Lectures on The Lambda Calculus (II)

Masahiko Sato
Graduate School of Informatics, Kyoto University

Autumn school "Proof and Computation"
Fischbachau, Germany
October 6, 2016

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.

These lectures are based on my work in progress.

de Bruijn algebra $\mathbb{D}$ vs. our algebra $\mathbb{L}$

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.

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

This is an instance of the following algebra which depends on algebra $\boldsymbol{\tau}$:

$$\mathbb{L}_{\boldsymbol{\tau}} = \boldsymbol{\tau} + (\mathbb{L}_{\boldsymbol{\tau}}\ \mathbb{L}_{\boldsymbol{\tau}})^{\mathbb{N}}$$

By putting $\boldsymbol{\tau} = \mathbb{T}$ we obtain $\mathbb{L}$.

de Bruijn algebra $\mathbb{D}$ vs. our algebra $\mathbb{L}$ (cont.)

$$\mathbb{L}_{\boldsymbol{\tau}} = \boldsymbol{\tau} + (\mathbb{L}_{\boldsymbol{\tau}}\ \mathbb{L}_{\boldsymbol{\tau}})^{\mathbb{N}}$$

By putting $\boldsymbol{\tau} = \mathbb{P}$ (closed threads), we obtain $\mathbb{L}_{\mathbf{0}}$ consisting exactly of *closed* $\mathbb{L}$*-expressions* as follows.

$$\mathbb{L}_{\mathbf{0}} = \mathbb{P} + (\mathbb{L}_{\mathbf{0}}\ \mathbb{L}_{\mathbf{0}})^{\mathbb{N}}$$

In $\mathbb{D}$, it is not as easy as in our case. One can only get $\mathbb{D}_{\mathbf{0}}$, consisting of closed de Bruijn terms, by solving the following infinite family of equations. We put
$\mathbb{N}_{\boldsymbol{i}} := \{\boldsymbol{n} \in \mathbb{N} \mid \boldsymbol{n} < \boldsymbol{i}\}$ $(\boldsymbol{i} \in \mathbb{N})$.

$$\begin{aligned}
\mathbb{D}_{\mathbf{0}} &= \mathbb{N}_{\mathbf{0}} + \boldsymbol{\lambda}\mathbb{D}_{\mathbf{1}} + (\mathbb{D}_{\mathbf{0}}\ \mathbb{D}_{\mathbf{0}}),\\
\mathbb{D}_{\mathbf{1}} &= \mathbb{N}_{\mathbf{1}} + \boldsymbol{\lambda}\mathbb{D}_{\mathbf{2}} + (\mathbb{D}_{\mathbf{1}}\ \mathbb{D}_{\mathbf{1}}),\\
\mathbb{D}_{\mathbf{2}} &= \mathbb{N}_{\mathbf{2}} + \boldsymbol{\lambda}\mathbb{D}_{\mathbf{3}} + (\mathbb{D}_{\mathbf{2}}\ \mathbb{D}_{\mathbf{2}}),\\
&\cdots
\end{aligned}$$

Embedding of de Bruijn algebra $\mathbb{D}$ into our algebra $\mathbb{L}$

We can save $\mathbb{D}$ from this situation by embedding $\mathbb{D}$ into our algebra $\mathbb{L}$ by defining the embedding function

$$[\cdot] : \mathbb{D} \to \mathbb{L}$$

as follows.

$$\begin{aligned} [n] &:= n, \\ [\lambda D] &:= \lambda[D], \\ [(D\ E)] &:= ([D]\ [E]). \end{aligned}$$

What is this function?

Embedding of de Bruijn algebra $\mathbb{D}$ into our algebra $\mathbb{L}$

We can save $\mathbb{D}$ from this situation by embedding $\mathbb{D}$ into our algebra $\mathbb{L}$ by defining the embedding function

$$[\cdot] : \mathbb{D} \to \mathbb{L}$$

as follows.

$$\begin{aligned} [n] &:= n, \\ [\lambda D] &:= \lambda [D], \\ [(D\ E)] &:= ([D]\ [E]). \end{aligned}$$

What is this function?

The identity function! $\mathbb{D}$ is indeed a subset of $\mathbb{L}$. (So far we can apply λ only to threads. But we will extend it to be applicable to any $\mathbb{L}$-expression.)

