Lectures on The Lambda Calculus (I)

Masahiko Sato
Graduate School of Informatics, Kyoto University

Autumn school "Proof and Computation"
Fischbachau, Germany
October 4, 2016

Overview

In these lectures, I give an introduction to the $\boldsymbol{\lambda}$-calculus from the following view point.

Although $\boldsymbol{\lambda}$-terms are defined by the well known 'inductive definition', it is also well known that it is far from easy to define the *substitution operation* on $\boldsymbol{\lambda}$-terms correctly.

In order to cope with this situation, we will develop elementary part of the $\boldsymbol{\lambda}$-calculus, entirely based on the *finitistic mathematics* (in the sense of Hilbert). This means that all the mathematical objects we discuss in the lectures are finitary objects created by finitistic methods.

Based on this foundational motivation, we will define $\boldsymbol{\lambda}$-terms as elements of a free algebra $\mathbb{L}$.

Variable and substitution

The notions of *variable* and *substituion* are two central notions in the $\boldsymbol{\lambda}$-calculus.

Variable and substitution

The notions of *variable* and *substituion* are two central notions in the $\boldsymbol{\lambda}$-calculus.

For that matter, these notions are indispensable in any branch of mathematics.

Variable and substitution

The notions of *variable* and *substituion* are two central notions in the $\boldsymbol{\lambda}$-calculus.

For that matter, these notions are indispensable in any branch of mathematics.

But, they are particularly important in the $\boldsymbol{\lambda}$-calculus since, as Church pointed out in his '*Foundations of a simple theory of types*', $\boldsymbol{\lambda}$-terms can be used to encode other mathemtical expressions involving variable binding (e.g. $\forall \boldsymbol{x}\boldsymbol{.}\ \boldsymbol{x} = \boldsymbol{x}$ or $\sum_{\boldsymbol{i}=\mathbf{1}}^{\mathbf{100}} \boldsymbol{i}$.)

Variable and substitution (cont.)

In these lectures, I will argue that Church's usage of variable and substitution are conceptually wrong although technically consistent.

I think that this is unfortunate, since $\boldsymbol{\lambda}$-calculus itself is a very important calculus. The problem is that Church failed to introduce it naturally.

I will develop $\boldsymbol{\lambda}$-calculus without using variables and substitutions in Church's sense.

Frege, Gentzen, Church and McCarthy

- Gottlob Frege (1848 – 1925)
  - Invented modern logic with quantification.
- Gerhard Gentzen (1905 – 1945)
  - Invented sequent calculus and natural deduction.
- Alonzo Church (1903 – 1995)
  - Invented $\boldsymbol{\lambda}$-calculus.
- John McCarthy (1927 – 2011)
  - Invented LISP programming language.
  - Introduced the notion of abstract syntax.
  - Introduced the notion of proof checking by a computer.

External syntax and internal syntax

External syntax is mainly used for human communications. Abbreviations, macros, syntactitic sugar etc. are examples of external syntax.

Internal syntax is mainly used to represent external syntax inside a computer. Programming languages *parse* programs written in external language into internal language. Internal syntax can be hard to read but convenient for computing by a computer.

McCarthy introduced the notion of abstract syntax which can be used to relate external syntax and internal syntax. In his paper *'A basis for a mathematical theory of computation'* (1963) McCarthy (essentially) characterized abstact syntax as free algebra.

What is important here is that one and same object can be written in various external or internal syntax, but abstract syntax gives a canonical notation for the object.

Plan of the lectures

I Background history, philosophy and *main idea*.

II The free algebra $\mathbb{T}$ of *threads*

III The free algebra $\mathbb{L}$ of $\mathbb{L}$*-expressions*. Church-Rosser Theorem and the pushout property.

We will also discuss relationship between $\mathbb{L}$, Church's $\boldsymbol{\lambda}$-terms and de Bruijn's noatation system.

These lectures are based on my work in progress.

Frege's view

In §§28 – 31 of *Grundgesetze der Arithmetic, volume 1* (1893), Frege tried to define the syntax and semantics (*Bedeutung*) of the language (*Begriffsschrift*) he used in the book.

Russell found a technical gap in Frege's definition (Russell Paradox), but it is interesting to note that Frege defined his well-formed expressions (*Eigennamen*), which include higher-order expressions, without starting from *variables*.

Therefore, I believe that Frege would have rejected the definition of raw lambda-terms given by Church:

$$\Lambda \ni M, N ::= x \mid \lambda_x M \mid (M\ N)$$

Raw λ-terms

Definition of raw lambda-terms.

$$\Lambda \ni M, N, P ::= x \mid \lambda_x M \mid (M\ N)$$

$(M\ N)$ stands for the application of (function) M to N.

We write $[x := N]M$ for the result of substituting N for x in M.

Problems with raw λ-terms

A problem with raw lambda-terms is that substitution is non-trivial.

Let M be $\lambda_y(x\ y)$. Then, what is $[x := y]M$?

$[x := y]\lambda_y(x\ y) = \lambda_y(y\ y)$ is not correct. y was a free variable before substitution, but it becomes a bound variable after susbstitution.

The problem is solved by renaming y in M to a fresh variable z. Then, $[x := y]\lambda_z(x\ z) = \lambda_z(y\ z)$.

We replaced $M = \lambda_y(x\ y)$ by $M' = \lambda_z(x\ z)$ hich is obtained by renaming. Such a pair of M and M' are called *α-equivalent*.

Problems with raw λ-terms (cont.)

A second problem with raw $\boldsymbol{\lambda}$-terms is that the notion of *immediate subterm* becomes obscure on (raw) $\boldsymbol{\lambda}$-terms.

For example what is (or, are) the immediate subterm(s) of

$$\boldsymbol{\lambda_x \lambda_y (x\ y)?}$$

You may say the answer is $\boldsymbol{\lambda_y (x\ y)}$ (with $\boldsymbol{x}$ free).

But, then what about

$$\boldsymbol{\lambda_y \lambda_x (y\ x)?}$$

Your answer should be $\boldsymbol{\lambda_x (y\ x)}$ (with $\boldsymbol{y}$ free).

Since two given terms are $\boldsymbol{\alpha}$-equivalent, the answers must also be $\boldsymbol{\alpha}$-equivalent. But, this is not the case here.

Problems with raw λ-terms (cont.)

All of these difficulties boil down to the following:

1. The raw $\boldsymbol{\lambda}$-terms $\boldsymbol{\lambda_x x}$ and $\boldsymbol{\lambda_y y}$ are two distinct raw $\boldsymbol{\lambda}$-terms (since they are syntactically different).
2. However, we somehow wish to identify them. And we do this by quotienting $\boldsymbol{\Lambda}$ by the $\boldsymbol{\alpha}$-equivalence relation.

