A Common Notation System for the Lambda-Calculus and Combinatory Logic

Masahiko Sato
Graduate School of Informatics, Kyoto University

The Second Workshop on
Mathematical Logic and its Applications
Kanazawa
March 9, 2018

What are the Lambda-Calculus and Combinatory Logic?

What are the Lambda-Calculus and Combinatory Logic?

The Preface of "Lambda-Calculus and Combinators, an Introduction" by J.R. Hindley and J.P. Seldin says:

The $\boldsymbol{\lambda}$-calculus and combinatory logic are two systems of logic which can also serve as abstract programming languages. They both aim to describe some very general properties of programs that can modify other programs, in an abstract setting not cluttered by details. In some ways they are rivals, in others they support each other.

What are the Lambda-Calculus and Combinatory Logic?

The Preface of "Lambda-Calculus and Combinators, an Introduction" by J.R. Hindley and J.P. Seldin says:

The $\boldsymbol{\lambda}$-calculus and combinatory logic are two systems of logic which can also serve as abstract programming languages. They both aim to describe some very general properties of programs that can modify other programs, in an abstract setting not cluttered by details. In some ways they are rivals, in others they support each other.

In this talk, I will argue that they are, in fact, one and the same calculus.

History of the calculi

History of the calculi

Again from the Preface of "Lambda-Calculus and Combinators, an Introduction".

The λ-calculus was invented around 1930 by an American logician Alonzo Church, as part of a comprehensive logical system which included higher-order operators (operators which act on other operators)...

History of the calculi

Again from the Preface of "Lambda-Calculus and Combinators, an Introduction".

The λ-calculus was invented around 1930 by an American logician Alonzo Church, as part of a comprehensive logical system which included higher-order operators (operators which act on other operators)...

Combinatory logic has the same aims as λ-calculus, and can express the same computational concepts, but its grammar is much simpler. Its basic idea is due to two people: Moses Shönfinkel, who first thought of it in 1920, and Haskell Curry, who independently re-discovered it seven years later and turned it into a workable technique.

Today's Key Phrases

Today's Key Phrases

Semantics of syntax

Today's Key Phrases

Semantics of syntax

What you see is (not) what you get

The syntax of the Lambda Calculus and Combinatory Logic

$$
\begin{array}{rcl}
\mathbb{X} & ::= & x, y, z, \cdots \\
M, N \in \Lambda & ::= & x \mid \lambda_x M \mid (M\ N)^0 \\
M, N \in \mathsf{CL} & ::= & x \mid \mathsf{I} \mid \mathsf{K} \mid \mathsf{S} \mid (M\ N)^0
\end{array}
$$

$(M\ N)^0$ stands for the application of the function M to its argument N. It is often written simply MN, but we will always use the notation $(M\ N)^0$ for the application.

the Lambda Calculus

$$M, N \in \Lambda \ ::= \ x \mid \lambda_x M \mid (M \ N)^0$$

$\lambda_x M$ stands for the function obtained from M by abstracting x in M.

β reduction rule

$$(\lambda_x M \ N)^0 \to [x := N]M$$

Example

$$\begin{aligned}(\lambda_x x \ M)^0 &\to [x := M]x = M \\ ((\lambda_{xy} x \ M)^0 \ N)^0 &\to ([x := M]\lambda_y x \ N)^0 = (\lambda_y M \ N)^0 \\ &\to [y := N]M = M\end{aligned}$$

Combinatory Logic

$$M, N \in \mathsf{CL} \ ::= \ x \mid \mathsf{I} \mid \mathsf{K} \mid \mathsf{S} \mid (M \ N)^0$$

Weak reduction rules

$$(\mathsf{I} \ M)^0 \to M$$
$$((\mathsf{K} \ M)^0 \ N)^0 \to M$$
$$(((\mathsf{S} \ M)^0 \ N)^0 \ P)^0 \to ((M \ P)^0 \ (N \ P)^0)^0$$

These rules suggest the following identities.

$$\mathsf{I} = \lambda_x x$$
$$\mathsf{K} = \lambda_{xy} x$$
$$\mathsf{S} = \lambda_{xyz}((x \ z)^0 \ (y \ z)^0)^0$$

By this identification, every combinatory term becomes a lambda term. Moreover, the above rewriting rules all hold in the lambda calculus.

Combinatory Logic (cont.)

What about the converse direction? We can translate every lambda term to a combinatory term as follow.

$$x^* = x$$
$$(\lambda_x M)^* = \lambda^*{}_x M^*$$
$$((M\ N)^0)^* = (M^*\ N^*)^0$$

We used $\lambda^* : \mathbb{X} \times \mathsf{CL} \to \mathsf{CL}$ above, which we define by:

$$\lambda^*_x x := \mathsf{I}$$
$$\lambda^*_x y := (\mathsf{K}\ y)^0 \quad \text{if } x \neq y$$
$$\lambda^*_x (M\ N)^0 := ((\mathsf{S}\ \lambda^*_x M)^0\ \lambda^*_x N)^0$$

Combinatory Logic (cont.)

The abstraction operator $\boldsymbol{\lambda^*}$ enjoys the following property.

$$(\boldsymbol{\lambda^*_x M\ N})^{\mathbf{0}} \rightarrow [\boldsymbol{x := N}]\boldsymbol{M}$$

So, CL can simulate the $\boldsymbol{\beta}$-reduction rule of the $\boldsymbol{\lambda}$-calculus. However, the simulation does not provide isomorphism. Therefore, for example, the Church-Rosser property for CL does not imply the CR property for the $\boldsymbol{\lambda}$-calculus.

Recall the syntax of $\boldsymbol{\Lambda}$ and CL.

$$\begin{aligned} \mathbb{X} &::= \boldsymbol{x, y, z, \cdots} \\ \boldsymbol{M, N} \in \boldsymbol{\Lambda} &::= \boldsymbol{x} \mid \boldsymbol{\lambda_x M} \mid \boldsymbol{(M\ N)^0} \\ \boldsymbol{M, N} \in \mathsf{CL} &::= \boldsymbol{x} \mid \mathsf{I} \mid \mathsf{K} \mid \mathsf{S} \mid \boldsymbol{(M\ N)^0} \end{aligned}$$

Differences between λ-calculus and Combinatory Logic

- In combinatory logic, if M is a normal term, then $(\mathsf{S}\ M)^0$ is also normal.
  But, in the $\boldsymbol{\lambda}$-calculus, it can be simplified as follows:

$$(\mathsf{S}\ M)^0 \to \lambda_{yz}((M\ z)^0\ (y\ z)^0)^0.$$

  This means that the $\boldsymbol{\lambda}$-calculus has a finer computational granularity.
- While variables are indispensable in the definition of closed $\boldsymbol{\lambda}$-terms, closed CL-terms can be constructed without using variables.
- In $\boldsymbol{\Lambda}$ we cannot avoid the notion of bound variables, but we don't have the notion in CL.

