

# Ordinals and Typed Lambda Calculus

## Definable Ordinals in the Simply Typed Lambda Calculus

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2018-2

(Last modification: 7th May 2025)

Alonzo Church (1903 – 1995)\*



---

\*Figures sources: [History of computers](#), [Wikipedia](#) and [MacTutor History of Mathematics](#).

## A FORMULATION OF THE SIMPLE THEORY OF TYPES

ALONZO CHURCH

The purpose of the present paper is to give a formulation of the simple theory of types<sup>1</sup> which incorporates certain features of the calculus of  $\lambda$ -conversion.<sup>2</sup> A complete incorporation of the calculus of  $\lambda$ -conversion into the theory of types is impossible if we require that  $\lambda x$  and juxtaposition shall retain their respective meanings as an abstraction operator and as denoting the application of function to argument. But the present partial incorporation has certain advantages from the point of view of type theory and is offered as being of interest on this basis (whatever may be thought of the finally satisfactory character of the theory of types as a foundation for logic and mathematics).

For features of the formulation which are not immediately connected with the incorporation of  $\lambda$ -conversion, we are heavily indebted to Whitehead and Russell,<sup>3</sup> Hilbert and Ackermann,<sup>4</sup> Hilbert and Bernays,<sup>5</sup> and to forerunners of these, as the reader familiar with the works in question will recognize.

**1. The hierarchy of types.** The class of *type symbols* is described by the rules that  $\iota$  and  $\sigma$  are each type symbols and that if  $\alpha$  and  $\beta$  are type symbols then  $(\alpha\beta)$  is a type symbol: it is the least class of symbols which contains the symbols  $\iota$  and  $\sigma$  and is closed under the operation of forming the symbol  $(\alpha\beta)$  from the symbols  $\alpha$  and  $\beta$ .

As exemplified in the statement just made, we shall use the Greek letters  $\alpha, \beta, \gamma$  to represent variable or undetermined type symbols. We shall abbreviate type symbols by omission of parentheses with the convention that association is to the left—so that, for instance,  $\sigma\iota$  will be an abbreviation for  $(\sigma\iota)$ ,  $\iota\iota$  for  $((\iota)\iota)$ ,  $\iota(\iota)$  for  $((\iota)(\iota))$ , etc. Moreover, we shall use  $\alpha'$  as an abbreviation for  $((\alpha\alpha)(\alpha\alpha))$ ,  $\alpha''$  as an abbreviation for  $((\alpha'\alpha')(\alpha'\alpha'))$ , etc.

The type symbols enter our formal theory only as subscripts upon variables and constants. In the interpretation of the theory it is intended that the

Received March 23, 1940.

# Simply Typed Lambda Calculus

---

## Convention

The simply typed  $\lambda$ -calculus will be denoted by  $\lambda^{\rightarrow}$ .

# Simply Typed Lambda Calculus

---

## Convention

The simply typed  $\lambda$ -calculus will be denoted by  $\lambda^{\rightarrow}$ .

## Definition

Let  $V$  a denumerable set of variables. The set of  **$\lambda$ -terms**, denoted by  $\Lambda$ , is inductively defined by

$x \in V \Rightarrow x \in \Lambda$	(variable)
$M, N \in \Lambda \Rightarrow (M N) \in \Lambda$	(application)
$M \in \Lambda, x \in V \Rightarrow (\lambda x.M) \in \Lambda$	( $\lambda$ -abstraction)

# Types

---

## Definition

Let  $B$  a set of base types (i.e. constant types). The **set of types** of  $\lambda^{\rightarrow}$ , denoted by  $\mathbb{T}$ , is inductively defined by

$$b \in B \Rightarrow b \in \mathbb{T} \quad (\text{constant types})$$

$$\sigma, \tau \in \mathbb{T} \Rightarrow (\sigma \rightarrow \tau) \in \mathbb{T} \quad (\text{function types})$$

## Remark

Usually, the set of types is defined by the abstract grammar

$$\begin{array}{l} \tau ::= b \in B \\ \quad | \tau \rightarrow \tau \end{array}$$

# Types

---

## Convention

Arbitrary types will be denoted by  $\sigma$  and  $\tau$ , with or without number-subscripts.

