COMPARISON OF TWO CATEGORICAL MODELS OF TYPED $ lambda $-CALCULUS

Otsuka, Hiroshi
Department of Mathematics, Kyushu University

https://doi.org/10.5109/13415

Bulletin of Informatics and Cybernetics, Vol. 24, No. 3~4, 1991

COMPARISON OF TWO CATEGORICAL MODELS OF TYPED λ-CALCULUS

By

Hiroshi OHTSUKA[*]

Abstract

This paper discusses a relation between two categorical models of typed λ-calculus which are both cartesian closed categories. One of them has a concept of variables in it. The other does not have such concept and based on de Bruijn's name-free expression. We show that the second one is obtained by certain construction over cartesian closed category and they are isomorphic from the categorical point of view.

1. Introduction

Category, particularly cartesian closed category (CCC for short), recently began to be treated as a model of typed λ-calculus by several authors [2], [4], [5], [9]. The work of this paper is a Comparison of two categorical models of typed λ-calculus which are introduced by Curien [2] and Koymans [4].

The first model has *indeterminate*, a concept of variables in λ-calculus, and is obtained by universal construction over certain CCC. That is, it assigns indeterminates to variables and translates λ-terms into not the CCC but so-called *polynomial category* of it (polynomial CCC [5], another term is free CCC [2]). It is important that free variables are treated as themselves. For example, in [9], they were handled as being bound and only closed λ-terms were considered.

On the other hand, the second model does not have such concept but it is based on de Bruijn's name-free expression to make up for it. That is, it assigns not indeterminates but projections characterized by so-called their *indexes* to variables and translates λ-terms into the CCC itself.

But our observation shows that the second model is also obtained by certain construction over certain CCC known as Kleisli's construction. *Kleisli category* gotten by his construction becomes extended CCC just as polynomial CCC does. Therefore, we can regard that the second one deals with λ-terms in Kleisli category of the CCC instead of in itself.

Moreover, functional completeness of CCC [5] tells us that Kleisli category is isomorphic to polynomial CCC. This means that two categorical models of typed λ-calculus are both not merely CCCs but also extended ones obtained by certain con-

* Department of Mathematics, Kyushu University 33, Fukuoka 812, Japan

structions over it and have certain universality. In particular, the second model has been considered as merely CCC in early categorical model and it has been hard to investigate the categorical property of it. We conclude that the second model is gotten by Kleisli's construction just as the first one does by universal construction, and consequently, two categorical models are equivalent from the categorical point of view.

Section 2 sketches the basic notions of categories, CCC, polynomial CCC, Kleisli's construction. Section 3 introduces typed λ-calculus and two translations of it into CCC. Section 4 gives some properties of them respectively. Section 5 contains the main result mentioned that these translations have reciprocal relationship from the categorical point of view. The functional completeness of CCC due to Lambek and Scott [5] is essential for this relation.

2. Cartesian Closed Categories

In this section, we recall some relevant notions and notations of CCC, in particular Kleisli's construction and universal construction over CCC and their relation.

DEFINITION 2.1. A CCC $\mathfrak{C}$ is the category which has some additional structure. Let C be a set of basic objects. Objects of $\mathfrak{C}$ are defined as follows.

1. Elements of C are objects of $\mathfrak{C}$.
2. The special object 1 (*terminal object*) $\notin C$ is an object of $\mathfrak{C}$.
3. If A and B are objects of $\mathfrak{C}$ then so are $A \times B$ (*product*) and B^A (*exponential*).

Morphisms of $\mathfrak{C}$ are defined as follows.

1. For any object A, there exists the identity $id_A : A \to A$.
2. For any object A, there exists $!_A : A \to 1$.
3. For any objects A and B, there exist first and second projections
$$\pi_{A,B} : A \times B \to A \text{ and } \pi'_{A,B} : A \times B \to B.$$
4. For any objects B and C, there exists evaluation $\varepsilon_{B,C} : B^C \times C \to B$.
5. If $f : B \to C$ and $g : A \to B$ then $f \circ g : A \to C$.
6. If $f : A \to B$ and $g : A \to C$ then $< f, g > : A \to B \times C$.
7. If $h : A \times C \to B$ then $\Lambda(h) : A \to B^c$.

We omit the required structure of cartesian closedness. It is referred [5], [7].
We omit the subscript of morphisms and the symbol of composition $\circ$ if no confusion occurs. In the rest of this section, $\mathfrak{A}$ and $\mathfrak{B}$ denote CCCs. If a functor $F : \mathfrak{A} \to \mathfrak{B}$ preserves cartesian closed structure, we call it cartesian closed functor (CCF for short). In the next paragraph, we present two constructions over CCC and certain relation between them [5].

First, we construct an adjoint pair of functors which induces a comonad $< S_A, \zeta_A, \eta_A >$ over an $\mathfrak{A}$ and an object A in $\mathfrak{A}$. The comonad consists of an endofunctor S_A over $\mathfrak{A}$ associated with two natural transformations

$$\zeta_A : S_A \to I_{\mathfrak{A}} \text{ (identity functor) and } \eta_A : S_A \to S^2 \text{ (composition of } S \text{ with } S\text{)},$$

which are defined as follows.

1. $S_A(B) = A \times B$, and $S_A(f) = < \pi_{A,B}, f\pi'_{A,B} >$ for $f: B \to C$.
2. $\zeta_A(B) = \pi'_{A,B}: A \times B \to B$.
3. $\eta_A(B) = < \pi_{A,B}, id_{A\times B} >: A \times B \to A \times (A \times B)$.

for any objects B, C in $\mathfrak{A}$. Then it is easy to show that $< S_A, \zeta_A, \eta_A >$ becomes comonad (c.f. [7], [8]).

It is known that inducing pairs of functors are not uniquely determined by the comonads. But there are two essentially different pairs of adjoint functors satisfying this condition and having certain universal property. These pairs were independently found by Eilenberg-Moore and Kleisli. As pointed out by Lambek and Scott, the Kleisli category of $< S_A, \zeta_A, \eta_A >$ over $\mathfrak{A}$ is isomorphic to polynomial CCC of $\mathfrak{A}$. That is why we present Kleisli's construction and its universal property.

Let $\mathfrak{A}_{S_A}$ be the the Kleisli category of $< S_A, \zeta_A, \eta_A >$, which has the same objects as $\mathfrak{A}$. A morphism $f: B \to C$ in $\mathfrak{A}_{S_A}$ is called Kleisli morphism $f: S_A(B) = A \times B \to C$ in $\mathfrak{A}$. Below, morphisms with superscript A denote Kleisli morphisms. In particular,

1. The identity id^A_B in $\mathfrak{A}_{S_A}$ *is* $\zeta_A(B) = \pi'_{A,B}$ in $\mathfrak{A}$.
2. The composition in Kleisli category is defined by

$$g\circ f = gS_A(f)\eta_A = g < \pi_{A,B}, f >: S_A(B) \to D$$

for $f: S_A(B) \to C$ and $g: S_A(C) \to D$ in $\mathfrak{A}$.