Our Claim

Our claim is that, albeit the differences in the surface syntax of $\boldsymbol{\lambda}$-calculus and Combinatory Logic, they are actually one and the same calculus (or algebra) which formalizes the abstract concept of computable function.

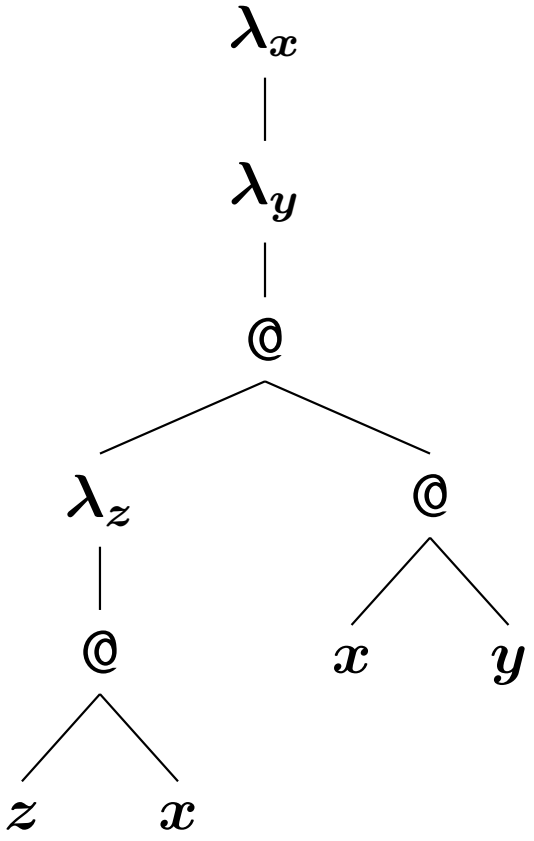
Our Claim

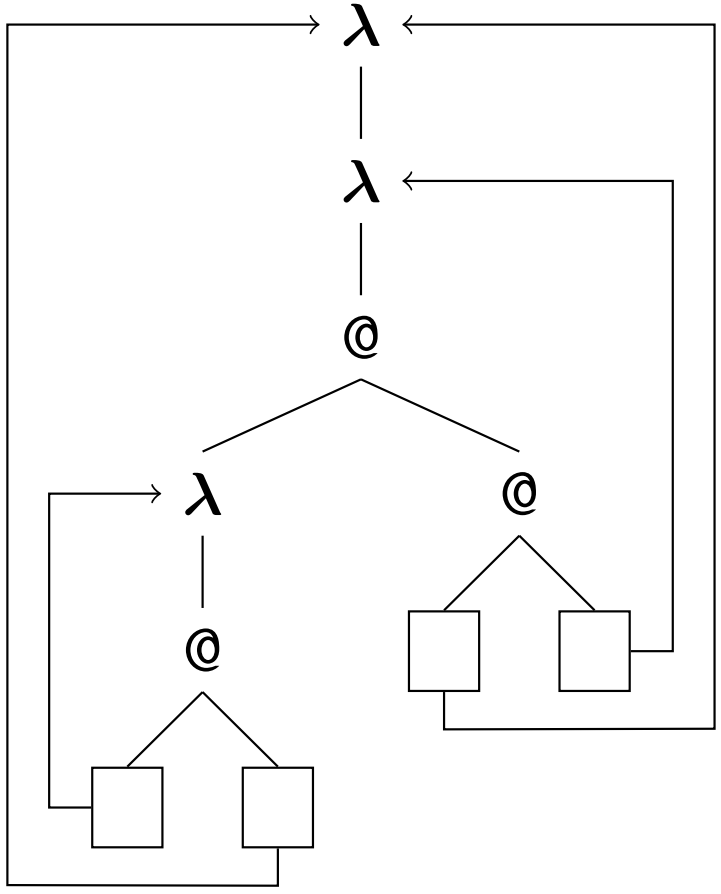
Our claim is that, albeit the differences in the surface syntax of $\boldsymbol{\lambda}$-calculus and Combinatory Logic, they are actually one and the same calculus (or algebra) which formalizes the abstract concept of computable function.

We reconcile the diffrences in the syntax by introducing a common syntactic extesion of the two calculi.

Church's syntax and Quine-Bourbaki notation (1)

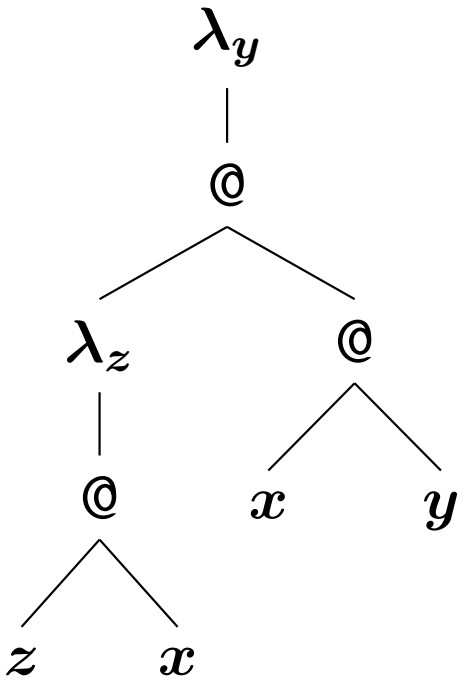
$$\lambda_x \lambda_y (\lambda_z (z\ x)^0\ (x\ y)^0)^0$$

