Duality between Call-by-name Recursion and Call-by-value Iteration

Yoshihiko Kakutani

Research Institute for Mathematical Sciences, Kyoto University
kakutani@kurims.kyoto-u.ac.jp

Abstract. We investigate the duality between call-by-name recursion and call-by-value iteration on the $\lambda\mu$-calculi. The duality between call-by-name and call-by-value was first studied by Filinski, and Selinger has studied the category-theoretic duality on the models of the call-by-name $\lambda\mu$-calculus and the call-by-value one. We extend the call-by-name $\lambda\mu$-calculus and the call-by-value one with a fixed-point operator and an iteration operator, respectively. We show that the dual translations constructed by Selinger can be expanded into our extended $\lambda\mu$-calculi, and we also discuss their implications to practical applications.

Introduction

Duality

In this thesis, we study duality between recursion and iteration in functional languages with first-class continuations.

The roots of duality between recursion and iteration are in the duality between values and continuations, which was first suggested by Filinski in [6]. A function f of the type $A \to B$ usually represents a computation which accepts a value $x : A$ and returns a value $f(x) : B$. However, such a function has another canonical meaning, *i.e.*, the meaning given by continuation semantics. Let R be the return type of continuations. A continuation of type A is a computation from A to R. While intuitively a value means the result of a computation in a point, a continuation means the rest of a computation in a point. A typical example of continuation usage is continuation passing style (*e.g.*, see [22]). In the continuation semantics, we can consider $f : A \to B$ transforms a continuation g of B to a continuation $g \circ f$ of A. This implies that the exchange of value paradigm for continuation paradigm makes the directions of computations reverse.

The duality between call-by-name and call-by-value is induced from the duality between values and continuations. Any β-redex can be reduced with no restriction in the call-by-name λ-calculus, while β-reduction in the call-by-value λ-calculus is restricted to the case of application to values. Conversely, in the practical call-by-value language enriched with first-class continuations, for instance, SML/NJ [11], continuation handling is rather freely allowed. Therefore the duality arises between call-by-name languages and call-by-value ones if we restrict call-by-name continuation handlings appropriately. The call-by-name λ-calculus and the call-by-value one have some symmetry, but it is not so obvious because of the lack of continuations.

It is the aim of this thesis to make explicit this duality about recursive computation and iterative computation.

First-class continuations and $\lambda\mu$-calculi

Advantage brought by continuations is not merely the beauty of symmetry. First-class continuations make functional languages more expressive [18], and allow us to describe more complicated programs. Indeed a '*call-with-current-continuation*' operator, which is one of the most typical examples of the usage of first-class continuations, enables λ-calculi to express the notion of 'goto' (*cf.* Landin's `J` operator [17]). Here we give an informal explanation of `callcc` that means 'call-with-current-continuation'. We shall consider a simply typed call-by-value λ-calculus. Although a practical language may be polymorphically typed (SML/NJ) or untyped (scheme), `callcc` is essentially not so different. We introduce the constants `callcc` and `throw` such that

$$\texttt{callcc} : (\neg A \to A) \to A,$$
$$\texttt{throw} : \neg A \to A \to B,$$

where $\neg A$ means the type of continuations of A, and their operational semantics are

$$\begin{aligned}\mathbf{E}[\texttt{callcc}\,V] &\to \mathbf{E}[V\,(\Lambda x^A.\,\mathbf{E}[x])],\\ \mathbf{E}[\texttt{throw}(\Lambda x^A.\,M)\,V] &\to M\,[V\,/x],\end{aligned}$$

where $\mathbf{E}[-]$ is an evaluation context and V is a value (the latter condition is needed by call-by-value only). $\Lambda x^A.\,M$, which is not included by the language but occurs in reduction steps, has the type $\neg A$. Intuitively `callcc` carries the current context into its argument, and `throw` aborts the current context and switches the context to first argument. We can also consider call-by-name version without a big change. While arguments in β-reductions are *more* restricted in a call-by-value language than in a call-by-name one, evaluation contexts are *less* restricted in a call-by-value language than in a call-by-name one. The following is an example of programs with `callcc`:

$$L0 \to (\lambda k.\lambda x.\ \texttt{throw}\,k\,(\lambda y.\,x))(\Lambda z.\,z0)\,0 \to^* \texttt{throw}(\Lambda z.\,z0)\,(\lambda y.\,0) \to (\lambda y.\,0)0 \to 0$$

where $L = \texttt{callcc}(\lambda k.\lambda x.\ \texttt{throw}\,k\,(\lambda y.\,x))$. (This L has an interesting feature that $\langle L, L\rangle$ is not equal to $(\lambda x.\langle x, x\rangle)L$ [28].)

$\lambda\mu$-calculi are extensions of λ-calculi with control operators. The call-by-value $\lambda\mu$-calculus is as expressive as other popular functional languages with first-class continuations. For example, above `callcc` and `throw` are expressed via the following encoding:

$$\begin{aligned}\overline{\neg A} &= \overline{A} \to \bot,\\ \overline{\texttt{callcc}\,M : A} &= \mu\alpha^{\overline{A}}.[\alpha]\,\overline{M}\,(\lambda x^{\overline{A}}.[\alpha]\,x) : \overline{A},\\ \overline{\texttt{throw}\,N\,M : B} &= \mu\beta^{\overline{B}}.\,\overline{N}\,\overline{M} : \overline{B}.\end{aligned}$$

Felleisen's $\mathcal{C}$ operator and $\mathcal{A}$ operator, which have been studied widely [3–5, 13], also can be expressed in the call-by-value $\lambda\mu$-calculus.

Since $\lambda\mu$-calculi deal with both values and continuations, the duality between call-by-name and call-by-value is expanded into $\lambda\mu$-calculi. Indeed, in [25] Selinger has given categorical models called control categories and co-control categories respectively to the call-by-name $\lambda\mu$-calculus (which we call the $\lambda\mu_{\mathrm{n}}$-calculus) and the call-by-value calculus (the $\lambda\mu_{\mathrm{v}}$-calculus), where co-control categories are the opposite categories of control categories. The call-by-name $\lambda\mu$-calculus includes the full call-by-name λ-calculus, and the call-by-value $\lambda\mu$-calculus is an extension of Moggi's λ_{c}-calculus [19], which is a standard calculus for reasoning about call-by-value programming.

Additionally, the following fact is well-known: Curry-Howard correspondences exist between $\lambda\mu$-calculi and classical logics [21, 20]. (The fact that control operators enable calculi to correspond with classical logics was first discovered by Griffin [10].) Proof-theoretically, the duality is a kind of De Morgan duality [2].

Recursion and iteration

Recursion is indispensable for programming languages and has been studied very extensively. However, most of such widely-known studies, which includes uniformity, dinaturality, and diagonal property [1], is for call-by-name languages rather than for call-by-value ones. Therefore, we are to add a recursion operator to the $\lambda\mu_{\mathrm{n}}$-calculus and to consider the duality.

By the duality between values and continuations, we can get recursion on continuations in the call-by-value calculus from call-by-name recursion. Recursion on continuations is just iteration because recursors on negative types and iterators have a bijective correspondence in the call-by-value calculus (strictly speaking, which requires stability axiom for recursors). This construction is based on Filinski's 'recursion from iteration' [7]. He showed that in the call-by-value language with first-class continuations recursion operators were definable by iteration operators and *vice versa*. Since the $\lambda\mu_{\mathrm{v}}$-calculus can express the first-class controls as mentioned above, the $\lambda\mu_{\mathrm{v}}$-calculus is fit for that schema.

The categorical investigation leads us to the duality between call-by-name recursion and call-by-value iteration more directly. Namely, a fixed-point operator on a control category is exactly dual for an iteration operator on a co-control category.

On the other hand, in [7], Filinski also proposed the uniformity principles for fixed-point operators and iteration operators. We refine and justify the uniformity principle in [12]. The $\lambda\mu_{\mathrm{v}}$-calculus can enjoy its results.

Overview

In this paper, we recall the $\lambda\mu$-calculi in Section 1, and their categorical semantics in Section 2. Section 2 also gives an investigation of the duality between call-by-name recursion and call-by-value iteration from the categorical point of view. We reach the duality on the $\lambda\mu$-calculi at Section 3 by introducing a fixed-point operator to the $\lambda\mu_{\mathrm{n}}$-calculus, and an iteration operator to the $\lambda\mu_{\mathrm{v}}$-calculus. We extend the dual translations between the $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus with the recursion operator and the iteration one. In Section 4, we consider some implication of our duality to the programming practice: first we study the correspondence between recursion and iteration in the call-by-value language, and second we give programming examples with inductive and co-inductive data types.

1 The $\lambda\mu$-calculi

1.1 Syntax

A $\lambda\mu$-calculus was first introduced by Parigot in [21]. $\lambda\mu$-calculi are extensions of λ-calculi with the notion of continuations. In this section, we define the syntax of the $\lambda\mu$-calculi, both the call-by-name calculus and the call-by-value one, and give CPS semantics. Our version of the $\lambda\mu$-calculi is based on Selinger's [25]:

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A \mid \Delta}\ (\mathrm{id}) \qquad \frac{}{\Gamma \vdash * : \top \mid \Delta}\ (\mathrm{top})$$

$$\frac{\Gamma, x : A \vdash M : B \mid \Delta}{\Gamma \vdash \lambda x^A.\, M : A \to B \mid \Delta}\ (\mathrm{abs}) \qquad \frac{\Gamma \vdash M : A \to B \mid \Delta \quad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash MN : B \mid \Delta}\ (\mathrm{app})$$

$$\frac{\Gamma \vdash M : A \mid \Delta \quad \Gamma \vdash N : B \mid \Delta}{\Gamma \vdash \langle M, N \rangle : A \wedge B \mid \Delta}\ (\mathrm{pair}) \qquad \frac{\Gamma \vdash M : A_1 \wedge A_2 \mid \Delta}{\Gamma \vdash \pi_i M : A_i \mid \Delta}\ (\mathrm{proj})$$

$$\frac{\Gamma \vdash M : \bot \mid \alpha : A, \Delta}{\Gamma \vdash \mu\alpha^A.\, M : A \mid \Delta}\ (\mathrm{capt}) \qquad \frac{\Gamma \vdash M : A \mid \Delta \quad \alpha : A \in \Delta}{\Gamma \vdash [\alpha]\, M : \bot \mid \Delta}\ (\mathrm{release})$$

$$\frac{\Gamma \vdash M : \bot \mid \beta : B, \alpha : A, \Delta}{\Gamma \vdash \mu(\alpha^A, \beta^B).\, M : A \vee B \mid \Delta}\ (\mathrm{inj}) \qquad \frac{\Gamma \vdash M : A \vee B \mid \Delta \quad \alpha : A, \beta : B \in \Delta}{\Gamma \vdash [\alpha, \beta]\, M : \bot \mid \Delta}\ (\mathrm{choice})$$

Fig. 1. The deduction rules of the $\lambda\mu$-calculi

including conjunction types and disjunction types. Conjunction types are dual notion of disjunction types, *i.e.*, call-by-name disjunctions play the role of call-by-value conjunctions, and call-by-value disjunctions play the role of call-by-name conjunctions.

The formal syntax is the following:

$$\begin{aligned}
A, B &::= \sigma \mid A \to B \mid \top \mid A \wedge B \mid \bot \mid A \vee B, \\
M, N &::= x \mid * \mid \lambda x^A.\, M \mid MN \mid \langle M, N \rangle \mid \pi_1 M \mid \pi_2 M \\
&\quad \mid \mu\alpha^A.\, M \mid [\alpha]\, M \mid \mu(\alpha^A, \beta^B).\, M \mid [\alpha, \beta]\, M, \\
V, W &::= x \mid * \mid \lambda x^A.\, M \mid \langle V, W \rangle \mid \pi_1 V \mid \pi_2 V \\
&\quad \mid \mu(\alpha^A, \beta^B).[\alpha]\, V \mid \mu(\alpha^A, \beta^B).[\beta]\, V,
\end{aligned}$$

where x ranges over variables, α and β range over names, and σ ranges over base types. A, M and V are called **types**, **terms** and **values**, respectively. Values make sense only for the call-by-value calculus. The symbol $*$ denotes a special constant with the type $\top$. We assume the usual associative strength among connectives ($\mu\alpha^A.\,(-)$ and $[\alpha]\,(-)$ is as strong as $\lambda x^A.\,(-)$), and the set denoted by $\mathrm{FV}(-)$ of free variables on terms is defined in the same way as λ-calculi. We also define $\mathrm{FN}(-)$, the set of free names on terms, where μ-abstractions bind names. For abbreviation, we may write $\neg A$ for $A \to \bot$ and in the call-by-value calculus we use $\mathtt{let}\, x^A\, \mathtt{be}\, M\, \mathtt{in}\, N$ as syntactic sugar for $(\lambda x^A.\, N)M$.

Every **judgement** takes the form $\Gamma \vdash M : A \mid \Delta$, where Γ denotes a sequence of pairs $x : A$, and Δ denotes a sequence of pairs $\alpha : A$. The typing rules are given in Figure 1, which are applied to both the call-by-name $\lambda\mu$-calculus and the call-by-value one. In this paper, we consider only derivable judgements and we may confuse a judgement itself with the predicate that means the judgement is deducible.

$$
\begin{array}{rl}
V_\sigma \equiv \underline{\sigma} & V_{A\to B} \equiv K_A \times V_B \quad K_A \equiv V_A \to R \\
V_\top \equiv 0 & V_\bot \equiv 1 \\
V_{A\wedge B} \equiv V_A + V_B & V_{A\vee B} \equiv V_A \times V_B
\end{array}
$$

$$
\begin{array}{rcll}
\widehat{x} & = & \lambda k^{V_A}.\, \underline{x} k & x : A \\
\widehat{*} & = & \lambda k^{V_\top}.\, \Box_R k & \\
\widehat{\lambda x^A.\, M} & = & \lambda \langle x^{K_A}, k^{V_B} \rangle.\, \widehat{M} k & M : B \\
\widehat{MN} & = & \lambda k^{V_B}.\, \widehat{M} \langle \widehat{N}, k \rangle & M : A \to B,\ N : A \\
\widehat{\langle M, N \rangle} & = & \lambda k^{V_{A\wedge B}}.\, [\widehat{M}, \widehat{N}]\, k & M : A,\ N : B \\
\widehat{\pi_i M} & = & \lambda k^{V_{A_i}}.\, \widehat{M} (\mathtt{in}_i k) & M : A_1 \wedge A_2 \\
\widehat{\mu \alpha^A.\, M} & = & \lambda \underline{\alpha}^{V_A}.\, \widehat{M} * & \\
\widehat{[\alpha]\, M} & = & \lambda k^{V_\bot}.\, \widehat{M} \underline{\alpha} & \\
\widehat{\mu(\alpha^A, \beta^B).\, M} & = & \lambda \langle \underline{\alpha}^{V_A}, \underline{\beta}^{V_B} \rangle.\, \widehat{M} * & \\
\widehat{[\alpha, \beta]\, M} & = & \lambda k^{V_\bot}.\, \widehat{M} \langle \underline{\alpha}, \underline{\beta} \rangle &
\end{array}
$$

