Unification of the Lambda-Calculus and Combinatory Logic

Masahiko Sato
Graduate School of Informatics, Kyoto University

j.w.w. Takafumi Sakurai and Helmut Schwichtenberg

TPP 2018
RIEC, Tohoku University
November 21, 2018

Objects, Concepts and Notations

- Natural numbers: 0, 1, 2+3, $\cdots$
- Rational numbers: $1/3$, $2/6$, $\cdots$
- Real numbers: Gray-code, Signed digit code, $\cdots$
- Computable functions: $\boldsymbol{\lambda}_{\boldsymbol{x}}\boldsymbol{x}$, I, $\boldsymbol{\lambda}_{\boldsymbol{xy}}\boldsymbol{x}$, K, $\cdots$

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 λ-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 $\boldsymbol{\lambda}$-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.

The syntax of the Lambda Calculus and Combinatory Logic

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

$(M\ N)^0$ 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-conversion 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

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

These rules suggest the following identities.

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

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 λ^* enjoys the following property.

$$(\lambda^*_x M\ N)^0 \to [x := N]M$$

So, CL can simulate the β-reduction rule of the λ-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 λ-calculus.

Recall the syntax of Λ and CL.

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

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:
  $$(\mathsf{S}\ M)^0 \to \boldsymbol{\lambda}_{yz}((M\ z)^0\ (y\ z)^0)^0.$$
  This means that the $\boldsymbol{\lambda}$-calculus has a finer computational granularity.
- While (free) 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 λ-calculus and Combinatory Logic, they are actually one and the same calculus (or algebra) which formalizes the abstract concept of computable function.

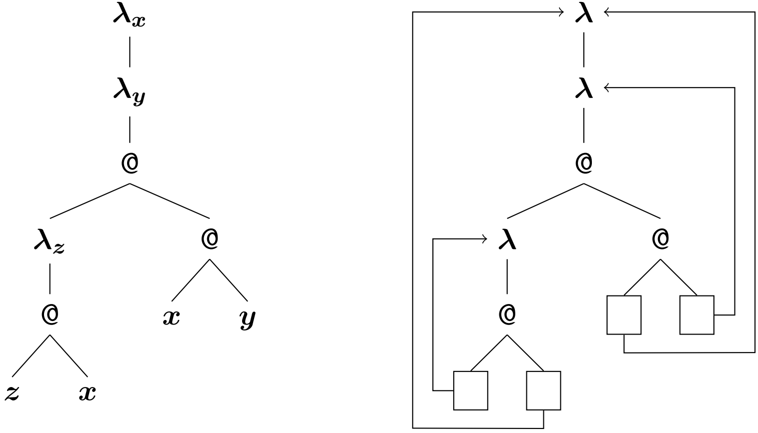
Our Claim

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.

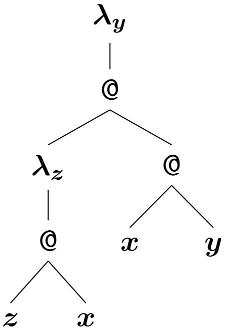
Church's syntax and Quine-Bourbaki notation (1)

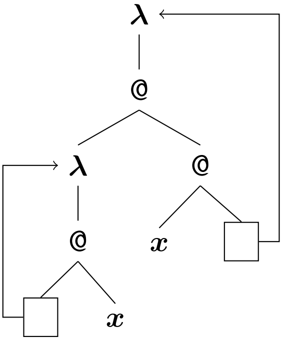
$$\lambda_x \lambda_y (\lambda_z (z\ x)^0\ (x\ y)^0)^0$$



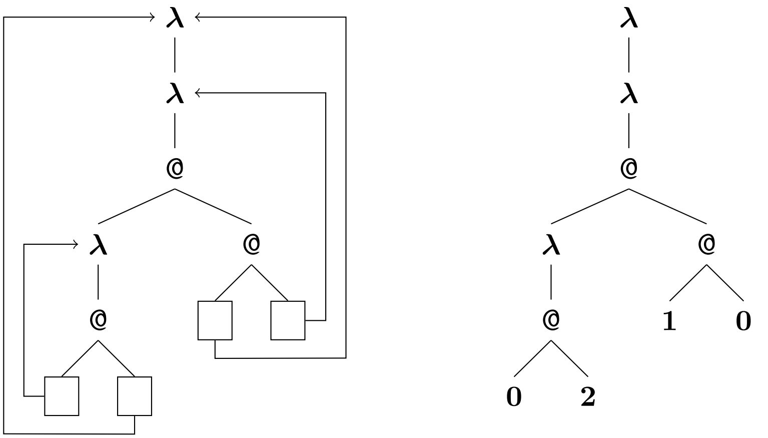
Church's syntax and Quine-Bourbaki notation (2)