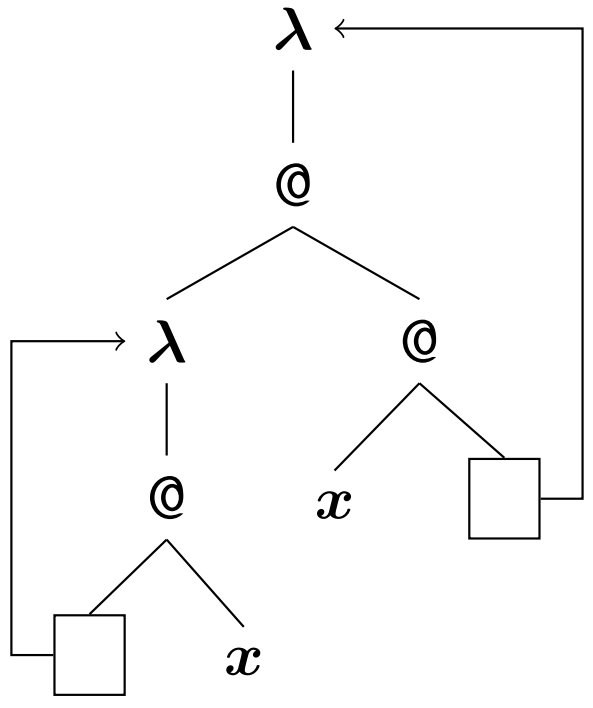




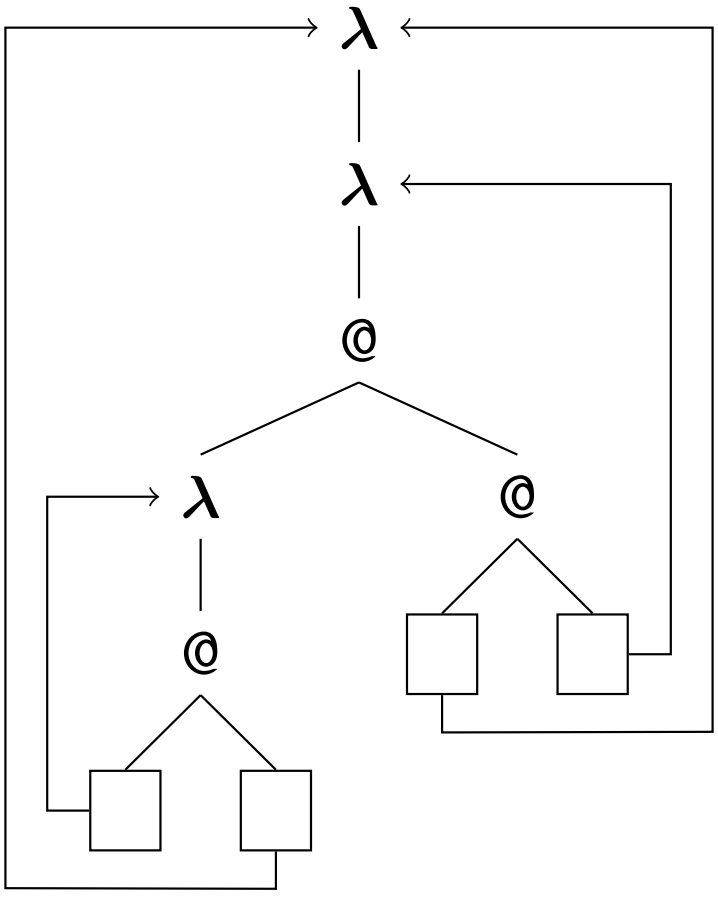
Church's syntax and Quine-Bourbaki notation (2)

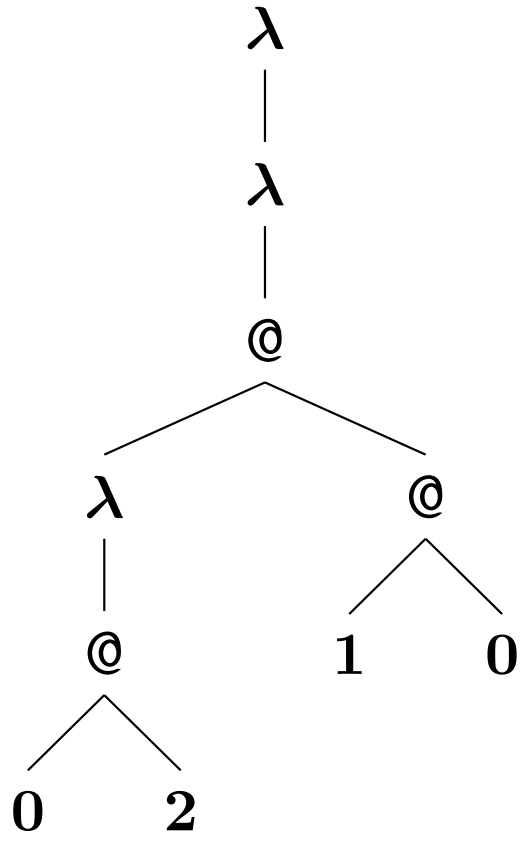
$$\lambda_y(\lambda_z(z\ x)^0\ (x\ y)^0)^0$$



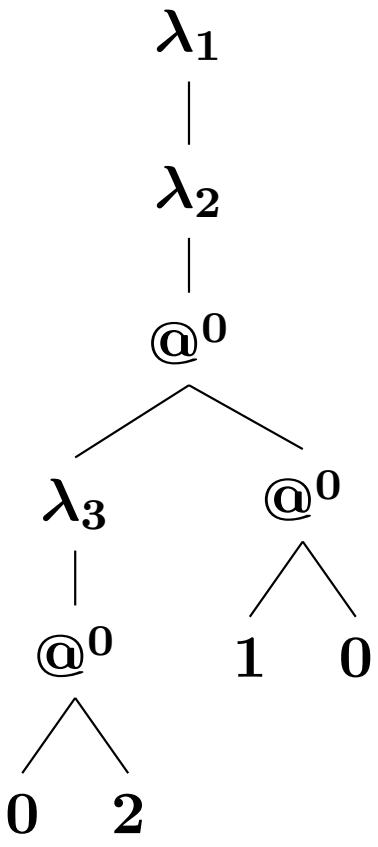


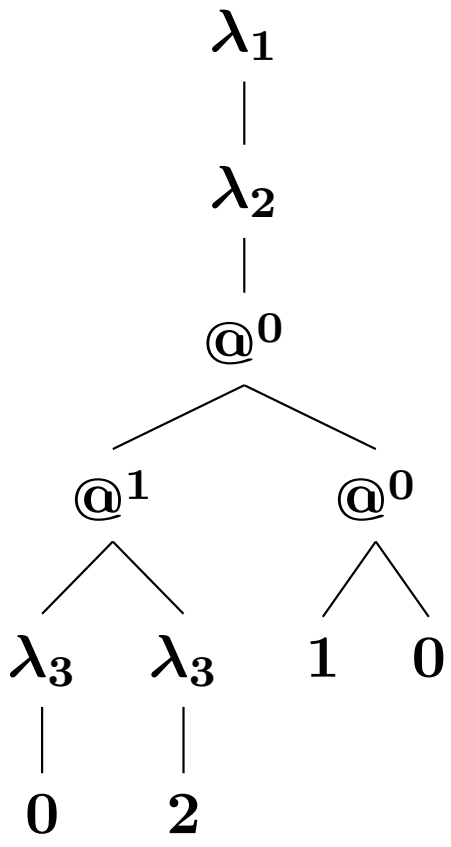
Quine-Bourbaki notation and de Bruijn notation