Structure of raw λ-terms

Recall that:

$$\Lambda \ni M, N, P ::= x \mid \lambda_x M \mid (M\ N)$$

By writing $\lambda_{x_1 x_2 \cdots x_n} M$ for $\lambda_{x_1} \lambda_{x_2} \cdots \lambda_{x_n} M$ $(n \geq 0)$, any λ-term can be uniquely written in one of the following two forms.

1. $\lambda_{x_1 x_2 \cdots x_n} y$.
2. $\lambda_{x_1 x_2 \cdots x_n} (M\ N)$.

We will call a term of the first form *thread*.

An alternative definition of Λ

Using the above classification, we can give an alternative definition of $\boldsymbol{\Lambda}$ as follows.

$$\frac{}{\boldsymbol{\lambda_{\bar{x}} y \in \Lambda}} \qquad \frac{\boldsymbol{\lambda_{\bar{x}} M \in \Lambda} \quad \boldsymbol{\lambda_{\bar{x}} N \in \Lambda}}{\boldsymbol{\lambda_{\bar{x}} (M\ N) \in \Lambda}}$$

This gives a correct definition of $\boldsymbol{\Lambda}$ since by these rules we can generate all the raw $\boldsymbol{\lambda}$-tersm.

But it is incovenient to use this as an official definition of raw $\boldsymbol{\lambda}$-term since it does not give us a free algebra.

However, it can be used to characterize closed $\boldsymbol{\lambda}$-terms and also to define $\boldsymbol{\alpha}$-equivalence.

The set Λ_0 of closed terms

We can define the subset $\boldsymbol{\Lambda_0}$ of $\boldsymbol{\Lambda}$, consiting of closed $\boldsymbol{\lambda}$-terms, as follows.

$$\frac{\boldsymbol{y \in \bar{x}}}{\boldsymbol{\lambda_{\bar{x}} y \in \Lambda_0}} \qquad \frac{\boldsymbol{\lambda_{\bar{x}} M \in \Lambda_0 \quad \lambda_{\bar{x}} N \in \Lambda_0}}{\boldsymbol{\lambda_{\bar{x}}(M\ N) \in \Lambda_0}}$$

Note that the above definition does not rely on the notion of free occurrences of a variable in a term.

This definition suggests that we should be able to develop proof theory of the $\boldsymbol{\lambda}$-calculus with free variables without appealing to the notion of bound variables, and of the $\boldsymbol{\lambda}$-caluculs of closed $\boldsymbol{\lambda}$-terms without using the notion of variables.

λ_β-calculus

$$\frac{}{(\lambda_x M\ N) \to_\beta M[x := N]}\ \beta$$

$$\frac{M \to_\beta M'}{(M\ N) \to_\beta (M'\ N)}\ \mathsf{L} \qquad \frac{N \to_\beta N'}{(M\ N) \to_\beta (M\ N')}\ \mathsf{R}$$

$$\frac{M \to_\beta N}{\lambda_x M \to_\beta \lambda_x N}\ \xi$$

$$\frac{}{M \to_\beta M}\ \mathsf{Rfl} \qquad \frac{M \to_\beta N \quad N \to_\beta P}{M \to_\beta P}\ \mathsf{Trn}$$

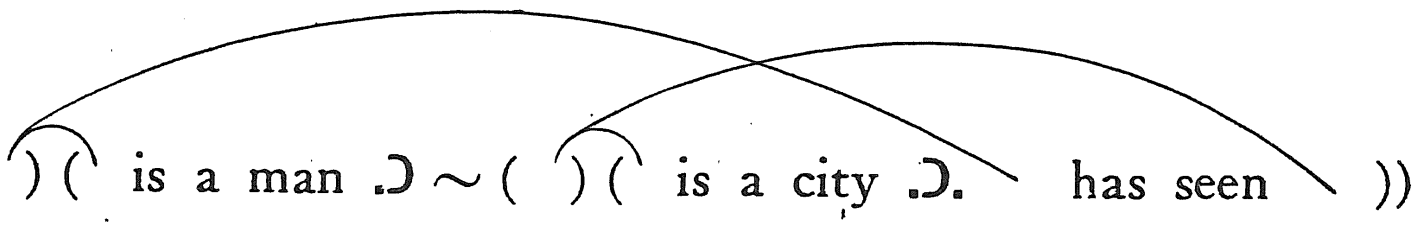
The β-rule captures the informal notion of function application.

History

- Frege, in his *Begriffsschrift* (1879), used Latin letters for global variables and used German letters for local variables.
- Gentzen (in the 30's) also used different sets of variables for global and local variables. He also introduced *eigen variable*.
- Whitehead-Russell (1910) and, later, Gödel and Church used only one sort of letters for both global and local variables. (I think Church made a *conceptual mistake* here.)
- Quine and Bourbaki (in the 50's) introduced *graphical (two dimensional) notation* for local variable binding.
- McCarthy (1963) introduced *abstract syntax*
- de Bruijn (1972) introduced his *indices* and provided a canonical notation for α-equivalent terms.