The data structure of threads

We can view the algebra $\mathbb{T}$ in the following two ways.

$$\mathbb{T} = \mathbb{N} + \boldsymbol{\lambda}\mathbb{T} \ \text{ or } \ \mathbb{T} = \mathbb{N} \times \mathbb{N}$$

In the first view, a typical element of $\mathbb{T}$ can be written as $\boldsymbol{\lambda}^{\boldsymbol{i}}\boldsymbol{k}$. This element is obtained from $\boldsymbol{k}$ by applying the constructor $\boldsymbol{\lambda}$ to $\boldsymbol{k}$ $\boldsymbol{i}$ times.

In the second view, the same element can be written $\boldsymbol{i}/\boldsymbol{k}$. From abstract syntax point of view, they are just two different notation (written in two different syntax) for the same *thread*.

For example, $\boldsymbol{k} = \boldsymbol{\lambda}^{\mathbf{0}}\boldsymbol{k}$ in the first view, corresponds to $\mathbf{0}/\boldsymbol{k}$ in the second view. For this reason we will also write $\boldsymbol{k}$ for $\mathbf{0}/\boldsymbol{k}$.

The datatype $\mathbb{T}$

For technical reason, we will officially define $\mathbb{T}$ by the following inductive definition, taking the second view above.

$$\frac{\boldsymbol{i} \in \mathbb{N} \quad \boldsymbol{k} \in \mathbb{N}}{\boldsymbol{i}/\boldsymbol{k} \in \mathbb{T}} \ \mathsf{Thrd}$$

We will use $\boldsymbol{q}, \boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t}$ as meta variables ranging over threads.

Now, any thread $\boldsymbol{t}$ can be uniquely written $\boldsymbol{t} = \boldsymbol{i}/\boldsymbol{k}$. In this case we say that *height* of $\boldsymbol{t}$, written $\mathsf{Ht}(\boldsymbol{t})$ is $\boldsymbol{i}$ and *depth* of $\boldsymbol{t}$, written $\mathsf{Dp}(\boldsymbol{t})$, is $\boldsymbol{k}$.

λ as an operator on $\mathbb{T}$

We do not have $\boldsymbol{\lambda}$ in $\mathbb{T}$, but we can *define* it as an operator on $\mathbb{T}$:

$$\boldsymbol{\lambda}\frac{i}{k} := \frac{i'}{k}.$$

In general, we define $\boldsymbol{\lambda}^n : \mathbb{T} \to \mathbb{T}$ by

$$\boldsymbol{\lambda}^n\frac{i}{k} := \frac{i+n}{k}.$$

So, $\boldsymbol{\lambda}^n t$ increases height of t by n keeping its depth. For example, we have:

$$\boldsymbol{\lambda}^i k = \boldsymbol{\lambda}^i\frac{0}{k} = \frac{0+i}{k} = i/k$$

Closed and open threads

A thread i/k is defined to be *closed* if $i > k$, and it is defined to be *open* if $i \leq k$.

Since $i/k = \lambda^i k$ we may visualize it as follows.

$$\lambda_{i-1}\lambda_{i-2}\cdots\lambda_1\lambda_0 k$$

So, recalling the de Bruijn notation, we see that it is a closed term if and only if $i > k$.

Closed threads are also called *projections*. We write $\mathbb{P}$ for the set $\{t \in \mathbb{T} \mid t \text{ is closed}\}$ of propositions.

Classification of $\mathbb{T}$ and $\mathbb{P}$ by height

We put

$$\mathbb{T}^n := \{t \in \mathbb{T} \mid \mathrm{Ht}(t) \geq n\},$$
$$\mathbb{P}^n := \{t \in \mathbb{P} \mid \mathrm{Ht}(t) \geq n\}.$$

We have:

$$\mathbb{T} = \mathbb{T}^0 \supsetneq \mathbb{T}^1 \supsetneq \mathbb{T}^2 \cdots$$
$$\mathbb{P} = \mathbb{P}^0 \supsetneq \mathbb{P}^1 \supsetneq \mathbb{P}^2 \cdots$$

We note that $\boldsymbol{\lambda}^i : \mathbb{T}^n \to \mathbb{T}^{n+i}$ and $\boldsymbol{\lambda}^i : \mathbb{P}^n \to \mathbb{P}^{n+i}$ (since $\mathbb{P}$ is closed under application of $\boldsymbol{\lambda}$). So, it is natural to write $\boldsymbol{\lambda}^i\mathbb{T}^n$ for $\mathbb{T}^{n+i}$ and $\boldsymbol{\lambda}^i\mathbb{P}^n$ for $\mathbb{P}^{n+i}$

Closing and opening

Recall that:

$$\boldsymbol{\lambda} : \mathbb{T}^{n} \to \mathbb{T}^{n+1} \;\; (n \in \mathbb{N})$$

Not only $\boldsymbol{\lambda}$ has this arity, it is also a *bijective* operator.