Moreover, $\mathfrak{A}_{S_A}$ becomes CCC by the following cartesian closed structure. Let B, C, D be any objects in $\mathfrak{A}_{S_A}$.

1. Following morphisms are Kleisli morphisms.

$$!^A_B = !_{A\times B},\ \pi^A_{B,C} = \pi_{B,C}\ \pi'_{A,B\times C},\ \pi'^A_{B,C} = \pi'_{B,C}\ \pi'_{A,B\times C} \text{ and } \varepsilon^A_{B,C} = \varepsilon_{B,C}\ \pi'_{A,B\times C}$$

2. If $f: B \to C$ and $g: B \to D$ are morphisms in $\mathfrak{A}_{S_A}$, then so is

$$< f, g >^A = < f, g >: B \to C \times D.$$

3. If $h: B \times C \to D$ is a morphism in $\mathfrak{A}_{S_A}$, then so is

$$\Lambda^A(h) = \Lambda(h\alpha): B \to D^C.$$

Where $\alpha = < \pi\ \pi, < \pi'\ \pi, \pi' >>: (A \times B) \times C \to A \times (B \times C)$.

Accordingly, the functors from $\mathfrak{A}$ into $\mathfrak{A}_{S_A}$ and vice versa can be defined. The CCF $F_{S_A}: \mathfrak{A} \to \mathfrak{A}_{S_A}$ is defined by

1. $F_{S_A}(B) = B$ for any objects B,
2. $F_{S_A}(f) = f\zeta_A(B) = f\pi'_{A,B}$ for any morphisms $f: B \to C$.

The inverse functor $G_{S_A}: \mathfrak{A}_{S_A} \to \mathfrak{A}$ is defined by

1. $G_{S_A}(B) = S_A(B) = A \times B$ for any objects B,
2. $G_{S_A}(g) = S_A(g)\eta_A(B) = < \pi_{A,B}, g >$ for any Kleisli morphisms $g: A \times B \to C$.

It is clear that F_{S_A} is the left adjoint of G_{S_A}. Moreover, above construction has the universal property known as Kleisli's construction. We present it without proof.

THEOREM 2.1. (*Comparison theorem of Kleisli's construction* [7], [8]) *Let* $< S, \zeta, \eta >$ *be a comonad over a category* $\mathfrak{C}$, $F_S : \mathfrak{C} \to \mathfrak{C}_s$ *and* $G_s : \mathfrak{C}_s \to \mathfrak{C}$ *are the pair of adjoint functors obtained by Kleisli's construction. If* $F : \mathfrak{C} \to \mathfrak{D}$ *and* $G : \mathfrak{D} \to \mathfrak{C}$ *are another pair of adjoint functors inducing the given comonad, then there exists uniquely determined functor* $L : \mathfrak{C}_s \to \mathfrak{D}$ *such that*

$$F = LF_S \qquad \textit{and} \qquad G_S = GL.$$

Of course, this theorem still holds in the situation of $< S_A, \zeta_A, \eta_A >$ over CCC $\mathfrak{A}$.

On the other hand, universal construction of CCC $\mathfrak{A}$ gives rise to polynomial CCC $\mathfrak{A}[x]$, where $x : A \to B$ is an *indeterminate* over $\mathfrak{A}$. A polynomial category $\mathfrak{A}[x]$ over $\mathfrak{A}$ is freely constructed by adding x as a morphism to $\mathfrak{A}$. Morphisms in $\mathfrak{A}[x]$ are called *polynomials*. We introduce equality '$=_x$' between polynomials such that rendering $\mathfrak{A}[x]$ CCC [5]. The subscript of equality is a set of indeterminates (if one element, then abbreviate like as above and for a set of indeterminates X, we write $=_X$).

PROPOSITION 2.1. (*Universality* [5], [6]) *Let* $x : A \to B$ *be an indeterminate over* $\mathfrak{A}$. *For any CCF* $F : \mathfrak{A} \to \mathfrak{B}$ *and a morphism* $b : F(A) \to F(B)$ *in* $\mathfrak{B}$, *there exists a unique CCF* $F' : \mathfrak{A}[x] \to \mathfrak{B}$ *such that*

$$F'(x) = b \qquad \textit{and} \qquad F'H_x = F.$$

Where $H_x : \mathfrak{A} \to \mathfrak{A}\,[x]$ *is the CCF which sends* $f : B \to C$ *onto the "constant" polynomial with the same name.*

PROPOSITION 2.1 which asserts that the polynomial CCC $\mathfrak{A}[x]$ has the universal property corresponds with Theorem 2.1. Moreover, $\mathfrak{A}[x]$ whose indeterminate is $x : 1 \to A$ has closed relation with Kleisli category $\mathfrak{A}_{S_A}$.

PROPOSITION 2.2. (*Functional completeness* [5], [6]) *For every polynomial* $\phi(x) : B \to C$ *in an indeterminate* $x : 1 \to A$ *over* $\mathfrak{A}$, *there is a unique morphism* $f : A \times B \to C$ *in* $\mathfrak{A}$ *such that*

$$f < x!_B, id_B > =_x \phi(x).$$

We only exhibit the algorithm finding morphism f for a polynomial $\phi(x)$. We take the following $\kappa_x\phi(x)$ as f satisfying above equation.

1. $\kappa_x k = k\,\pi'_{A,B} : A \times B \to C$ for constant polynomial $k : B \to C$.
2. $\kappa_x x = \pi_{A,B} : A \times 1 \to A$ for indeterminate $x : 1 \to A$.
3. $\kappa_x < \phi(x), \psi(x) > = < \kappa_x\phi(x), \kappa_x\psi(x) > : A \times B \to C \times D$ for $\phi(x) : B \to C$ and $\psi(x) : B \to D$.
4. $\kappa_x(\psi(x)\phi(x)) = \kappa_x\psi(x) < \pi_{A,B}, \kappa_x\phi(x) > : A \times B \to A \times C \to D$ for $\phi(x) : B \to C$ and $\psi(x) : C \to D$.
5. $\kappa_x\Lambda(\chi(x)) = \Lambda((\kappa_x\chi(x))\alpha) : A \times B \to C^D$ for $\chi(x) : B \times D \to C$.

The next proposition is taken another look at the Proposition 2.2 ([5], [6]).

PROPOSITION 2.3. *For an indeterminate* $x : 1 \to A$ *over* $\mathfrak{A}$,

$$\mathfrak{A}[x] \cong \mathfrak{A}_{S_A}.$$

The isofunctor $\mathfrak{A}[x] \to \mathfrak{A}_{S_A}$ is defined through the algorithm in the proof of Proposition 2.2. The isomorphic situation is made mention by the functional completeness.

Although we concern only one indeterminate above, the following situation holds in general:

$$\begin{aligned}\mathfrak{A}[x_0, \ldots, x_{n-1}] &\cong (\ldots (\mathfrak{A}[x_0])\ldots)[x_{n-1}]\\ &\cong (\ldots (\mathfrak{A}_{S_{A_0}})\ldots)_{S_{A_{n-1}}}\end{aligned}$$

where $x_i : 1 \to A_i (0 \leq i \leq n-1)$.