Quine's notation

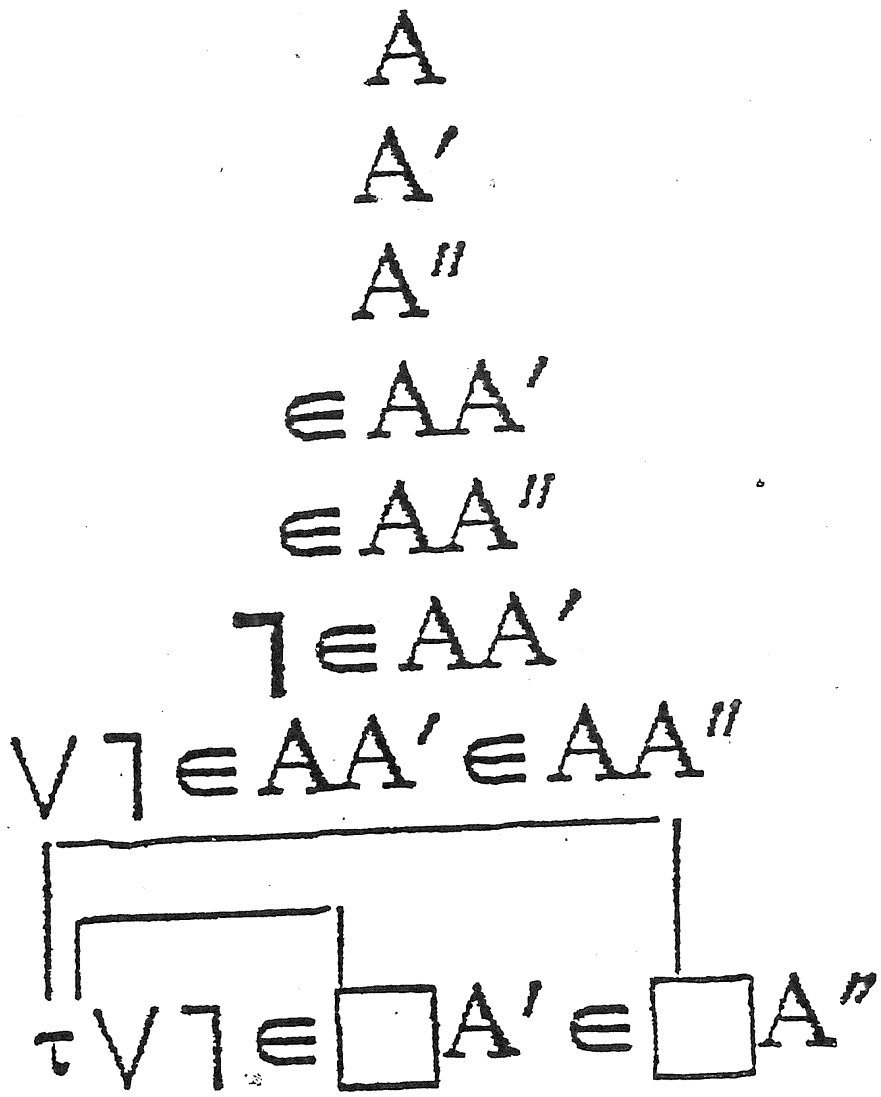
sideration for established usage, the "variation" connoted belongs to a vague metaphor which is best forgotten. The variables have no meaning beyond the pronominal sort of meaning which is reflected in translations such as (20); they serve merely to indicate cross-references to various positions of quantification. Such cross-references could be made instead by curved lines or *bonds*; e.g., we might render (27) thus:



and (26) thus:

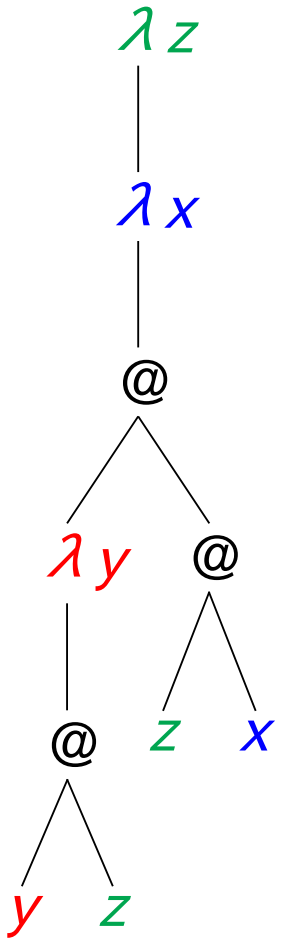
Bourbaki's notation

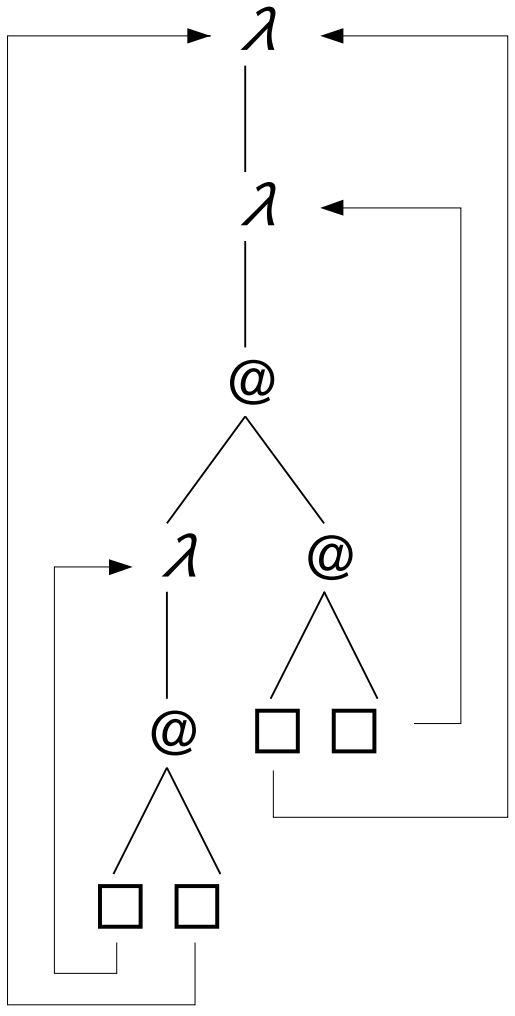
See next slide.


A
A′
A″
∈AA′
∈AA″
¬∈AA′
V¬∈AA′∈AA″
τV¬∈□A′∈□A″

From Church to Quine-Bourbaki

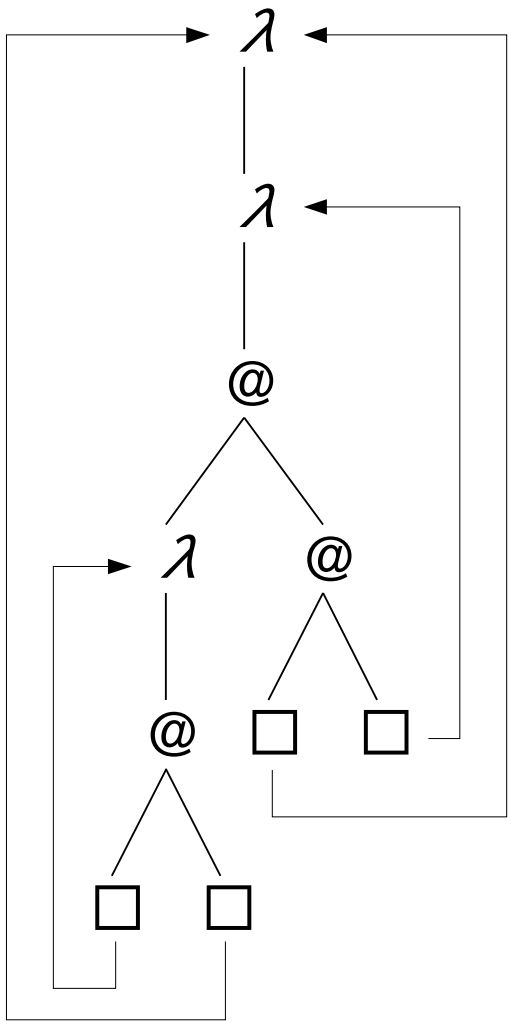
$$\lambda_{zx}(\lambda_{y}(y\ z)\ (z\ x))$$

