n+1} =_\beta \mathsf{d}_n.$$

Numeral Systems

Theorem

A numeral system d is adequate iff there exists a combinator $\mathsf{pred_d}$ such that for all $n \in \mathbb{N}$ [Barendregt (1981) 2004, Theorem 6.4.3],

$$\mathsf{pred_d}\ \mathsf{d}_{n+1} =_\beta \mathsf{d}_n.$$

Remark

There are various adequate numeral systems in the literature. See, e.g. Barendregt [(1981) 2004], Goldberg [2000] and Jansen [2013].

References


Barendregt, H. P. [1981] (2004). The Lambda Calculus. Its Syntax and Semantics. Revised edition, 6th impression. Vol. 103. Studies in Logic and the Foundations of Mathematics. Elsevier (cit. on pp. 15–18, 21–24).


Goldberg, Mayer (2000). An Adequate and Efficient Left-Associated Binary Numeral System in the λ-Calculus. Journal of Functional Programming 10.6, pp. 607–623. DOI: 10.1017/S095679680000380 (cit. on pp. 23, 24).


Hindley, J. Roger and Seldin, Jonathan P. (2008). Lambda-Calculus and Combinators. An Introduction. Cambridge University Press (cit. on pp. 15, 16).


Jansen, Jan Martin (2013). Programming in the λ-Calculus: From Church to Scott and Back. In: The Beauty of Functional Code. Ed. by Achten, Peter and Koopman, Pieter. Vol. 8106. Lecture Notes in Computer Science. Springer, pp. 168–180. DOI: 10.1007/978-3-642-40355-2_12 (cit. on pp. 23, 24).