Fig. 2. The CPS translation of the $\lambda\mu_{\mathrm{n}}$-calculus

1.2 The CPS semantics

We will give semantics to the $\lambda\mu$-calculi by **CPS translations**. The acronym 'CPS' is the abbreviation of '*continuation passing style*'. Continuation passing style is one of the popular way to give denotational and operational semantics to programming languages. The CPS target language is the call-by-name λ-calculus with finite products, finite sums and a distinguished type R, whose variables include all $\underline{x}$ and $\underline{\alpha}$. In this paper, we adopt the syntax defined by $M, N ::= x \mid * \mid \Box_A \mid \lambda x^A.\, M \mid MN \mid \pi_i M \mid \langle M, N \rangle \mid \mathtt{in}_i M \mid [M, N]$, and consider the normal $\beta\eta$-equality. $\lambda x^A.\, * : A \to \top$ represents the terminal map, while $\Box_A : \bot \to A$ represents the initial map. If you like the usual ML-like notation, $[M, N]$ is denoted by $\lambda k^{A+B}.\, \mathtt{case}\, k\, \mathtt{of}\, \mathtt{in}_1 k_1 \Rightarrow M k_1 \mid \mathtt{in}_2 k_2 \Rightarrow N k_2$. When $\lambda \langle x^A, y^B \rangle . M$ is an abbreviation for $\lambda z^{A\times B}.(\lambda x^A.\lambda y^B.\, M)(\pi_1 z)\,(\pi_2 z)$ and $\lambda [\alpha^{A\to C}, \beta^{B\to C}].\, M$ is an abbreviation for $\lambda \gamma^{A+B\to C}.(\lambda \alpha^{A\to C}.\lambda \beta^{B\to C}.\, M)(\lambda x^A.\, \gamma(\mathtt{in}_1 x))\,(\lambda y^A.\, \gamma(\mathtt{in}_2 y))$, the CPS translations are described in Figure 2 and 3. One is for the call-by-name $\lambda\mu$-calculus, and the other is for the call-by-value one. In the call-by-name semantics, the judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$ is translated to the judgement

$$\underline{x_1} : K_{B_1}, \ldots, \underline{x_1} : K_{B_1}, \underline{\alpha_1} : V_{A_1}, \ldots, \underline{\alpha_m} : V_{A_m} \vdash \widehat{M} : K_A,$$

and in the call-by-value semantics, the judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$ is translated to the judgement

$$\underline{x_1} : V_{B_1}, \ldots, \underline{x_n} : V_{B_n}, \underline{\alpha_1} : K_{A_1}, \ldots, \underline{\alpha_m} : K_{A_m} \vdash \widetilde{M} : C_A.$$

The call-by-name CPS translation is based on that of Hofmann and Streicher [14] rather than Plotkin's original translation [22], and the call-by-value CPS translation is essentially the same on λ-terms as Plotkin's [22]. In the same

$$
\begin{array}{rlll}
V_\sigma \equiv \underline{\sigma} & V_{A\to B} \equiv V_A \times K_B \to R & K_A \equiv V_A \to R & \\
V_\top \equiv 1 & V_\bot \equiv 0 & C_A \equiv K_A \to R & \\
V_{A\wedge B} \equiv V_A \times V_B & V_{A\vee B} \equiv V_A + V_B & & \\
\widetilde{x} = \lambda k^{K_A}.\, k\, \underline{x} & & & x : A \\
\widetilde{*} = \lambda k^{K_\top}.\, k* & & & \\
\widetilde{\lambda x^A.\, M} = \lambda k^{K_{A\to B}}.\, k(\lambda\langle x^{V_A}, c^{K_B}\rangle.\, \widetilde{M}c) & & & M : B \\
\widetilde{MN} = \lambda k^{K_B}.\, \widetilde{M}(\lambda m^{V_{A\to B}}.\, \widetilde{N}(\lambda n^{V_A}.\, m\langle n, k\rangle)) & & & M : A \to B,\ N : A \\
\widetilde{\langle M, N\rangle} = \lambda k^{K_{A\wedge B}}.\, \widetilde{M}(\lambda m^{V_A}.\, \widetilde{N}(\lambda n^{V_B}.\, k\langle m, n\rangle)) & & & M : A,\ N : B \\
\widetilde{\pi_i M} = \lambda k^{K_{A_i}}.\, \widetilde{M}(\lambda m^{V_{A_1\wedge A_2}}.\, k(\pi_i m)) & & & M : A_1 \wedge A_2 \\
\widetilde{\mu\alpha^A.\, M} = \lambda \underline{\alpha}^{K_A}.\, \widetilde{M}\Box_R & & & \\
\widetilde{[\alpha]\, M} = \lambda k^{K_\bot}.\, \widetilde{M}\underline{\alpha} & & & \\
\widetilde{\mu(\alpha^A, \beta^B).\, M} = \lambda[\underline{\alpha}^{K_A}, \underline{\beta}^{K_B}].\, \widetilde{M}\Box_R & & & \\
\widetilde{[\alpha, \beta]\, M} = \lambda k^{K_\bot}.\, \widetilde{M}[\underline{\alpha}, \underline{\beta}] & & &
\end{array}
$$

Fig. 3. The CPS translation of the $\lambda\mu_{\mathrm{v}}$-calculus

way as the equational theory of A-normal form [8] gives sound and complete axioms for the call-by-value λ-calculus, we can axiomatize the equality on the $\lambda\mu$-judgements that is sound and complete for each CPS semantics. The equality axioms are defined by Figure 4 and 5, where an expression like $[N/x]$ is a usual substitution for free variables or names, but an expression $[\mathbf{C}[(-)]/[\alpha](-)]$, called a mixed substitution, not only replaces all free $[\alpha]\, M$ by $\mathbf{C}[M]$, but also replaces $[\alpha, \beta]\, M$ and $[\beta, \alpha]\, M$ by $\mathbf{C}[\mu\alpha^A.[\alpha, \beta]\, M]$ and $\mathbf{C}[\mu\alpha^A.[\beta, \alpha]\, M]$ respectively. In the figures, a form $M = N : A$ means that $\Gamma \vdash M : A \mid \Delta$ equals $\Gamma \vdash N : A \mid \Delta$ for any Γ and Δ. The equalities are induced into judgements from these axioms by canonical way, that is, constructing reflective transitive and compatible closure. The $\lambda\mu$-calculus with the equational theory derived from Figure 4 is named **the $\boldsymbol{\lambda\mu_{\mathrm{n}}}$-calculus**, and the $\lambda\mu$-calculus with the equational theory derived from Figure 5 is named **the $\boldsymbol{\lambda\mu_{\mathrm{v}}}$-calculus**. In [25], the $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus are called the call-by-name $\lambda\mu$-calculus and the call-by-value $\lambda\mu$-calculus respectively. We write $\Gamma \vdash M =_{\mathrm{n}} N : A \mid \Delta$ for $\Gamma \vdash M : A \mid \Delta$ is equal to $\Gamma \vdash N : A \mid \Delta$ in the $\lambda\mu_{\mathrm{n}}$-calculus. Moreover, we may write simply $M =_{\mathrm{n}} N$ when A, Γ and Δ are obvious from context. So is the call-by-value case too: $M =_{\mathrm{v}} N$ means $\Gamma \vdash M : A \mid \Delta$ is equal to $\Gamma \vdash N : A \mid \Delta$ in the $\lambda\mu_{\mathrm{v}}$-calculus.

Remark 1. The $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus are variants of Parigot's $\lambda\mu$-calculi [21] and Ong-Stewart's $\lambda\mu_{\mathrm{v}}$-calculus [20]. Parigot has shown that his $\lambda\mu$-calculus has Church-Rosser property and the strong normalization property, and Ong and Stewart as well. Additionally, although it is natural from the coincidence of the CPS translations, we stress that the $\lambda\mu_{\mathrm{v}}$-calculus is an extension of Moggi's λ_{c}-calculus.

Remark 2. We can naturally extend the $\lambda\mu$-calculi with constants or control constants. However, when we consider the duality, function types will be identified with non-function types. Therefore an equation for such constant may be

$$
\begin{array}{lll}
(\beta_{\to})\ (\lambda x^A.\, M)N = M\,[N\,/\,x] & :B & \\
(\eta_{\to})\ \lambda x^A.\, Mx = M & :B & x \notin \mathrm{FV}(M) \\
(\beta_{\wedge})\ \pi_i\langle M_1, M_2\rangle = M_i & :A & \\
(\eta_{\wedge})\ \langle \pi_1 M, \pi_2 M\rangle = M & :A \wedge B & \\
(\eta_{\top})\ * = M & :\top & \\
(\beta_{\mu})\ [\beta]\mu\alpha^A.\, M = M\,[\beta\,/\,\alpha] & :\bot & \\
(\eta_{\mu})\ \mu\alpha^A.[\alpha]\, M = M & :A & \alpha \notin \mathrm{FN}(M) \\
(\beta_{\vee})\ [\gamma,\delta]\mu(\alpha^A,\beta^B).\, M = M\,[\beta\,/\,\alpha, \delta\,/\,\gamma] & :\bot & \\
(\eta_{\vee})\ \mu(\alpha^A,\beta^B).[\alpha,\beta]\, M = M & :A \vee B & \alpha,\beta \notin \mathrm{FN}(M) \\
(\beta_{\bot})\ [\beta]\, M = M & :\bot & \\
(\zeta_{\to})\ (\mu\alpha^{A\to B}.\, M)N = \mu\beta^B.\, M\,[[\beta](-)N\,/\,[\alpha](-)] & :B & \\
(\zeta_{\wedge})\ \pi_i(\mu\alpha^{A_1\wedge A_2}.\, M) = \mu\beta^{A_i}.\, M\,[[\beta]\pi_i(-)\,/\,[\alpha](-)] & :A_i & \\
(\zeta_{\vee})\ [\gamma,\delta]\mu\alpha^{A\vee B}.\, M = M\,[[\gamma,\delta](-)\,/\,[\alpha](-)] & :\bot &
\end{array}
$$

Fig. 4. The axioms of the $\lambda\mu_{\mathrm{n}}$-calculus

translated to a complicated form. If we deal with the duality syntactically, it is a little necessary to devise the equality relations. We will revisit this point in Section 3.

2 Category-theoretic Approach

2.1 Control categories

According to Selinger, the $\lambda\mu_{\mathrm{n}}$-calculus has a complete class of models called **control categories** [25] (whose definition is recalled in Appendix A). Control categories have finite products and finite classical disjunctions. As contrasted with it, the $\lambda\mu_{\mathrm{v}}$-calculus can be seen as an internal language for a co-control category. A co-control category is the opposite category of a control category. So it is natural that there exists the dual correspondence between the $\lambda\mu_{\mathrm{v}}$-calculus and the $\lambda\mu_{\mathrm{n}}$-calculus.

Following Selinger, we shall characterize control categories by response categories. Let $\mathcal{C}$ be a category that has distributive finite products and coproducts and distinguished object R such that R^A exists for any A. We call $\mathcal{C}$ a response category (and call R its object of responses), if $\mathcal{C}$ satisfies the mono requirement, *i.e.*, for any A, the canonical morphism $\partial_A : A \to R^{R^A}$ is monic. Given a response category $\mathcal{C}$, we define its **category of continuations** $R^{\mathcal{C}}$ by $R^{\mathcal{C}}(A,B) = \mathcal{C}(R^A, R^B)$. Here we remark that the opposite category of continuations $(R^{\mathcal{C}})^{\mathrm{op}}$ can be considered as Kleisli category of $\mathcal{C}$ on the continuation monad $R^{R^{(-)}}$.

It can be seen that a category of continuations has a cartesian closed structure. (The reader can be referred to [16] to recall the definition of cartesian closed categories.) Indeed, in terms of $\mathcal{C}$,

$$1 \cong R^0, \qquad R^A \times R^B \cong R^{A+B}, \qquad (R^B)^{R^A} \cong R^{B\times R^A}$$

$$
\begin{array}{rll}
(\beta_{\to}) & \texttt{let}\ x^A\ \texttt{be}\ V\ \texttt{in}\ M = M\,[V\,/\,x] & :B \\
(\eta_{\to}) & \lambda x^A.\, V x = V & :A \to B \quad x \notin \mathrm{FV}(V) \\
(\beta_{\wedge}) & \pi_i \langle V_1, V_2 \rangle = V_i & :A \\
(\eta_{\wedge}) & \langle \pi_1 V, \pi_2 V \rangle = V & :A \wedge B \\
(\eta_{\top}) & * = V & :\top \\
(\beta_{\mu}) & [\beta]\mu\alpha^A.\, M = M\,[\beta\,/\,\alpha] & :\bot \\
(\eta_{\mu}) & \mu\alpha^A.[\alpha]\, M = M & :A \quad \alpha \notin \mathrm{FN}(M) \\
(\beta_{\vee}) & [\gamma, \delta]\mu(\alpha^A, \beta^B).\, M = M\,[\beta\,/\,\alpha, \delta\,/\,\gamma] & :\bot \\
(\eta_{\vee}) & \mu(\alpha^A, \beta^B).[\alpha, \beta]\, M = M & :A \vee B \quad \alpha, \beta \notin \mathrm{FN}(M) \\
(\beta_{\bot}) & [\beta]\, M = M & :\bot \\
(\mathrm{let}_{\to}) & M N = \texttt{let}\ x^{A \to B}\ \texttt{be}\ M\ \texttt{in}\ \texttt{let}\ y^A\ \texttt{be}\ N\ \texttt{in}\ x y & :B \quad x \notin \mathrm{FV}(N) \\
(\mathrm{let}_{\wedge}) & \langle M, N \rangle = \texttt{let}\ x^A\ \texttt{be}\ M\ \texttt{in}\ \texttt{let}\ y^B\ \texttt{be}\ N\ \texttt{in}\ \langle x, y \rangle & :A \wedge B \quad x \notin \mathrm{FV}(N) \\
(\mathrm{let}_{\pi}) & \pi_i M = \texttt{let}\ x^{A_1 \wedge A_2}\ \texttt{be}\ M\ \texttt{in}\ \pi_i x & :A_i \\
(\mathrm{let}_{\bot}) & [\alpha]\, M = \texttt{let}\ x^A\ \texttt{be}\ M\ \texttt{in}\ [\alpha]\, x & :\bot \\
(\mathrm{let}_{\vee}) & [\alpha, \beta]\, M = \texttt{let}\ x^{A \vee B}\ \texttt{be}\ M\ \texttt{in}\ [\alpha, \beta]\, x & :\bot \\
(\mathrm{comp}) & \texttt{let}\ y^B\ \texttt{be}\ (\texttt{let}\ x^A\ \texttt{be}\ M\ \texttt{in}\ N)\ \texttt{in}\ L & \\
 & = \texttt{let}\ x^A\ \texttt{be}\ M\ \texttt{in}\ \texttt{let}\ y^B\ \texttt{be}\ N\ \texttt{in}\ L & :C \quad x \notin \mathrm{FV}(L) \\
(\mathrm{id}) & \texttt{let}\ x^A\ \texttt{be}\ M\ \texttt{in}\ x = M & :B \\
(\zeta) & V(\mu\alpha^A.M) = \mu\beta^B.\, M\,[[\beta]\, V(-)\,/[\alpha](-)] & :B
\end{array}
$$

Fig. 5. The axioms of the $\lambda\mu_{\mathrm{v}}$-calculus

hold. Moreover $R^{\mathcal{C}}$ has a premonoidal structure [23] $\oplus$:

$\bot$ can be defined as R^1 and $R^A \oplus R^B$ can be defined as $R^{A \times B}$.

Proposition 1 (Selinger [25]). *Let $\mathcal{C}$ be a response category with R. The category of continuations $R^{\mathcal{C}}$ is a control category.*

The proposition above claims that a category of continuations is an example of a control category, but Selinger has shown that any control category essentially arises as a category of continuations.

Theorem 2 (Selinger [25]). *Any control category is equivalent to a category of continuations.*

2.2 Models of the $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus

In this subsection, we outline the interpretation from the $\lambda\mu_{\mathrm{n}}(\lambda\mu_{\mathrm{v}})$-calculus to a (co-)control category.