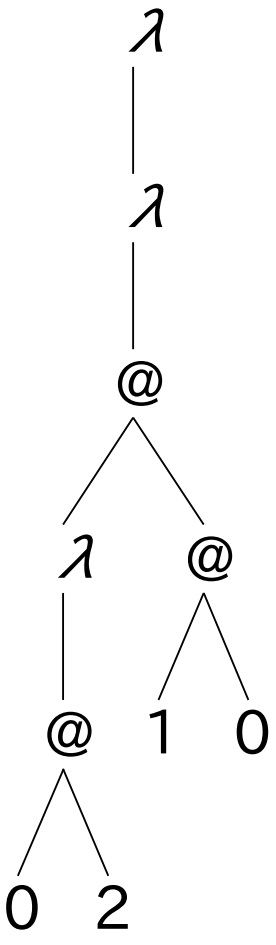




From Quine-Bourbaki to de Bruijn

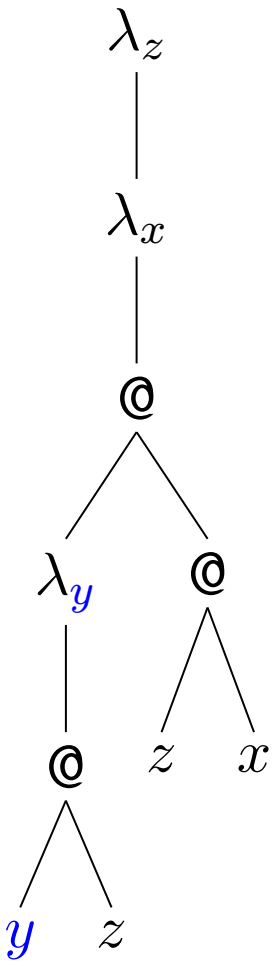
$$\lambda_{zx}(\,\lambda_{y}(y\ z)\ (z\ x))$$