# Types

---

## Convention

Arbitrary types will be denoted by  $\sigma$  and  $\tau$ , with or without number-subscripts.

## Example

The simple theory of types of Church [Chu1940] only included two base types:

$\iota$ : the type of individuals and

$o$ : the type of propositions.



# Type-Assignments

---

## Definition

A **formula** or **type-assignment** in  $\lambda^{\rightarrow}$  is of the form

$$M : \sigma,$$

which means that  $\lambda$ -term  $M$  has type  $\sigma$ .

# Type-Assignments

---

## Example

The simple theory of types of Church [Chu1940] included the following type-assignments:

$N : o \rightarrow o$	(negation)
$A : o \rightarrow o \rightarrow o$	(disjunction)
$\Pi : (\sigma \rightarrow o) \rightarrow o$	(universal quantifier)
$0_{\sigma'} : \sigma' \rightarrow \sigma'$	(zero)
$S_{\sigma'} : (\sigma' \rightarrow \sigma') \rightarrow \sigma' \rightarrow \sigma'$	(successor)

where  $\sigma'$  denotes  $(\sigma \rightarrow \sigma) \rightarrow (\sigma \rightarrow \sigma)$ .

# Type-Contexts

---

## Definition

A **type-context** (or **basis**)  $\Gamma$  is a **finite** set of type-assignments

$$\Gamma = \{ x_1 : \sigma_1, \dots, x_n : \sigma_n \}$$

in which there is no a term-variable with more than one assignment.

## Remark

Note that a type-context only includes typed term-variables.

# Typing Relation

---

## Definition

The **typing relation**  $\Gamma \vdash M : \sigma$  means that  $\lambda$ -term  $M$  has type  $\sigma$  in the context  $\Gamma$ .

# Typing Rules

---

## Rules of type-assignment

The typing rules for  $\lambda^{\rightarrow}$  (relative to a context  $\Gamma$ ) are given by\*

$$\Gamma, x : \sigma \vdash x : \sigma \quad (\text{axiom})$$

$$\frac{\Gamma \vdash M : (\sigma \rightarrow \tau) \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (M N) : \tau} \quad (\rightarrow\text{-elimination})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x.M) : (\sigma \rightarrow \tau)} \quad (\rightarrow\text{-introduction})$$

---

\*See, e.g. [Bar1992; Hin2008].

# Typing Rules

---

## Example

To prove that  $\lambda x.x$  has type  $\sigma \rightarrow \sigma$ , for all  $\sigma \in \mathbb{T}$ .

# Typing Rules

---

## Example

To prove that  $\lambda x.x$  has type  $\sigma \rightarrow \sigma$ , for all  $\sigma \in \mathbb{T}$ .

## Proof

$$\frac{x : \sigma \vdash x : \sigma}{\vdash \lambda x.x : \sigma \rightarrow \sigma} \rightarrow I$$

# Typing Rules

---

## Example

To prove that  $\lambda x y. x$  has type  $\sigma \rightarrow (\tau \rightarrow \sigma)$ , for all  $\sigma, \tau, \in \mathbb{T}$ .

## Proof

$$\frac{\frac{x : \sigma, y : \tau \vdash x : \sigma}{x : \sigma \vdash \lambda y. x : \tau \rightarrow \sigma} \rightarrow\text{I}}{\vdash \lambda x y. x : \sigma \rightarrow (\tau \rightarrow \sigma)} \rightarrow\text{I}$$



# Typing Rules

## Example

The composition operator  $\lambda f g x. f (g x)$  has type  $(\sigma_2 \rightarrow \sigma_3) \rightarrow (\sigma_1 \rightarrow \sigma_2) \rightarrow \sigma_1 \rightarrow \sigma_3$ , for all  $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{T}$ .