3. Typed λ-calculus

We abbreviate typed $\lambda\beta\eta$-calculus with product types (another term is surjective pairing) to typed λ-calculus. In this section, we present the system of typed λ-calculus and two translations of it into CCC. The formal system of typed λ-calculus consists of *types*, *λ-terms* of each type and certain *rules* between λ-terms of the same type ([1], [2], [6]).

Definition 3.1. Types have the same structure as objects of CCC. Let C be a set of basic types. Types are defined as follows.

1. Elements of C are types.
2. The special object $1 \notin C$ is a type.
3. If A and B are types so are $A \times B$ and A^B.

Λ_A (v_A) denotes the set of typed λ-terms (variables resp) of type A. For each type A, there are countably many variables of type A. λ-terms are inductively defined by

1. $v_A \in \Lambda_A$,
2. $M \in \Lambda_{B^A}$, $N \in \Lambda_A \Rightarrow (MN) \in \Lambda_B$,
3. $M \in \Lambda_B$, $x \in v_A \Rightarrow (\lambda x.M) \in \Lambda_{B^A}$,
4. $M \in \Lambda_A$, $N \in \Lambda_B \Rightarrow (M, N) \in \Lambda_{A\times B}$,
5. $M \in \Lambda_{A\times B} \Rightarrow (fst(M)) \in \Lambda_A$, $(snd(M)) \in \Lambda_B$,
6. $* \in \Lambda_1$

Free and bound variables are defined as usual. We use $FV(M)$ to denote the set of free variables of λ-term M. A term of the form (MN) is an *application* (of M to N). A term $(\lambda x.M)$ is a *λ-abstraction*, x is called *binder* and M is *body*. A term (M, N) is a *pairing*, a term $(fst(M))$ is a *1st projection* and $(snd(M))$ is a *2nd projection*. We adopt usual brackets convention if confusion may not occur and use '≡' to indicate syntactical equality.

Substitution is the replacement of all the free occurrence of a variable in a λ-term by another λ-term. The result of substituting N for the variable x in M is denoted by $M[x := N]$. In general, it is assumed that substitution does not influence the binding structure.

Definition 3.2. Substitution is defined as follows.

$$x[x := N] \equiv N.$$

$$y[x := N] \equiv y \qquad \text{if } x \not\equiv y.$$

$$(PQ)[x := N] \equiv (P[x := N])(Q[x := N]).$$

$$(P,Q)[x := N] \equiv (P[x := N], Q[x := N]).$$

$$(\lambda x.P)[x := N] \equiv \lambda x.P.$$

$$(\lambda y.P)[x := N] \equiv \lambda y.(P[x := N]), \quad \text{if } x \not\equiv y \text{ and } y \notin FV(M).$$

$$(\lambda y.P)[x := N] \equiv \lambda z.((P[y := z])[x := N]),$$

$$\text{if } x \not\equiv y \text{ and } y \in FV(M), \text{ choose } z \notin FV(P) \cup FV(N).$$

$$fst(P)[x := N] \equiv fst(P[x := N]).$$

$$snd(P)[x := N] \equiv snd(P[x := N]).$$

$$*[x := N] \equiv *.$$

Axiom 3.1. λ-terms of the same type are to satisfy the following rules.

1. $\lambda x.M = \lambda y.M[x := y]$, if $y \notin FV(M)$. (α-conversion)
2. $(\lambda x.M)N = M[x := N]$. ($\beta$-conversion)
3. $\lambda x.Mx = M$, if $x \notin FV(M)$. (η-conversion)
4. $M = *$ for $M \in \Lambda_1$. (π-conversion)
   $fst(P,Q) = P$, $snd(P,Q) = Q$.
   $(fst(M), snd(M)) = M$.
5. Standard compatible rules (see Barendregt [1], Lambek and Scott [6]).

We denote induced equation $M = N$ by above axioms in typed λ-calculus for $\lambda \vdash M = N$.

Next, we present two translations of typed λ-calculus into the CCC $\mathfrak{A}$ which has the same objects as types. Because we concern categorical properties of them, we restrict free variables in typed λ-calculus to finite. They are gathered up and make right associative ordered list,

$$FV = (x_0, (x_1, (\ldots, (x_{n-1},*)\ldots))) = (x_0, x_1, \ldots, x_{n-1},*)$$

Remark that numbering is started with 0. If we assume a finite list of variables as several pairings, it can have the same type of the pairing. Particularly, FV has the type $A_0 \times (A_1 \times (\ldots \times (A_{n-1} \times 1) \ldots))$, where each variable x_i has type A_i. We denote typed λ-calculus whose free variables are included in FV for $\Lambda(FV)$

The first translation translates λ-terms into polynomial CCC, that is, indeterminates are assigned to variables. This is due to Lambek and Scott [5] and introduced by Curien [2]. In Poigne [9], free variables are handled as being bound in the sense that his method assigns id_A to all free variables with same type A. Thus, λ-terms have been translated not into polynomial CCC but into CCC. We extend his method to be able to treat free variables themselves and adopt polynomial CCC. Moreover, this is a simple version of Curien's one which translates λ-terms into polynomials via so-called *Categorical Combinatory Logic*. First of all, we add the indeterminates, having the same name as variables in FV,

$$x_0 : 1 \to A_0,\ x_1 : 1 \to A_1,\ \ldots,\ x_{n-1} : 1 \to A_{n-1}$$

to CCC $\mathfrak{A}$ and make $\mathfrak{A}[x_0, \ldots, x_{n-1}]$. We denote the set of indeterminates which have the same name as variables in FV for $\{FV\}$.

DEFINITION 3.3. We identify the basis of objects in CCC $\mathfrak{A}$ to that of types in typed λ-calculus, and define polynomial

$$\mathcal{L}(M,\ ENV) : 1 \to A$$

for λ-term M of type A in $\Lambda(ENV)$ as follows.

$$\mathcal{L}(x,\ ENV) = x_i : 1 \to A,$$

where x_i in ENV has the same name as x.

$$\mathcal{L}(MN,\ ENV) = \varepsilon < \mathcal{L}(M,\ ENV), \mathcal{L}(N,\ ENV) >.$$

$$\mathcal{L}((M,\ N),\ ENV) = < \mathcal{L}(M,\ ENV), \mathcal{L}(N,\ ENV) >.$$

$$\mathcal{L}(\lambda x.M,\ ENV) = \Lambda(\kappa_x \mathcal{L}(M,\ (x,\ ENV)) < \pi'_{1,A}, \pi_{1,A} >),$$

where x has type A.

$$\mathcal{L}(fst(M),\ ENV) = \pi\, \mathcal{L}(M,\ ENV).$$

$$\mathcal{L}(snd(M),\ ENV) = \pi' \mathcal{L}(M,\ ENV).$$

$$\mathcal{L}(*,\ ENV) = id_1.$$

A list ENV is equal with FV at the top level.
Obviously, if $FV(\lambda x.M)$ is included in ENV then so is $FV(M)$ in $(x,\ ENV)$ and $\mathcal{L}$ does not depend on the order of variables in ENV. It is easy to show that if FV' is another list of variables and let M be the λ-term whose free variables are included in both FV and FV', then

$$\mathcal{L}(M,\ FV) =_{\{FV\}\cap\{FV'\}} \mathcal{L}(M,\ FV')$$

It is essential that $FV(M)$ is included in FV. We denote above translation for $\mathcal{L}_{FV}$: $\Lambda(FV) \to \mathfrak{A}[x_0, \ldots, x_{n-1}]$.