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

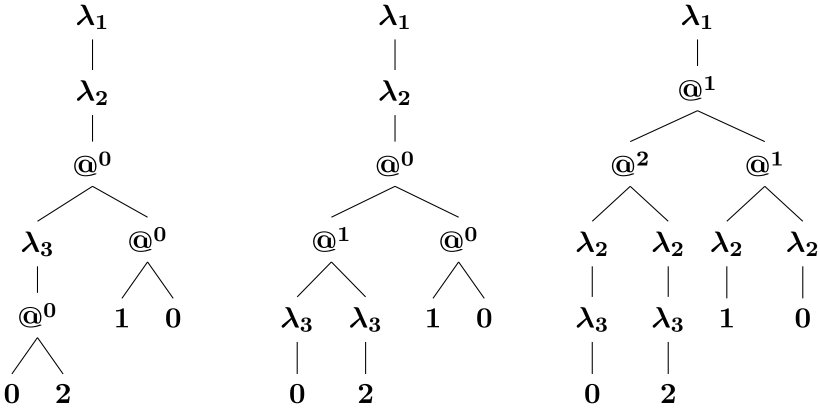




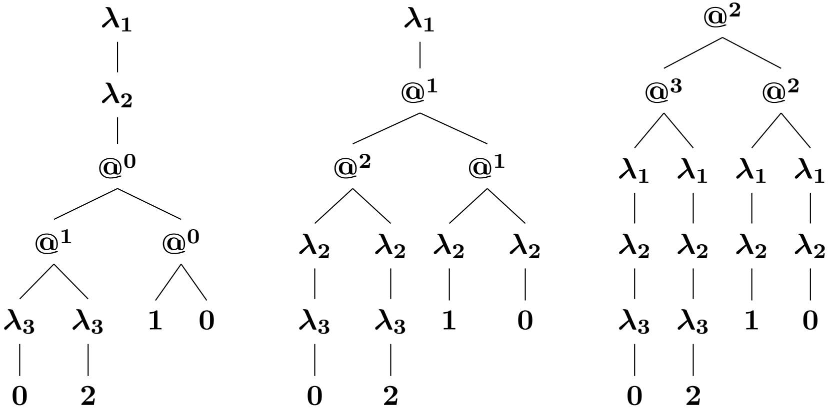
Quine-Bourbaki notation and de Bruijn notation



Generalized de Bruijn notation (1)



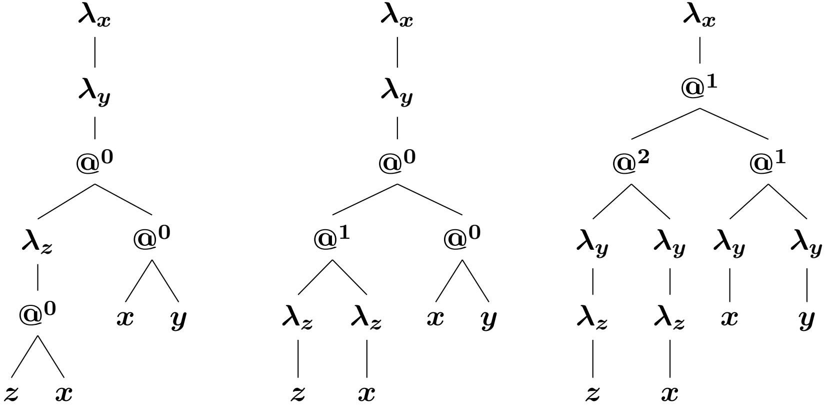
Generalized de Bruijn notation (2)