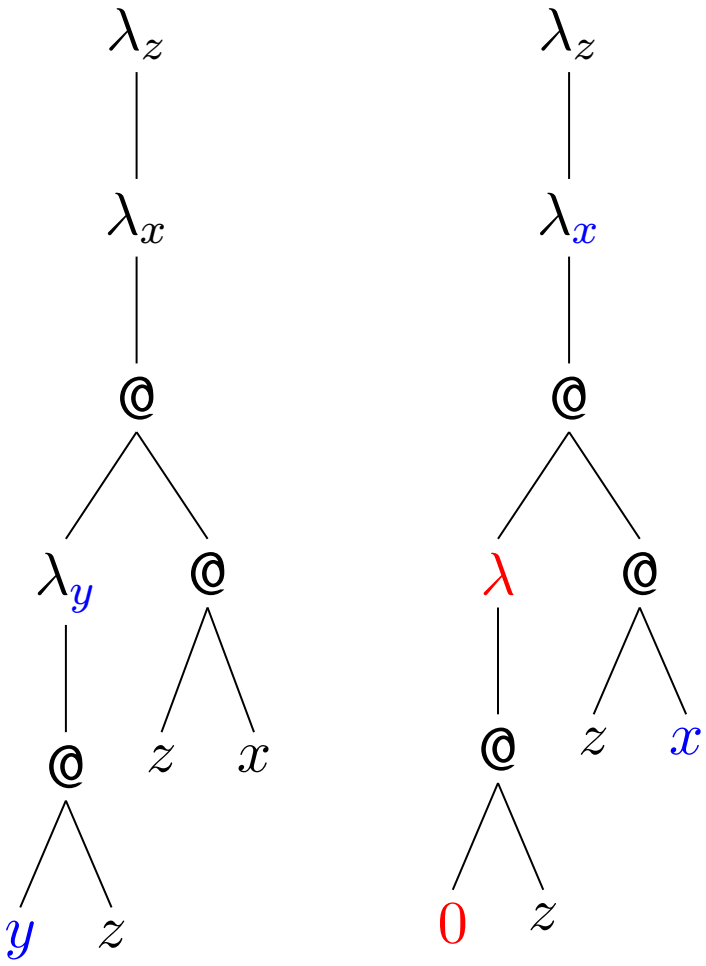
From Church to de Bruijn

$$\lambda_{zx}(\lambda_y(y\ z)\ (z\ x)) = \lambda^2(\lambda(0\ 2)\ (1\ 0))$$



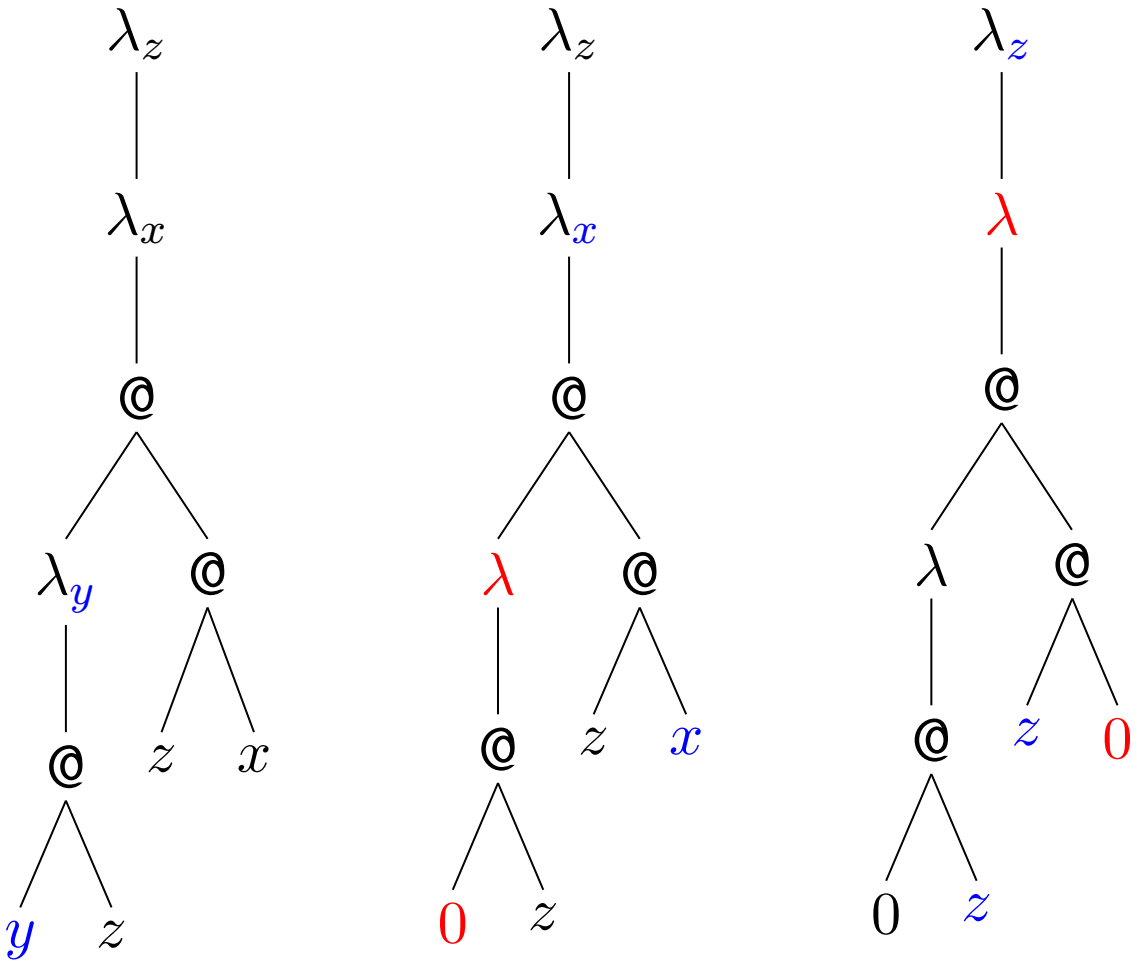
From Church to de Bruijn

$$\lambda_{zx}(\lambda_y(y\ z)\ (z\ x)) = \lambda^2(\lambda(0\ 2)\ (1\ 0))$$



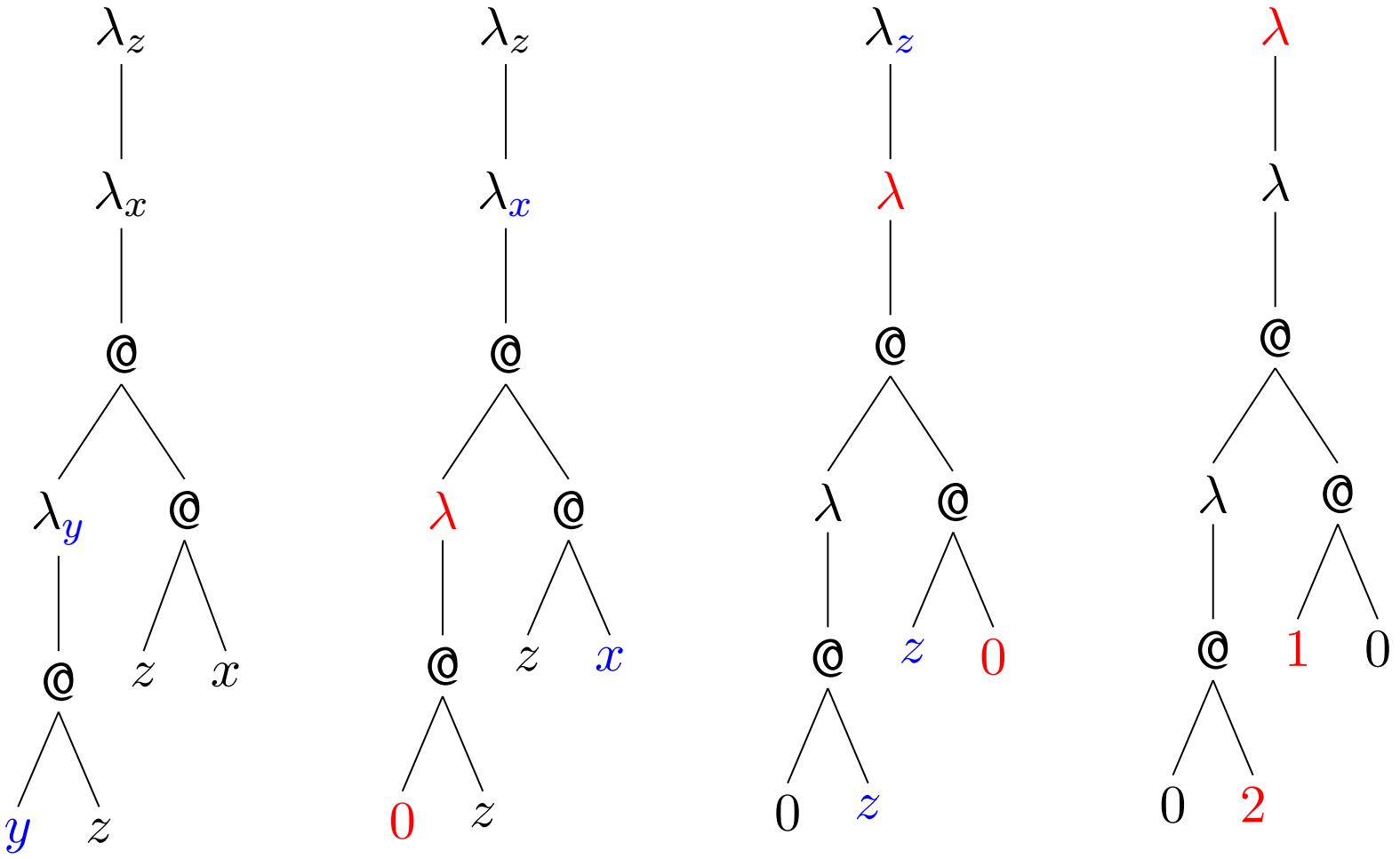
From Church to de Bruijn

$$\lambda_{zx}(\lambda_y(y\ z)\ (z\ x)) = \lambda^2(\lambda(0\ 2)\ (1\ 0))$$