The second translation translates λ-terms not into polynomial CCC but into CCC. This method is given by Curien [2] and typed version of Koymans's one [4]. Although it does not use indeterminates, it treats free variables as themselves using so-caled *indexes* of them instead of their names. Therefore, it may be based on the variable concept of de Bruijn's name-free expression.

DEFINITION 3.4. We identify the basis of objects in CCC $\mathfrak{A}$ to that of types in typed λ-calculus, and define morphism,

$$\mathfrak{C}(M,\ ENV) : X \to A$$

for λ-term M of type A in $\Lambda(ENV)$, where ENV has type X, as follows.

$$\mathfrak{C}(x,\ ENV) = \pi\pi'^{i} : X \to A,$$

where i is the minimum such that $x = x_i$ in ENV.

$$\mathfrak{C}(MN,\ ENV) = \varepsilon < \mathfrak{C}(M,\ ENV), \mathfrak{C}(N,\ ENV) >.$$

$$\mathfrak{C}((M, N), ENV) = < \mathfrak{C}(M, ENV), \mathfrak{C}(N, ENV) >.$$

$$\mathfrak{C}(\lambda x.M, ENV) = \Lambda(\mathfrak{C}(M, (x, ENV)) < \pi'_{X,A}, \pi_{X,A} >),$$

where x has type A and ENV has X.

$$\mathfrak{C}(fst(M), ENV) = \pi\mathfrak{C}(M, ENV).$$

$$\mathfrak{C}(snd(M), ENV) = \pi'\mathfrak{C}(M, ENV).$$

$$\mathfrak{C}(*, ENV) = !_X.$$

$\pi\pi'^i$ means $\pi_{A_i,Y}\pi'_{A_{i-1},A_i\times Y} \dots \pi'_{A_0,A_1\times Z}$, where $Y = A_{i+1} \times(\dots \times 1)$ and $Z = A_2 \times (\dots \times 1)$. And if $i = 0$ then π'^i becomes id. A list ENV is equal with FV at the top level.

Remark that $\mathfrak{C}$ does depend on the order of variables in FV contrary to $\mathcal{L}$. We denote above translation for $\mathfrak{C}_{FV}: \Lambda(FV) \to \mathfrak{A}$. We give some properties of $\mathfrak{C}$ relating the manipulation of list. First, we provide the following notations of list.

1. $ENV(i, m, +)$ denotes the list which is made by adding m variables in the preceding position of the ith variable in ENV. We restrict the added variables to ones which can not appear in ENV or ones which appear in the left positions of the ith variable in ENV.
2. $ENV(i, m, -)$ denotes the list which is made by removing m variables starting with the order i in ENV.

Of course, we assume that there are enough variables in ENV to be able to define above definitions. Next, we define the morphisms corresponding with above notations of list. Let X be the type of FV (or $FV(i, m, +)$) and $X_{i,m}$ be $FV(i, m, -)$ (or FV resp). We define $\gamma_{i,m} : X \to X_{i,m}$ as follows.

$$\gamma_{i,m} = < \pi, \gamma_{i-1,m}\pi' > \quad : \quad X_0 \times Y \to X_0 \times Z$$

where $\gamma_{i-1,m} : Y \to Z$

and X_0 is the type of the first variable in FV.

$$\gamma_{0,m} = \pi'^m \quad : \quad X_0 \times (\dots (X_{m-1} \times Y)\dots) \to Y,$$

where $\gamma_{0,0} = id : Y \to Y$.

Finally, we get the following property of manipulation of list.

PROPOSITION 3.1. *Let $M \in \Lambda(FV)$ be the λ-term of type A. If FV has type Y and $FV(i, m, +)$ has X, then,*

$$\mathfrak{C}(M, FV(i, m, +)) = \mathfrak{C}(M, FV)\gamma_{i,m} : X \to A,$$

where $\gamma_{i,m} : X \to Y$.

We prove only two cases, variable and λ-abstraction. If $M \equiv x$ and x is the jth variable in ENV, then the first case $0 \leq j < i$,

$$\mathfrak{C}(x, ENV(i, m, +)) = \pi\pi'^j.$$

$$\begin{aligned}
\mathfrak{C}(x, ENV)\gamma_{i,m} &= \pi\pi'^{j} < \pi, \gamma_{i-1,m}\pi' > \\
&= \pi\pi'^{j-1}\gamma_{i-1,m}\pi' \\
&\vdots \\
&= \pi\gamma_{i-j,m}\ \pi'^{j} \\
&= \pi\pi'^{j}.
\end{aligned}$$

The second case $i \leq j$, we can assume that x can not appear the added variables in $ENV(i,\ m,\ +)$.

$$\begin{aligned}
\mathfrak{C}(x, ENV(i, m, +)) &= \pi\pi'^{j+m}. \\
\mathfrak{C}(x, ENV)\gamma_{i,m} &= \pi\pi'^{j}\gamma_{i,m} \\
&= \pi\pi'^{j-1}\gamma_{i-1,m}\pi' \\
&\vdots \\
&= \pi\pi'^{j-i}\gamma_{0,m}\pi'^{i} \\
&= \pi\pi'^{j-i}\pi'^{m}\pi'^{i} \\
&= \pi\pi'^{j+m}.
\end{aligned}$$

If $M \equiv \lambda y.P$, then

$$\begin{aligned}
&\mathfrak{C}((\lambda y.P), ENV(i, m, +)) \\
&= \Lambda(\mathfrak{C}(P, (y, ENV(i, m, +)) < \pi', \pi >)) \\
&= \Lambda(\mathfrak{C}(P, (y, ENV)(i + 1, m, +)) < \pi', \pi >). \\
&\mathfrak{C}((\lambda y.P), ENV)\gamma_{i,m} \\
&= \Lambda(\mathfrak{C}(P, (y, ENV)) < \pi', \pi >)\gamma_{i,m} \\
&= \Lambda(\mathfrak{C}(P,(y, ENV)) < \pi', \gamma_{i,m}\ \pi >) \\
&= \Lambda(\mathfrak{C}(P, (y, ENV)) < \pi, \gamma_{i,m}\pi' >< \pi', \pi >) \\
&= \Lambda(\mathfrak{C}(P, (y, ENV))\ \gamma_{i+1,m} < \pi', \pi >).
\end{aligned}$$

By inductive assumption, $\mathfrak{C}(P,\ (y,\ ENV)(i + 1,\ m,\ +)) = \mathfrak{C}(P,\ (y,\ ENV))\gamma_{i+1,m}$ hence above two formulas are identical. □

Next, we consider the order of variables in the list of them. ENV_i denotes the list which is made by exchanging ith variable with $(i + 1)$th one in the list ENV. Of course, we assume there are enough variables in X to be able to define ENV_i. On the other hand, we define the morphisms corresponding with above notation. Let X be the type of FV and X_i be FV_i. We define $\delta_i : X \to X_i$ as follows.

$$\delta_i \doteq < \pi, \delta_{i-1}\pi' > \quad : \quad X_0 \times Y \to X_0 \times Z$$

$$\text{where} \quad \delta_{i-1} : Y \to Z$$
$$\text{and} \quad X_0 \text{ is the type of the first variable in } FV.$$
$$\delta_0 = <\pi\pi', <\pi, \pi'^2>> \quad : \quad X_0 \times (X_1 \times Y) \to X_1 \times (X_0 \times Y).$$

Then, we get the following property.