The type interpretation of the $\lambda\mu_{\mathrm{n}}$-calculus is defined by

$$\begin{aligned}
[\![\sigma]\!]_{\mathrm{n}} &\equiv \underline{\sigma}, \\
[\![A \to B]\!]_{\mathrm{n}} &\equiv [\![B]\!]_{\mathrm{n}}^{[\![A]\!]_{\mathrm{n}}}, \\
[\![\top]\!]_{\mathrm{n}} &\equiv 1, \\
[\![A \wedge B]\!]_{\mathrm{n}} &\equiv [\![A]\!]_{\mathrm{n}} \times [\![B]\!]_{\mathrm{n}}, \\
[\![\bot]\!]_{\mathrm{n}} &\equiv \bot, \\
[\![A \vee B]\!]_{\mathrm{n}} &\equiv [\![A]\!]_{\mathrm{n}} \oplus [\![B]\!]_{\mathrm{n}},
\end{aligned}$$

while the type interpretation of the $\lambda\mu_{\mathrm{v}}$-calculus is defined by

$$\begin{aligned}
[\![\sigma]\!]_{\mathrm{v}} &\equiv \underline{\sigma}, \\
[\![A \to B]\!]_{\mathrm{v}} &\equiv [\![A]\!]_{\mathrm{v}} \multimap [\![B]\!]_{\mathrm{v}}, \\
[\![\top]\!]_{\mathrm{v}} &\equiv \top, \\
[\![A \wedge B]\!]_{\mathrm{v}} &\equiv [\![A]\!]_{\mathrm{v}} \otimes [\![B]\!]_{\mathrm{v}}, \\
[\![\bot]\!]_{\mathrm{v}} &\equiv 0, \\
[\![A \vee B]\!]_{\mathrm{v}} &\equiv [\![A]\!]_{\mathrm{v}} + [\![B]\!]_{\mathrm{v}},
\end{aligned}$$

where $\underline{\sigma}$ is an object assigned to each base type σ.

A $\lambda\mu_{\mathrm{n}}$-judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$ is interpreted to a morphism from $[\![B_1]\!]_{\mathrm{n}} \times [\![B_2]\!]_{\mathrm{n}} \times \ldots \times [\![B_n]\!]_{\mathrm{n}}$ to $[\![A]\!]_{\mathrm{n}} \oplus [\![A_1]\!]_{\mathrm{n}} \oplus [\![A_2]\!]_{\mathrm{n}} \oplus \ldots \oplus [\![A_m]\!]_{\mathrm{n}}$ in a control category. On the other hand, a $\lambda\mu_{\mathrm{v}}$-judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$ is interpreted to a morphism from $[\![B_1]\!]_{\mathrm{v}} \otimes [\![B_2]\!]_{\mathrm{v}} \otimes \ldots \otimes [\![B_n]\!]_{\mathrm{v}}$ to $[\![A]\!]_{\mathrm{v}} + [\![A_1]\!]_{\mathrm{v}} + [\![A_2]\!]_{\mathrm{v}} \ldots + [\![A_m]\!]_{\mathrm{v}}$ in a co-control category. We shall omit the detail of the interpretation, for which the reader is referred to [25].

We have seen all the call-by-name CPS target terms are restricted to the negative type terms. Since control categories are equivalent to categories of continuations, we can get an interpretation to a control category in the way that the CPS target calculus is interpreted to response categories. For the call-by-value language, the following isomorphism is useful:

$$\mathcal{C}(R^B, R^A) \cong \mathcal{C}(A, R^{R^B}) \quad \text{in the response category } \mathcal{C}.$$

Hence in the similar way to the $\lambda\mu_{\mathrm{n}}$-calculus, the $\lambda\mu_{\mathrm{v}}$-calculus can be interpreted to a co-control category via the CPS translation. In fact, this means that the $\lambda\mu_{\mathrm{v}}$-calculus is interpreted to a Kleisli category on the continuation monad. It is widely known that the call-by-value λ-calculus enriched with first-class continuations is axiomatizable such that its axioms are sound and complete for the CPS translation or interpretations to Kleisli categories [24, 19]. Our (Selinger's) CPS translation from the $\lambda\mu_{\mathrm{v}}$-calculus is essentially the same as its CPS translation from the call-by-value λ-calculus, and the interpretation to a control category is just the interpretation to a Kleisli category on the continuation monad.

2.3 Recursion and Iteration

Because the call-by-name λ-calculus is the subset of the $\lambda\mu_{\mathrm{n}}$-calculus, models of the $\lambda\mu_{\mathrm{n}}$-calculus are also models of the call-by-name λ-calculus. Properties of fixed-point operators on the call-by-name λ-calculus have been studied widely, for example, uniformity, dinaturality, diagonal property and so on (see [26] for a survey). So it is natural to study recursion in the call-by-name $\lambda\mu$-calculus and its models, *i.e.*, control categories.

Remark 3. It is a well-known fact that if a distributive category (which has finite products and finite coproducts such that products distribute coproducts) has a fixed-point operator, then it gets trivial [15]. However, in this case, disjunction types are not coproduct types, so a control category enriched with a recursion operator is not trivial. We also note that, considering a category of continuations $R^{\mathcal{C}}$, the fixed-point operator in $R^{\mathcal{C}}$ is a fixed-point operator restricted on negative types in $\mathcal{C}$. The following example guarantees that recursion on a control category is not senseless.

Example 4. We consider the category of ω-cpos and ω-continuous maps as a control category $\mathcal{C}$. Let R be an ω-cpo that has a bottom element. Then $R^{\mathcal{C}}$ can be a control category. Because each R^A has a bottom, every $f\colon R^A \to B$ has a least fixed-point. Hence there exists a fixed-point operator on $R^{\mathcal{C}}$.

We introduce call-by-value iteration from call-by-name recursion by the duality between control categories and co-control categories. Let $\mathcal{P}$ be a control category. The following two are dual for each other:

$$\frac{f \in \mathcal{P}(A, A)}{f^{\dagger} \in \mathcal{P}(1, A),} \qquad \frac{f \in \mathcal{P}^{\mathrm{op}}(A, A)}{f_{\dagger} \in \mathcal{P}^{\mathrm{op}}(A, 0).}$$

Now, we assume $(-)^{\dagger}$ denotes **a fixed-point operator** on a control category. The following dual equations are induced:

$$f^{\dagger} = f \circ f^{\dagger} = f \circ f \circ f^{\dagger} = f \circ \cdots \circ f \circ f \circ f^{\dagger},$$
$$f_{\dagger} = f_{\dagger} \circ f = f_{\dagger} \circ f \circ f = f_{\dagger} \circ f \circ f \circ \cdots \circ f.$$

We call such $(-)_{\dagger}$ **an iteration operator**. This duality seems to make programs inside-out. While the duality between call-by-name and call-by-value appears in duality between demand-passing and data-passing, the duality between recursion and iteration appears in duality between 'outside to inside' and 'inside to outside'.

In the next section, we shall consider this duality syntactically in the $\lambda\mu$-calculi, extending them with a call-by-name recursion operator and a call-by-value iteration operator.

3 The $\lambda\mu$-calculi with Recursion

3.1 Fixed-point operators and iteration operators

We add a family of constants $\{\mathtt{fix}^A \mid A \text{ is a type}\}$ to the $\lambda\mu_{\mathrm{n}}$-calculus, and a family of constants $\{\mathtt{loop}^A \mid A \text{ is a type}\}$ to the $\lambda\mu_{\mathrm{v}}$-calculus, where $\{\mathtt{fix}^A\}$ and $\{\mathtt{loop}^A\}$ are called **a fixed-point operator** and **an iteration operator** respectively.

The typing rules and the equality axioms are standard ones:

$$\frac{}{\Gamma \vdash \mathtt{fix}^A : (A \to A) \to A \mid \Delta}\ (\mathrm{fix}), \qquad \frac{}{\Gamma \vdash \mathtt{loop}^A : (A \to A) \to A \to \bot \mid \Delta}\ (\mathrm{loop}),$$

and

$$\begin{array}{rll} (\mathrm{fix}) & \Gamma \vdash \mathtt{fix}^A =_{\mathrm{n}} \lambda m^{A\to A}.\, m(\mathtt{fix}^A\, m) & : (A \to A) \to A \quad \mid \Delta, \\ (\mathrm{loop}) & \Gamma \vdash \mathtt{loop}^A =_{\mathrm{v}} \lambda f^{A\to A}.\lambda x^A.\, \mathtt{loop}^A\, f\, (f x) & : (A \to A) \to A \to \bot \mid \Delta. \end{array}$$

It follows $\mathtt{fix}\, M =_{\mathrm{n}} M(\mathtt{fix}^A\, M)$ holds for any $\lambda\mu_{\mathrm{n}}$-term $M : A \to A$, and $\mathtt{loop}\, F =_{\mathrm{v}} (\mathtt{loop}^A\, F) \circ F$ holds for any $\lambda\mu_{\mathrm{v}}$-value $F : A \to A$. This implies that naive ideas of a fixed-point operator and an iteration operator as is mentioned in the previous section are represented by $\mathtt{fix}^A$ and $\mathtt{loop}^A$ in the $\lambda\mu$-calculi.

3.2 The dual translations

In order to deal with the duality plainly, we assume that the set of names in the $\lambda\mu_{\mathrm{n}}$-calculus is the same as the set of variables in the $\lambda\mu_{\mathrm{v}}$-calculus, while the set of variables in the $\lambda\mu_{\mathrm{n}}$-calculus is the same as the set of names in the $\lambda\mu_{\mathrm{v}}$-calculus. In the $\lambda\mu_{\mathrm{n}}$-calculus, we also assume $\diamond$ denotes the constant that plays the role of $*$. We will define translations between the $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus. This dual translations are just the syntactic incarnation of the categorical duality. Now especially we shall pick up the recursion part:

$$\begin{aligned} (\!|\, \mathtt{loop}^A \,|\!) &= \lambda\kappa^{(\!|(A\to A)\to A\to\bot|\!)}.\, \kappa(\lambda\gamma^{\neg(\top\to(\!|A|\!))}.\lambda\phi^{(\!|A|\!)\to(\!|A|\!)}.\, \gamma(\lambda\tau^{\top}.\, \mathtt{fix}^{(\!|A|\!)}\, \phi)), \\ (\!\langle\, \mathtt{fix}^A \,\rangle\!) &= \lambda k^{\langle\!|(A\to A)\to A|\!\rangle}.\, \mathtt{loop}^{\langle\!|A|\!\rangle}(\lambda x^{\langle\!|A|\!\rangle}.\mu\beta^{\langle\!|A|\!\rangle}.\, \pi_1 k\, \langle \lambda y^{\langle\!|A|\!\rangle}.[\beta]\, y, x\rangle)\, (\pi_2 k). \end{aligned}$$

The rest of the definitions, due to Selinger [25], is in Figure 6 and 7.

The following propositions guarantee that these translations are sound for the typing and for the equational theories.

Proposition 1. *For any $\lambda\mu_{\mathrm{v}}$-judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$,*

$$\alpha_1 : (\!|A_1|\!), \ldots, \alpha_m : (\!|A_m|\!), \alpha : (\!|A|\!) \vdash (\!|M|\!)\alpha : \bot \mid x_1 : (\!|B_1|\!), \ldots, x_n : (\!|B_n|\!),$$

and for any $\lambda\mu_{\mathrm{n}}$-judgement $\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash M : A \mid x_1 : A_1, \ldots, x_m : A_m$,

$$x_1 : \langle\!|A_1|\!\rangle, \ldots, x_m : \langle\!|A_m|\!\rangle, x : \langle\!|A|\!\rangle \vdash \langle\!|M|\!\rangle x : \bot \mid \alpha_1 : \langle\!|B_1|\!\rangle, \ldots, \alpha_n : \langle\!|B_n|\!\rangle.$$

$$
\begin{array}{rcl}
(\!|\sigma|\!) & \equiv & \sigma \\
(\!|A \to B|\!) & \equiv & ((\!|B|\!) \to (\!|A|\!)) \to \bot \\
(\!|\top|\!) & \equiv & \bot
\end{array}
\qquad
\begin{array}{rcl}
(\!|A \wedge B|\!) & \equiv & (\!|A|\!) \vee (\!|B|\!) \\
(\!|\bot|\!) & \equiv & \top \\
(\!|A \vee B|\!) & \equiv & (\!|A|\!) \wedge (\!|B|\!)
\end{array}
$$

$$
\begin{array}{rcll}
(\!|x|\!) & = & \lambda\kappa^{(\!|A|\!)}.[x]\,\kappa & x : A \\
(\!|\,*\,|\!) & = & \lambda\kappa^{(\!|\top|\!)}.\,\kappa & \\
(\!|\lambda x^A.\,M|\!) & = & \lambda\kappa^{(\!|A\to B|\!)}.\,\kappa(\lambda\beta^{(\!|B|\!)}.\mu x^{(\!|A|\!)}.\,(\!|M|\!)\beta) & M : B \\
(\!|M\,N|\!) & = & \lambda\kappa^{(\!|B|\!)}.\,(\!|M|\!)(\lambda\gamma^{(\!|B|\!)\to(\!|A|\!)}.\,(\!|N|\!)(\gamma\kappa)) & M : A \to B,\ \ N : A \\
(\!|\langle M, N\rangle|\!) & = & \lambda\kappa^{(\!|A\wedge B|\!)}.\,(\!|M|\!)(\mu x^{(\!|A|\!)}.\,(\!|N|\!)(\mu y^{(\!|B|\!)}.[x,y]\,\kappa)) & M : A,\ \ N : B \\
(\!|\pi_i M|\!) & = & \lambda\kappa^{(\!|A_i|\!)}.\,(\!|M|\!)(\mu(x_1^{A_1}, x_2^{A_2}).[x_i]\,\kappa) & M : A_1 \wedge A_2 \\
(\!|\mu\alpha^A.\,M|\!) & = & \lambda\kappa^{(\!|A|\!)}.\,(\lambda\alpha^{(\!|A|\!)}.\,(\!|M|\!)\diamond)\kappa & \\
(\!|[\alpha]\,M|\!) & = & \lambda\kappa^{(\!|\bot|\!)}.\,(\!|M|\!)\,\alpha & \\
(\!|\mu(\alpha^A, \beta^B).\,M|\!) & = & \lambda\kappa^{(\!|A\vee B|\!)}.\,(\lambda\alpha^{(\!|A|\!)}.\lambda\beta^{(\!|B|\!)}.\,(\!|M|\!)\diamond)(\pi_1\kappa)\,(\pi_2\kappa) & \\
(\!|[\alpha,\beta]\,M|\!) & = & \lambda\kappa^{(\!|\bot|\!)}.\,(\!|M|\!)\langle\alpha,\beta\rangle &
\end{array}
$$