So it has its inverse

$$\overline{\boldsymbol{\lambda}} : \mathbb{T}^{n+1} \to \mathbb{T}^{n} \;\; (n \in \mathbb{N})$$

with the property $\overline{\boldsymbol{\lambda}}\boldsymbol{\lambda} t = t$ for all $t \in \mathbb{T}$ and $\boldsymbol{\lambda}\overline{\boldsymbol{\lambda}} t = t$ for all $t \in \boldsymbol{\lambda}\mathbb{T}$.

Note that

$$\boldsymbol{\lambda} : \mathbb{T} \to \boldsymbol{\lambda}\mathbb{T} \;\; \text{and} \;\; \overline{\boldsymbol{\lambda}} : \boldsymbol{\lambda}\mathbb{T} \to \mathbb{T}$$

Given any thread, by applying $\boldsymbol{\lambda}$ sufficiently many times, it becomes a closed thread. So we will call $\boldsymbol{\lambda}$ a *closing* operator. Similarly $\overline{\boldsymbol{\lambda}}$ will be called an *opening* operator.

Instantiation operation

We wish to define the *instantiation* operation which is a binary function of the form:

$$\langle \boldsymbol{\lambda} \cdot \ \cdot \rangle : \boldsymbol{\lambda}\mathbb{T} \times \mathbb{T} \to \mathbb{T}$$

This form imposes a natural condition that $\langle \cdot\ \boldsymbol{t} \rangle$ is meaningful only if the first argument $\cdot$ is of the form $\boldsymbol{\lambda r}$.

So, for any threads $\boldsymbol{r}$ and $\boldsymbol{t}$, we wish to know what $\langle \boldsymbol{\lambda r}\ \boldsymbol{t} \rangle$ means. Our intuition is that it means the result of *applying* the function $\boldsymbol{\lambda r}$ to its argument $\boldsymbol{t}$.

Our idea is to define it by defining yet another binary function of the form:

$$\cdot \leftarrow \cdot : \mathbb{T} \times \mathbb{T} \to \mathbb{T},$$

and then put:

$$\langle \boldsymbol{\lambda r}\ \boldsymbol{t} \rangle := \boldsymbol{r} \leftarrow \boldsymbol{t}.$$

Filling operation

We wish to define the *filling* operation which is a binary function of the form:

$$\cdot \leftarrow \cdot : \mathbb{T} \times \mathbb{T} \to \mathbb{T}$$

So, for any threads r and t, we wish to know what $r \leftarrow t$ means.

Our idea is to define it by case analysis on the form of r. Namely, we say that r is *balanced* if $\mathsf{Ht}(r) = \mathsf{Dp}(r)$, and define the filling operation according as r is balanced or not.

Filling operation: Balanced case

In this case, $r = \lambda^i k$ where $i = \mathsf{Ht}(r) = \mathsf{Dp}(r) = k$.

Filling *succeeds* in this case, and we put:

$$r \leftarrow t := \Uparrow^r t.$$

Here, $\Uparrow^r \; : \mathbb{T} \to \mathbb{T}$ is a *lifting* operation defined by:

$$\Uparrow^r \frac{j}{\ell} := \begin{cases} \frac{j+k}{\ell} & \text{if } j > \ell, \\ \frac{j+k}{\ell+k} & \text{if } j \leq \ell. \end{cases}$$