PROPOSITION 3.2. *Let $M \in \Lambda(FV)$ be the λ-term of type A. If FV has type X and FV_i has X_i. Moreover, ith and $(i+1)$th variables do not appear in the left position of ith one in FV. Then,*

$$\mathfrak{C}(M, FV_i) = \mathfrak{C}(M, FV)\delta_i : X_i \to A,$$

where $\delta_i : X_i \to X$.

We prove only two cases, variable and λ-abstraction. If $M \equiv x$ and x is the ith variable in ENV, then,

$$\begin{aligned}
\mathfrak{C}(x, ENV_i) &= \pi\pi'^{i+1}. \\
\mathfrak{C}(x, ENV)\delta_i &= \pi\pi'^{i} < \pi, \delta_{i-1}\pi' > \\
&= \pi\pi'^{i-1}\, \delta_{i-1}\pi' \\
&\ \vdots \\
&= \pi\delta_0\pi'^{i} \\
&= \pi\pi'^{i+1}.
\end{aligned}$$

If $M \equiv y$ and y is the $(i+1)$th variable in ENV, then,

$$\begin{aligned}
\mathfrak{C}(y, ENV_i) &= \pi\pi'^{i}. \\
\mathfrak{C}(y, ENV)\delta_i &= \pi\pi'^{i+1} < \pi, \delta_{i-1}\pi' > \\
&= \pi\pi'^{i-1}\delta_{i-1}\pi' \\
&\ \vdots \\
&= \pi\pi'\delta_0\pi'^{i} \\
&= \pi\pi'^{i}.
\end{aligned}$$

If $M \equiv z$ and z is the jth variable in ENV and $j < i$, then,

$$\begin{aligned}
\mathfrak{C}(z, ENV_i) &= \pi\pi'^{j}. \\
\mathfrak{C}(z, ENV)\delta_i &= \pi\pi'^{j} < \pi, \delta_{i-1}\pi' > \\
&= \pi\pi'^{j-1}\, \delta_{i-1}\pi' \\
&\ \vdots \\
&= \pi\delta_{i-j}\pi'^{j} \\
&= \pi\pi'^{j}.
\end{aligned}$$

If $i + 1 < j$, then,

$$\begin{aligned}
\mathfrak{C}(z, ENV_i) &= \pi\pi'^j. \\
\mathfrak{C}(z, ENV)\delta_i &= \pi\pi'^j < \pi, \delta_{i-1}\pi' > \\
&= \pi\pi'^{j-1}\delta_{i-1}\pi' \\
&\vdots \\
&= \pi\pi'^{j-i}\delta_0\pi'^i \\
&= \pi\pi'^{j-i-2}\pi'^2\pi'^i \\
&= \pi\pi'^j.
\end{aligned}$$

We must only consider $M \equiv \lambda z.P$, where $z \not\equiv x$ and $z \not\equiv y$, then,

$$\begin{aligned}
&\mathfrak{C}((\lambda z.P), ENV_i) \\
&= \Lambda(\mathfrak{C}(P, (z, ENV_i)) < \pi', \pi >) \\
&= \Lambda(\mathfrak{C}(P, (z, ENV)_{i+1}) < \pi', \pi >). \\
&\mathfrak{C}((\lambda z.P), ENV)\delta_i \\
&= \Lambda(\mathfrak{C}(P, (z, ENV)) < \pi', \pi >)\delta_i \\
&= \Lambda(\mathfrak{C}(P, (z, ENV)) < \pi, \delta_i\pi' >< \pi', \pi >) \\
&= \Lambda(\mathfrak{C}(P, (z, ENV))\delta_{i+1} < \pi', \pi >).
\end{aligned}$$

By inductive assumption, $\mathfrak{C}(P, (z, ENV)_{i+1}) = \mathfrak{C}(P, (z, ENV))\delta_{i+1}$ hence above two formulas are identical. □

4. Some Properties of Two Categorical Models

Before we compare $\mathcal{L}_{FV}$ with $\mathfrak{C}_{FV}$ in the categorical situation, in this section, we investigate some properties of them respectively. First, we extend the algorithm κ introduced in Section 2 to be able to treat several indeterminates. We get the following commutativity for the order of several applications of algorithm κ.

PROPOSITION 4.1. *Let $\phi(x, y)$ be a polynomial $C \to D$ in indeterminates $x : 1 \to A$ and $y : 1 \to B$ over CCC $\mathfrak{A}$. Then,*

$$\kappa_x(\kappa_y\phi(x, y)) = \kappa_y(\kappa_x\phi(x, y))\delta_0.$$

δ_0 is isomorphism which replaces the order of products. That is, the essential part of above equation is that the order of several applications of algorithm κ only effects the order of products of the domain of the resulting morphism. For $\phi(x_0, \ldots, x_{n-1})$ in polynomial CCC $\mathfrak{A}[x_0, \ldots, x_{n-1}]$, we abbreviate several applications of κ, $\kappa_{x_{n-1}}(\ldots(\kappa_{x_0}\phi(x_0, \ldots, x_{n-1}))\ldots)$, to $\kappa_{x_{n-1}, \ldots, x_0}\,\phi(x_0, \ldots, x_{n-1})$.

The first translation $\mathcal{L}_{FV}$ gives the following identities.

PROPOSITION 4.2. *Let M, N be λ-terms in $\Lambda(FV)$ and x be a variable in FV. Then*
1. For substitution, we have

$$\mathcal{L}(M[x := N], FV) =_{\{FV\}} (\kappa_x \mathcal{L}(M,(x, FV))) < \mathcal{L}(N, FV), id >.$$

2. For β-conversion, we have

$$\mathcal{L}((\lambda x.M)N, FV) =_{\{FV\}} \mathcal{L}(M[x := N], FV).$$

3. For η-conversion, if $x \notin FV(M)$, we have

$$\mathcal{L}(\lambda x.(Mx), FV) =_{\{FV\}} \mathcal{L}(M, FV).$$

We prove only two cases, variable and λ-abstraction, of 1 in Proposition 4.2. If $M \equiv x$, then

$$\mathcal{L}(x[x := N], FV) =_{\{FV\}} \mathcal{L}(N, FV).$$

$$\begin{aligned}(\kappa_x \mathcal{L}(x, (x, FV))) < \mathcal{L}(N, FV), id > &=_{\{FV\}} \pi < \mathcal{L}(N, FV), id > \\ &=_{\{FV\}} \mathcal{L}(N, FV).\end{aligned}$$