## Proof

Let  $\Gamma = \{ f : \sigma_2 \rightarrow \sigma_3, g : \sigma_1 \rightarrow \sigma_2, x : \sigma_1 \}$ . Then

$$\begin{array}{c} \frac{\Gamma \vdash f : \sigma_2 \rightarrow \sigma_3 \quad \frac{\Gamma \vdash g : \sigma_1 \rightarrow \sigma_2 \quad \Gamma \vdash x : \sigma_1}{\Gamma \vdash g x : \sigma_2} \rightarrow E}{\Gamma \vdash f (g x) : \sigma_3} \rightarrow E \\ \frac{}{f : \sigma_2 \rightarrow \sigma_3, g : \sigma_1 \rightarrow \sigma_2 \vdash \lambda x. f (g x) : \sigma_1 \rightarrow \sigma_3} \rightarrow I \\ \frac{}{f : \sigma_2 \rightarrow \sigma_3 \vdash \lambda g x. f (g x) : (\sigma_1 \rightarrow \sigma_2) \rightarrow (\sigma_1 \rightarrow \sigma_3)} \rightarrow I \\ \frac{}{\vdash \lambda f g x. f (g x) : (\sigma_2 \rightarrow \sigma_3) \rightarrow (\sigma_1 \rightarrow \sigma_2) \rightarrow (\sigma_1 \rightarrow \sigma_3)} \rightarrow I \end{array}$$

# Typing Rules

---

## Example

Let  $\Gamma = \{y : \sigma\}$  be a basis. To prove that  $\Gamma \vdash (\lambda x.x) y : \sigma$ .

Proof

$$\frac{\frac{\Gamma, x : \sigma \vdash x : \sigma}{\Gamma \vdash \lambda x.x : \sigma \rightarrow \sigma} \rightarrow I \quad \Gamma \vdash y : \sigma}{\Gamma \vdash (\lambda x.x) y : \sigma} \rightarrow E$$

# Typing Rules

---

## Example

The self-application  $xx$  has no a type assigned in  $\lambda^{\rightarrow}$ .

---

\*See, e.g. [BDS2013, Proposition 2.4.24].

# Typing Rules

---

## Example

The self-application  $xx$  has no a type assigned in  $\lambda^{\rightarrow}$ .

## Example

The fixed-point combinator  $Y := \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$  does not have a type in  $\lambda^{\rightarrow}.$ \*

---

\*See, e.g. [BDS2013, Proposition 2.4.24].

# Typing Rules

---

## Example

The self-application  $xx$  has no a type assigned in  $\lambda^{\rightarrow}$ .

## Example

The fixed-point combinator  $Y := \lambda f. (\lambda x. f (x x)) (\lambda x. f (x x))$  does not have a type in  $\lambda^{\rightarrow}.$ \*

## Example

The  $\lambda$ -term  $\Omega := (\lambda x. x x) (\lambda x. x x)$  does not have a type in  $\lambda^{\rightarrow}$  (because it has no a  $\beta$ -normal form).

---

\*See, e.g. [BDS2013, Proposition 2.4.24].

# Definable Number-Theoretic Functions

---

## Remark

Recall that a number-theoretic function is a function whose signature is

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

# Definable Number-Theoretic Functions

---

## Example

The following functions are number-theoretic functions:

$$z(x) = 0$$

(**zero** function)

$$s(x) = x + 1$$

(**successor** function)

$$p_k^n(x_1, \dots, x_n) = x_k$$

( **$n$ -ary projection** functions)

$$k_n(x) = n$$

( **$n$ -constant** functions)

$$\text{discr}(x, y, z) = \begin{cases} y, & \text{if } x \neq 0 \\ z, & \text{if } x = 0 \end{cases}$$

(**discriminator** function)