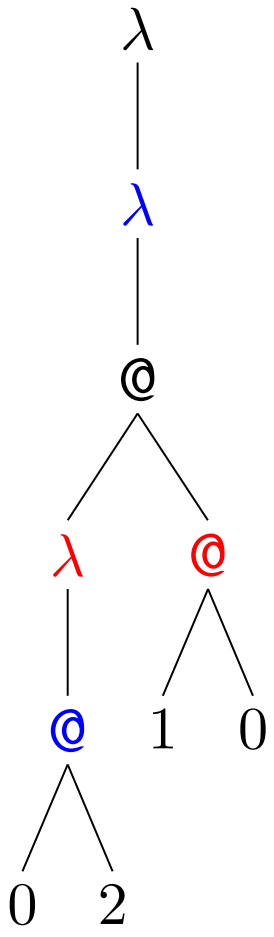
From Church to de Bruijn

$$\lambda_{zx}(\lambda_y(y\ z)\ (z\ x)) = \lambda^2(\lambda(0\ 2)\ (1\ 0))$$



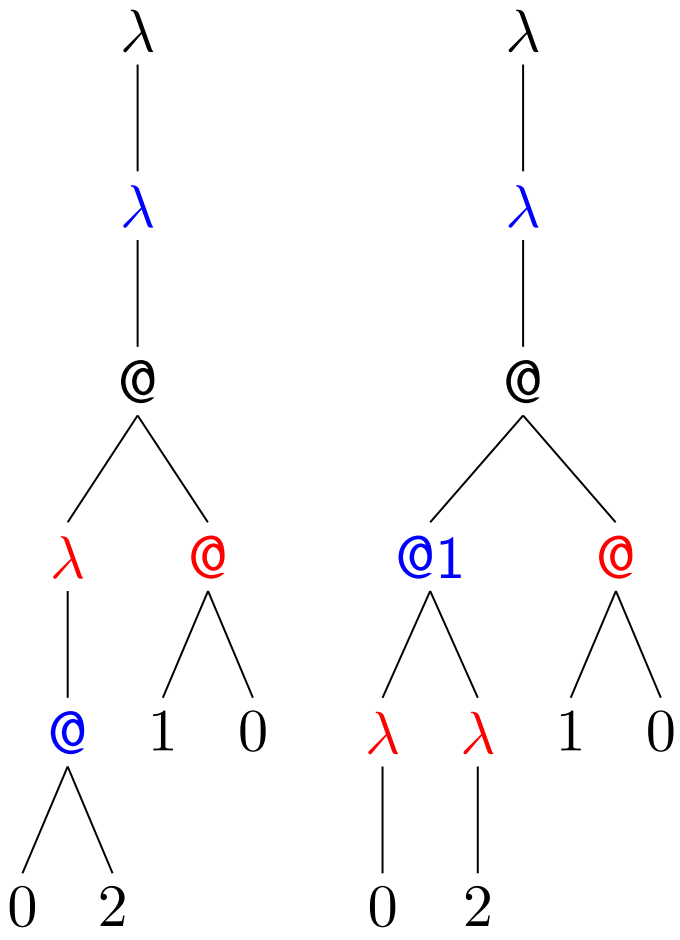
From de Bruijn to our approach

$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$



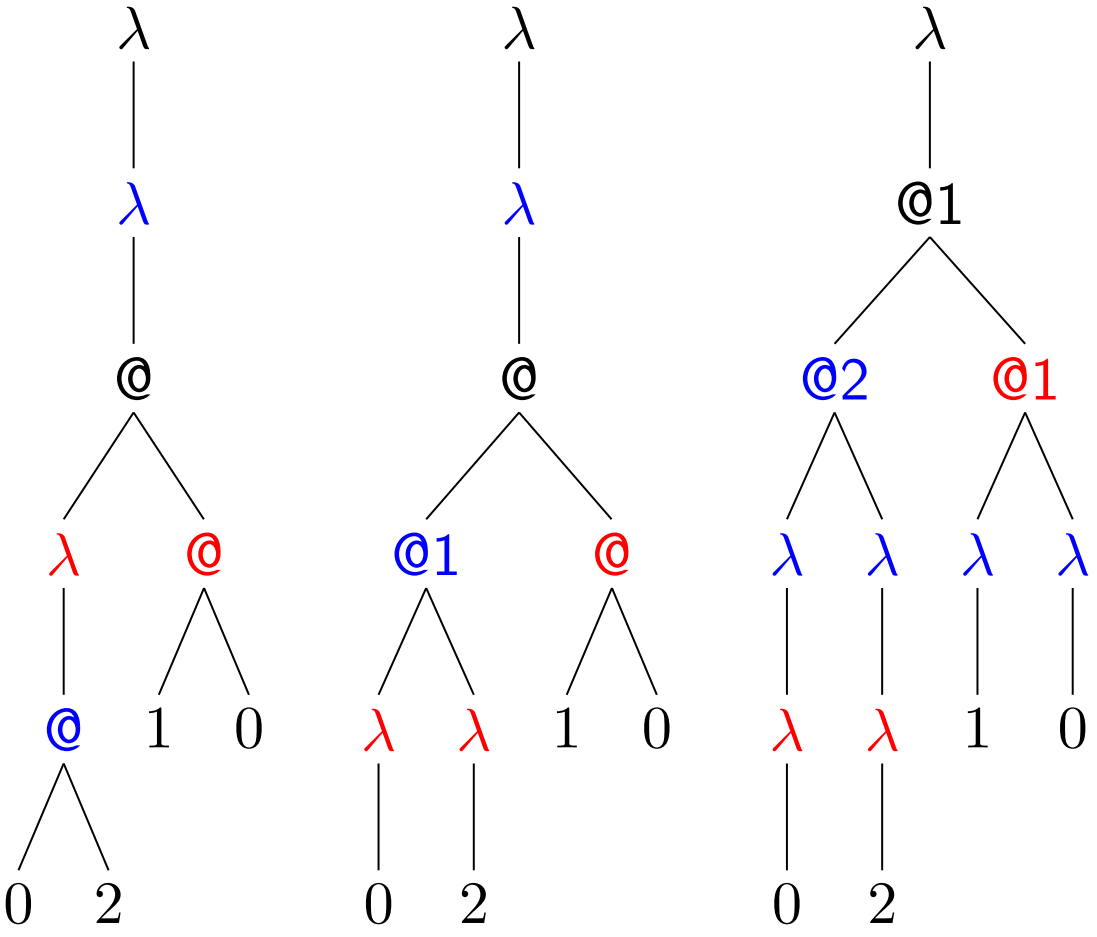
From de Bruijn to our approach

$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$



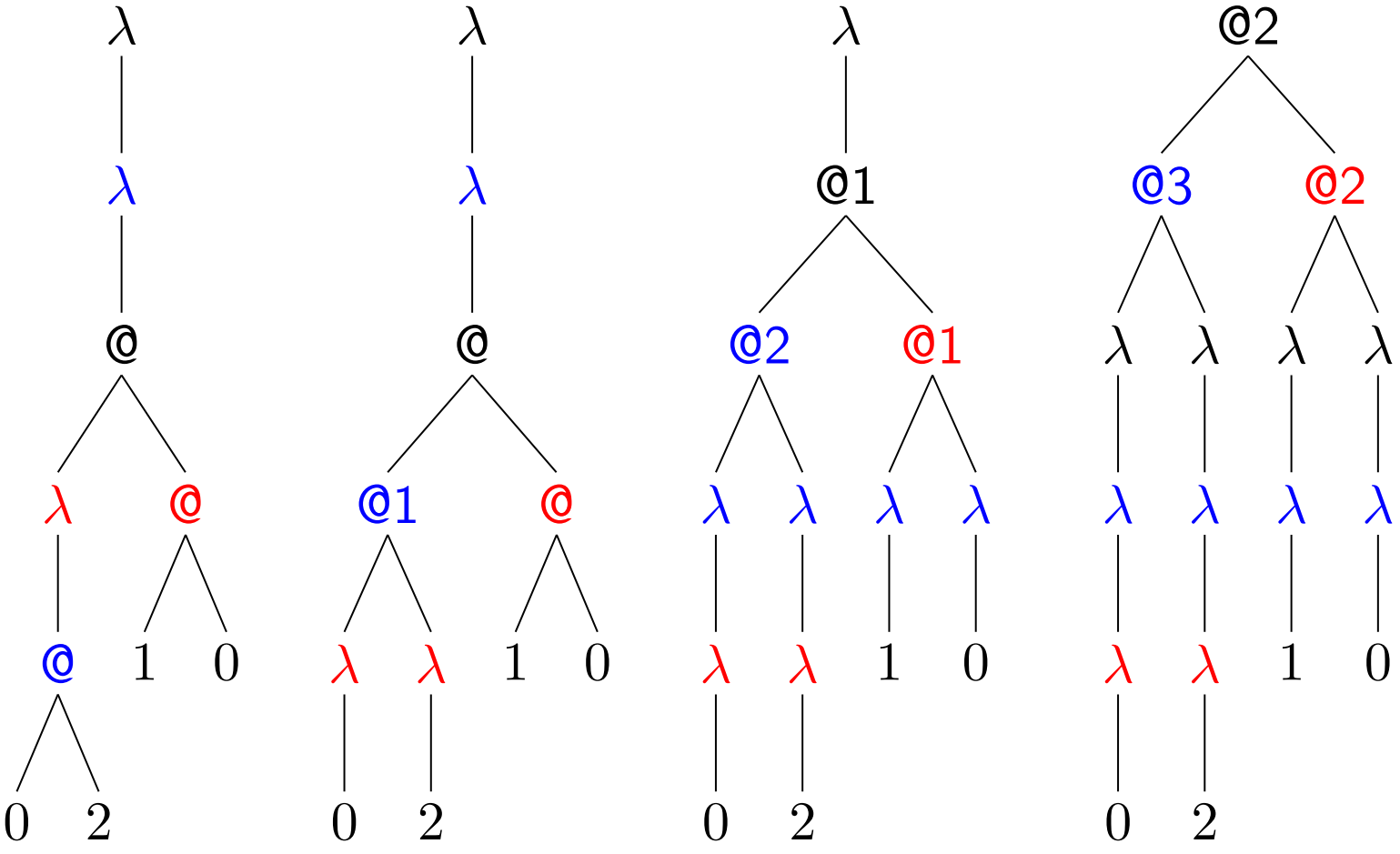