Note that for any $t \in \mathbb{T}$, $\Uparrow^r t$ is closed (open) iff t is closed (open), and $\Uparrow^r t = \lambda^{\mathsf{Ht}(r)} t$ if t is closed. Also:

$$\Uparrow^r \; : \mathbb{T}^n \to \mathbb{T}^{n+\mathsf{Ht}(r)}.$$

Filling operation: Unbalanced case

In this case, $r = \lambda^i k$ where $i = \text{Ht}(r) \neq \text{Dp}(r) = k$.
Filling *fails* in this case, and we put:

$$r \leftarrow t := \Downarrow r.$$

Here, $\Downarrow\ : \mathbb{T} \rightarrow \mathbb{T}$ is a *lowering* operation defined by:

$$\Downarrow \frac{i}{k} := \begin{cases} \frac{i}{k} & \text{if } i \geq k, \\ \frac{i}{k-1} & \text{if } i < k. \end{cases}$$

The lowering function lowers depth of r by one only when r is open and $\text{Dp}(r) > 0$. Note that for any $r \in \mathbb{T}$, $\Downarrow r$ is closed (open) iff r is closed (open). Also:

$$\Downarrow\ : \mathbb{T}^n \rightarrow \mathbb{T}^n.$$

Filling operation (cont.)

Combining the balanced and unbalanced cases, we get the following definition of filling operation.

$$r \leftarrow t := \begin{cases} \Uparrow^r t & \text{if } r \text{ is balanced,} \\ \Downarrow r & \text{if } r \text{ is unbalanced.} \end{cases}$$

We can spell out the explicit definition as follows.

$$\frac{i}{k} \leftarrow \frac{j}{\ell} := \begin{cases} \frac{i}{k} & \text{if } i > k, \\ \frac{j+k}{\ell} & \text{if } i = k \text{ and } j > \ell, \\ \frac{j+k}{\ell+k} & \text{if } i = k \text{ and } j \leq \ell, \\ \frac{i}{k-1} & \text{if } i < k. \end{cases}$$

Definition of instantiation operation

Our plan was to define instantiation operation with arity:

$$\langle \boldsymbol{\lambda} \cdot \ \cdot \rangle : \boldsymbol{\lambda}\mathbb{T} \times \mathbb{T} \to \mathbb{T}$$

in terms of the filling operation with arity

$$\cdot \leftarrow \cdot : \mathbb{T} \times \mathbb{T} \to \mathbb{T}$$

by putting

$$\langle \boldsymbol{\lambda} \boldsymbol{r} \ \boldsymbol{t} \rangle := \boldsymbol{r} \leftarrow \boldsymbol{t}.$$

Here, we define instantiation as follows.

Definition (Instantiaton $\langle \cdot \ \cdot \rangle : \boldsymbol{\lambda}\mathbb{T} \times \mathbb{T} \to \mathbb{T}$)

$$\langle \boldsymbol{r} \ \boldsymbol{t} \rangle := \overline{\boldsymbol{\lambda}} \boldsymbol{r} \leftarrow \boldsymbol{t}$$

Definition of instantiation operation (cont.)

Putting $r = i/k$ $(i > 0)$ and $t = j/\ell$ we have:

$$\langle \frac{i}{k}\ \frac{j}{\ell} \rangle := \frac{i-1}{k} \leftarrow \frac{j}{\ell} = \begin{cases} \frac{i-1}{k} & \text{if } i-1 > k, \\ \frac{j+k}{\ell} & \text{if } i-1 = k \text{ and } j > \ell, \\ \frac{j+k}{\ell+k} & \text{if } i-1 = k \text{ and } j \leq \ell, \\ \frac{i-1}{k-1} & \text{if } i-1 < k. \end{cases}$$

Examples of instantiation operatation: Case 1

We wish to see the informal correctness of our definition of instantiation based on our intuitive understanding of threads.

Here, we consider the following case:

$$i > 0 \text{ and } i - 1 > k$$

$$\langle \frac{i}{k} r \rangle := \frac{i-1}{k} \leftarrow r = \frac{i-1}{k}$$

Let us say that $i = 4$ and $k = 2$, so that we have

$$\langle \lambda^4 2 \; r \rangle = \lambda^3 2 \leftarrow r = \lambda^3 2$$

Or, equivalently:

$$\langle \lambda^4 2 \; r \rangle = \langle \lambda_{xyzu} y \; r \rangle = \lambda_{yzu} y = \lambda^3 2$$