# Definable Number-Theoretic Functions

---

## Definition

Let's work with the  $\lambda^{\rightarrow}$  with only one base type, denoted type  $o$ . The **type of natural numbers** is defined by

$$\text{Nat} := (o \rightarrow o) \rightarrow (o \rightarrow o).$$



# Definable Number-Theoretic Functions

---

## Example

The  $\lambda$ -term  $c_0 := \lambda f x. x$  has type **Nat**.

$$\frac{\frac{f : o \rightarrow o, x : o \vdash x : o}{f : o \rightarrow o \vdash \lambda x. x : o \rightarrow o} \rightarrow I}{\vdash \lambda f x. x : (o \rightarrow o) \rightarrow o \rightarrow o} \rightarrow I$$

# Definable Number-Theoretic Functions

## Example

The  $\lambda$ -term  $\text{succ} := \lambda n f x. f (n f x)$  has type  $\text{Nat} \rightarrow \text{Nat}$ .

Let  $\Gamma = \{n : \text{Nat}, f : o \rightarrow o, x : o\}$ . Then

$$\begin{array}{c} \frac{\Gamma \vdash n : \text{Nat} \quad \Gamma \vdash f : o \rightarrow o}{\Gamma \vdash n f : o \rightarrow o} \rightarrow E \quad \Gamma \vdash x : o \\ \frac{\Gamma \vdash f : o \rightarrow o \quad \Gamma \vdash n f x : o}{\Gamma \vdash f (n f x) : o} \rightarrow E \\ \frac{\Gamma \vdash f (n f x) : o}{n : \text{Nat}, f : o \rightarrow o \vdash \lambda x. f (n f x) : o \rightarrow o} \rightarrow I \\ \frac{n : \text{Nat}, f : o \rightarrow o \vdash \lambda x. f (n f x) : o \rightarrow o}{n : \text{Nat} \vdash \lambda f x. f (n f x) : \text{Nat}} \rightarrow I \\ \frac{n : \text{Nat} \vdash \lambda f x. f (n f x) : \text{Nat}}{\vdash \lambda n f x. f (n f x) : \text{Nat} \rightarrow \text{Nat}} \rightarrow I \end{array}$$

# Definable Number-Theoretic Functions

---

## Example

$$\begin{aligned}c_{n+1} &: \text{Nat} && \quad := \text{succ } c_n, \\ \text{add} &: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat} && \quad := \lambda m n f x. m f (n f x), \\ \text{mult} &: \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat} && \quad := \lambda m n f. m (n f).\end{aligned}$$

# Definable Number-Theoretic Functions

---

## Definition

Let  $\varphi : \mathbb{N}^k \rightarrow \mathbb{N}$  be a number-theoretic function. The function  $\varphi$  is **definable respect to Nat** in the  $\lambda \rightarrow$  iff there exists a  $\lambda$ -term  $F : \mathbf{Nat}^n \rightarrow \mathbf{Nat}$  such that for all  $n_1, \dots, n_k \in \mathbb{N}$ ,

$$\varphi(n_1, \dots, n_k) = a \Rightarrow F \mathbf{c}_{n_1} \dots \mathbf{c}_{n_k} =_{\beta} \mathbf{c}_a.$$

# Definable Number-Theoretic Functions

---

## Definition

The class of the so-called **extended polynomials** is the smallest class of number-theoretic functions including the constant functions  $k_0$  and  $k_1$ , the projection functions, addition, multiplication, the discrimination function and closed under composition.\*

---

\*See, e.g. [Sta1979; Dan1999]. A different but equivalent definition is given in [TS2000].

# Definable Number-Theoretic Functions

---

## Theorem

If  $\varphi$  is an extended polynomial, then  $\varphi$  is definable (respect to the type **Nat**) in the  $\lambda^{\rightarrow}$ .

---

\*The theorem was proved by Schwichtenberg [Sch1976] and stated by Statman [Sta1979] as pointed out by Danner [Dan1999].