If $M \equiv y \not\equiv x$, then

$$\begin{aligned}\mathcal{L}(y[x := N], FV) &=_{\{FV\}} \mathcal{L}(y, FV) \\ &=_{\{FV\}} y. \\ (\kappa_x \mathcal{L}(y, (x, FV))) < \mathcal{L}(N, FV), id > &=_{\{FV\}} y\pi' < \mathcal{L}(N, FV), id > \\ &=_{\{FV\}} y.\end{aligned}$$

If $M \equiv \lambda x.P$, then

$$\begin{aligned}&\mathcal{L}((\lambda x.P)[x := N], FV) \\ =_{\{FV\}}\ &\mathcal{L}(\lambda x.P, FV) \\ =_{\{FV\}}\ &\Lambda(\kappa_x \mathcal{L}(P,(x, FV)) < \pi', \pi >).\end{aligned}$$

On the other hand,

$$\begin{aligned}&(\kappa_x \mathcal{L}((\lambda x.P),(x, FV))) < \mathcal{L}(N, FV), id > \\ =_{\{FV\}}\ &\kappa_x(\Lambda(\kappa_x \mathcal{L}(P, (x, x, \mathrm{FV})) < \pi', \pi >)) < \mathcal{L}(N, FV), id > \\ =_{\{FV\}}\ &\Lambda(\kappa_x \mathcal{L}(P, (x, x, FV)) < \pi', \pi >)\ \pi' < \mathcal{L}(N, FV), id > \\ =_{\{FV\}}\ &\Lambda(\kappa_x \mathcal{L}(P, (x, x, FV)) < \pi', \pi >).\end{aligned}$$

Because $\{(x, FV)\} = \{(x, x, FV)\}$, above two formulas are identical.
If $M \equiv \lambda y.P$ and $y \notin FV(N)$, then

$$\begin{aligned}&\mathcal{L}((\lambda y.P)[x := N], FV) \\ =_{\{FV\}}\ &\mathcal{L}(\lambda y.P[x := N], FV)\end{aligned}$$

$= _{\{FV\}} \Lambda(\kappa_y(\mathcal{L}(P[x:=N], (y, FV))) < \pi', \pi >)$

$= _{\{FV\}} \Lambda(\kappa_y(\kappa_x(\mathcal{L}(P, (x, y, FV))) < \mathcal{L}(N, (y, FV)), id >) < \pi', \pi >)$

$= _{\{FV\}} \Lambda(\kappa_{yx}(\mathcal{L}(P, (x, y, FV))) < \pi, \kappa_y < \mathcal{L}(N, (y, FV)), id >>< \pi', \pi >)$

$= _{\{FV\}} \Lambda(\kappa_{y,x}(\mathcal{L}(P, (x, y, FV))) < \pi, < \mathcal{L}(N, (y, FV)), id > \pi' >< \pi', \pi >)$

$= _{\{FV\}} \Lambda(\kappa_{y,x}(\mathcal{L}(P, (x, y, FV))) < \pi', < \mathcal{L}(N, (y, FV)), id > \pi >)$

$= _{\{FV\}} \Lambda(\kappa_{x,y}(\mathcal{L}(P, (x, y, FV))) < \mathcal{L}(N, (y, FV))\, \pi, < \pi', \pi >>)$ (By Proposition 4.1).

On the other hand,

$$(\kappa_y \mathcal{L}((\lambda y.P), (x, FV))) < \mathcal{L}(N, FV), id >$$

$$= _{\{FV\}} \kappa_x(\Lambda(\kappa_y \mathcal{L}(P, (y, x, FV)) < \pi', \pi >)) < \mathcal{L}(N, FV), id >$$

$$= _{\{FV\}} \Lambda(\kappa_x(\kappa_y \mathcal{L}(P, (y, x, FV)) < \pi', \pi >)\alpha) < \mathcal{L}(N, FV), id >$$

$$= _{\{FV\}} \Lambda(\kappa_{x,y} \mathcal{L}(P, (y, x, FV)) < \mathcal{L}(N, FV)\, \pi, < \pi', \pi >>).$$

Because $y \notin FV(N)$, $\mathcal{L}(N, (y, FV)) = _{\{FV\}} \mathcal{L}(N, FV)$, hence above two formulas are identical. If $M \equiv \lambda y.P$ and $y \in FV(N)$, then $z \notin FV(N) \cup FV(P)$,

$\mathcal{L}((\lambda y.P)[x:=N], FV)$

$= _{\{FV\}} \mathcal{L}(\lambda z.P[y:=z][x:=N], FV)$

$= _{\{FV\}} \Lambda(\kappa_z(\kappa_x(\kappa_y(\mathcal{L}(P, (y, x, z, FV))) < z, id >) < \mathcal{L}(N, (z, FV)), id >) < \pi', \pi >)$

$= _{\{FV\}} \Lambda(\kappa_{z,x,y} \mathcal{L}(P, (y, x, z, FV))) < \pi', < \mathcal{L}(N, (z, FV))\, \pi, < \pi', \pi >>>)$

$= _{\{FV\}} \Lambda(\kappa_{x,y}(\mathcal{L}(P, (y, x, z, FV))) < (\mathcal{L}(N, (z, FV))\, \pi, < \pi' \pi > >).$

On the other hand,

$$(\kappa_x \mathcal{L}((\lambda y.P), (x, FV))) < \mathcal{L}(N, FV), id >$$

$$= _{\{FV\}} \Lambda(\kappa_{x,y} \mathcal{L}(P, (y, x, FV)) < \mathcal{L}(N, FV)\pi, < \pi', \pi >>).$$

Because $z \notin FV(N) \cup FV(P)$, we get $\mathcal{L}(P, (y, x, FV)) =_{\{(y,\ x,\ FV)\}} \mathcal{L}(P, (y, x, z, FV))$ and $\mathcal{L}(N, FV) = _{\{FV\}} \mathcal{L}(P, (z, FV))$, hence above two formulas are identical.

Similarly, the second translation $\mathfrak{C}_{FV}$ gives the following identities.

PROPOSITION 4.3. *Let M, N, be λ-terms in $\Lambda(FV)$ and x be a variable in FV. Then*

1. *For substitution, we have*

$$\mathfrak{C}(M[x:=N], FV) = \mathfrak{C}(M, (x, FV)) < \mathfrak{C}(N, FV), id >.$$

2. *For β-conversion, we have*

$$\mathfrak{C}((\lambda x.M)N, FV) = \mathfrak{C}(M[x:=N], FV).$$

3. *For η-conversion, if $x \notin FV(M)$, we have*

$$\mathfrak{C}(\lambda x.(Mx), FV) = \mathfrak{C}(M, FV).$$

Remark that above equation holds not in polynomial CCC but in CCC. We prove only two cases, variable and λ-abstraction, of 1 in Proposition 4.3. If $M \equiv x$, then

$$\begin{aligned}\mathfrak{C}(x[x:=N], FV) &= \mathfrak{C}(N, FV).\\ \mathfrak{C}(x, (x, FV)) < \mathfrak{C}(N, FV), id > &= \pi < \mathfrak{C}(N, FV), id >\\ &= \mathfrak{C}(N, FV).\end{aligned}$$