Fig. 6. The dual translation from the $\lambda\mu_{\mathrm{v}}$ to the $\lambda\mu_{\mathrm{n}}$

$$
\begin{array}{rcl}
\langle\!|\sigma|\!\rangle & \equiv & \sigma \\
\langle\!|A \to B|\!\rangle & \equiv & (\langle\!|A|\!\rangle \to \bot) \wedge \langle\!|B|\!\rangle \\
\langle\!|\top|\!\rangle & \equiv & \bot
\end{array}
\qquad
\begin{array}{rcl}
\langle\!|A \wedge B|\!\rangle & \equiv & \langle\!|A|\!\rangle \vee \langle\!|B|\!\rangle \\
\langle\!|\bot|\!\rangle & \equiv & \top \\
\langle\!|A \vee B|\!\rangle & \equiv & \langle\!|A|\!\rangle \wedge \langle\!|B|\!\rangle
\end{array}
$$

$$
\begin{array}{rcll}
\langle\!|\alpha|\!\rangle & = & \lambda k^{\langle\!|A|\!\rangle}.[\alpha]\,k & \alpha : A \\
\langle\!|\,\diamond\,|\!\rangle & = & \lambda k^{\langle\!|\top|\!\rangle}.\,k & \\
\langle\!|\lambda\alpha^A.\,M|\!\rangle & = & \lambda k^{\langle\!|A\to B|\!\rangle}.\,\pi_1 k\,(\mu\alpha^{\langle\!|A|\!\rangle}.\,\langle\!|M|\!\rangle(\pi_2 k)) & M : B \\
\langle\!|M\,N|\!\rangle & = & \lambda k^{\langle\!|B|\!\rangle}.\,\langle\!|M|\!\rangle\langle\langle\!|N|\!\rangle, k\rangle & M : A \to B,\ \ N : A \\
\langle\!|\langle M, N\rangle|\!\rangle & = & \lambda k^{\langle\!|A\wedge B|\!\rangle}.\,\langle\!|M|\!\rangle(\mu\alpha^{\langle\!|A|\!\rangle}.\,\langle\!|N|\!\rangle(\mu\beta^{\langle\!|B|\!\rangle}.[\alpha,\beta]\,k)) & M : A,\ \ N : B \\
\langle\!|\pi_i M|\!\rangle & = & \lambda k^{\langle\!|A_i|\!\rangle}.\,\langle\!|M|\!\rangle(\mu(\alpha_1^{\langle\!|A_1|\!\rangle}, \alpha_2^{\langle\!|A_2|\!\rangle}).[\alpha_i]\,k) & M : A_1 \wedge A_2 \\
\langle\!|\mu x^A.\,M|\!\rangle & = & \lambda k^{\langle\!|A|\!\rangle}.\,(\lambda x^{\langle\!|A|\!\rangle}.\,\langle\!|M|\!\rangle *)k & \\
\langle\!|[x]\,M|\!\rangle & = & \lambda k^{\langle\!|\bot|\!\rangle}.\,\langle\!|M|\!\rangle x & \\
\langle\!|\mu(x^A, y^B).\,M|\!\rangle & = & \lambda k^{\langle\!|A\vee B|\!\rangle}.\,(\lambda x^{\langle\!|A|\!\rangle}.\lambda y^{\langle\!|B|\!\rangle}.\,\langle\!|M|\!\rangle *)(\pi_1 k)\,(\pi_2 k) & \\
\langle\!|[x,y]\,M|\!\rangle & = & \lambda k^{\langle\!|\bot|\!\rangle}.\,\langle\!|M|\!\rangle\langle x, y\rangle &
\end{array}
$$

Fig. 7. The dual translation from the $\lambda\mu_{\mathrm{n}}$ to the $\lambda\mu_{\mathrm{v}}$

Proposition 2. *$(\!|(-)|\!)$ and $\langle\!|(-)|\!\rangle$ are sound, that is, each equality is preserved by the translation.*

Proof. Check all the equality axioms. For example, (loop) case is followed by the equation

$$(\!|\, \mathtt{loop}^A\, f\, x|\!)\kappa =_{\mathrm{n}} [f](\lambda\phi^{(\!|A|\!)\to(\!|A|\!)}.[x]\ \mathtt{fix}^{(\!|A|\!)}\,\phi)$$

and the equation (fix). □

These translations fit not only into the categorical duality but also into the CPS translations. Let $\mathtt{Y}$ be a fixed point combinator on the λ-calculus. The CPS forms of $\mathtt{fix}$ and $\mathtt{loop}$ are the following:

$$\widehat{\mathtt{fix}^A} = \lambda\langle g^{K_A\times V_A\to R}, k^{V_A}\rangle.\ \mathtt{Y}(\lambda f^{K_A}.\lambda x^{V_A}.\, g\langle f, x\rangle)\, k,$$
$$\widetilde{\mathtt{loop}^A} = \lambda k^{K_{(A\to A)\to\neg A}}.\, k(\lambda\langle g^{V_{A\to A}}, h^{K_{\neg A}}\rangle.\, h(\lambda\langle x^{V_A}, t^{K_\perp}\rangle.\ \mathtt{Y}(\lambda f^{K_A}.\lambda y^{V_A}.\, g\langle y, f\rangle)))).$$

(Recall that it is sufficient that $\mathtt{Y}$ is only applied to negative types.) It can be easily checked that the dual translations preserve the CPS transforms up to some primitive isomorphisms (*e.g.*, $A \cong A \times (0 \to R)$). Thus from the semantic point of view, the translations preserve the equalities exactly. In the next subsection, we formalize and show it syntactically.

3.3 Call-by-value to call-by-name to call-by-value

By the definition of the type translation, we can easily see that the composite $\langle\!|(\!|(-)|\!)|\!\rangle$ is not the identity generally, but the type $\langle\!|(\!|A|\!)|\!\rangle$ looks isomorphic to A. Indeed the following isomorphism exists in a co-control category:

$$\begin{aligned}
[\![\langle\!|(\!|A|\!)|\!\rangle \to \langle\!|(\!|B|\!)|\!\rangle]\!]_{\mathrm{v}} &\equiv [\![\langle\!|(\!|A|\!)|\!\rangle]\!]_{\mathrm{v}} \multimap [\![\langle\!|(\!|B|\!)|\!\rangle]\!]_{\mathrm{v}} \\
&\cong (((([\![\langle\!|(\!|B|\!)|\!\rangle]\!]_{\mathrm{v}} \multimap 0) \otimes [\![\langle\!|(\!|A|\!)|\!\rangle]\!]_{\mathrm{v}}) \multimap 0) \otimes \top \\
&\equiv [\![\langle\!|(\!|A \to B|\!)|\!\rangle]\!]_{\mathrm{v}}.
\end{aligned}$$

According to this categorical investigation, we construct a type-indexed family of isomorphisms in the $\lambda\mu_{\mathrm{v}}$-calculus.

Define

$$\begin{aligned}
&\mathrm{I}_0^{A\to B} : (A \to B) \to (((B \to \bot) \wedge A) \to \bot) \wedge \top \\
&\quad \equiv \lambda f^{A\to B}.\langle\lambda k^{\neg B\wedge A}.(\pi_1 k)(f(\pi_2 k)), *\rangle, \\
&\mathrm{J}_0^{A\to B} : (((B \to \bot) \wedge A) \to \bot) \wedge \top \to A \to B \\
&\quad \equiv \lambda m^{\neg(\neg B\wedge A)\wedge\top}.\lambda x^A.\mu\beta^B.(\pi_1 m)\langle\lambda y^B.[\beta]\, y, x\rangle.
\end{aligned}$$

(This $\mathrm{J}_0^{A\to B}$ is a subspecies of switch defined by Filinski.) Indeed these terms are isomorphisms, *i.e.*, $\mathrm{J}_0^{A\to B}(\mathrm{I}_0^{A\to B} f) =_{\mathrm{v}} f$ and $\mathrm{I}_0^{A\to B}(\mathrm{J}_0^{A\to B} m) =_{\mathrm{v}} m$ hold.

Using them, we can define isomorphisms A to $\langle\!(\!|A|\!)\!\rangle$ and *vice versa*:

$$
\begin{aligned}
\mathrm{I}_\sigma &\equiv \lambda f^\sigma.f,\\
\mathrm{J}_\sigma &\equiv \lambda m^{\langle\!(\!|\sigma|\!)\!\rangle}.m,\\
\mathrm{I}_{A\to B} &\equiv \lambda f^{A\to B}.\mathrm{I}_0^{\langle\!(\!|A|\!)\!\rangle\to\langle\!(\!|B|\!)\!\rangle}(\lambda x^{\langle\!(\!|A|\!)\!\rangle}.\mathrm{I}_B(f(\mathrm{J}_A x))),\\
\mathrm{J}_{A\to B} &\equiv \lambda m^{\langle\!(\!|A\to B|\!)\!\rangle}.\lambda x^A.\mathrm{J}_B(\mathrm{J}_0^{\langle\!(\!|A|\!)\!\rangle\to\langle\!(\!|B|\!)\!\rangle}m(\mathrm{I}_A x)),\\
\mathrm{I}_\top &\equiv \lambda f^\top.f,\\
\mathrm{J}_\top &\equiv \lambda m^{\langle\!(\!|\top|\!)\!\rangle}.m,\\
\mathrm{I}_{A\wedge B} &\equiv \lambda f^{A\wedge B}.\langle \mathrm{I}_A(\pi_1 f), \mathrm{I}_B(\pi_2 f)\rangle,\\
\mathrm{J}_{A\wedge B} &\equiv \lambda m^{\langle\!(\!|A\wedge B|\!)\!\rangle}.\langle \mathrm{J}_A(\pi_1 m), \mathrm{J}_B(\pi_2 m)\rangle,\\
\mathrm{I}_\perp &\equiv \lambda f^\perp.f,\\
\mathrm{J}_\perp &\equiv \lambda m^{\langle\!(\!|\perp|\!)\!\rangle}.m,\\
\mathrm{I}_{A\vee B} &\equiv \lambda f^{A\vee B}.\mu(\alpha^{\langle\!(\!|A|\!)\!\rangle}, \beta^{\langle\!(\!|B|\!)\!\rangle}).[\alpha]\,\mathrm{I}_A(\mu\gamma^A.[\beta]\,\mathrm{I}_B(\mu\delta^B.[\gamma,\delta]\,f)),\\
\mathrm{J}_{A\vee B} &\equiv \lambda m^{\langle\!(\!|A\vee B|\!)\!\rangle}.\mu(\alpha^A, \beta^B).[\beta]\,\mathrm{J}_B(\mu\delta^{\langle\!(\!|B|\!)\!\rangle}.[\alpha]\,\mathrm{J}_A(\mu\gamma^{\langle\!(\!|A|\!)\!\rangle}.[\gamma,\delta]\,m)).
\end{aligned}
$$

It follows by easy induction that the terms defined above are isomorphisms.

We shall use these I_A and J_A to represent the notion of 'up to natural isomorphisms' syntactically. The following proposition is the exact phrase of its notion.

Proposition 3. *In the $\lambda\mu_{\mathrm{v}}$-calculus without* `loop`*, the composite $\mu\kappa^A.\,\langle\!(\!|(-)|\!)\!\rangle\kappa\rangle*$ is equivalent to the identity in the following sense:*
for any judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$,

$$
\begin{aligned}
&x_1 : B_1, \ldots, x_n : B_n \vdash\\
&\quad \mathrm{J}_A(\mu\kappa^{\langle\!(\!|A|\!)\!\rangle}.\,\langle\!(\!|M|\!)\!\rangle\kappa\rangle*)\,[\mathrm{I}_{B_1}x_1\,/x_1, \ldots, [\alpha_1]\,\mathrm{J}_{A_1}(-)\,/[\alpha_1]\,(-), \ldots] =_{\mathrm{v}} M\\
&\quad : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m,
\end{aligned}
$$

and in other words,

$$
\begin{aligned}
&x_1 : \langle\!(\!|B_1|\!)\!\rangle, \ldots, x_n : \langle\!(\!|B_n|\!)\!\rangle \vdash\\
&\quad \mu\kappa^{\langle\!(\!|A|\!)\!\rangle}.\,\langle\!(\!|M|\!)\!\rangle\kappa\rangle* =_{\mathrm{v}} \mathrm{I}_A M\,[\mathrm{J}_{B_1}x_1\,/x_1, \ldots, [\alpha_1]\,\mathrm{I}_{A_1}(-)\,/[\alpha_1]\,(-), \ldots]\\
&\quad : \langle\!(\!|A|\!)\!\rangle \mid \alpha_1 : \langle\!(\!|A_1|\!)\!\rangle, \ldots, \alpha_m : \langle\!(\!|A_m|\!)\!\rangle.
\end{aligned}
$$

Proof. By induction. □

Remark 4. One would expect the substitutions in the foregoing proposition to mean *parallel* substitutions, but, for example, $[[\alpha_i]\,\mathrm{J}_{A_i}(-)\,/[\alpha_i]\,(-), [\alpha_j]\,\mathrm{J}_{A_j}(-)\,/[\alpha_j]\,(-)]$ is problematic if $[\alpha_j, \alpha_i]$ occurs in the target term freely. Hence we try defining the multi-substitution as a sequential composition of single-substitutions. Comparing a term $M[\ldots, [\alpha_i]\,\mathrm{J}_{A_i}(-)\,/[\alpha_i]\,(-), [\alpha_j]\,\mathrm{J}_{A_j}(-)\,/[\alpha_j]\,(-), \ldots]$ with a term $M[\ldots, [\alpha_j]\,\mathrm{J}_{A_j}(-)\,/[\alpha_j]\,(-), [\alpha_i]\,\mathrm{J}_{A_i}(-)\,/[\alpha_i]\,(-), \ldots]$, we see that these two terms are equal to each other on the $\lambda\mu_{\mathrm{v}}$-theory, and furthermore, we also get the term equal to them even if we apply the substitution replacing $[\alpha_i, \alpha_j](-)$ by $[\alpha_i, \alpha_j]\,\mathrm{J}_{A_i\vee A_j}(-)$, because $\mathrm{J}_{A_i\vee A_j}$ is an isomorphism. Thus, there is no need to take care whether some substitutions conflict in a parallel substitution.

If the iterator $\mathtt{loop}^A$ occurs in M, then $\mathtt{loop}^{\langle\!|(\!|A|\!)\!|\rangle}$ occurs in $\mu\kappa.\langle\!|(\!|M|\!)\kappa|\!\rangle *$. Therefore we have to extend the substitutions to include the replacement of $\mathtt{loop}^{\langle\!|(\!|A|\!)\!|\rangle}$ by $\lambda f.\lambda x.\ \mathtt{loop}^A(\lambda z.\,\mathrm{J}_A(f(\mathrm{I}_A z)))\,(\mathrm{J}_A x)$.

We define

$$\mathrm{I}^A_{\mathrm{loop}} \equiv \lambda l^{(A\to A)\to\neg A}.\lambda f^{\langle\!|(\!|A|\!)\!|\rangle\to\langle\!|(\!|A|\!)\!|\rangle}.\lambda x^{\langle\!|(\!|A|\!)\!|\rangle}.\,l\,(\lambda z^A.\,\mathrm{J}_A(f(\mathrm{I}_A z)))\,(\mathrm{J}_A x) \qquad \text{and}$$
$$\mathrm{J}^A_{\mathrm{loop}} \equiv \lambda l^{(\langle\!|(\!|A|\!)\!|\rangle\to\langle\!|(\!|A|\!)\!|\rangle)\to\neg\langle\!|(\!|A|\!)\!|\rangle}.\lambda f^{A\to A}.\lambda x^A.\,l\,(\lambda z^{\langle\!|(\!|A|\!)\!|\rangle}.\,\mathrm{I}_A(f(\mathrm{J}_A z)))\,(\mathrm{I}_A x).$$

$\mathrm{J}^A_{\mathrm{loop}}\circ\mathrm{I}^A_{\mathrm{loop}}$ and $\mathrm{I}^A_{\mathrm{loop}}\circ\mathrm{J}^A_{\mathrm{loop}}$ are not identities but the terms applied to $\mathtt{loop}^A$ are equal to $\mathtt{loop}^A$, that is, $\mathrm{J}^A_{\mathrm{loop}}(\mathrm{I}^A_{\mathrm{loop}}\,\mathtt{loop}^A) =_{\mathrm{v}} \mathtt{loop}^A$ and $\mathrm{I}^A_{\mathrm{loop}}(\mathrm{J}^A_{\mathrm{loop}}\,\mathtt{loop}^{\langle\!|(\!|A|\!)\!|\rangle}) =_{\mathrm{v}} \mathtt{loop}^{\langle\!|(\!|A|\!)\!|\rangle}$.