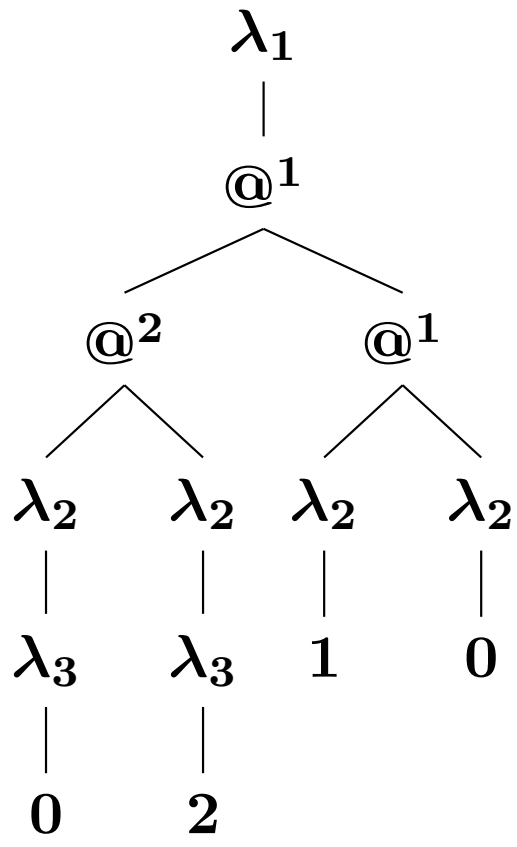




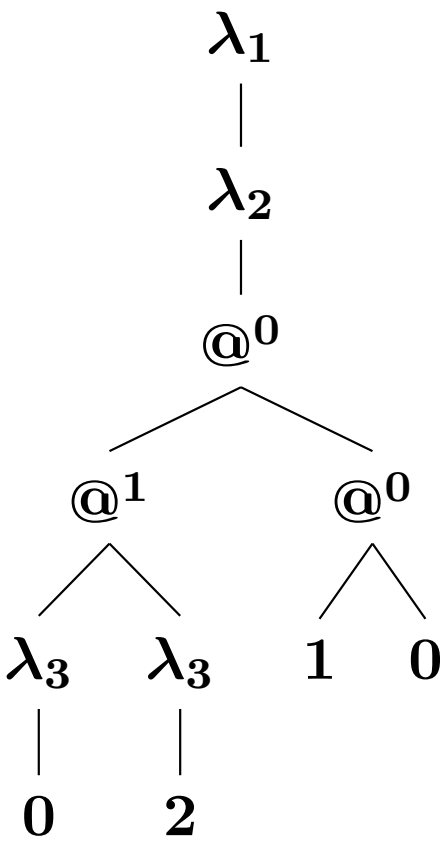
Generalized de Bruijn notation (1)

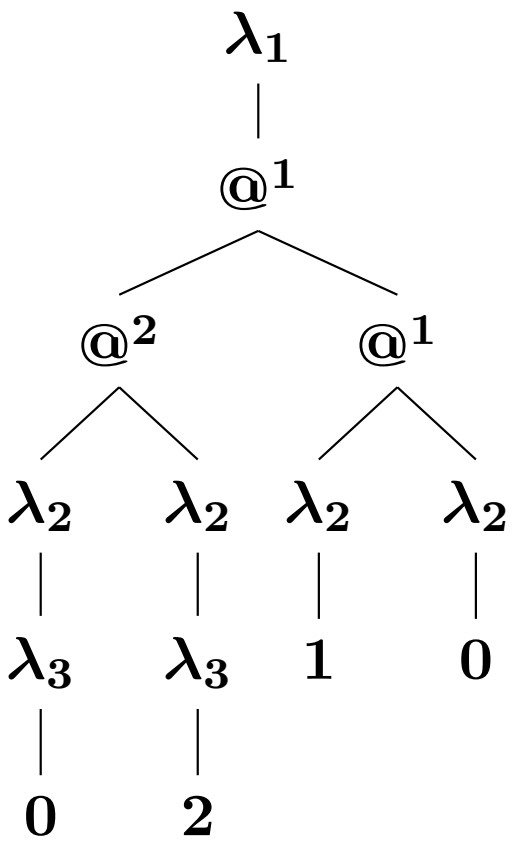


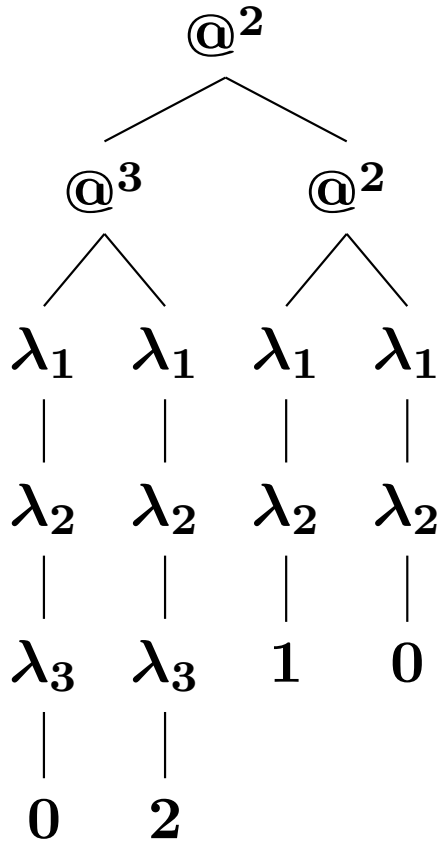




Generalized de Bruijn notation (2)