Examples of instantiation operatation: Case 2

Here, we consider the following case:

$$i > 0,\ i - 1 = k \text{ and } j > l$$

$$\langle \frac{i}{k}\ \frac{j}{\ell} \rangle := \frac{i-1}{k} \leftarrow \frac{j}{\ell} = \frac{j+k}{\ell}$$

Let us say that $i = 3$, $k = 2$, $j = 1$ and $\ell = 0$ so that we have

$$\langle \lambda^3 2\ \lambda^1 0 \rangle = \lambda^2 2 \leftarrow \lambda^1 0 = \lambda^3 0$$

Or, equivalently:

$$\langle \lambda^3 2\ \lambda^1 0 \rangle = \langle \lambda_{xyz} x\ \lambda_u u \rangle = \lambda_{yz} \lambda_u u = \lambda_{yzu} u = \lambda^3 0$$

Examples of instantiation operatation: Case 3

Here, we consider the following case:

$$\boldsymbol{i > 0,\ i - 1 = k} \ \text{ and } \ \boldsymbol{j \le l}$$

$$\langle \frac{\boldsymbol{i}}{\boldsymbol{k}} \ \frac{\boldsymbol{j}}{\boldsymbol{\ell}} \rangle := \frac{\boldsymbol{i-1}}{\boldsymbol{k}} \leftarrow \frac{\boldsymbol{j}}{\boldsymbol{\ell}} = \frac{\boldsymbol{j+k}}{\boldsymbol{\ell+k}}$$

Let us say that $\boldsymbol{i = 3}$, $\boldsymbol{k = 2}$, $\boldsymbol{j = 1}$ and $\boldsymbol{\ell = 1}$ so that we have

$$\langle \boldsymbol{\lambda^3 2\ \lambda^1 1} \rangle = \boldsymbol{\lambda^2 2 \leftarrow \lambda^1 1 = \lambda^3 3}$$

Or, equivalently:

$$\langle \boldsymbol{\lambda^3 2\ \lambda^1 1} \rangle = \langle \boldsymbol{\lambda_{xyz} x\ \lambda_u 1} \rangle = \boldsymbol{\lambda_{yz} \lambda_u 3 = \lambda_{yzu} 3 = \lambda^3 3}$$

We changed $\boldsymbol{1}$ to $\boldsymbol{3}$ to avoid capturing by $\boldsymbol{\lambda_{yz}}$.

Examples of instantiation operatation: Case 4

Here, we consider the following case:

$$i > 0 \text{ and } i - 1 < k$$

$$\langle \frac{i}{k}\ r\rangle := \frac{i-1}{k} \leftarrow r = \frac{i-1}{k-1}$$

Let us say that $i = 2$ and $k = 2$, so that we have

$$\langle \lambda^2 2\ r\rangle = \lambda^1 2 \leftarrow r = \lambda^1 1$$

Or, equivalently:

$$\langle \lambda^2 2\ r\rangle = \langle \lambda_{xy} 2\ r\rangle = \lambda_y 1 = \lambda^1 1$$

We changed 2 to 1 since it is not in the scope of λ_x anymore.

Instantiation under λ

Consider $\boldsymbol{\lambda_z \underline{(\lambda_{xy}(x\ y)\ z)}}$. In the tradtitional $\boldsymbol{\lambda}$-calculus, we can convert the underlined $\boldsymbol{\beta}$-redex using the $\boldsymbol{\xi}$-rule as follows.

$$\dfrac{\overline{(\lambda_{xy}(x\ y)\ z) \rightarrow_{\beta} \lambda_y(z\ y)}\ \beta}{\lambda_z(\lambda_{xy}(x\ y)\ z) \rightarrow_{\beta} \lambda_{zy}(z\ y)}\ \xi$$

In our calculus, we wish to eliminate the $\boldsymbol{\xi}$-rule, by extending the $\boldsymbol{\beta}$-rule so that we can reduce the inner $\boldsymbol{\beta}$-redex directly as shown below. Here, we note that

$$\lambda_z(\lambda_{xy}(x\ y)\ z) = ((\lambda^3 1\ \lambda^3 0)^3\ \lambda^1 0)^1$$
$$\lambda_{zy}(z\ y) = (\lambda^2 1\ \lambda^2 0)^2$$