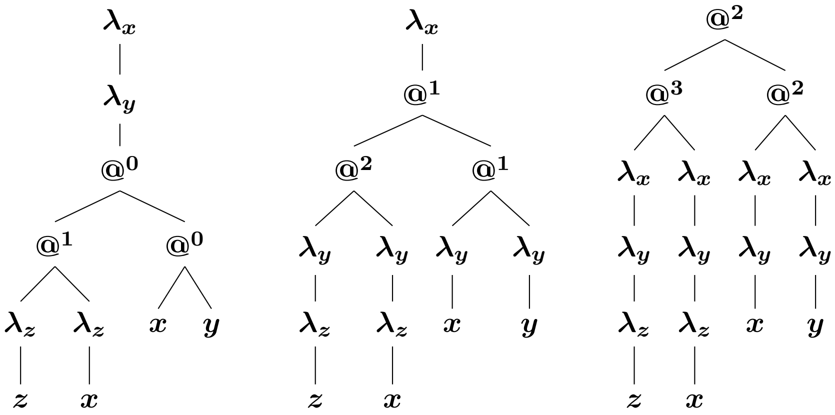
Nameless binder and distributive law

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

Generalized Church's syntax (1)

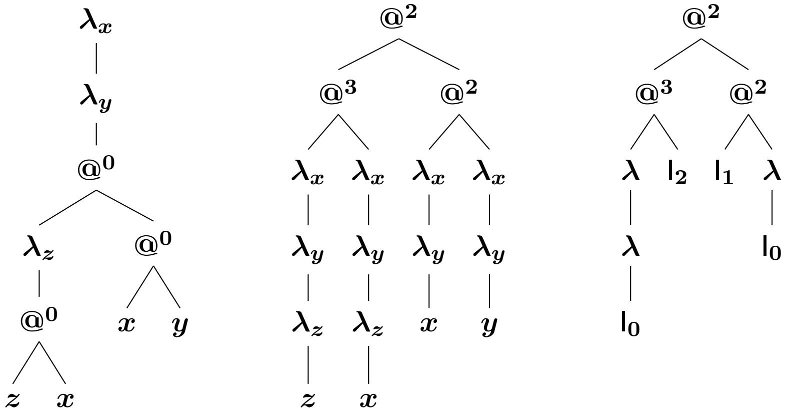


Generalized Church's syntax (2)



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 $\mathbb{M}$, $\boldsymbol{\Lambda}$ and CL)

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

We will write $\mathbb{M}_0$ for the subset $\{M \in \mathbb{M} \mid M \text{ is closed}\}$ of $\mathbb{M}$.
Combinators I, K and S are definable in $\mathbb{M}_0$ as abbreviations:

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

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

In order to make

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

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

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

α-reduction

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

$$\frac{}{\lambda_x \lambda^i \mathsf{l}_k \to_{1\alpha} \lambda^{i+1} \mathsf{l}_k}\ \mathsf{E}_1$$

$$\frac{}{\lambda_x \lambda^i x \to_{1\alpha} \mathsf{l}_i}\ \mathsf{E}_2 \qquad \frac{x \neq y}{\lambda_x \lambda^i y \to_{1\alpha} \lambda^{i+1} y}\ \mathsf{E}_3$$

$$\frac{}{\lambda_*(M\ N)^i \to_{1\alpha} (\lambda_* M\ \lambda_* N)^{i+1}}\ \mathsf{D} \qquad \frac{M \to_{1\alpha} M'}{\lambda_* M \to_{1\alpha} \lambda_* M'}\ \mathsf{C}_1$$

$$\frac{M \to_{1\alpha} M'}{(M\ N)^i \to_{1\alpha} (M'\ N)^i}\ \mathsf{C}_2 \qquad \frac{N \to_{1\alpha} N'}{(M\ N)^i \to_{1\alpha} (M\ N')^i}\ \mathsf{C}_3$$

Definition (α-nf)

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

α-reduction (cont.)

Example

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

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

α-reduction (cont.)

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.
- We have $\boldsymbol{\alpha}$-equality $=_{\boldsymbol{\alpha}}$ by writing $\boldsymbol{M} =_{\boldsymbol{\alpha}} \boldsymbol{N}$ for $\boldsymbol{M_\alpha} = \boldsymbol{N_\alpha}$.
- The $\boldsymbol{\alpha}$-normalizing function $(-)_{\boldsymbol{\alpha}} : \mathbb{M} \to \mathbb{M}$ is idempotent, and we will write $\mathbb{L}$ for its image.
- Traditional $\boldsymbol{\lambda}$-calculus studied the structure of the setoid $(\boldsymbol{\Lambda}, =_{\boldsymbol{\alpha}})$. We will work on $\mathbb{L}$ which is a pure datatype, free from the concept of $\boldsymbol{\alpha}$-equality.