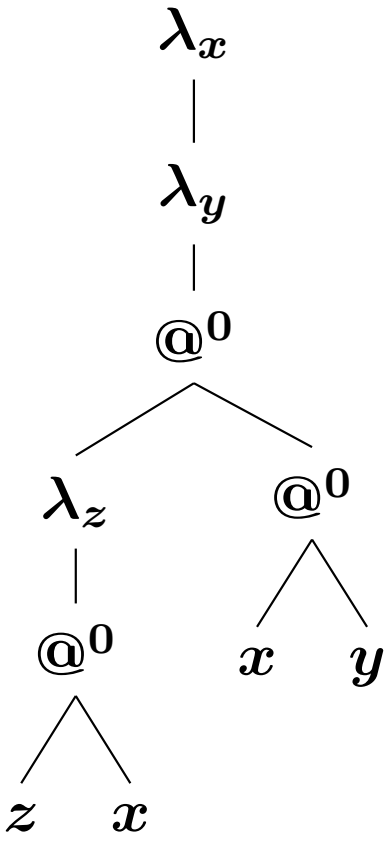


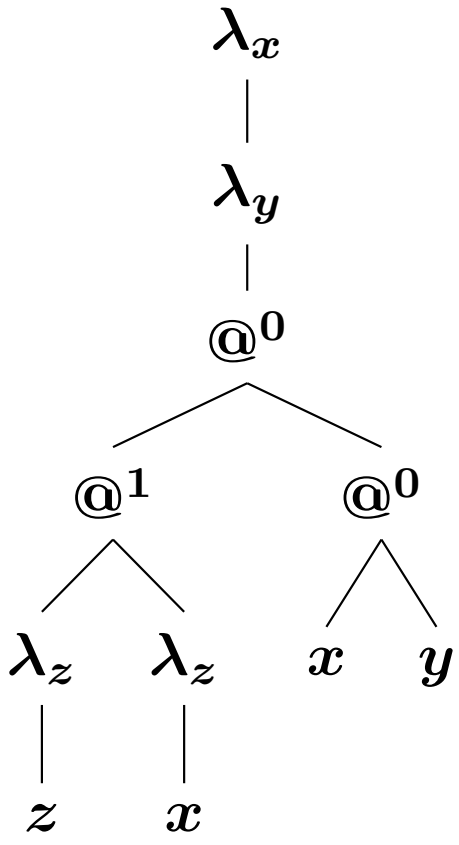


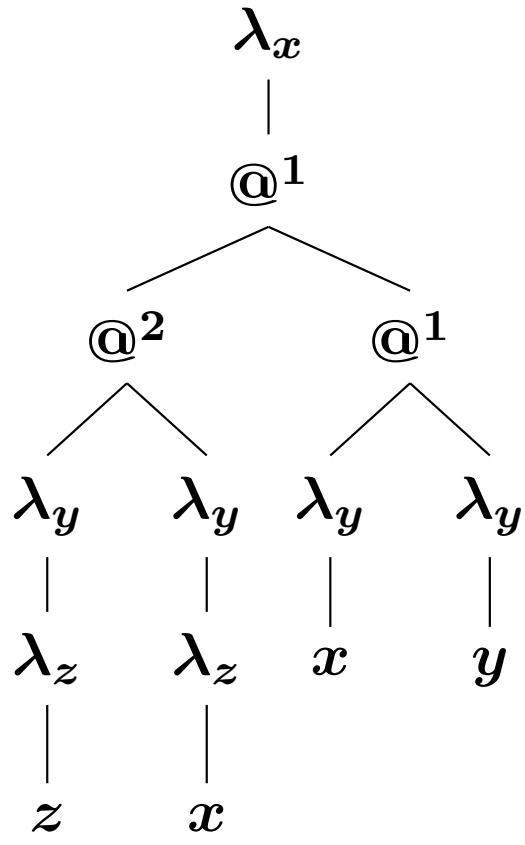
Nameless binder and distributive law

$$\lambda(D\ E)^n = (\lambda D\ \lambda E)^{n+1}$$

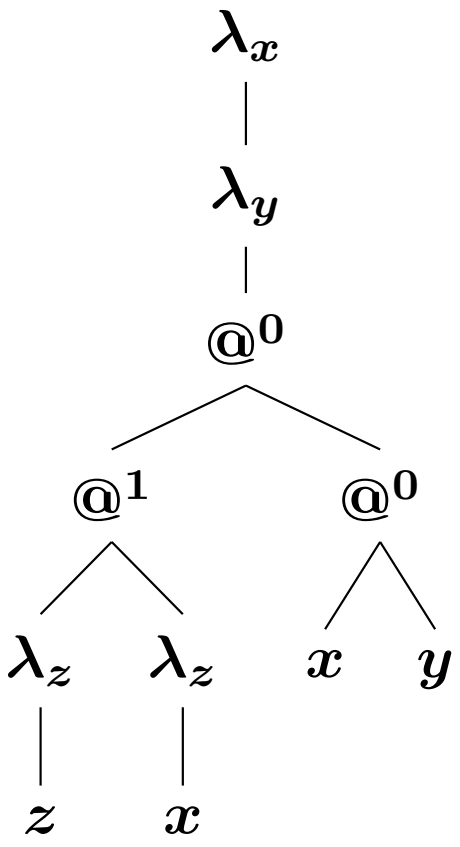
Generalized Church's syntax (1)

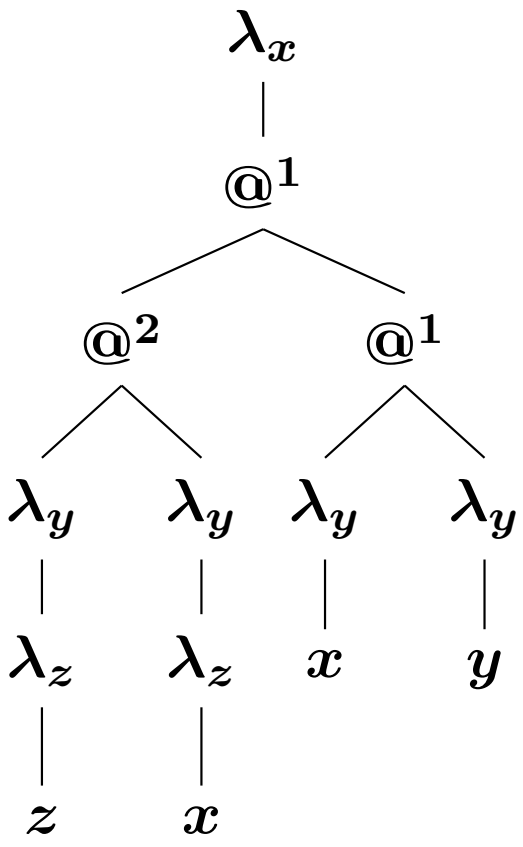


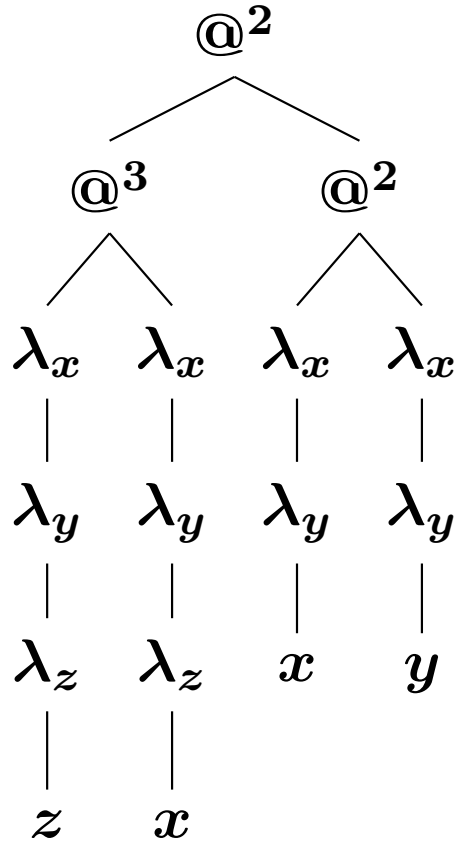




Generalized Church's syntax (2)