If $M \equiv y \not\equiv x$ and y is the ith element in FV, then

$$\begin{aligned}\mathfrak{C}(y[x:=N], FV) &= \mathfrak{C}(y, FV)\\ &= \pi\pi'^{i}.\\ \mathfrak{C}(y, (x, FV)) < \mathfrak{C}(N, FV), id > &= \pi\pi'^{i+1} < \mathfrak{C}(N, FV), id >\\ &= \pi\pi'^{i}.\end{aligned}$$

If $M \equiv \lambda x.P$, then

$$\begin{aligned}&\mathfrak{C}((\lambda x.P)[x:=N], FV)\\ &= \mathfrak{C}(\lambda x.P, FV)\\ &= \Lambda(\mathfrak{C}(P, (x, FV)) < \pi', \pi >).\end{aligned}$$

On the other hand,

$$\begin{aligned}&\mathfrak{C}(\lambda x.P, (x, FV)) < \mathfrak{C}(N, FV), id >\\ &= \Lambda(\mathfrak{C}(P, (x, x, FV)) < \pi', \pi >)) < \mathfrak{C}(N, FV), id >\\ &= \Lambda(\mathfrak{C}(P, (x, FV))\ \gamma_{1,1} < \pi', \pi >) < \mathfrak{C}(N, FV), id > \quad \text{(By Proposition 3.1)}\\ &= \Lambda(\mathfrak{C}(P, (x, FV)) < \pi', \pi >).\end{aligned}$$

If $M \equiv \lambda y.P$ and $y \notin FV(N)$, then

$$\begin{aligned}&\mathfrak{C}((\lambda y.P)[x:=N], FV)\\ &= \Lambda(\mathfrak{C}(P[x:=N], (y, FV)) < \pi', \pi >)\\ &= \Lambda(\mathfrak{C}(P, (x, y, FV)) < \mathfrak{C}(N, (y, FV)), id >< \pi', \pi >)\\ &= \Lambda(\mathfrak{C}(P, (x, y, FV)) < \mathfrak{C}(N, FV)\ \pi, < \pi', \pi >>) \quad \text{(By Proposition 3.1).}\end{aligned}$$

On the other hand,

$$\begin{aligned}&\mathfrak{C}(\lambda y.P, (x, FV)) < \mathfrak{C}(N, FV), id >\\ &= \Lambda(\mathfrak{C}(P, (y, x, FV)) < \pi', \pi >) < \mathfrak{C}(N, FV), id >\\ &= \Lambda(\mathfrak{C}(P, (x, y, FV))\delta_0 < \pi', \pi >) < \mathfrak{C}(N, FV), id > \quad \text{(By Proposition 3.2)}\\ &= \Lambda(\mathfrak{C}(P, (x, y, FV)) < \mathfrak{C}(N, FV)\ \pi, < \pi', \pi >>).\end{aligned}$$

If $M \equiv \lambda y.P$ and $y \in FV(N)$, then $z \notin FV(N) \cup FV(P)$,

$$\begin{aligned}
&\mathfrak{C}((\lambda y.P)[x := N], FV)\\
&= \mathfrak{C}(\lambda z.P[y := z][x := N], FV)\\
&= \Lambda(\mathfrak{C}(P, (y, x, z, FV)) < \mathfrak{C}(z, (x, z, FV)), id >< \mathfrak{C}(N, (z, FV)), id >< \pi', \pi >)\\
&= \Lambda(\mathfrak{C}(P, (y, x, FV))\gamma_{2,1} < \pi, < \mathfrak{C}(N, FV)\gamma_{0,1}, id >>< \pi', \pi >)\\
&= \Lambda(\mathfrak{C}(P, (y, x, FV)) < \pi', < \mathfrak{C}(N, FV), id > \pi >). \qquad \text{(By Proposition 3.1)}
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&\mathfrak{C}((\lambda y.P), (x, FV)) < \mathfrak{C}(N, FV), id >\\
&= \Lambda(\mathfrak{C}(P, (y, x, FV)) < \pi', < \mathfrak{C}(N, FV), id > \pi >). \quad \square
\end{aligned}$$

Finally, we easily arrive at the next theorem by using above propositions.

THEOREM 4.1. *Let M, N be λ-terms having same type in $\Lambda(FV)$. Then*

$$\begin{aligned}
\lambda \vdash M = N \quad \Rightarrow \quad &\mathcal{L}(M, FV) =_{\{FV\}} \mathcal{L}(N, FV),\\
&\mathfrak{C}(M, FV) = \mathfrak{C}(N, FV).
\end{aligned}$$

5. Comparison of Two Categorical Models

In this section, we compare $\mathcal{L}_{FV}$ with $\mathfrak{C}_{FV}$ in the categorical situation. $\mathcal{L}_{FV}$ gives the polynomial from 1 to $\mathfrak{A}$ (as an object) in $\mathfrak{A}[x_0, \ldots, x_{n-1}]$ to λ-term M of type A in $\Lambda(FV)$. On the other hand, $\mathfrak{C}_{FV}$ does the morphism in $\mathfrak{A}$,

$$\mathfrak{C}(M, FV) : A_0 \times (A_1 \times (\ldots(A_{n-1} \times 1)\ldots)) \to A,$$

to it, where $FV = (x_0, x_1, \ldots, x_{n-1}, *)$ and each x_i has type A_i. But we can show $\mathfrak{C}_{FV}$ gives not simply the morphism in $\mathfrak{A}$ but one in $(\ldots(\mathfrak{A}_{s_{A_{n-1}}}\ldots)_{s_{A_0}}$ to M of type A in $\Lambda(FV)$.

For the purpose of it, we sequentially apply several algorithms κ to $\mathcal{L}(M, FV)$ and compare the resulting morphism with $\mathfrak{C}(M, FV)$. The order of applications of κ is descendant, that is, first apply κ_{n-1}, second κ_{n-2}, $\ldots$, and finally κ_0. Then, we get the morphism,

$$\kappa_{x_0, x_1, \ldots, x_{n-1}} \mathcal{L}(M, FV) : A_0 \times (A_1 \times (\ldots(A_{n-1} \times 1)\ldots)) \to A,$$

which does not include indeterminates. We arrive at the following main result of this paper.