Theorem 5. *The composite $\mu\kappa^A.\langle\!|(\!|(-)|\!)\kappa|\!\rangle *$ is equivalent to the identity in the following sense:*
for any judgement $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$,

$$\begin{aligned}&x_1 : B_1, \ldots, x_n : B_n \vdash\\ &\quad \mathrm{J}_A(\mu\kappa^{\langle\!|(\!|A|\!)\!|\rangle}.\,\langle\!|(\!|M|\!)\kappa|\!\rangle *)\,[\ldots, \mathrm{I}^{D_k}_{\mathrm{loop}}\,\mathtt{loop}^{D_k}/\mathtt{loop}^{\langle\!|(\!|D_k|\!)\!|\rangle}, \ldots] =_{\mathrm{v}} M\\ &: A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m,\end{aligned}$$

and in other words,

$$\begin{aligned}&x_1 : \langle\!|(\!|B_1|\!)\!|\rangle, \ldots, x_n : \langle\!|(\!|B_n|\!)\!|\rangle \vdash\\ &\quad \mu\kappa^{\langle\!|(\!|A|\!)\!|\rangle}.\,\langle\!|(\!|M|\!)\kappa|\!\rangle * =_{\mathrm{v}} \mathrm{I}_A M\,[\ldots, \mathrm{J}^{D_k}_{\mathrm{loop}}\,\mathtt{loop}^{\langle\!|(\!|D_k|\!)\!|\rangle}/\mathtt{loop}^{D_k}, \ldots]\\ &: \langle\!|(\!|A|\!)\!|\rangle \mid \alpha_1 : \langle\!|(\!|A_1|\!)\!|\rangle, \ldots, \alpha_m : \langle\!|(\!|A_m|\!)\!|\rangle,\end{aligned}$$

where D_k ranges over all types D such that $\mathtt{loop}^D$ occurs in M.

Proof. Essentially the same as the previous proposition, except for the case of `loop`.

$$
\begin{aligned}
&\mu\kappa^{\langle\!|(\!|(A\to A)\to A\to\bot|\!|\rangle}.\langle\!|(\!|\,\mathtt{loop}\,|\!|)\kappa|\!|\rangle * \\
&\quad =_{\mathrm{v}} \mu\kappa^{\langle\!|(\!|(A\to A)\to\neg A|\!|\rangle}.\langle\!|\kappa(\lambda\gamma^{(\!|\neg A|\!|}.\lambda\phi^{(\!|A|\!|\to(\!|A|\!|}.\gamma(\lambda\tau^{\top}.\,\mathtt{fix}^{(\!|A|\!|}\,\phi))|\!|\rangle\, * \\
&\quad =_{\mathrm{v}} \langle\lambda k^{\neg\langle\!|(\!|\neg A|\!|\rangle\wedge\langle\!|(\!|A\to A|\!|\rangle}.(\pi_1 k)(\mu\gamma^{\langle\!|(\!|\neg A|\!|\rangle}.\langle\!|\lambda\phi^{(\!|A|\!|\to(\!|A|\!|}.\gamma(\lambda\tau^{\top}.\,\mathtt{fix}^{(\!|A|\!|}\,\phi)|\!|\rangle(\pi_2 k)), *\rangle \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle} \\
&\qquad (\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mu\gamma^{\langle\!|(\!|\neg A|\!|\rangle}.\langle\!|\lambda\phi^{(\!|A|\!|\to(\!|A|\!|}.\gamma(\lambda\tau^{\top}.\,\mathtt{fix}^{(\!|A|\!|}\,\phi)|\!|\rangle m) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mu\gamma^{\langle\!|(\!|\neg A|\!|\rangle}. \\
&\qquad (\pi_1 m)(\mu\phi^{\langle\!|(\!|A|\!|\to(\!|A|\!|\rangle}.[\gamma]\langle\langle\!|\lambda\tau^{\top}.\,\mathtt{fix}^{(\!|A|\!|}\,\phi|\!|\rangle, \pi_2 m\rangle)) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mu\gamma^{\langle\!|(\!|\neg A|\!|\rangle}.(\pi_1 m)(\mu\phi^{\langle\!|(\!|A|\!|\to(\!|A|\!|\rangle}.[\gamma] \\
&\qquad \langle\lambda h^{\langle\!|\top\to(\!|A|\!|\rangle}.(\pi_1 h)(\mu\tau^{\bot}.\,\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle}\langle\langle\!|\phi|\!|\rangle, *\rangle)(\pi_2 h)), *\rangle)) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mu\gamma^{\langle\!|(\!|\neg A|\!|\rangle}.\mu\phi^{\bot}.[\gamma] \\
&\qquad \langle\lambda h^{\langle\!|\top\to(\!|A|\!|\rangle}.(\pi_1 h)(\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle}\langle[\phi]\,\pi_1 m, *\rangle)(\pi_2 h)), *\rangle) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}. \\
&\qquad \langle\lambda h^{\neg\bot\wedge\langle\!|(\!|A|\!|\rangle}.(\pi_1 h)(\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle} m)(\pi_2 h)), *\rangle) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mathrm{I}_0^{\langle\!|(\!|A|\!|\rangle\to\bot}(\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle} m))) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}. \\
&\qquad \mathrm{I}_0^{\langle\!|(\!|A|\!|\rangle\to\bot}(\lambda y^{\langle\!|(\!|A|\!|\rangle}.\mathrm{I}_{\bot}(\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle} m)(\mathrm{I}_A(\mathrm{J}_A y))))) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}. \\
&\qquad \mathrm{I}_{\neg A}(\lambda x^A.\,\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle} m)(\mathrm{I}_A x))) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mathrm{I}_{\neg A} \\
&\qquad (\lambda x^A.\,\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\lambda z^{\langle\!|(\!|A|\!|\rangle}.\mathrm{I}_A(\mathrm{J}_A(\mathrm{J}_0^{\langle\!|(\!|A|\!|\rangle\to\langle\!|(\!|A|\!|\rangle} m(\mathrm{I}_A(\mathrm{J}_A z)))))(\mathrm{I}_A x))) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mathrm{I}_{\neg A} \\
&\qquad (\lambda x^A.\,\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}(\lambda z^{\langle\!|(\!|A|\!|\rangle}.\mathrm{I}_A((\mathrm{J}_{A\to A} m)(\mathrm{J}_A z)))(\mathrm{I}_A x))) \\
&\quad =_{\mathrm{v}} \mathrm{I}_0^{\langle\!|(\!|A\to A|\!|\rangle\to\langle\!|(\!|\neg A|\!|\rangle}(\lambda m^{\langle\!|(\!|A\to A|\!|\rangle}.\mathrm{I}_{\neg A}((\mathrm{J}^A_{\mathtt{loop}}\,\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle})(\mathrm{J}_{A\to A} m))) \\
&\quad =_{\mathrm{v}} \mathrm{I}_{(A\to A)\to A\to\bot}(\mathrm{J}^A_{\mathtt{loop}}\,\mathtt{loop}^{\langle\!|(\!|A|\!|\rangle}).
\end{aligned}
$$

□

As was mentioned before, for the dual translations, there is another simple way to add some constants. Let c^A be a constant with type A, and ϱ^A be a name constant with type A in the call-by-value $\lambda\mu$-calculus. As Selinger has done in [25], define the translations as follows:

$$
\begin{aligned}
(\!|c^A|\!) &= \lambda\kappa^{(\!|A|\!|}.[c^{(\!|A|\!|}]\,\kappa, & \langle\!|\varrho^A|\!|\rangle &= \lambda k^{\langle\!|A|\!|\rangle}.[\varrho^{\langle\!|A|\!|\rangle}]\,k, \\
(\!|[\varrho^A]\,M|\!) &= \lambda\kappa^{\bot}.(\!|M|\!)\varrho^{(\!|A|\!|}, & \langle\!|[c^A]\,M|\!|\rangle &= \lambda k^{\bot}.\langle\!|M|\!|\rangle c^{\langle\!|A|\!|\rangle}.
\end{aligned}
$$

In this case, the target language must have the constants with respect to the source language. Constants are not so different from variables unless they have proper axioms, but otherwise the translations have to preserve the axioms. For instance, if $c^{A\to\top}$ satisfies $c^{A\to\top}x =_{\mathrm{v}} *$, it is necessary that $\mu\alpha^{\top}.\langle\!(c^{A\to\top}x)\!\rangle\alpha\rangle * = \mu\alpha^{\top}.(\pi_1 c^{(\!|\langle\!|A|\!\rangle\!|)})\langle\lambda t^{\top}.[\alpha]\,t, y\rangle$ is equal to $*$ in the $\lambda\mu_{\mathrm{v}}$-calculus.

3.4 Call-by-name to call-by-value to call-by-name

In the same way as the previous case, we shall define isomorphisms between A and $(\!|\langle\!|A|\!\rangle\!|)$ in the $\lambda\mu_{\mathrm{n}}$-calculus through the following isomorphisms:

$$\begin{aligned}
&\mathrm{G}_0^{A\to B}\colon (A\to B)\to((1\to A)\to\bot)\vee B\\
&\quad\equiv \lambda\phi^{A\to B}.\mu(x^{\neg(1\to A)}, y^B).[x]\,\lambda\gamma^{1\to A}.[y]\,\phi(\gamma\diamond),\\
&\mathrm{H}_0^{A\to B}\colon ((1\to A)\to\bot)\vee B\to(A\to B)\\
&\quad\equiv \lambda\psi^{\neg(1\to A)\vee B}.\lambda\alpha^A.\mu y^B.(\mu x^{\neg(1\to A)}.[x,y]\,\psi)(\lambda\tau^{\top}.\alpha),
\end{aligned}$$

which is motivated by the following categorical isomorphism:

$$\begin{aligned}
[\![(\!|\langle\!|A|\!\rangle\!|)\to(\!|\langle\!|B|\!\rangle\!|)]\!]_{\mathrm{n}} &\equiv [\![(\!|\langle\!|A|\!\rangle\!|)]\!]_{\mathrm{n}}\Rightarrow[\![(\!|\langle\!|B|\!\rangle\!|)]\!]_{\mathrm{n}}\\
&\cong ((1\Rightarrow[\![(\!|\langle\!|A|\!\rangle\!|)]\!]_{\mathrm{n}})\Rightarrow\bot)\oplus[\![(\!|\langle\!|B|\!\rangle\!|)]\!]_{\mathrm{n}}\\
&\equiv [\![(\!|\langle\!|A\to B|\!\rangle\!|)]\!]_{\mathrm{n}}.
\end{aligned}$$

In the $\lambda\mu_{\mathrm{n}}$-calculus, we define the isomorphisms G_A and H_A by just the same mutual recursion as that of the $\lambda\mu_{\mathrm{v}}$-calculus.

$$\begin{aligned}
\mathrm{G}_\sigma &\equiv \lambda\phi^\sigma.\phi,\\
\mathrm{H}_\sigma &\equiv \lambda\psi^{(\!|\langle\!|\sigma|\!\rangle\!|)}.\psi,\\
\mathrm{G}_{A\to B} &\equiv \lambda\phi^{A\to B}.\mathrm{G}_0^{(\!|\langle\!|A|\!\rangle\!|)\to(\!|\langle\!|B|\!\rangle\!|)}(\lambda\alpha^{(\!|\langle\!|A|\!\rangle\!|)}.\mathrm{G}_B(\phi(\mathrm{H}_A\alpha))),\\
\mathrm{H}_{A\to B} &\equiv \lambda\psi^{(\!|\langle\!|A\to B|\!\rangle\!|)}.\lambda\alpha^A.\mathrm{H}_B(\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!|)\to(\!|\langle\!|B|\!\rangle\!|)}\psi(\mathrm{G}_A\alpha)),\\
\mathrm{G}_\top &\equiv \lambda\phi^\top.\phi,\\
\mathrm{H}_\top &\equiv \lambda\psi^{(\!|\langle\!|\top|\!\rangle\!|)}.\psi,\\
\mathrm{G}_{A\wedge B} &\equiv \lambda\phi^{A\wedge B}.\langle\mathrm{G}_A(\pi_1\phi),\mathrm{G}_B(\pi_2\phi)\rangle,\\
\mathrm{H}_{A\wedge B} &\equiv \lambda\psi^{(\!|\langle\!|A\wedge B|\!\rangle\!|)}.\langle\mathrm{H}_A(\pi_1\psi),\mathrm{H}_B(\pi_2\psi)\rangle,\\
\mathrm{G}_\bot &\equiv \lambda\phi^\bot.\phi,\\
\mathrm{H}_\bot &\equiv \lambda\psi^{(\!|\langle\!|\bot|\!\rangle\!|)}.\psi,\\
\mathrm{G}_{A\vee B} &\equiv \lambda\phi^{A\vee B}.\mu(x^{(\!|\langle\!|A|\!\rangle\!|)}, y^{(\!|\langle\!|B|\!\rangle\!|)}).[x]\,\mathrm{G}_A(\mu w^A.[y]\,\mathrm{G}_B(\mu z^B.[w,z]\,\phi)),\\
\mathrm{H}_{A\vee B} &\equiv \lambda\psi^{(\!|\langle\!|A\vee B|\!\rangle\!|)}.\mu(x^A, y^B).[y]\,\mathrm{H}_B(\mu z^{(\!|\langle\!|B|\!\rangle\!|)}.[x]\,\mathrm{H}_A(\mu w^{(\!|\langle\!|A|\!\rangle\!|)}.[w,z]\,\psi)).
\end{aligned}$$

Therefore we can get the dual result for the previous subsection.

Proposition 6. *In the $\lambda\mu_{\mathrm{n}}$-calculus without* `fix`*, the composite $\mu k^A.(\!|\langle\!|(-)|\!\rangle\!|k)\diamond$ is equivalent to the identity in the following sense:*

for any judgement $\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash M : A \mid x_1 : A_1, \ldots, x_m : A_m$,

$$\begin{aligned}&\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash \\ &\quad \mathrm{H}_A(\mu k^{(|\langle|A|\rangle|)}.\, (|\langle|M|\rangle k|)\diamond)\, [\mathrm{G}_{B_1}\alpha_1 \,/\alpha_1, \ldots, [x_1]\, \mathrm{H}_{A_1}(-) \,/[x_1]\,(-), \ldots] =_{\mathrm{n}} M \\ &\quad : A \mid x_1 : A_1, \ldots, x_m : A_m,\end{aligned}$$

and in other words,

$$\begin{aligned}&\alpha_1 : (|\langle|B_1|\rangle|), \ldots, \alpha_n : (|\langle|B_n|\rangle|) \vdash \\ &\quad \mu k^{(|\langle|A|\rangle|)}.\, (|\langle|M|\rangle k|)\diamond =_{\mathrm{n}} \mathrm{G}_A M\, [\mathrm{H}_{B_1}\alpha_1 \,/\alpha_1, \ldots, [x_1]\, \mathrm{G}_{A_1}(-) \,/[x_1]\,(-), \ldots] \\ &\quad : (|\langle|A|\rangle|) \mid x_1 : (|\langle|A_1|\rangle|), \ldots, x_m : (|\langle|A_m|\rangle|).\end{aligned}$$

Moreover, in the similar way of the `loop` case, we introduce the type translators from $(A \to A) \to A$ to $(|\langle|(A \to A) \to A|\rangle|)$, and *vice versa*.

$$\begin{aligned}\mathrm{G}^A_{\mathrm{fix}} &\equiv \lambda\nu^{(A\to A)\to A}.\lambda\phi^{(|\langle|A|\rangle|)\to(|\langle|A|\rangle|)}.\, \mathrm{G}_A(\nu(\lambda\alpha^A.\, \mathrm{H}_A(\phi(\mathrm{G}_A\alpha)))), \\ \mathrm{H}^A_{\mathrm{fix}} &\equiv \lambda\nu^{((|\langle|A|\rangle|)\to(|\langle|A|\rangle|))\to(|\langle|A|\rangle|)}.\lambda\phi^{A\to A}.\, \mathrm{H}_A(\nu(\lambda\alpha^{(|\langle|A|\rangle|)}.\, \mathrm{G}_A(\phi(\mathrm{H}_A\alpha)))).\end{aligned}$$

Unlike the call-by-value case, $\mathrm{H}^A_{\mathrm{fix}}$ is exactly the inverse of $\mathrm{G}^A_{\mathrm{fix}}$ in the $\lambda\mu_{\mathrm{n}}$-calculus. We can check easily that $\mathrm{H}^A_{\mathrm{fix}} \circ \mathrm{G}^A_{\mathrm{fix}}$ and $\mathrm{G}^A_{\mathrm{fix}} \circ \mathrm{H}^A_{\mathrm{fix}}$ are the identities.