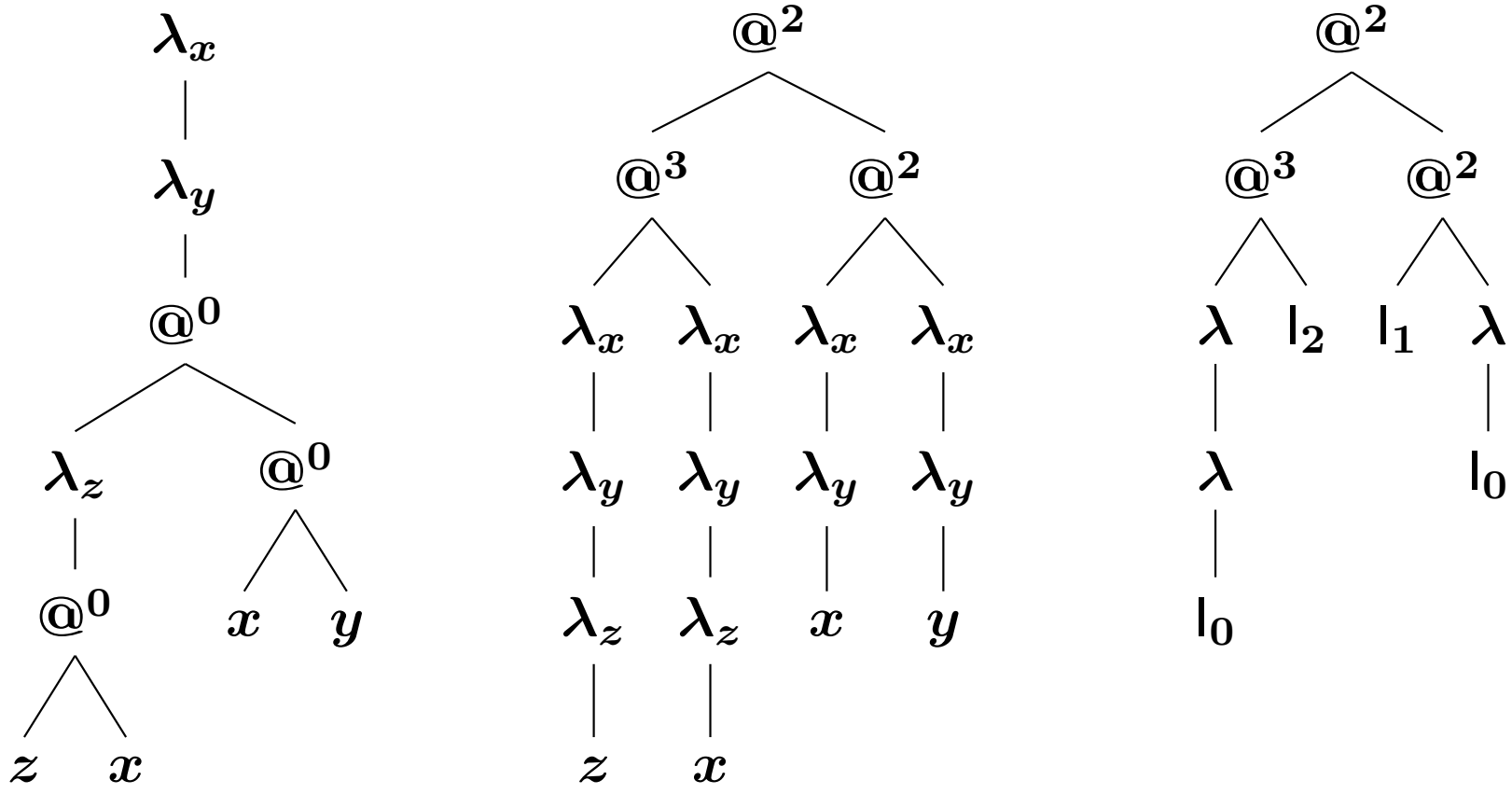






Distributive Law: $\lambda_x(D\ E)^n = (\lambda_x D\ \lambda_x E)^{n+1}$.

α-reduction



$$\lambda_x x \to_\alpha \mathsf{I}_0,\ \lambda_x\lambda_y x \to_\alpha \mathsf{I}_1,\ \lambda_x\lambda_y\lambda_z x \to_\alpha \mathsf{I}_2, \ldots$$

$$\lambda_x \mathsf{I}_k \to_\alpha \lambda \mathsf{I}_k,\ \lambda_x\lambda \mathsf{I}_k \to_\alpha \lambda\lambda \mathsf{I}_k,\ \lambda_x\lambda\lambda \mathsf{I}_k \to_\alpha \lambda\lambda\lambda \mathsf{I}_k, \ldots$$

α-reduction rules can compute α normal form.
To achieve this, we must extend Church's syntax!

Common extension of lambda calculus and combinatory logic

Definition (The datatypes $\mathbb{M}$, Λ and CL)

$$
\begin{array}{rcl}
M, N \in \mathbb{M} & ::= & x \mid \mathsf{I}_k \mid \lambda_x M \mid \lambda M \mid (M\ N)^i \quad (i, k \in \mathbb{N}) \\
M, N \in \Lambda & ::= & x \mid \lambda_x M \mid (M\ N)^0 \\
M, N \in \mathsf{CL} & ::= & x \mid \mathsf{I} \mid \mathsf{K} \mid \mathsf{S} \mid (M\ N)^0
\end{array}
$$

Combinators I, K and S are definable in $\mathbb{M}$ as abbreviations:

$$
\begin{array}{rcl}
\mathsf{I} & := & \mathsf{I}_0 \\
\mathsf{K} & := & \mathsf{I}_1 \\
\mathsf{S} & := & ((\mathsf{I}_2\ \lambda\lambda\mathsf{I}_0)^3\ (\lambda\mathsf{I}_1\ \lambda\lambda\mathsf{I}_0)^3)^3
\end{array}
$$

$\mathbb{M}$ as an extension of combinatory logic

In order to make

$$\boldsymbol{M, N} \in \mathbb{M} \ ::= \ \boldsymbol{x} \mid \mathsf{I}_{\boldsymbol{k}} \mid \boldsymbol{\lambda_x M} \mid \boldsymbol{\lambda M} \mid \boldsymbol{(M\ N)^i}$$

an extension of combinatory logic, we embedded the S combinator in $\mathbb{M}$ by the following informal computation.

$$\begin{aligned}
\mathsf{S} &= \boldsymbol{\lambda_{xyz}((x\ z)^0\ (y\ z)^0)^0} \\
&= \boldsymbol{((\lambda_{xyz}x\ \lambda_{xyz}z)^3\ (\lambda_{xyz}y\ \lambda_{xyz}z)^3)^3} \\
&= \boldsymbol{((\mathsf{I}_2\ \lambda\lambda\mathsf{I}_0)^3\ (\lambda\mathsf{I}_1\ \lambda\lambda\mathsf{I}_0)^3)^3}
\end{aligned}$$