The datatype $\mathbb{L}$

We have written $\mathbb{L}$ 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 definition).

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.

The datatype $\mathbb{L}_0$

Recall that $\mathbb{M}$ is defined by:

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

We will write $\mathbb{L}_0$ for the following subset of $\mathbb{M}_0$.

$$\mathbb{L}_0 := \{M \in \mathbb{M}_0 \mid M \text{ is an } \alpha\text{-nf}\}$$

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

$$M, N \in \mathbb{L}_0 \ ::= \ \lambda^i \mathsf{I}_k \mid (M\ N)^i$$

If we write I_k^i for $\lambda^i \mathsf{I}_k$, we have

$$M, N \in \mathbb{L}_0 \ ::= \ \mathsf{I}_k^i \mid (M\ N)^i$$

Height and Thickness of $\mathbb{L}_0$-terms

$$M, N \in \mathbb{L}_0 \ ::= \ \mathsf{I}_k^i \mid (M \ N)^i$$

Definition (Height (Ht) and Thickness (Th) of $\mathbb{L}_0$-terms)

$$\begin{aligned}
\mathsf{Ht}(\mathsf{I}_k^i) &:= i + k + 1, \\
\mathsf{Ht}((M \ N)^i) &:= i. \\
\mathsf{Th}(\mathsf{I}_k^i) &:= 0, \\
\mathsf{Th}((M \ N)^i) &:= \mathsf{Th}(M) + \mathsf{Th}(N) + 1.
\end{aligned}$$

Remark

Thickness of M is obtained by counting the number of applications in M. Since all the $\mathbb{L}_0$ terms are constructed from natural numbers by projections and applications, all the metamathematical arguments about $\mathbb{L}_0$ boil down to arguments about natural numbers $(=, <, +)$.

Well-formed $\mathbb{L}_0$-terms

We define well-formed $\mathbb{L}_\mathbf{0}$-terms inductively as follows.

1. I^{i}_{k} is well-formed.
2. $(M\ N)^{i}$ is well-formed, if M, N are well-formed, $\mathsf{Ht}(M) \geq i$ and $\mathsf{Ht}(N) \geq i$,

We will write $\mathbb{L}^{+}_{\mathbf{0}}$ for the set of well-formed $\mathbb{L}_\mathbf{0}$-terms. Well-formed $\mathbb{L}_\mathbf{0}$-terms exacltly correspond to traditional closed $\boldsymbol{\lambda}$-terms. Namely, give an $\mathbb{L}_\mathbf{0}$-term M, it is the $\boldsymbol{\alpha}$-nf of a closed $\boldsymbol{\lambda}$-term.

Well-formed $\mathbb{L}_0$-terms (cont.)

We will study the $\boldsymbol{\lambda}$-calculus enitrely working within the set $\mathbb{L}_{\mathbf{0}}^{+}$ of well-formed $\mathbb{L}_{\mathbf{0}}$-terms. We will do this in the following order:

1. Eliminate $\boldsymbol{\xi}$-rule.
2. Eliminate $\boldsymbol{\eta}$-equality. Just as we defined $\boldsymbol{\alpha}$-equality on $\mathbb{M}$, we define $\boldsymbol{\eta}$-equality on $\mathbb{L}_{\mathbf{0}}$ and will work on the setoid $\mathbb{L}_{\mathbf{0}\boldsymbol{\eta}} := (\mathbb{L}_{\mathbf{0}}, =_{\boldsymbol{\eta}})$.
3. Reformulate $\boldsymbol{\beta}$-rule by eliminating substitution and introducing instantiation. The reformulated $\boldsymbol{\beta}$-rule allows us to apply the rule without the need of applying $\boldsymbol{\xi}$-rule first.