Theorem 7. *The composite* $\mu k^A.\, (|\langle|(-)|\rangle k|)\diamond$ *is equivalent to the identity in the following sense:*
for any judgement $\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash M : A \mid x_1 : A_1, \ldots, x_m : A_m$,

$$\begin{aligned}&\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash \\ &\quad \mathrm{H}_A(\mu k^{(|\langle|A|\rangle|)}.\, (|\langle|M|\rangle k|)\diamond)\, [\ldots, \mathrm{G}^{D_k}_{\mathrm{fix}}\, \mathtt{fix}^{D_k} \,/\mathtt{fix}^{(|\langle|D_k|\rangle|)}, \ldots] =_{\mathrm{n}} M \\ &\quad : A \mid x_1 : A_1, \ldots, x_m : A_m,\end{aligned}$$

and in other words,

$$\begin{aligned}&\alpha_1 : (|\langle|B_1|\rangle|), \ldots, \alpha_n : (|\langle|B_n|\rangle|) \vdash \\ &\quad \mu k^{(|\langle|A|\rangle|)}.\, (|\langle|M|\rangle k|)\diamond =_{\mathrm{n}} \mathrm{G}_A M\, [\ldots, \mathrm{H}^{D_k}_{\mathrm{fix}}\, \mathtt{fix}^{(|\langle|D_k|\rangle|)} \,/\mathtt{fix}^{D_k}, \ldots] \\ &\quad : (|\langle|A|\rangle|) \mid x_1 : (|\langle|A_1|\rangle|), \ldots, x_m : (|\langle|A_m|\rangle|),\end{aligned}$$

where D_k *ranges over all types* D *such that* $\mathtt{fix}^D$ *occurs in* M.

Proof. By induction. Here we demonstrate `fix` case only.

$$
\begin{aligned}
&\mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\langle\!|\,\mathtt{fix}\,|\!\rangle k|\!)\diamond\\
&\quad=_{\mathrm{n}}\ \mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\,\mathtt{loop}^{\langle\!|A|\!\rangle}(\lambda x^{\langle\!|A|\!\rangle}.\mu\beta^{\langle\!|A|\!\rangle}.(\pi_1 k)\langle\lambda y^{\langle\!|A|\!\rangle}.[\beta]\,y,x\rangle)(\pi_2 k)|\!)\diamond\\
&\quad=_{\mathrm{n}}\ \mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\,\mathtt{loop}^{\langle\!|A|\!\rangle}\,|\!)(\lambda\delta^{(\!|\neg\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\to\langle\!|A|\!\rangle\!)}.\\
&\qquad\quad(\!|\lambda x^{\langle\!|A|\!\rangle}.\mu\beta^{\langle\!|A|\!\rangle}.(\pi_1 k)\langle\lambda y^{\langle\!|A|\!\rangle}.[\beta]\,y,x\rangle|\!)(\delta(\lambda\gamma^{\top\to(\!|\langle\!|A|\!\rangle\!)}.(\!|\pi_2 k|\!)(\gamma\diamond))))\\
&\quad=_{\mathrm{n}}\ \mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\lambda x^{\langle\!|A|\!\rangle}.\mu\beta^{\langle\!|A|\!\rangle}.(\pi_1 k)\langle\lambda y^{\langle\!|A|\!\rangle}.[\beta]\,y,x\rangle|\!)\\
&\qquad\quad(\lambda\phi^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}.(\lambda\gamma^{\top\to(\!|\langle\!|A|\!\rangle\!)}.(\!|\pi_2 k|\!)(\gamma\diamond))(\lambda\tau^{\top}.\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}\,\phi))\\
&\quad=_{\mathrm{n}}\ \mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\lambda x^{\langle\!|A|\!\rangle}.\mu\beta^{\langle\!|A|\!\rangle}.(\pi_1 k)\langle\lambda y^{\langle\!|A|\!\rangle}.[\beta]\,y,x\rangle|\!)\\
&\qquad\quad(\lambda\phi^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}.(\!|\pi_2 k|\!)(\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}\,\phi))\\
&\quad=_{\mathrm{n}}\ \mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\pi_2 k|\!)(\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}\\
&\qquad\quad(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mu x^{(\!|\langle\!|A|\!\rangle\!)}.(\!|\mu\beta^{\langle\!|A|\!\rangle}.(\pi_1 k)\langle\lambda y^{\langle\!|A|\!\rangle}.[\beta]\,y,x\rangle|\!)\zeta))\\
&\quad=_{\mathrm{n}}\ \mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\pi_2 k|\!)(\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mu x^{(\!|\langle\!|A|\!\rangle\!)}.\\
&\qquad\quad(\!|\pi_1 k|\!)(\lambda\nu^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.(\mu w^{\neg(\top\to(\!|\langle\!|A|\!\rangle\!))}.[x]\,\mu z^{(\!|\langle\!|A|\!\rangle\!)}.[w,z]\,\nu\diamond)(\lambda\tau^{\top}.\zeta))))\\
&\quad=_{\mathrm{n}}\ \mu(u^{(\!|\neg\langle\!|A\to A|\!\rangle\!)},v^{(\!|\langle\!|A|\!\rangle\!)}).[u,v]\,\mu k^{(\!|\langle\!|(A\to A)\to A|\!\rangle\!)}.(\!|\pi_2 k|\!)(\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mu x^{(\!|\langle\!|A|\!\rangle\!)}.\\
&\qquad\quad(\!|\pi_1 k|\!)(\lambda\nu^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.[x]\,\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\nu\diamond)\zeta)))\\
&\quad=_{\mathrm{n}}\ \mu(u^{(\!|\neg\langle\!|A\to A|\!\rangle\!)},v^{(\!|\langle\!|A|\!\rangle\!)}).[v]\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mu x^{(\!|\langle\!|A|\!\rangle\!)}.\\
&\qquad\quad[u]\,\lambda\nu^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.[x]\,\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\nu\diamond)\zeta)\\
&\quad=_{\mathrm{n}}\ \mu(u^{(\!|\neg\langle\!|A\to A|\!\rangle\!)},v^{(\!|\langle\!|A|\!\rangle\!)}).[u]\,\lambda\epsilon^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.(\mu s^{(\!|\neg\langle\!|A\to A|\!\rangle\!)}.\\
&\qquad\quad[v]\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mu x^{(\!|\langle\!|A|\!\rangle\!)}.[s]\,\lambda\nu^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.[x]\,\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\nu\diamond)\zeta))\epsilon\\
&\quad=_{\mathrm{n}}\ \mu(u^{(\!|\neg\langle\!|A\to A|\!\rangle\!)},v^{(\!|\langle\!|A|\!\rangle\!)}).[u]\,\lambda\epsilon^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.\mu s^{\perp}.\\
&\qquad\quad[v]\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mu x^{(\!|\langle\!|A|\!\rangle\!)}.[s](\lambda\nu^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.[x]\,\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\nu\diamond)\zeta)\epsilon)\\
&\quad=_{\mathrm{n}}\ \mu(u^{(\!|\neg\langle\!|A\to A|\!\rangle\!)},v^{(\!|\langle\!|A|\!\rangle\!)}).[u]\,\lambda\epsilon^{\top\to(\!|\langle\!|A\to A|\!\rangle\!)}.\\
&\qquad\quad[v]\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\epsilon\diamond)\zeta)\\
&\quad=_{\mathrm{n}}\ \mathrm{G}_0^{(\!|\langle\!|A\to A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\lambda\alpha^{(\!|\langle\!|A\to A|\!\rangle\!)}.\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}\alpha\,\zeta))\\
&\quad=_{\mathrm{n}}\ \mathrm{G}_0^{(\!|\langle\!|A\to A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}\\
&\qquad\quad(\lambda\alpha^{(\!|\langle\!|A\to A|\!\rangle\!)}.\,\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mathrm{G}_A(\mathrm{H}_A(\mathrm{H}_0^{(\!|\langle\!|A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}\alpha(\mathrm{G}_A(\mathrm{H}_A\zeta))))))\\
&\quad=_{\mathrm{n}}\ \mathrm{G}_0^{(\!|\langle\!|A\to A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}\\
&\qquad\quad(\lambda\alpha^{(\!|\langle\!|A\to A|\!\rangle\!)}.\mathrm{G}_A(\mathrm{H}_A(\mathtt{fix}^{(\!|\langle\!|A|\!\rangle\!)}(\lambda\zeta^{(\!|\langle\!|A|\!\rangle\!)}.\mathrm{G}_A(\mathrm{H}_{A\to A}\alpha(\mathrm{H}_A\zeta))))))\\
&\quad=_{\mathrm{n}}\ \mathrm{G}_0^{(\!|\langle\!|A\to A|\!\rangle\!)\to(\!|\langle\!|A|\!\rangle\!)}(\lambda\alpha^{(\!|\langle\!|A\to A|\!\rangle\!)}.\mathrm{G}_A(\mathrm{H}^A_{\mathtt{fix}}\,\mathtt{fix}^A(\mathrm{H}_{A\to A}\alpha)))\\
&\quad=_{\mathrm{n}}\ \mathrm{G}_{(A\to A)\to A}(\mathrm{H}^A_{\mathtt{fix}}\,\mathtt{fix}^A).
\end{aligned}
$$

□

Here it has shown that the $\lambda\mu_{\mathrm{n}}$-calculus extended with a fixed-point operator is exactly dual for the $\lambda\mu_{\mathrm{v}}$-calculus extended with an iteration operator. The syntactic translations between recursion and iteration enable us to apply the

duality to practical programming: for example, we can use techniques of call-by-name recursive programming in a call-by-value language or can verify call-by-value programs in the call-by-name setting.

4 Applications

4.1 Call-by-value fixed-point operators

What is described in this subsection is the joint work with M. Hasegawa. Categorical discussion and more details are in [12].

Though we have discussed iteration in the call-by-value languages, iteration is less familiar than recursion with functional languages. However, Filinski demonstrated in [7] that iteration operators have bijective correspondence with recursion operators under *a certain condition* in a call-by-value calculus with first-class continuations. Since the $\lambda\mu_{\mathrm{v}}$-calculus can be considered as a call-by-value language with first-class continuations, the $\lambda\mu_{\mathrm{v}}$-calculus with an iteration operator is as expressive as the one with a fixed-point operator.

First, we consider what call-by-value recursion is. Unlike the call-by-name calculus, call-by-value recursion operators are not able to satisfy the standard fixed-point equation: for any value $V : A \to A$,

$$\mathtt{fix}_{\mathrm{v}}^{A}\, V = V\,(\mathtt{fix}_{\mathrm{v}}^{A}\, V).$$

This equation implies that the evaluation of $\mathtt{fix}_{\mathrm{v}}^{A}\, V$ never terminates, so it does not make sense. A usual solution to this problem is to restrict the applied types to function types and η-expand the right-hand-side of the equation. Namely,

$$\mathtt{fix}_{\mathrm{v}}^{A\to B}\, F = \lambda x^{A}.\, F\,(\mathtt{fix}_{\mathrm{v}}^{A\to B}\, F)\, x$$

for any value $F : (A \to B) \to (A \to B)$. Since the right-hand-side of this equation is a value, the evaluation of $\mathtt{fix}_{\mathrm{v}}^{A\to B}\, F$ terminates. However, the foregoing equation is not sufficient for reasoning about call-by-value recursive computation. There are many useful equations that are not derivable from this equation alone.

Hence, we propose more axioms for call-by-value fixed-point operators. One is called **the stability axiom**: for any value $F : (A \to B) \to (A \to B)$,

$$\mathtt{fix}_{\mathrm{v}}^{A\to B}\, F = \mathtt{fix}_{\mathrm{v}}^{A\to B}\,(\lambda f^{A\to B}.\lambda x^{A}.\, F\, f\, x)$$

It looks rather simple and intuitively natural, but we believe stability is significant for axiomatization of call-by-value recursors. The other axiom is **the uniformity axiom**. Our stable uniform fixed-point operator coincides with the uniform T-fixed-point operator of Simpson and Plotkin [26], when T is the continuation monad $R^{R^{(-)}}$. Our uniformity axiom is the following:

> for any values $F : (A \to B) \to (A \to B)$, $G : (A' \to B') \to (A' \to B')$, and any rigid value $H : (A \to B) \to (A' \to B')$, if $H \circ F = G \circ H$ holds, then $H\,(\mathtt{fix}_{\mathrm{v}}^{A\to B}\, F) = \mathtt{fix}_{\mathrm{v}}^{A'\to B'}\, G$.

Here $H\colon (A \to B) \to (A' \to B')$ is **rigid** if $H(\lambda x^A.\, M x) = \lambda y^{A'}.\, H M\, y$ holds for any $M\colon A \to B$. This definition of 'rigid' is different from Filinski's 'rigid' definition [7] but our stable uniformity is equivalent to Filinski's uniformity, and we can construct a bijective correspondence between *stable uniform* fixed-point operators and *uniform* iteration operators.