# Definable Number-Theoretic Functions

---

## Theorem

If  $\varphi$  is an extended polynomial, then  $\varphi$  is definable (respect to the type **Nat**) in the  $\lambda^{\rightarrow}$ .

## Theorem

If a number-theoretic function  $\varphi$  is definable (respect to the type **Nat**) in the  $\lambda^{\rightarrow}$ , then the function  $\varphi$  is an extended polynomial.\*

---

\*The theorem was proved by Schwichtenberg [Sch1976] and stated by Statman [Sta1979] as pointed out by Danner [Dan1999].

# Definable Number-Theoretic Functions

---

## Remark

A function

$\text{pred} : \text{Nat} \rightarrow \text{Nat}$

$\text{pred } c_{n+1} =_{\beta} c_n.$

cannot be defined in the  $\lambda^{\rightarrow}.$ \*

---

\*See, e.g. [BDS2013, Proposition 2.4.22].



# Definable Number-Theoretic Functions

---

## Remark

Let  $P := (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o$ . The type of the predecessor function defined in the untyped  $\lambda$ -calculus is

$$\text{pred} : (P \rightarrow P) \rightarrow o \rightarrow o \rightarrow o$$

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

# Definable Number-Theoretic Functions

---

## Remark

Let  $P := (o \rightarrow o) \rightarrow (o \rightarrow o) \rightarrow o$ . The type of the predecessor function defined in the untyped  $\lambda$ -calculus is

$$\begin{aligned}\text{pred} &: (P \rightarrow P) \rightarrow o \rightarrow o \rightarrow o \\ \text{pred} &:= \lambda n f x. n (\lambda g h. h (g f)) (\lambda u. x) (\lambda u. u).\end{aligned}$$

## Remark

There are not functions defined by (primitive) recursion in the  $\lambda^{\rightarrow}$ .

# Definable Number-Theoretic Functions

---

## Remark

Troelstra and Schwichtenberg [TS2000, p. 21–22] point out that

*However, if we permit ourselves the use of Church numerals of different types, and in particular liberalize the notion of representation of a function by permitting numerals of different types for the input and the output, we can represent more than extended polynomials.*

# Definable Number-Theoretic Functions

---

## Example

Let  $\text{Nat}_A$  be the type of natural numbers of type  $A$

$$\text{Nat}_A := (A \rightarrow A) \rightarrow (A \rightarrow A).$$

and let  $\overline{n}_A$  be a numeral of type  $\text{Nat}_A$ . We can define exponentiation by

$$\text{exp} : \text{Nat}_A \rightarrow \text{Nat}_{A \rightarrow A} \rightarrow \text{Nat}_A := \lambda m n f x. (n m) f x,$$

where

$$\text{exp } \overline{m}_A \overline{n}_{A \rightarrow A} =_{\beta} \overline{m^n}_A \ (n > 0).$$

# Definable Ordinals

---

## Abstractions on the representation of natural numbers

- No abstraction

We can define two constants

$$\begin{aligned}Z &: o, \\S &: o \rightarrow o,\end{aligned}$$

so natural numbers are represented by terms of the form

$$S(\dots (S Z) \dots).$$

Using this representation the definable number-theoretic functions are those definable by iterating successor (e.g. projection and constant functions) [Dan1999].

(continued on next slide)

## Abstractions on the representation of natural numbers (continuation)

- Abstraction on zero

By abstracting on zero, natural numbers are represented by terms of the form

$$\lambda z. S (\dots (S z) \dots).$$

Using this representation the definable number-theoretic functions are those definable by iterating successor and addition [Dan1999].

(continued on next slide)

## Abstractions on the representation of natural numbers (continuation)

- Abstraction on zero and successor

By abstracting on zero and successor, natural numbers are represented by the Church numerals

$$\lambda s z. s (\dots (s z) \dots).$$

And we know which are the definable number-theoretic functions using this representation (i.e. the extended polynomials).