Elimination of ξ-rule

The $\boldsymbol{\xi}$-rule in the $\boldsymbol{\lambda}$-calculus is the following rule:

$$\frac{\Gamma, x \vdash M \rightarrow_{\beta} N}{\Gamma \vdash \lambda_x M \rightarrow_{\beta} \lambda_x N}$$

In case we can assign simple types to terms, the rule becomes:

$$\frac{\Gamma, x : \sigma \vdash M : \tau \rightarrow_{\beta} N : \tau}{\Gamma \vdash \lambda_x M : \sigma \supset \tau \rightarrow_{\beta} \lambda_x N : \sigma \supset \tau}$$

But, we have no variables in $\mathbb{L}_{\mathbf{0}}^{+}$. So we cannot even formulate the $\boldsymbol{\xi}$-rule! The real problem is how can we develop $\boldsymbol{\lambda}$-calculus without $\boldsymbol{\xi}$? We can solve this problem by moving from Gentzen-Martin-Löf style hypothetical judgements to Frege-Hilbert style categorical judgments.

Elimination of ξ-rule (cont.)

In the framework of hyothetical judgements, the implication introduction rule is:

$$\frac{\boldsymbol{\Gamma, \sigma \vdash \tau}}{\boldsymbol{\Gamma \vdash \sigma \supset \tau}}$$

But, in the framework of categorical judgments, it bocomes trivial:

$$\frac{\vdash \boldsymbol{\Gamma} \supset \boldsymbol{\sigma} \supset \boldsymbol{\tau}}{\vdash \boldsymbol{\Gamma} \supset \boldsymbol{\sigma} \supset \boldsymbol{\tau}}$$

We can see the equivalence of these formulations by deduction theorem of propositional logic. In the same way, we can develop $\boldsymbol{\lambda}$ without ξ-rule.

η-equality

In the $\boldsymbol{\lambda}$-calculus, $\boldsymbol{\eta}$-conversion rule is:

$$\boldsymbol{\Gamma \vdash \lambda_x (M\ x)^0 \rightarrow_\eta M}$$

where $\boldsymbol{x}$ is not free in $\boldsymbol{M}$.

In $\mathbb{L}_{\mathbf{0}}^{+}$, noting that $\boldsymbol{\lambda_x (M\ x)^0 = (\lambda_x M\ \lambda_x x)^1 = (\lambda M\ \mathsf{I}_0)^1}$, the rule becomes:

$$\boldsymbol{\Gamma \vdash (\lambda M\ \mathsf{I})^1 \rightarrow_\eta M}$$

It is necessary to rewrite this in the form of categorical judgment, but we skip the details. After showing confluence of $\boldsymbol{\eta}$-reduction, we can introduce $\boldsymbol{\eta}$-equalrity $\boldsymbol{=_\eta}$ as an equivalence relation on $\mathbb{L}_{\mathbf{0}}^{+}$.

In traditional $\boldsymbol{\lambda}$-calculus, $\boldsymbol{\eta}$-conversion is introduced after $\boldsymbol{\beta}$-conversion is introduced. However since we think that extensionality is an essential property of (computable) functions, we introduced $\boldsymbol{\eta}$-equality before introducing $\boldsymbol{\beta}$-conversion.

β-conversion

In the $\boldsymbol{\lambda}$-calculus, $\boldsymbol{\beta}$-conversion rule is:

$$\boldsymbol{\Gamma \vdash (\lambda_x M\ P)^0 \rightarrow_\beta [x := P]M}$$

In $\mathbb{L}_{\mathbf{0}}^{+}$, we replace substitution by instantiation ($\langle \boldsymbol{M\ P} \rangle^{\boldsymbol{i}}$) and define the rule by:

$$\boldsymbol{\Gamma \vdash (M\ P)^0 \rightarrow_\beta \langle M\ P \rangle^0}$$

where $\mathrm{Ht}(\boldsymbol{M}) > \mathbf{0}$. Rewriting it into catgorical judgment form, we have:

$$\boldsymbol{\vdash (M\ P)^i \rightarrow_\beta \langle M\ P \rangle^i}$$

where $\mathrm{Ht}(\boldsymbol{M}) > \boldsymbol{i}$. So, our $\boldsymbol{\beta}$-rule can be applied directly under $\boldsymbol{\lambda}$-binders without using $\boldsymbol{\xi}$-rule.

Instantiation at level n

If $M, P \in \mathbb{L}_0^+$ and $\mathrm{Ht}(M) > n$, then $\langle M\ P\rangle^n$ is defined by the following equations.