Filinski has demonstrated in [7] that recursors and iterators are mutually definable. We demonstrate it on the $\lambda\mu_{\mathrm{v}}$-calculus:

$$\begin{aligned}
\mathtt{loop}^{A}_{\mathtt{fix}}\, f &= \mathtt{fix}^{\neg A}_{\mathrm{v}}(\mathtt{pets}_{A,A}\, f),\\
\mathtt{fix}^{A\to B}_{\mathtt{loop}}\, F &= \mathtt{switch}_{A,B}(\mathtt{loop}^{\neg B\wedge A}(\mathtt{step}_{\neg B\wedge A,\neg B\wedge A}(\mathtt{switch}^{-1}_{A,B} \circ F \circ \mathtt{switch}_{A,B}))),
\end{aligned}$$

where

$$\begin{aligned}
\mathtt{step}_{A,B} &\equiv \lambda f^{\neg B\to\neg A}.\lambda x^A.\mu\beta^B.\, f(\lambda y^B.[\beta]\, y)\, x &&: (\neg B \to \neg A) \to A \to B,\\
\mathtt{pets}_{A,B} &\equiv \lambda f^{A\to B}.\lambda k^{\neg A}.\lambda x^B.\, k(f x) &&: (A \to B) \to \neg B \to \neg A,\\
\mathtt{switch}_{A,B} &\equiv \lambda m^{\neg(\neg B\wedge A)}.\lambda x^A.\mu\beta^B.\, m\langle \lambda y^B.[\beta]\, y, x\rangle &&: \neg(\neg B \wedge A) \to A \to B.
\end{aligned}$$

Properties of $\mathtt{loop}^{A}_{\mathtt{fix}}$ and $\mathtt{fix}^{A\to B}_{\mathtt{loop}}$ are guaranteed by the equations,

$$\begin{aligned}
\mathtt{step}_{A,B} \circ \mathtt{pets}_{A,B} &=_{\mathrm{v}} \mathrm{id}_{A\to B},\\
\mathtt{pets}_{A,B} \circ \mathtt{step}_{A,B} &=_{\mathrm{v}} \lambda F^{\neg B\to\neg A}.\lambda f^{\neg B}.\lambda x^A.\, F f\, x,\\
\mathtt{switch}^{-1}_{A,B} &=_{\mathrm{v}} \lambda f^{A\to B}.\lambda\langle k^{\neg B}, x^A\rangle.\, k(f x).
\end{aligned}$$

On the translations above, the stability absorbs a gap between $\mathtt{pets}_{A,B} \circ \mathtt{step}_{A,B}$ and the identity. If we ignore $\mathtt{switch}_{A,\bot}$ and $\mathtt{switch}^{-1}_{A,\bot}$, then $\mathtt{fix}^{\neg A}_{\mathtt{loop}}$ is considered as $\mathtt{loop}^{A} \circ \mathtt{step}_{A,\bot}$. Therefore, a bijective correspondence arises between iteration operators and stable fixed-point operators restricted on negative types.

However yet uniformity is necessary for the perfect bijective correspondence between recursors and iterators. For the iteration uniformity, we begin with defining the totality: $H\colon A' \to A$ is **total** if for any value $V\colon A'$, HV is **focal**. 'focal' intuitively means 'effect-free'. The word 'total' (which misleadingly does not mean 'discardable') is due to Filinski [7], but he defined 'total' with the condition that HV is a *value*. In the $\lambda\mu_{\mathrm{v}}$-calculus, $M\colon A$ is focal if and only if for any $N\colon B$ and $L\colon C$, $\mathtt{let}\, x^A\, \mathtt{be}\, M\, \mathtt{in}\, \mathtt{let}\, y^B\, \mathtt{be}\, N\, \mathtt{in}\, L = \mathtt{let}\, y^B\, \mathtt{be}\, N\, \mathtt{in}\, \mathtt{let}\, x^A\, \mathtt{be}\, M\, \mathtt{in}\, L$ holds. It follows that any value is focal, so a total function in Filinski's sense is also total in our sense. The above defined $\mathtt{step}$ sends a rigid function to a total function, and $\mathtt{pets}$ sends a total function to a rigid function. Then we define the uniformity for iteration:

> for any values $F\colon A' \to A'$ and $G\colon A \to A$, and any total value $H\colon A' \to A$,
> if $H \circ F = G \circ H$, then $(\mathtt{loop}^{A}\, G) \circ H = \mathtt{loop}^{A'}\, F$ holds.

If an iteration operator satisfies this condition, it is called a **uniform** iterator.

Theorem 1 (Hasegawa and Kakutani [12]). *In the $\lambda\mu_{\mathrm{v}}$-calculus, there is a bijective correspondence between stable uniform fixed-point operators and uniform iteration operators.*

4.2 Inductive data types and co-inductive data types

In this subsection, we give some applications of the duality for programming practice. We use $\mathtt{zero}$ for a name of the type $\top$, and use $\mathtt{suc}$ for a name of the type $\mathbb{N}$. If the type $\mathbb{N}$ satisfies the equation

$$\mathbb{N} = \top + \mathbb{N},$$

$\mathbb{N}$ can be the type of natural numbers by the following scheme. We define

$$\begin{aligned}
0 &\equiv \mu(\mathtt{zero}^{\top}, \mathtt{suc}^{\mathbb{N}}).[\mathtt{zero}]\ *\ ,\\
1 &\equiv \mu(\mathtt{zero}^{\top}, \mathtt{suc}^{\mathbb{N}}).[\mathtt{suc}]\,0,\\
2 &\equiv \mu(\mathtt{zero}^{\top}, \mathtt{suc}^{\mathbb{N}}).[\mathtt{suc}]\,1,\\
&\vdots \quad ,\\
n{+}1 &\equiv \mu(\mathtt{zero}^{\top}, \mathtt{suc}^{\mathbb{N}}).[\mathtt{suc}]\,n,\\
&\vdots \quad .
\end{aligned}$$

Because $\top + \mathbb{N} = \mathbb{N}$, $\Gamma \vdash 0 : \mathbb{N} \mid \Delta$. So, it follows $\Gamma \vdash n : \mathbb{N} \mid \Delta$ for any natural number n. We can see easily that $n{-}1 =_{\mathrm{v}} \mu\mathtt{suc}.[\mathtt{zero}, \mathtt{suc}]\,n$ holds for any positive number n. After this, we may confuse meta-notation $M{+}1$ and $M{-}1$ with the terms, $\mu(\mathtt{zero}, \mathtt{suc}).[\mathtt{suc}]\,M$ and $\mu(\mathtt{zero}, \mathtt{suc}).[\mathtt{zero}]\,M$ respectively. According to previous subsection, we can use recursive definition of functions in the $\lambda\mu_{\mathrm{v}}$-calculus. For example, $\mathtt{plus} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ that satisfies the equation $\mathtt{plus}\, m\, n =_{\mathrm{v}} m{+}n$ is definable.

$$\begin{aligned}
\mathtt{plus} &\equiv \lambda m^{\mathbb{N}}.\lambda n^{\mathbb{N}}.\mu\beta^{\mathbb{N}}.\ \mathtt{loop}^{\neg\mathbb{N}\wedge\mathbb{N}}(\lambda\langle k^{\neg\mathbb{N}}, l^{\mathbb{N}}\rangle.\, \mathrm{P}\langle k, l\rangle)\ \langle \lambda y^{\mathbb{N}}.[\beta]\, y, n\rangle,\\
\mathrm{P}\langle k, l\rangle &\equiv \langle \lambda x^{\mathbb{N}}.\, k(x{+}1), \mu\alpha^{\mathbb{N}}.\ (\lambda l_1^{\mathbb{N}}.[\alpha]\, l_1)(\mu\mathtt{suc}.\ (\lambda t^{\top}.\, km)(\mu\mathtt{zero}.[\mathtt{zero}, \mathtt{suc}]\, l))\rangle.
\end{aligned}$$

Another basic example is $\mathtt{fact} : \mathbb{N} \to \mathbb{N}$, which computes factorials. When we assume $\mathtt{times}$ is defined,

$$\begin{aligned}
\mathtt{fact} &\equiv \lambda n^{\mathbb{N}}.\mu\beta^{\mathbb{N}}.\ \mathtt{loop}^{\neg\mathbb{N}\wedge\mathbb{N}}(\lambda\langle k^{\neg\mathbb{N}}, l^{\mathbb{N}}\rangle.\, \mathrm{F}\langle k, l\rangle)\ \langle \lambda y^{\mathbb{N}}.[\beta]\, y, n\rangle,\\
\mathrm{F}\langle k, l\rangle &\equiv \langle \lambda x^{\mathbb{N}}.\, k(\mathtt{times}\, x\, l), \mu\alpha^{\mathbb{N}}.\ (\lambda l_1^{\mathbb{N}}.[\alpha]\, l_1)(\mu\mathtt{suc}.\ (\lambda t^{\top}.\, k1)(\mu\mathtt{zero}.[\mathtt{zero}, \mathtt{suc}]\, l))\rangle.
\end{aligned}$$

These continuation based programs are similar to usual recursive programs with accumulators. The names, $\mathtt{zero}$ and $\mathtt{suc}$, are used for branching: this technique is not a trick because in a usual functional language that has coproduct types, 0 is expressed by $\mathtt{in}_1 *$, $n{+}1$ is expressed by $\mathtt{in}_2 n$, and $N(\mu\mathtt{suc}.\, M\,(\mu\mathtt{zero}.[\mathtt{zero}, \mathtt{suc}]\, n))$ is expressed by $\mathtt{case}\, n\, \mathtt{of}\, \mathtt{in}_1 t \Rightarrow M t \mid \mathtt{in}_2 n_1 \Rightarrow N n_1$.

If we treat booleans only, type structure gets simpler, *i.e.*, $\mathrm{Bool} = \top + \top$. In this case, $\mu(\mathtt{false}, \mathtt{true}).[\mathtt{true}]\, *$ is the true value, and $\mu(\mathtt{false}, \mathtt{true}).[\mathtt{false}]\, *$ is the false value. The branching process is just the same as natural numbers.

On the other hand, by the duality, we can expand this theory in the call-by-name setting. Assume $\mathbb{N} = \bot \times \mathbb{N}$, which is the dual equation of the call-by-value

$\mathbb{N} = \top + \mathbb{N}$, holds in the $\lambda\mu_{\mathrm{n}}$-calculus. Natural numbers are translated to the $\lambda\mu_{\mathrm{n}}$-terms:

$$(\!|0|\!) =_{\mathrm{n}} \lambda\kappa^{\mathbb{N}}.\, \pi_1\kappa, \qquad (\!|1|\!) =_{\mathrm{n}} \lambda\kappa^{\mathbb{N}}.\, \pi_1(\pi_2\kappa),$$

more generally,

$$(\!|n|\!) =_{\mathrm{n}} \lambda\kappa^{\mathbb{N}}.\, \pi_1(\pi_2^n\kappa).$$

In fact, $\mathbb{N}$ is the type of infinite streams and $(\!|0|\!)$ is a function to get the head of a stream. As an analogy, for booleans, 'true' and 'false' are translated to π_2 and π_1 respectively. π_2 and π_1 are indeed 'true' and 'false' in canonical sense of the call-by-name calculus.

This result of the duality matches with the categorical investigation. Let $\mathcal{C}$ be a response category. If $\mathcal{C}$ has a natural numbers object [16] $1 + \mathbb{N} \xrightarrow[{[z,s]}]{} \mathbb{N}$, then in the category of continuations $R^{\mathcal{C}}$, $\langle R^z, R^s\rangle : \mathbb{N} \to \bot \times \mathbb{N}$ is an isomorphism. It follows the consistency of the existence of $\mathbb{N}$.

A type of streams enables the $\lambda\mu_{\mathrm{n}}$-calculus to express functions on natural numbers. For example, we shall demonstrate the call-by-name version of the factorial computation. Let

$$n' \equiv (\!|n|\!) =_{\mathrm{n}} \lambda\kappa^{\mathbb{N}}.\, \pi_1(\pi_2^n\kappa) : \mathbb{N} \to \bot.$$

We will define $\mathtt{fact}' : \neg\mathbb{N} \to \neg\mathbb{N}$:

$$\begin{aligned} \mathtt{fact}' &\equiv \lambda\nu^{\neg\mathbb{N}}.\lambda\kappa^{\mathbb{N}}.\, \nu(\mathtt{fix}^{\mathbb{N}\to\mathbb{N}}(\lambda\phi^{\mathbb{N}\to\mathbb{N}}.\, F\phi)\,\kappa), \\ F\phi &\equiv \lambda\eta^{\mathbb{N}}.\mu l^{\mathbb{N}}.[l]\,\langle 1'\eta, \phi(\mu x^{\mathbb{N}}.\ \mathtt{times}'(\lambda\alpha^{\mathbb{N}}.[x]\,\alpha)\,(\lambda\beta^{\mathbb{N}}.[l]\,\beta)\,\eta)\rangle, \end{aligned}$$

where $\mathtt{times}'\, m'n' =_{\mathrm{n}} (m\times n)'$ holds. This program is exactly the dual of the call-by-value program defined above. Namely $\mathtt{fact}'$ can be constructed from $\mathtt{fact}$ by the dual translation $(\!|(-)|\!)$. Thus the dual translations help us to program with recursion, and the comparison of call-by-name operations of a program with call-by-value operations of its dual program points out the essential behavior of programs in some sense.

Another implication of the examples is the duality of data types. In the call-by-value case, $\mathbb{N}$ is often given by the inductive limit of the diagram

$$0 \xrightarrow{i_0} \top \xrightarrow{i_1} \top + \top \xrightarrow{i_2} \top + (\top + \top) \xrightarrow{i_3} \top + (\top + (\top + \top)) \xrightarrow{i_4} \cdots,$$

where $i_0 = \Box_\top$ ($\Box_X : 0 \to X$ denotes the initial map), $i_1 = \iota_1$ (ι_k denotes the injection of coproducts), and $i_{n+2} = \mathrm{id}_\top + i_{n+1}$. Dually in the call-by-name case, $\mathbb{N}$ can be given by the projective limit of

$$1 \xleftarrow{p_0} \bot \xleftarrow{p_1} \bot \times \bot \xleftarrow{p_2} \bot \times (\bot \times \bot) \xleftarrow{p_3} \bot \times (\bot \times (\bot \times \bot)) \xleftarrow{p_4} \cdots,$$

where $p_0 = \Diamond_\bot$ ($\Diamond_X : 1 \to X$ denotes the terminal map), $p_1 = \pi_1$ (π_k denotes the projection of products), $p_{n+2} = \mathrm{id}_\bot \times p_{n+1}$. This implies that the duality between an inductive type and a co-inductive type has coincided with the duality between call-by-name and call-by-value. The duality of data types also matches with the dual translations on the $\lambda\mu$-calculi.

Conclusion

Summary

In this thesis, we have investigated the duality between call-by-name recursion and call-by-value iteration via the duality between values and continuations.

In [25], Selinger has shown the model-theoretic duality between the call-by-name $\lambda\mu$-calculus and the call-by-value one, and introduced the syntactic duality from it. Along the line that Selinger has taken, we studied the duality between call-by-name recursion and call-by-value iteration extending the $\lambda\mu$-calculi with a call-by-name fixed-point operator and a call-by-value iteration operator.

Because the syntactic translations have dealt with recursion and iteration, there is some possibilities that they applied to practical programs. For example, we expect that the translations may be used for verification of programs or compiling.

Further work

In Section 4, we considered duality between an inductive type and a co-inductive type. However, it is still *ad hoc*. We guess the natural numbers type is sufficiently worth studying, but the study of more general inductive data types will give much implication for the type of natural numbers. For the study of general inductive data types, further categorical investigation and more detailed syntactic treatment should be required. The duality between inductive types and co-inductive types follows duality between the languages enriched with data types. Because data types play an important role of practical programming as is demonstrated in Section 4, the study of data-type duality will give us more useful applications.

Furthermore, the duality between inductive types and co-inductive types is not all category-theoretic duality. More general structure of data types are waiting to be studied along this line.

Uniformity

We have proposed an axiomatization for call-by-value fixed-point operators in Section 4 (really in [12]). Our three axioms (the usual fixed-point axiom, the stability axiom, and the uniformity axiom) are justified in two ways: a category-theoretic axiomatics (of Simpson and Plotkin [26]) and an actual program construction (of Filinski [7]). In the same paper, we also proposed the uniformity for call-by-value iteration operators such that there is a bijective correspondence between fixed-point operators and iteration operators. Therefore, our uniform iterator is justified in the same sense as the stable uniform fixed-point operator is. So, the uniformity for call-by-name fixed-point operators is induced as its mirror image, and is also justified through the duality. However, in direct call-by-name semantics, the call-by-name uniformity looks somewhat strange. The property of uniformity in the $\lambda\mu_{\mathrm{n}}$-calculus remains to be studied.