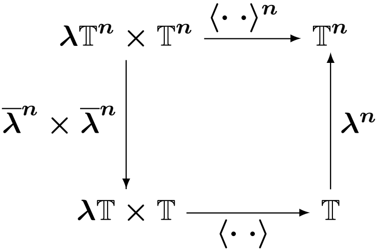
$$((\lambda^3 1\ \lambda^3 0)^3\ \lambda^1 0)^1 \rightarrow_{\beta} \langle(\lambda^3 1\ \lambda^3 0)^3\ \lambda^1 0\rangle^1 = (\lambda^2 1\ \lambda^2 0)^2$$

Instantiation at level n

What we will do here is to generalize the instantiation operation $\langle \boldsymbol{r}\ \boldsymbol{t} \rangle$ (which operates on $\boldsymbol{r} \in \boldsymbol{\lambda}\mathbb{T}$ and $\boldsymbol{t} \in \mathbb{T}$) to $\langle \boldsymbol{r}\ \boldsymbol{t} \rangle^{\boldsymbol{n}}$ with arity:

$$\langle \cdot\ \cdot \rangle^{\boldsymbol{n}} : \boldsymbol{\lambda}\mathbb{T}^{\boldsymbol{n}} \times \mathbb{T}^{\boldsymbol{n}} \to \mathbb{T}^{\boldsymbol{n}}$$

We define this operation so that the following diagram commutes:



Namely,

$$\langle \boldsymbol{r}\ \boldsymbol{t} \rangle^{\boldsymbol{n}} := \boldsymbol{\lambda}^{\boldsymbol{n}} \langle \overline{\boldsymbol{\lambda}}^{\boldsymbol{n}} \boldsymbol{r}\ \overline{\boldsymbol{\lambda}}^{\boldsymbol{n}} \boldsymbol{t} \rangle$$

Instantiation Lemma

Lemma (Instantiation Lemma for threads)

$$n < m, r \in \mathbb{T}^{m+1}, s \in \mathbb{T}^{m}, t \in \mathbb{T}^{n} \vdash$$
$$\langle\langle r\ s\rangle^{m}\ t\rangle^{n} = \langle\langle r\ t\rangle^{n}\ \langle s\ t\rangle^{n}\rangle^{m-1}.$$

The above lemma is derivable from the following lemma.

Lemma (special case of the above lemma)

$$0 < m, r \in \mathbb{T}^{m+1}, s \in \mathbb{T}^{m}, t \in \mathbb{T} \vdash$$
$$\langle\langle r\ s\rangle^{m}\ t\rangle = \langle\langle r\ t\rangle\ \langle s\ t\rangle\rangle^{m-1}.$$

Substitution and Instantiation

$$\boldsymbol{x} \neq \boldsymbol{y}, \boldsymbol{x} \notin \mathrm{FV}(\boldsymbol{M}) \vdash$$
$$\boldsymbol{K}[\boldsymbol{x} := \boldsymbol{L}][\boldsymbol{y} := \boldsymbol{M}] = \boldsymbol{K}[\boldsymbol{y} := \boldsymbol{M}][\boldsymbol{x} := \boldsymbol{L}[\boldsymbol{y} := \boldsymbol{M}]].$$

$$\boldsymbol{K} \in \mathbb{T}^{\mathbf{2}}, \boldsymbol{L} \in \mathbb{T}^{\mathbf{1}}, \boldsymbol{M} \in \mathbb{T} \vdash \langle\langle \boldsymbol{K}\ \boldsymbol{L}\rangle^{\mathbf{1}}\ \boldsymbol{M}\rangle = \langle\langle \boldsymbol{K}\ \boldsymbol{M}\rangle\ \langle \boldsymbol{L}\ \boldsymbol{M}\rangle\rangle.$$

Or, equivalently,

$$\langle\langle \boldsymbol{\lambda}^{\mathbf{2}}\boldsymbol{K}\ \boldsymbol{\lambda}^{\mathbf{1}}\boldsymbol{L}\rangle^{\mathbf{1}}\ \boldsymbol{M}\rangle = \langle\langle \boldsymbol{\lambda}^{\mathbf{2}}\boldsymbol{K}\ \boldsymbol{M}\rangle\ \langle \boldsymbol{\lambda}^{\mathbf{1}}\boldsymbol{L}\ \boldsymbol{M}\rangle\rangle.$$

We can see that Instantiation operation naturally represents $\boldsymbol{\beta}$-conversion rule as an algebraic operation.