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

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

Lift $\uparrow_n^k$ is defined by

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

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

Current status and future plan

- We have defined and proved most of the results in this talk in Minlog. For example, we proved Church-Rosser property of $\mathbb{L}_{\boldsymbol{\beta}}$ by the residual method.
- Several properties of $\boldsymbol{\eta}$-equality are still to be formally proved.
- It is easy to internalize instantiation operation. By internalizing it we expect to have a natural first-order axiomatization of $\boldsymbol{\lambda_\beta}$-calculus.
- Formally prove the expected connection between $\mathbb{M}_{\mathbf{0}}$ and $\mathbb{L}_{\mathbf{0}}$

Conclusion: $\mathbb{M}$ and $\mathbb{L}_{\mathbf{0}}^{+}$

We may think of $\mathbb{M}$ as a common notation system for both $\boldsymbol{\lambda}$-calculus and Combinatory Logic, and its sublanguage $\mathbb{L}_{\mathbf{0}}^{+}$ as a notation system for the pure Combinatory Logic.

$$\boldsymbol{M, N} \in \mathbb{M} \;::=\; \boldsymbol{x} \mid \mathsf{I}_{\boldsymbol{k}} \mid \boldsymbol{\lambda_x M} \mid \boldsymbol{\lambda M} \mid \boldsymbol{(M\ N)^{i}}$$
$$\boldsymbol{M, N} \in \mathbb{L}_{\mathbf{0}}^{+} \;::=\; \mathsf{I}_{\boldsymbol{k}}^{\boldsymbol{i}} \mid \boldsymbol{(M\ N)^{i}}$$

In $\mathbb{L}_{\mathbf{0}}^{+}$ we can have the best of both $\boldsymbol{\lambda}$-calculus and Combinatory Logic. For example, substitution is replaced by instantiation, and proof of CR for $\mathbb{L}$ implies proof of CR for $\mathbb{M}$ (and hence for $\boldsymbol{\Lambda}$).

External syntax Λ for positive $\mathbb{L}$-terms

We use prefixes $[u_1 \cdots u_n]$ $(n \geq 0)$ (also written $[\bar{u}]$) in the following definition of raw terms:

$$\text{Raw terms} \ni K, L ::= [u_1 \cdots u_n]z \mid [u_1 \cdots u_n](K\ L)$$

External syntax Λ is defined inductively as follows.

$$\frac{x \text{ occurs in } \bar{u}}{[\bar{u}]x : \Lambda} \qquad \frac{[\bar{u}]K : \Lambda \quad [\bar{u}]L : \Lambda}{[\bar{u}](K\ L) : \Lambda}$$

$[\bar{u}]K$ ($[\bar{u}]L$) is call the car (resp., cdr) of $[\bar{u}](K\ L)$

Remark

Raw terms allow open terms, but Λ defines exactly the closed λ-terms. Note that the prefix part of a Λ-term contains one or more variables.

Analysis of projections

The terms created by the rule below are called projections. They are indeed projection functions used in the theory of pritimitive recursive functions.

$$\frac{}{[u_1 \cdots u_i \; x \; v_1 \cdots v_k]x : \Lambda},$$

where x may appear in $u_1, \ldots, u_i$, but may not appear in $v_1, \ldots, v_k$.

After taking $i + k + 1$ arguments, $U_1, \ldots, U_i, X, V_1, \ldots, V_k$, the function $[u_1 \cdots u_i \; x \; v_1 \cdots v_k]x$ returns X. Here, we call k the de Bruijn index of the projection, and $i + k + 1$ the height of the projection. Since a projection is completely characterized by its height and index, we will use this fact to define the notion of α-equality of Λ-terms.

α-equality on Λ

Writing $\bar{u}$, $\bar{v}$ for sequences of variables, we define the α-equality of Λ-terms as follows.

1. Two projections are α-equal $\Longleftrightarrow$ they have the same height and index.
2. $[\bar{u}](K\ L) =_\alpha [\bar{v}](K'\ L') \Longleftrightarrow$
   $[\bar{u}]K =_\alpha [\bar{v}]K'$ and $[\bar{u}]L =_\alpha [\bar{v}]L'$.