Acknowledgements

I am deeply grateful to Masahito Hasegawa. The discussion with him motivated me to write this thesis and lots of his suggestions have shown right directions of the research.

I would like to thank Susumu Nishimura for helpful comments about the construction of this thesis.

I thank Jacques Garrigue for the appropriate advice of writing the paper.

I also thank Prof. Reiji Nakajima for his encouragement. Deep appreciation goes to Akira Itou for his encouragement and special talk.

A Appendix: Categorical Background

A.1 Control categories

We have used some technical terms without sufficient explanation through this paper. In this appendix, we give some definitions omitted in Section 2. Most definitions and theorems are from [25].

Definition 1. *A category with finite products is called a cartesian closed category if for each object A, the functor $(-) \times A$ has a right-adjoint functor $(-)^A$.*

ev denotes the counit of this adjunction, and π_1, π_2 denote the projections of binary products.

Definition 2. *A binoidal category is a category $\mathcal{P}$ together with a binary operation $\oplus$ such that $A \oplus (-)$ and $(-) \oplus B$ are functorial.*

Definition 3. *A morphism $f\colon A \to A'$ in a binoidal category is central if for every morphism $g\colon B \to B'$, the following diagrams commute:*

$$\begin{tikzcd} A \oplus B \arrow[r, "A \oplus g"] \arrow[d, "f \oplus B"'] & A \oplus B' \arrow[d, "f \oplus B'"] \\ A' \oplus B \arrow[r, "A' \oplus g"] & A' \oplus B' \end{tikzcd} \qquad \begin{tikzcd} B \oplus A \arrow[r, "g \oplus A"] \arrow[d, "B \oplus f"'] & B \oplus A' \arrow[d, "B' \oplus f"] \\ B' \oplus A \arrow[r, "g \oplus A'"] & B' \oplus A' \end{tikzcd}$$

Definition 4. *A premonoidal category is a binoidal category $\mathcal{P}$ together with an object $\perp$ and central natural isomorphisms $a_{A,B,C}\colon (A \oplus B) \oplus C \to A \oplus (B \oplus C)$, $l_A\colon A \to A \oplus \perp$ and $r_A\colon A \to \perp \oplus A$, such that the following diagrams commute:*

$$\begin{tikzcd} ((A \oplus B) \oplus C) \oplus D \arrow[r, "a_{(A\oplus B),C,D}"] \arrow[d, "a_{A,B,C} \oplus D"] & (A \oplus B) \oplus (C \oplus D) \arrow[r, "a_{A,B,(C\oplus D)}"] & A \oplus (B \oplus (C \oplus D)) \\ (A \oplus (B \oplus C)) \oplus D \arrow[rr, "a_{A,(B\oplus C),D}"] & & A \oplus ((B \oplus C) \oplus D) \arrow[u, "A \oplus a_{B,C,D}"'] \end{tikzcd}$$

$$\begin{tikzcd}
A \oplus B \arrow[r, "l_A \oplus B"] \arrow[d, "A \oplus r_B"'] & (A \oplus \bot) \oplus B \arrow[dl, "a_{A,\bot,B}"] \\
A \oplus (\bot \oplus B) &
\end{tikzcd}$$

A symmetric premonoidal category is a premonoidal category in addition with central natural isomorphisms $c_{A,B} : A \oplus B \to B \oplus A$ such that the following diagrams commute:

$$\begin{tikzcd}
A \oplus B \arrow[d, "c_{A,B}"'] \arrow[dr, "\mathrm{id}_{A \oplus B}"] & \\
B \oplus A \arrow[r, "c_{B,A}"] & A \oplus B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "l_A"] \arrow[d, "r_A"'] & A \oplus \bot \arrow[dl, "c_{A,\bot}"] \\
\bot \oplus A &
\end{tikzcd}$$

$$\begin{tikzcd}
(A \oplus B) \oplus C \arrow[r, "a_{A,B,C}"] \arrow[d, "c_{A,B} \oplus C"'] & A \oplus (B \oplus C) \arrow[r, "c_{A,B \oplus C}"] & (B \oplus C) \oplus A \arrow[d, "a_{B,C,A}"'] \\
(B \oplus A) \oplus C \arrow[r, "a_{B,A,C}"] & B \oplus (A \oplus C) \arrow[r, "B \oplus c_{A,C}"] & B \oplus (C \oplus A)
\end{tikzcd}$$

The subcategory of a premonoidal category $\mathcal{P}$ formed by the central morphisms is called the center of $\mathcal{P}$ and denoted by $\mathcal{P}^\bullet$. The center is the largest subcategory on which $\oplus$ restricts to a bifunctorial tensor product.

Definition 5. *In a symmetric premonoidal category, a symmetric monoid is given by an object A together with two central morphisms $i_A : \bot \to A$ and $\nabla_A : A \oplus A \to A$ such that the following diagrams commute:*

$$\begin{tikzcd}
A \oplus \bot \arrow[r, "A \oplus i_A"] \arrow[dr, "l_A^{-1}"'] & A \oplus A \arrow[d, "\nabla_A"] & \bot \oplus A \arrow[l, "i_A \oplus A"'] \arrow[dl, "r_A^{-1}"] \\
& A &
\end{tikzcd}
\qquad
\begin{tikzcd}
A \oplus A \arrow[d, "\nabla_A"'] \arrow[dr, "c_{A,A}"] & \\
A & A \oplus A \arrow[l, "\nabla_A"']
\end{tikzcd}$$

$$\begin{tikzcd}
(A \oplus A) \oplus A \arrow[rr, "a_{A,A,A}"] \arrow[d, "\nabla_A \oplus A"'] & & A \oplus (A \oplus A) \arrow[d, "A \oplus \nabla_A"] \\
A \oplus A \arrow[r, "\nabla_A"] & A & A \oplus A \arrow[l, "\nabla_A"']
\end{tikzcd}$$

Definition 6. *A codiagonal is a family of symmetric monoids $\langle A, i_A, \nabla_A \rangle$ such that the following diagrams commute:*

$$\begin{tikzcd} \bot \arrow[d, shift right, "i_\bot"'] \arrow[d, shift left, "\mathrm{id}_\bot"] \\ \bot \end{tikzcd} \qquad \begin{tikzcd} \bot \arrow[d, "l_\bot"'] \arrow[dr, "i_{A\oplus B}"] & \\ \bot \oplus \bot \arrow[r, "i_A \oplus i_B"] & A \oplus B \end{tikzcd} \qquad \begin{tikzcd} A \oplus B \oplus A \oplus B \arrow[r, "\nabla_{A\oplus B}"] \arrow[d, "A \oplus c_{A,B} \oplus B"'] & A \oplus B \\ A \oplus A \oplus B \oplus B \arrow[ur, "\nabla_A \oplus \nabla_B"'] & \end{tikzcd}$$

Definition 7. *A morphism $f : A \to B$ is discardable if the following diagram commutes:*

$$\begin{tikzcd} \bot \arrow[r, "i_A"'] \arrow[d, "i_B"'] & A \arrow[dl, "f"] \\ B & \end{tikzcd}$$

A morphism $f : A \to B$ is copyable if the following diagram commutes:

$$\begin{tikzcd} A \oplus A \arrow[r, "f \oplus f"'] \arrow[d, "\nabla_A"'] & B \oplus B \arrow[d, "\nabla_B"] \\ A \arrow[r, "f"] & B \end{tikzcd}$$

A morphism is focal if it is discardable, copyable and central.

The focal morphisms form a subcategory of a premonoidal category $\mathcal{P}$, which is called the focus of $\mathcal{P}$ and denoted by $\mathcal{P}^{\#}$. The focus is a subcategory of the center and it is closed under $\oplus$.

Lemma 8. *The focus has a canonical coproduct structure.*

Definition 9. *Let $\mathcal{P}$ be a symmetric premonoidal category with codiagonals. And suppose $\mathcal{P}$ has finite products. $\mathcal{P}$ is distributive if each projection $\pi_i : A_1 \times A_2 \to A_i$ is focal and for any object C, $(-) \oplus C$ preserves finite products.*

Lemma 10. *If $\mathcal{P}$ is a distributive symmetric premonoidal category with codiagonals, then the focus of $\mathcal{P}$ is closed under finite products.*

Notions of 'focal', 'central', 'discardable', and 'copyable' are important for analyses of call-by-value languages. Führmann gives a further investigation of them in [9]. An interesting example `twicecc`, which is discardable but not copyable, has appeared in [28].

Definition 11. *Let $\mathcal{P}$ be a distributive symmetric premonoidal category with codiagonals and also be cartesian-closed. $\mathcal{P}$ is a control category if $s_{A,B,C} : B^A \oplus C \to (B \oplus C)^A$, obtained by currying the morphism*

$$(B^A \oplus C) \times A \xrightarrow{(B^A \oplus C) \times w^l} (B^A \oplus C) \times (A \oplus C) \xrightarrow{d_{B^A,A,C}} (B^A \times A) \oplus C \xrightarrow{\mathrm{ev} \oplus C} B \oplus C,$$

is a natural isomorphism in A, B, and C, and the following diagrams commute where $s'_{A,B,C} : B \oplus C^A \to (B \oplus C)^A$ *is* $c^A_{C,B} \circ s_{A,C,B} \circ c_{B,C^A}$*:*

$$\begin{tikzcd}
B^A \oplus C^D \arrow[r, "s'_{D,B^A,C}"] \arrow[d, "s_{A,B,C^D}"] & (B^A \oplus C)^D \arrow[d, "(s_{A,B,C})^D"'] \\
(B \oplus C^D)^A \arrow[r, "(s'_{D,B,C})^A"] & ((B \oplus C)^D)^A \cong ((B \oplus C)^A)^D
\end{tikzcd}$$

$$\begin{tikzcd}
B^A \oplus B^A \arrow[r, "s'_{A,B^A,B}"] \arrow[dr, "\nabla_{B^A}"'] & (B^A \oplus B)^A \arrow[r, "(s_{A,B,B})^A"] & (B \oplus B)^{A \times A} \arrow[dl, "\nabla_B{}^{\Delta_A}"] \\
& B^A &
\end{tikzcd}
\qquad
\begin{tikzcd}
\bot \arrow[d, shift right, "i_{B^A}"'] \arrow[d, shift left, "(i_B)^{\diamond_A}"] \\
B^A
\end{tikzcd}$$

A.2 More structures and co-control categories

For the interpretation of the $\lambda\mu_{\mathrm{v}}$-calculus, we examine the structures of control categories.

Lemma 12. *In a control category, any central morphism is discardable and copyable.*

This implies that the center and the focus coincide in a control category. This is very useful for reasoning about call-by-value computation.

Lemma 13. *In a control category $\mathcal{P}$,*

$$\mathcal{P}(B, C \oplus A) \cong \mathcal{P}^{\#}(\bot^{A^B}, C)$$

holds, and the isomorphisms are natural in A, B, and central C.

Through this adjunction, in the opposite category of a control category, which is called a co-control category, $\bot^{A^{(-)}}$ is similar to an exponential functor for the functor $(-) \oplus A$. Hence, we write $B \circ\!\!- A$ for $\bot^{A^B}$ in control categories, and use in co-control categories the notation whose correspondence to control categories is described in Table 1.

Every co-control category is, in fact, a $\otimes\neg$-category [27]. $\otimes\neg$-categories have only the structures $A \otimes B$ and $\neg A$, but B_A and $A \multimap B$ are expressed by $\neg A \otimes B$ and $\neg(A \otimes \neg B)$, and conversely, $\neg A$ is denoted by $A \multimap 0$ in co-control categories. Thus, it has been justified to consider co-control categories as the models of call-by-value calculi. While Thielecke have taken no notice for coproducts in $\otimes\neg$-categories, the presence of coproducts plays an important role for the duality as noted by Selinger.

Table 1. The correspondence between control categories and co-control categories

control categories	$A \times B$	1	$A \oplus B$	$\bot$	B^A	$B \circ\!\!- A$
co-control categories	$A + B$	0	$A \otimes B$	$\top$	B_A	$A \multimap B$

References

1. S. Bloom and Z. Ésik. *Iteration Theories.* EATC Monographs on Theoretical Computer Science. Springer-Verlarg, 1993.
2. P. L. Curien and Herbelin H. The duality of computation. In *Proceedings of 5th ACM SIGPLAN International Conference on Functional Programming.* ACM Press, 2000.
3. M. Felleisen. *The Calculi of λ_v-conversion: A Syntactic Theory of Control and State in Imperative Higher-Order Programming Languages.* PhD thesis, Indiana University, 1986.
4. M. Felleisen, D. P. Friedman, E. Kohlbecker, and B. F. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.
5. M. Felleisen and R. Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 102:235–271, 1992.
6. A. Filinski. Declarative continuations and categorical duality. Master's thesis, Computer Science Department, University of Copenhagen, 1989.
7. A. Filinski. Recursion from iteration. *Lisp and Symbolic Computation*, 7:11–38, 1994.
8. C. Flanagan, A. Sabry, B. F. Duba, and M. Felleisen. The essence of compiling with continuations. In *Programming Language Design and Implementation*, volume 28, pages 237–247, 1993.
9. C. Führmann. *The structure of call-by-value.* PhD thesis, University of Edinburgh, 2000.
10. T. G. Griffin. A formulae-as-types notion of control. In *Proceedings of 17th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 47–58. ACM Press, 1990.
11. R. Harper, B. F. Duba, and D. MacQueen. Typing first-class continuations in ML. *Journal of Functional Programming*, 3(4):465–484, 1993.
12. M. Hasegawa and Y. Kakutani. Axioms for recursion in call-by-value. In *Foundation of Software Science and Computation Structures*, volume 2030 of *LNCS*, pages 246–260. Springer-Verlag, 2001.
13. M. Hofmann. Sound and complete axiomatisations of call-by-value control operators. *Mathematical Structures in Computer Science*, 5(4):461–482, 1995.
14. M. Hofmann and T. Streicher. Continuation models are universal for $\lambda\mu$-calculus. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, 1997.
15. H. Huwig and A. Poigne. A note on inconsistencies caused by fixpoints in a cartesian closed category. *Theoretical Computer Science*, 73(1):101–112, 1990.
16. J. Lambek and P. J. Scott. *Introduction to higher order categorical logic.* Number 7 in Cambridge studies in advanced mathematics. Cambridge University Press, 1986.
17. P. J. Landin. A generalization of jumps and labels. *Higher-Order and Symbolic Computation*, 11(2), 1998. reprint of Report, UNIVAC System Programming Research, 1965.

18. J. Longley. When is a functional program not a functional program? In *Proceedings of 4th ACM SIGPLAN International Conference on Functional Programming*. ACM Press, 1999.
19. E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-66, Department of Computer Science, Edinburgh University, 1988.
20. C. H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principle of Programming Languages, Paris, January 1997*. ACM Press, 1997.
21. M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning, St. Petersburg*, volume 624 of *LNCS*, pages 190–201. Springer-Verlag, 1992.
22. G. D. Plotkin. Call-by-name, call-by-value and the lambda calculus. *Theoretical Computer Science*, 1:125–159, 1975.
23. A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997.
24. A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6:286–360, 1993.
25. P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. To appear, 2001.
26. A. K. Simpson and G. D. Plotkin. Complete axioms for categorical fixedpoint operators. In *Proceedings of 15th Annual Symposium on Logic and Computer Science*, 2000.
27. H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997. Also available as technical report ECS-LFCS-97-376.
28. H. Thielecke. Using a continuation twice and its implications for the expressive power of `call/cc`. *Higher-Order and Symbolic Computation*, 12(1):47–74, 1999.