However, Atsushi Igarashi noted that S may also be computed as follows (using η-conversion):

$$\begin{aligned}
\mathsf{S} &= \boldsymbol{\lambda_{xy}\lambda_z((x\ z)^0\ (y\ z)^0)^0} \\
&= \boldsymbol{\lambda_{xy}(\lambda_z(x\ z)^0\ \lambda_z(y\ z)^0)^1 = \lambda_{xy}(x\ y)^1} \\
&= \boldsymbol{(\lambda_{xy}x\ \lambda_{xy}y)^3 = (\mathsf{K}\ \lambda\mathsf{I})^3}
\end{aligned}$$

Definition (One step α-reduction on $\mathbb{M}$)

$$\frac{}{\lambda_x \lambda^i |_k \to_{1\alpha} \lambda^{i+1} |_k} \; \mathsf{E_1}$$

$$\frac{}{\lambda_x \lambda^i x \to_{1\alpha} |_i} \; \mathsf{E_2} \qquad \frac{x \neq y}{\lambda_x \lambda^i y \to_{1\alpha} \lambda^{i+1} y} \; \mathsf{E_3}$$

$$\frac{}{\lambda_* (M\ N)^i \to_{1\alpha} (\lambda_* M\ \lambda_* N)^{i+1}} \; \mathsf{D} \qquad \frac{M \to_{1\alpha} M'}{\lambda_* M \to_{1\alpha} \lambda_* M'} \; \mathsf{C_1}$$

$$\frac{M \to_{1\alpha} M'}{(M\ N)^i \to_{1\alpha} (M'\ N)^i} \; \mathsf{C_2} \qquad \frac{N \to_{1\alpha} N'}{(M\ N)^i \to_{1\alpha} (M\ N')^i} \; \mathsf{C_3}$$

Definition (α-nf)

M is an α-nf if M cannot be simplified by one step α-reduction.

Example

This example shows how the variable-binders $\boldsymbol{\lambda_x}$ and $\boldsymbol{\lambda_y}$ are eliminated by one step $\boldsymbol{\alpha}$-reductions.

$$
\begin{aligned}
\boldsymbol{\lambda_x \lambda_y (y\ x)^0} &\boldsymbol{\to_{1\alpha}\ \lambda_x (\lambda_y y\ \lambda_y x)^1} \\
&\boldsymbol{\to_{1\alpha}\ \lambda_x (\mathsf{I}\ \lambda_y x)^1} \\
&\boldsymbol{\to_{1\alpha}\ \lambda_x (\mathsf{I}\ \lambda x)^1} \\
&\boldsymbol{\to_{1\alpha}\ (\lambda_x \mathsf{I}\ \lambda_x \lambda x)^2} \\
&\boldsymbol{\to_{1\alpha}\ (\lambda \mathsf{I}\ \lambda_x \lambda x)^2} \\
&\boldsymbol{\to_{1\alpha}\ (\lambda \mathsf{I}\ \mathsf{K})^2} \quad \square
\end{aligned}
$$

Remark

Every $\boldsymbol{M} \in \mathbb{M}$ can be reduced to a unique $\boldsymbol{\alpha}$-nf, and we will write $\boldsymbol{M_\alpha}$ for it.

The datatype $\mathbb{L}$

We will write $\mathbb{L}$ for the following subset of $\mathbb{M}$.

$$\mathbb{L} := \{M \in \mathbb{M} \mid M \text{ is an } \alpha\text{-nf}\}$$

We can also define $\mathbb{L}$ directly by the following grammar (inductive defitnition).

Definition (The datatypes $\mathbb{T}$ and $\mathbb{L}$)

$$\begin{aligned} t \in \mathbb{T} &::= \lambda^i \mathsf{I}_k \mid \lambda^i x \\ M, N \in \mathbb{L} &::= t \mid (M\ N)^i \end{aligned}$$

Elements of $\mathbb{T}$ are called threads.

$\mathbb{L}$ as a Combinatory Logic

Definition (The datatypes $\mathbb{T}$ and $\mathbb{L}$)

$$t \in \mathbb{T} \ ::= \lambda^i \mathsf{I}_k \mid \lambda^i x$$
$$M, N \in \mathbb{L} \ ::= \ t \mid (M \ N)^i$$

By observing that

1. The fresh binder λ is a disguised form of K.
2. $\mathsf{I}_{n+1} = \lambda_x(\mathsf{K}\ \mathsf{K}^n x)^0 = (\lambda_x \mathsf{K}\ \lambda_x \mathsf{K}^n x)^1 = ((\mathsf{K}\ \mathsf{K})^0\ \mathsf{I}_n)^1$

we can replace the above definition by the following definition.

Definition (The datatype $\mathbb{L}$)

$$M, N \in \mathbb{L} \ ::= \ x \mid \mathsf{I} \mid \mathsf{K} \mid (M \ N)^i$$

α-reduction

Definition (α-reduction on $\mathbb{M}$ and α-equality)

$$\frac{M_0 \to_{1\alpha} M_1 \quad M_1 \to_{1\alpha} M_2 \quad \cdots \quad M_{n-1} \to_{1\alpha} M_n}{M_0 \to_\alpha M_n}$$

When we have $M_0 \to_\alpha M_n$ by this rule, we say that M_0 *α-reduces to M_n in n steps*.

$$\frac{M \to_\alpha P \quad N \to_\alpha P}{M =_\alpha N}$$

$=_\alpha$ is a decidable equivalence relation

Theorem

Given any $\mathbb{M}$-term M, there uniquely exists an N such that $M \to_\alpha N$ and N is an α-nf.

Remark

1. $(-)_{\alpha} : \mathbb{M} \to \mathbb{M}$ is idempotent, i.e., $(M_{\alpha})_{\alpha} = M_{\alpha}$ and image of $(-)_{\alpha}$ is $\mathbb{L}$.
2. For any $M \in \mathbb{M}$, $M =_{\alpha} M_{\alpha}$.
3. For any $M \in \mathbb{M}$, $M = M_{\alpha}$ iff $M \in \mathbb{L}$.
4. $M =_{\alpha} N$ iff $M_{\alpha} = N_{\alpha}$.

Thus M_{α} is a natural representative of the equivalence class $\{N \in \mathbb{M} \mid N =_{\alpha} M\}$ containing M.

Moreover, we can think of

$$(-)_{\alpha} : \mathbb{M} \to \mathbb{L}$$

as a semantic function which translates $\mathbb{M}$-terms to $\mathbb{L}$-terms. In other words $\mathbb{M}$ extends $\mathbb{L}$ by providing abbreviations (macros, syntax sugar) for $\mathbb{L}$ terms.