Here is an example:

$$\dfrac{\dfrac{\text{height} = 2,\ \text{index} = 1}{[x\ y]x =_\alpha [y\ x]y} \quad \dfrac{\text{height} = 2,\ \text{index} = 0}{[x\ y]y =_\alpha [y\ x]x}}{[x\ y](x\ y) =_\alpha [y\ x](y\ x)}$$

Height of Λ-terms

1. $\text{Ht}([\bar{u}]x) = n$, where n is the length of the variable sequence $\bar{u} = u_1, \ldots, u_n$.
2. $\text{Ht}([\bar{u}](K\ L))$ is the length of $\bar{u}$.

A term has height n if it has a prefix of length n and then followed by a variable or by an application.

Height of a term is an extremely simple and natural concept on Λ-terms, but it plays a very important role in the study of the λ-calculus.

β-conversion on Λ

Given two terms M and N such that $\mathrm{Ht}(M) > n$ and $\mathrm{Ht}(N) \geq n$, we define the instantiation of M by N at height n, written $\langle M\ N\rangle^n$ inductively as follows. We first give the definition for the case $n = 0$

1. $\langle [x\bar{v}]x\ L\rangle^0 := [\bar{v}]L$.
2. $\langle [x\bar{v}]y\ L\rangle^0 := [\bar{v}]y$, if $x \neq y$.
3. $\langle [x\bar{v}](K\ K')\ L\rangle^0 := [\bar{v}](J\ J')$, if $\langle [x\bar{v}]K\ L\rangle^0 = [\bar{v}]J$ and $\langle [x\bar{v}]K'\ L\rangle^0 = [\bar{v}]J'$.

The β-conversion rule for the case $n = 0$ is defined as follows. We assume that $\mathrm{Ht}(M) > 0$.

$$(M\ P) \rightarrow_\beta \langle M\ P\rangle^0$$

β-conversion on Λ

Given two terms M and N such that $\mathrm{Ht}(M) > n$ and $\mathrm{Ht}(N) \geq n$, we define the instantiation of M by N at height n, written $\langle M\ N\rangle^n$, inductively as follows. We assume that the length of $\bar{u}$ is n.

1. $\langle[\bar{u}x\bar{v}]x\ [\bar{u}]L\rangle^n := [\bar{u}\bar{v}]L$.
2. $\langle[\bar{u}x\bar{v}]y\ [\bar{u}]L\rangle^n := [\bar{u}\bar{v}]y$, if $x \neq y$.
3. $$\frac{\langle[\bar{u}x\bar{v}]K\ [\bar{u}]L\rangle^n = [\bar{u}\bar{v}]J \quad \langle[\bar{u}x\bar{v}]K'\ [\bar{u}]L\rangle^n = [\bar{u}\bar{v}]J'}{\langle[\bar{u}x\bar{v}](K\ K')\ [\bar{u}]L\rangle^n = [\bar{u}\bar{v}](J\ J')}$$

The β-conversion rule at height n is as follows. We assume that the length of $\bar{u}$ is n.

$$[\bar{u}]([x\bar{v}]K\ L) \to_\beta \langle[\bar{u}x\bar{v}]K\ [\bar{u}L]\rangle^n$$

Translation of Λ-terms into positive $\mathbb{L}$-terms

We can translate each $\boldsymbol{\Lambda}$-term M into a positive $\mathbb{L}$-term $(\ M\ )_{\boldsymbol{\Lambda}\to\mathbb{L}}$ as follows. The translation is bijective module $\boldsymbol{\alpha}$-equality.

1. $(\ [u_1 \cdots u_i\ x\ v_1 \cdots v_k]x\ )_{\boldsymbol{\Lambda}\to\mathbb{L}} := \mathsf{I}^i_k$.
2. $(\ [\bar{u}](K\ L)\ )_{\boldsymbol{\Lambda}\to\mathbb{L}} := ((\ [\bar{u}]K\ )_{\boldsymbol{\Lambda}\to\mathbb{L}}\ (\ [\bar{u}]L\ )_{\boldsymbol{\Lambda}\to\mathbb{L}})^n$, where n is the length of $\bar{u}$.

Related works

- Frege's Begriffsschrift.
- Gentzens' natural decuction system and sequent calculus.
- Brigitte Pientka: Modal Context calculus, Explict Context. Beluga proof assistant.