From de Bruijn to our approach

$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$



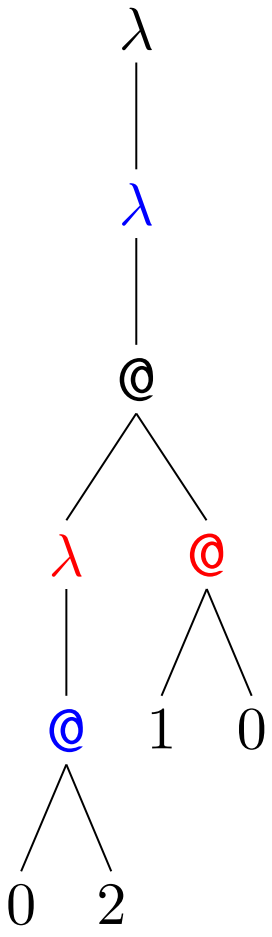
From de Bruijn to our approach

$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$



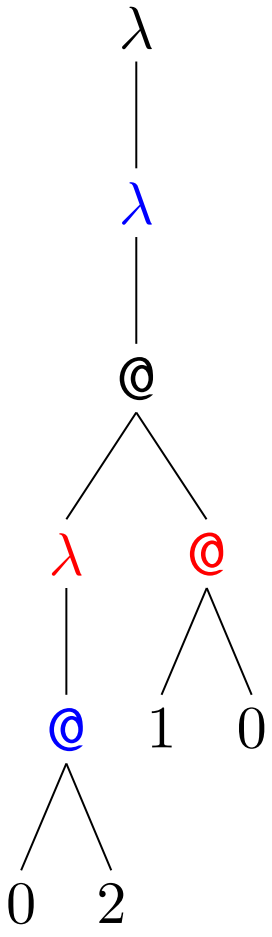
From de Bruijn to our approach (cont.)

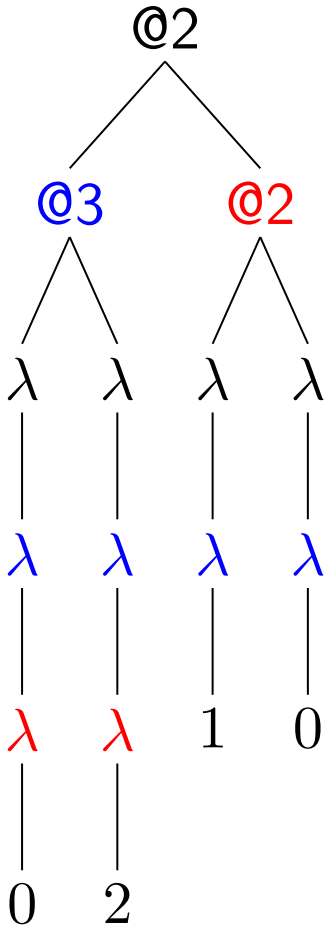
$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$



From de Bruijn to our approach (cont.)