# Definable Ordinals

---

## Representations of ordinals below $\omega^\omega$

From the possible abstractions for representing natural numbers, Danner [Dan1999] defines four representations (called canonical, additive, arithmetic and intensional) for ordinals below  $\omega^\omega$ .



# Definable Ordinals

---

## Notation

For  $n \in \mathbb{N}$ , we define

$$\begin{aligned} M^0 N &:= N, \\ M^{n+1} N &:= M (M^n N). \end{aligned}$$

# Definable Ordinals

---

An  $\omega$ -fold iterator (informally)

We introduced a constant  $L$  which ‘represents’ an  $\omega$ -fold iterator.

## Example

- The  $\lambda$ -term  $LS$  represents the function  $\cdot + w$  (the limit of iterating successor).
- The  $\lambda$ -term  $L^2S$  represents the function  $\cdot + w^2$  (the limit of iterating  $\cdot + w$ ).
- The  $\lambda$ -term  $L^nS$  represents the function  $\cdot + w^n$ .
- The  $\lambda$ -term  $S(S(LS(L^2SZ)))$  represents the ordinal  $\omega^2 + w + 2$ .

# Definable Ordinals

---

## Definition

The **type of ordinal numbers** is defined by

$$\text{ON} := (o \rightarrow o) \rightarrow o \rightarrow o.$$

# Definable Ordinals

---

## Canonical representations

Let  $L$  be of type  $ON$  and let  $\alpha$  be an ordinal below  $\omega^\omega$

$$\alpha = \omega^{n_r} + \dots + \omega^{n_0},$$

where  $n_r \geq n_{r-1} \geq \dots \geq n_0$  and  $n_i \in \mathbb{N}$ .

The **canonical representation** for  $\alpha$ , denoted by  $\bar{\alpha}$ , is defined by

$$\begin{aligned}\alpha^* &:= (L^{n_0} s) ((L^{n_1} s) (\dots ((L^{n_r} s) z) \dots)), \\ \bar{\alpha} &:= \lambda s. \lambda z. \alpha^*,\end{aligned}$$

where  $s : ON \rightarrow ON$  and  $z : ON$ .

# Definable Ordinal Functions

---

## Definition

An ordinal function  $\varphi(\alpha_1, \dots, \alpha_n)$  is **canonically, additively, arithmetically or intensionally definable respect to ON** in the  $\lambda \rightarrow$  iff there exists a closed  $\lambda$ -term  $F : \text{ON}^n \rightarrow \text{ON}$  such that

$$\varphi(\alpha_1, \dots, \alpha_n) = \beta \Rightarrow F A_1 \dots A_n \beta \eta\text{-reduces to } B,$$

where  $A_1, \dots, A_n$  and  $B$  are the canonical, additive, arithmetic or intensional representations for  $\alpha_1, \dots, \alpha_n$  and  $\beta$ , respectively [Dan1999, p. 191].

# Definable Ordinal Functions

---

## Notation

Let  $A$  be a set of ordinal functions. The smallest class of ordinal functions below  $\omega^\omega$  including the constant functions, the projection functions, the set  $A$  and closed under composition is denoted by  $[A]$ .

# Definable Ordinal Functions

---

Theorem (Danner [Dan1999], Theorem 4.11)

The ordinal functions canonically definable are **exactly** the functions

$$[\alpha \mapsto \alpha + 1, \alpha \mapsto \omega\alpha].$$

# Definable Ordinal Functions

---

Theorem (Danner [Dan1999], Theorem 4.11)

The ordinal functions canonically definable are **exactly** the functions

$$[\alpha \mapsto \alpha + 1, \alpha \mapsto \omega\alpha].$$

Theorem (Danner [Dan1999], Theorem 4.15)

The ordinal functions additively definable are **exactly** the functions

$$[+, \alpha \mapsto \omega\alpha].$$



# Definable Ordinal Functions

---

Theorem (Danner [Dan1999], Theorem 4.11)