Abstraction operator in $\mathbb{L}$

In $\mathbb{L}$, we can introduce the abstraction operation:

$$\boldsymbol{\lambda}^* : \mathbb{X} \times \mathbb{L} \to \mathbb{L}$$

as a macro as follows.

$$\begin{aligned}
\boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{\lambda}^{\boldsymbol{i}}\mathsf{I}_{\boldsymbol{k}} &:= \boldsymbol{\lambda}^{\boldsymbol{i+1}}\mathsf{I}_{\boldsymbol{k}} \\
\boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{\lambda}^{\boldsymbol{i}}\boldsymbol{x} &:= \mathsf{I}_{\boldsymbol{i}} \\
\boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{\lambda}^{\boldsymbol{i}}\boldsymbol{y} &:= \boldsymbol{\lambda}^{\boldsymbol{i+1}}\boldsymbol{y} \text{ if } \boldsymbol{x} \neq \boldsymbol{y} \\
\boldsymbol{\lambda}^*_{\boldsymbol{x}}(\boldsymbol{M}\ \boldsymbol{N})^{\boldsymbol{n}} &:= (\boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{M}\ \boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{N})^{\boldsymbol{n+1}}
\end{aligned}$$

Recall that, for CL, it was defined by:

$$\begin{aligned}
\boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{x} &:= \mathsf{I} \\
\boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{y} &:= (\mathsf{K}\ \boldsymbol{y})^{\mathbf{0}} \text{ if } \boldsymbol{x} \neq \boldsymbol{y} \\
\boldsymbol{\lambda}^*_{\boldsymbol{x}}(\boldsymbol{M}\ \boldsymbol{N})^{\mathbf{0}} &:= ((\mathsf{S}\ \boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{M})^{\mathbf{0}}\ \boldsymbol{\lambda}^*_{\boldsymbol{x}}\boldsymbol{N})^{\mathbf{0}}
\end{aligned}$$

Conclusion: $\mathbb{M}$ and $\mathbb{L}$

We may think of $\mathbb{M}$ as a common notation system for both $\boldsymbol{\lambda}$-calculus and Combinatory Logic, and its sublanguage $\mathbb{L}$ as a notation system for the pure Combinatory Logic.

$$\begin{aligned} \boldsymbol{M, N} \in \mathbb{M} &::= \boldsymbol{x} \mid \mathsf{I}_{\boldsymbol{k}} \mid \boldsymbol{\lambda_x M} \mid \boldsymbol{\lambda M} \mid \boldsymbol{(M\ N)^i} \\ \boldsymbol{t} \in \mathbb{T} &::= \boldsymbol{\lambda^i \mathsf{I}_k} \mid \boldsymbol{\lambda^i x} \\ \boldsymbol{M, N} \in \mathbb{L} &::= \boldsymbol{t} \mid \boldsymbol{(M\ N)^i} \end{aligned}$$

In $\mathbb{L}$ we can have the best of both $\boldsymbol{\lambda}$-calculus and Combinatory Logic. For example, substitution is always capture free, and proof of CR for $\mathbb{L}$ easily implies proof of CR for $\mathbb{M}$ (and hence for $\boldsymbol{\Lambda}$).

Height of $\mathbb{L}$-terms

Definition (Height (Ht) of $\mathbb{L}$-terms)

$$\begin{aligned}
\mathsf{Ht}(\boldsymbol{\lambda}^{\boldsymbol{i}}\mathsf{I}_{\boldsymbol{k}}) &:= \boldsymbol{i} + \boldsymbol{k} + \mathbf{1} \\
\mathsf{Ht}(\boldsymbol{\lambda}^{\boldsymbol{i}}\boldsymbol{x}) &:= \boldsymbol{i} \\
\mathsf{Ht}((\boldsymbol{M}\ \boldsymbol{N})^{\boldsymbol{i}}) &:= \mathbf{min}\{\boldsymbol{i}, \mathsf{Ht}(\boldsymbol{M}), \mathsf{Ht}(\boldsymbol{N})\}
\end{aligned}$$

Instantiation

Definition (Instantiation of threads at level n)

If $t \in T^{n+1}$ and $u \in T^n$, then $\langle t\ u \rangle^n$ can be computed by the following equations.

$$\langle \lambda^i \mathsf{I}_k\ \lambda^j \mathsf{I}_\ell \rangle^n := \begin{cases} \lambda^{i-1} \mathsf{I}_k & \text{if } n < i, \\ \lambda^{j+k} \mathsf{I}_\ell & \text{if } n = i \leq j, \\ \lambda^{j} \mathsf{I}_{\ell+k} & \text{if } n = i > j, \\ \lambda^{i} \mathsf{I}_{k-1} & \text{if } n > i. \end{cases}$$

$$\langle \lambda^i \mathsf{I}_k\ \lambda^j x \rangle^n := \begin{cases} \lambda^{i-1} \mathsf{I}_k & \text{if } n < i, \\ \lambda^{j+k} x & \text{if } n = i, \\ \lambda^{i} \mathsf{I}_{k-1} & \text{if } n > i. \end{cases}$$

$$\langle \lambda^i x\ t \rangle^n := \lambda^{i-1} x$$

Instantiation at level n

Define lift $\uparrow_n^k : \mathbb{L}^n \to \mathbb{L}^{n+k}$ by

$$\uparrow_n^k \lambda^j \mathsf{I}_\ell := \begin{cases} \lambda^{j+k}\mathsf{I}_\ell & \text{if } n \leq j, \\ \lambda^j \mathsf{I}_{\ell+k} & \text{if } n > j. \end{cases}$$

$$\uparrow_n^k \lambda^j x := \lambda^{j+k} x$$

$$\uparrow_n^k (M\ N)^j := (\uparrow_n^k M\ \uparrow_n^k N)^{j+k}.$$

Definition (Instantiation at level n)

If $M \in \mathbb{L}^{n+1}$ and $P \in \mathbb{L}^n$, then $\langle M\ P\rangle^n$ is defined by the following equations.

$$\langle \lambda^i \mathsf{I}_k\ P\rangle^n := \begin{cases} \lambda^{i-1}\mathsf{I}_k & \text{if } n < i, \\ \uparrow_n^k P & \text{if } n = i, \\ \lambda^i \mathsf{I}_{k-1} & \text{if } n > i. \end{cases}$$

$$\langle \lambda^i x\ P\rangle^n := \lambda^{i-1} x.$$

$$\langle (M\ N)^{i+1}\ P\rangle^n := (\langle M\ P\rangle^n\ \langle N\ P\rangle^n)^i.$$

Acknowledgement

We thank the Japan Society for the Promotion of Science (JSPS), Core-to-Core Program (A. Advanced Research Networks) for supporting the research.