THEOREM 5.1. *Let M be λ-term of type A in $\Lambda(FV)$ and $FV = (x_0, x_1, \ldots, x_{n-1}, *)$. Then,*

$$\kappa_{x_0, x_1, \ldots, x_{n-1}} \mathcal{L}(M, FV) = \mathfrak{C}(M, FV).$$

We prove only two cases, variable and λ-abstraction. Let FV be $(x_0, x_1, \ldots, x_{n-1}, *)$ and each x_i has type A_i. If $M \equiv x_i$ in $FV(0 \le i < n)$, then,

$$\begin{aligned}
&\kappa_{x_0, \ldots, x_{n-1}} \mathcal{L}(x_i, FV)\\
&= \kappa_{x_0, \ldots, x_{n-1}} x_i\\
&= \kappa_{x_0, \ldots, x_{n-2}} x_i \pi'_{A_{n-1}, 1}\\
&\ \vdots\\
&= \kappa_{x_0, \ldots, x_i} x_i \pi'^{n-1-i}\\
&= \kappa_{x_0, \ldots, x_{i-1}} \pi_{A_i, 1} < \pi, \pi'^{n-1-i} \pi' >\\
&= \kappa_{x_0, \ldots, x_{i-1}} \pi_{A_i, A_{i+1} \times (\ldots (A_{n-1} \times 1) \ldots)}\\
&\ \vdots\\
&= \pi \pi'^{i}.
\end{aligned}$$

If $M \equiv \lambda y.P$, y may occur in FV, then we α-convert $\lambda y.P$ into $\lambda z.P[y := z]$ where z is not included in FV, which is assured by Theorem 4.1.

$$\begin{aligned}
&\kappa_{x_0, \ldots, x_{n-1}} \mathcal{L}(\lambda y.P, FV)\\
&= \kappa_{x_0, \ldots, x_{n-1}} \Lambda(\kappa_y \mathcal{L}(P, (y, FV)) < \pi', \pi >)\\
&= \kappa_{x_0, \ldots, x_{n-2}} \Lambda(\kappa_{x_{n-1}} \kappa_y \mathcal{L}(P, (y, FV)) < \pi, < \pi', \pi > \pi' > \alpha)\\
&= \kappa_{x_0, \ldots, x_{n-2}} \Lambda(\kappa_{x_{n-1}, y} \mathcal{L}(P, (y, FV)) < \pi\pi, < \pi', \pi'\pi >>)\\
&= \kappa_{x_0, \ldots, x_{n-2}} \Lambda(\kappa_{y, x_{n-1}} \mathcal{L}(P, (y, FV))\delta_0 < \pi\pi, < \pi', \pi'\pi >>) \quad \text{(By Proposition 4.1)}\\
&= \kappa_{x_0, \ldots, x_{n-2}} \Lambda(\kappa_{y, x_{n-1}} \mathcal{L}(P, (y, FV)) < \pi', \pi >)\\
&\ \vdots\\
&= \Lambda(\kappa_{x_0, \ldots, x_{n-1}} \mathcal{L}(P, (y, FV)) < \pi', \pi >)\\
&= \Lambda(\mathfrak{C}(P, (y, FV)) < \pi', \pi >) \quad \text{(By inductive assumption)}\\
&= \mathfrak{C}(\lambda y.P, FV). \quad \square
\end{aligned}$$

According to Theorem 2.2 (functional completeness) in Section 2, $\mathfrak{C}(M, FV)$ may be considered as Kleisli morphism i.e. morphism in Kleisli category in the following sense.

$$\begin{aligned}
\mathfrak{A}[x_0, \ldots, x_{n-1}] &\cong (\ldots(\mathfrak{A}[x_{n-1}])\ldots)[x_0]\\
&\cong (\ldots(\mathfrak{A}_{S_{A_{n-1}}})\ldots)_{S_{A_0}}
\end{aligned}$$

where $FV = (x_0, \ldots, x_{n-1}, *)$ and $x_i : 1 \to A_i (0 \le i < n)$. Additionally, we get the following corollary.

Corollary 5.1. *Let M be λ-term of type A in $\Lambda(FV)$ and $FV = (x_0, x_1, \ldots, x_{n-1}, *)$. Then,*

$$\mathcal{L}(M,\ FV) = \mathfrak{C}(M,\ FV) < x_0, < \ldots < x_{n-1},\ id_1 > \ldots >>.$$

Moreover, it is important that above Kleisli category is an extended CCC and has the *cartesian closed structure*. This theorem asserts that $\mathfrak{C}_{FV}$ translates λ-terms into not only morphisms in the CCC but also ones in Kleisli category of it. It means that from the categorical point of view, $\mathfrak{C}_{FV}$ essentially adopt Kleisli category, while $\mathcal{L}_{FV}$ does polynomial CCC.

However polynomial CCC does have indeterminates, Kleisli category does not have them. This difference corresponds with that between λ-calculus and de Bruijn's name-free expression. Moreover so-called *indexes* of variables in FV, which characterize the composition of several projections, correspond with the minimal components in de Bruijn's name-free expression. That is, it gives rise to not only de Bruijn's name-free expression in λ-calculus but also Kleisli category (or it's construction) in the categorical model.

6. Conclusion

We have compared two categorical models of typed λ-calculus which are both CCC. It is clear that the first model (polynomial CCC) is not merely CCC but extended one and has certain universal property. We show that the second one is also does. Consequently, two categorical models are both extended CCC having certain universality and equivalent from the categorical point of view. Of course, categorical models presented in this paper are quite simpler than others in Curien [2], Koymans [4]. But essential parts of them are same those of others. For example, it is indeterminate corresponding with variable for the first model. On the other hand, it is so-called index of variable based on de Bruijn's name-free expression for the second model.

For the purpose that we will investigate the categorical properties and relation of models, we adopt simple models instead of original but complicated ones in Curien [2], Koymans [4].

The studies of categorical investigation of λ-calculus are actively accomplished by Curien [3] et al. They would treat CCC or C-monoid (type free version of CCC) as syntactic systems, e.g., categorical combinatory logic, while we do as description of semantics of λ-calculus. They extend classical combinatory logic to categorical one and try to reconstruct several properties, e.g., syntactic equivalence theorem, confluency under some systems of several axioms. Because our investigation mainly consider CCC as a (categorical) semantics of typed λ-calculus, it would not be immediately concatenated with their studies.

By the way, because Kleisli category of some comonad over CCC becomes categorical model of typed λ-calculus and it is based on de Bruijn's name-free expression, we can consider C-monoid (not polynomial C-monoid) as a categorical model of not λ-calculus but de Bruijn's name-free expression in the type free situation. Moreover, diverting Kleisli's method to certain construction of C-monoid, we will deal with extended C-monoid (of course, this is not polynomial C-monoid) as a categorical model of it.

Acknowledgment

I am grateful thanks to Yasuo Kawahara for some useful discussions and comments.

References

[1] Barendregt, H.: *The lambda calculus: syntax and semantics, rev. ed,* North Holland, Amsterdam, (1984).

[2] Curien, P.-L.: *Categorical combinators.* Inform. Control, **69** (1985), 188–254.

[3] Curien, P.-L.: *Categorical combinators, sequential algorithms and functional programming*, Pitman, London, (1985).

[4] Koymans, C. P. J.: *Models of the lambda calculus.* Inform. Control, **52** (1982), 306–332.

[5] Lambek, J. and Scott, P. J.: *Higher order categorical logic, part i: cartesian closed categories and lambda calculus*, (1986).

[6] Lambek, J. and Scott, P. J.: *Introduction to higher order categorical logic,* Cambridge University Press, (1985).

[7] MacLane, S.: *Categories for the working mathematician*, Springer-Verlag, New York/Berlin, (1972).

[8] Pareigis, B.: *Categories and Functors,* (1970).

[9] Poigne, A.: *On specifications, theories, and models with higher types*, Inform. Control, **68** (1986), 1–46.

Received September 7, 1990
Communicated by Y. Kawahara