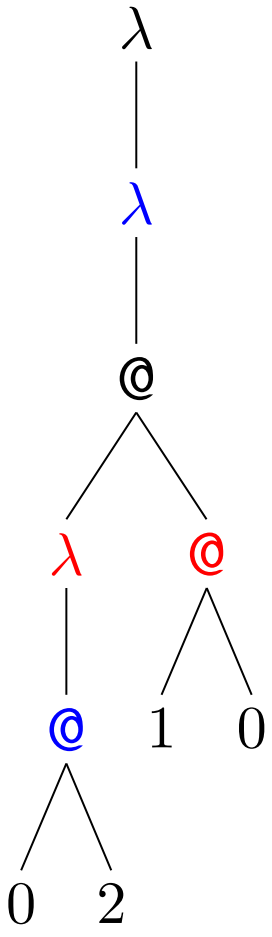
$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$

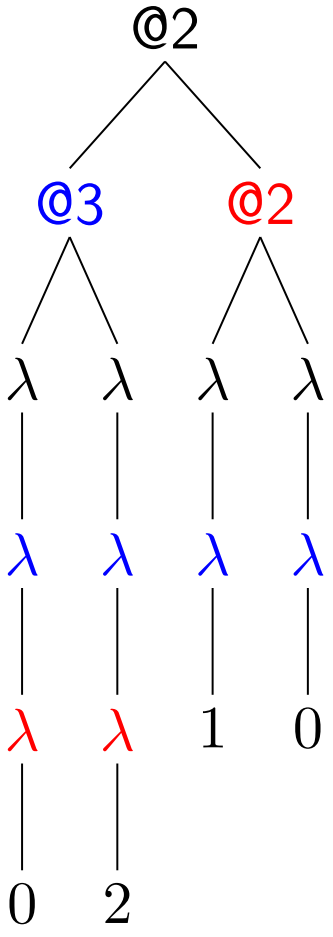


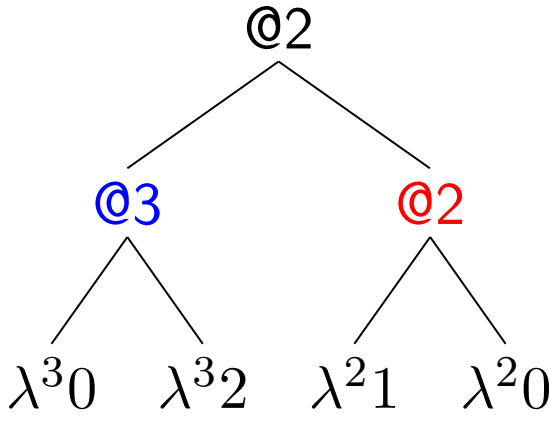


From de Bruijn to our approach (cont.)

$$\lambda^2(\lambda(0\ 2)\ (1\ 0)) = ((\lambda^3 0\ \lambda^3 2)^3\ (\lambda^2 1\ \lambda^2 0)^2)^2$$







Finitistic mathematics and free algebra

Finitistic mathematics (initiated by Hilbert) is deeply connected to (finitistic) free algebras.

Indeed all the finitistic mathematical objects are inductively generated as elements of some finitistic free algebras.

So, in finitistic mathematics:

- We can analyze every objects completely.
- We can prove properties of objects of a free algebra by using the induction principle associated with the algebra.

The free algebra $\mathbb{N}$ of natural numbers

$$\frac{}{\mathbf{0} \in \mathbb{N}}\ \text{Zero} \qquad \frac{\boldsymbol{k} \in \mathbb{N}}{\boldsymbol{k'} \in \mathbb{N}}\ \text{Succ}$$

This algebra has the following signature.

$$\begin{aligned} \text{Zero} &: \quad \to \mathbb{N} \\ \text{Succ} &: \mathbb{N} \to \mathbb{N} \end{aligned}$$

Note that we have

$$\begin{aligned} \mathbf{0} &= \quad\ \text{Zero}, \\ \mathbf{1} = \mathbf{0'} &= \text{Succ}(\text{Zero}), \\ \mathbf{2} = \mathbf{0''} &= \text{Succ}(\text{Succ}(\text{Zero})), \\ &\ldots \end{aligned}$$

de Bruijn algebra $\mathbb{D}$

$$\frac{\boldsymbol{k} \in \mathbb{N}}{\boldsymbol{k} \in \mathbb{D}} \qquad \frac{\boldsymbol{D} \in \mathbb{D}}{\boldsymbol{\lambda D} \in \mathbb{D}}\ \text{Abs} \qquad \frac{\boldsymbol{D} \in \mathbb{D} \quad \boldsymbol{E} \in \mathbb{D}}{\boldsymbol{(D\ E)} \in \mathbb{D}}\ \text{App}$$

The de Buijn algebra enjoys the following equation:

$$\mathbb{D} = \mathbb{N} + \boldsymbol{\lambda}\mathbb{D} + \boldsymbol{(}\mathbb{D}\ \mathbb{D}\boldsymbol{)}$$

$\mathbb{D}$ vs. $\mathbb{L}$

The de Buijn algebra enjoys the following equation:

$$\mathbb{D} = \mathbb{N} + \boldsymbol{\lambda}\mathbb{D} + (\mathbb{D}\ \mathbb{D})$$

We define the algebra $\mathbb{L}$ of $\mathbb{L}$-expressions by the following two equations.

$$\mathbb{T} = \mathbb{N} + \boldsymbol{\lambda}\mathbb{T},\ \mathbb{L} = \mathbb{T} + (\mathbb{L}\ \mathbb{L})^{\mathbb{N}}$$

Note that

$$\begin{aligned}\mathbb{T} &= \mathbb{N} + \boldsymbol{\lambda}\mathbb{T} = \mathbb{N} + \boldsymbol{\lambda}(\overline{\mathbb{N}} + \boldsymbol{\lambda}\mathbb{T}) = \mathbb{N} + \boldsymbol{\lambda}\mathbb{N} + \boldsymbol{\lambda^2}\mathbb{T} \\ &= \cdots = \sum_{\boldsymbol{n}\in\mathbb{N}} \boldsymbol{\lambda^n}\mathbb{N} \\ &\simeq \mathbb{N} \times \mathbb{N}\end{aligned}$$

$\mathbb{D}$ vs. $\mathbb{L}$ (cont.)

We now know that:

$$\mathbb{L} = \mathbb{T} + (\mathbb{L}\ \mathbb{L})^{\mathbb{N}} \quad \text{and} \quad \mathbb{T} = \mathbb{N} + \boldsymbol{\lambda}\mathbb{T} \simeq \mathbb{N} \times \mathbb{N}$$

So, technically, we can define the algebra $\mathbb{L}$ to enjoy the equation:

$$\mathbb{L} = \mathbb{N} \times \mathbb{N} + (\mathbb{L}\ \mathbb{L})^{\mathbb{N}}$$

Compare this with:

$$\mathbb{D} = \mathbb{N} + \boldsymbol{\lambda}\mathbb{D} + (\mathbb{D}\ \mathbb{D})$$

Note that the abstraction constructor $\boldsymbol{\lambda}$ in $\mathbb{D}$ is missing in $\mathbb{L}$.