The ordinal functions canonically definable are **exactly** the functions

$$[\alpha \mapsto \alpha + 1, \alpha \mapsto \omega\alpha].$$

Theorem (Danner [Dan1999], Theorem 4.15)

The ordinal functions additively definable are **exactly** the functions

$$[+, \alpha \mapsto \omega\alpha].$$

Theorem (Danner [Dan1999], Theorem 4.21)

The ordinal functions arithmetically definable are **exactly** the functions

$$[+, \times].$$

# Definable Ordinal Functions

---

## Definition

The **weak discriminator** function on ordinals is the function defined by [Dan1999, p. 198]

$$\text{wdiscr}(\alpha, \beta, \delta) := \begin{cases} \beta, & \text{if } \alpha > 0; \\ \delta, & \text{if } \alpha = 0. \end{cases}$$

# Definable Ordinal Functions

---

## Definition

The **weak discriminator** function on ordinals is the function defined by [Dan1999, p. 198]

$$\text{wdiscr}(\alpha, \beta, \delta) := \begin{cases} \beta, & \text{if } \alpha > 0; \\ \delta, & \text{if } \alpha = 0. \end{cases}$$

Theorem (Danner [Dan1999], Theorem 4.24)

The ordinal functions intensionally definable are **exactly** the functions

$$[+, \times, \text{wdiscr}].$$

## References

---

- [Bar1992] Henk Barendregt. Lambda Calculi with Types. In: Handbook of Logic in Computer Science. Volume 2. Ed. by S. Abramsky, Dov M. Gabbay and T. S. E. Maibaum. Clarendon Press, 1992, pp. 117–309 (cit. on p. [13](#)).
- [BDS2013] Henk Barendregt, Wil Dekkers and Richard Statman. Lambda Calculus with Types. Cambridge University Press, 2013. DOI: [10.1017/CB09781139032636](#) (cit. on pp. [19–21](#), [32](#)).
- [Chu1940] Alonzo Church. A Formulation of the Simple Theory of Types. The Journal of Symbolic Logic 5.2 (1940), pp. 55–68. DOI: [10.2307/2266170](#) (cit. on pp. [7](#), [8](#), [10](#)).
- [Dan1999] N. Danner. Ordinals and Ordinal Functions Representable in the Simply Typed Lambda Calculus. Annals of Pure and Applied Logic 97.1-3 (1999), pp. 179–201. DOI: [10.1016/S0168-0072\(98\)00046-3](#) (cit. on pp. [29–31](#), [37](#), [38](#), [40](#), [45](#), [47–51](#)).
- [Hin2008] J. Roger Hindley. Basic Simple Type Theory. Digitally printed version with corrections. Vol. 42. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2008 (1997) (cit. on p. [13](#)).

# References

- [Sch1976] H. Schwichtenberg. Functions Definable in the Simply-Typed Lambda Calculus. Translation by R. J. Irwin of *Definierbare Funktionen im  $\lambda$ -Kalkül mit Typen*. Arch. Math. Logic, 1976, vol 17(3–4), pp. 113–114. 1976. URL: <http://www.cis.syr.edu/~royer/htc96/fdst.ps> (cit. on pp. 30, 31).
- [Sta1979] Richard Statman. The Typed  $\lambda$ -Calculus Is Not Elementary Recursive. Theoretical Computer Science 9.1 (1979), pp. 73–81. DOI: [10.1016/0304-3975\(79\)90007-0](https://doi.org/10.1016/0304-3975(79)90007-0) (cit. on pp. 29–31).
- [TS2000] A. S. Troelstra and H. Schwichtenberg. Basic Proof Theory. 2nd ed. Vol. 44. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000 (1996). DOI: [10.1017/CB09781139168717](https://doi.org/10.1017/CB09781139168717) (cit. on pp. 29, 35).