

A Proof Theoretical Account of Continuation Passing Style

Ichiro Ogata

Information Technology Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 JAPAN
i.ogata@aist.go.jp http://staff.aist.go.jp/i.ogata
phone. +81 298 61 5906 fax. +81 298 61 5909

Abstract. We study the “classical proofs as programs” paradigm in Call-By-Value (CBV) setting. Specifically, we show the CBV normalization for **CND** (Parigot 92) can be simulated by the cut-elimination procedure for **LKQ** (Danos-Joinet-Schellinx 93), namely the q-protocol. We use a proof-term assignment system to prove this fact. The term calculus for **CND** we use follows Parigot’s $\lambda\mu$ -calculus and is closely related to Ong-Stewart’s (Ong-Stewart 97). A new term calculus for **LKQ** is presented as a variant of λ -calculus with a let-construct. We then define a translation from **CND** into **LKQ** and prove simulation theorem. We also show the translation we use can be thought of a familiar CBV CPS-translation without translation on types.

keywords: Classical Logic, Classical Natural Deduction, **LKQ**, Call-By-Value, CPS-translation, classical proof theory.

1 Introduction

Classical Natural Deduction: It has long been thought that classical logic cannot be put to use for computational purposes. It is because, in general, the normalization procedure for the the proof of classical logic has a lot of critical pairs. Hence classical logic in general, as a rewrite system, is not Church-Rosser (CR). Church’s λ -calculus is widely accepted as the logical basis of functional programming. It is also well known that typed λ -calculus has Curry-Howard correspondence with natural deduction-style intuitionistic logic. Parigot extends this idea to a classical logic. Its computational interpretation is a natural extension of Call-By-Name (CBN) λ -calculus, called $\lambda\mu$ -calculus.

We develop a CBV variant of Parigot’s $\lambda\mu$ -calculus, namely $\lambda\mu_V$. Our $\lambda\mu_V$ is a general CBV language in the sense that one can simulate CBV λ -calculus with continuations (catch/throw) and exception handling (handle/raise) by our $\lambda\mu_V$. However these investigations are not new, since Ong and Stewart describe these in [16]. What we do here is to improve Ong-Stewart’s CBV $\lambda\mu$ -calculus to be compatible to the q-protocol. Specifically, we introduce only two symmetric reduction rules, namely β_V and ζ_V . λ -variables are substituted by **values** in β_V ,

while μ -names are substituted by **evaluation contexts** in ζ_V . That is, both values and evaluation contexts are first class (i.e., functional) objects. Moreover, with the help of this refinement, we also get a simple, intuitive proof of the CR-property by using standard parallel reduction method[24]. Our $\lambda\mu_V$, being different from Ong-Stewart's, does not contain CBV λ -calculus as a sub-calculus. Instead we have a simple encoding of CBV λ -calculus into our $\lambda\mu_V$ which is given elsewhere. Also our $\lambda\mu_V$ does not model η -conversion, while Ong-Stewart's does. This is because η -conversion seems to have no relevancy to the cut-elimination procedure.

LKQ: **LKQ** is a variant of Gentzen's sequent-style classical logic **LK**. Gentzen's *Hauptsatz* states that any **LK** proof with cuts can be reduced into a cut-free proof. Numerous cut-elimination procedures have been described in the literature. However, all of them have problems in common — they intrinsically have non-deterministic choices which lead us to critical pairs. **LKQ** is an answer. It is equipped with SN and CR cut-elimination procedure, called the q-protocol[5]. CR property is recovered by adding some restrictions on logical rules of **LK**. Despite of these restrictions, soundness and completeness w.r.t. classical provability is still retained. What we do here is to develop a term calculus for **LKQ**, namely $\overline{\lambda\mu}_{\text{let}}$. The set of reduction rules of $\overline{\lambda\mu}_{\text{let}}$ are set to be compatible to the q-protocol. It is presented as a classically typed λ -calculus with a let-construct.

Translation and Simulation: The main result of this paper is the simulation theorem; the CBV normalization procedure for Classical Natural Deduction (**CND**)[17] is shown to be simulated by the q-protocol. First, we define a translation from $\lambda\mu_V$ to $\overline{\lambda\mu}_{\text{let}}$. This translation can be considered as a variant of CBV CPS-translation *without* translation on types. In previously known CPS-translations, there are so called *administrative reductions* in the target language[19]. That is, some superfluous redexes are produced by the translation, and they have nothing to do with any redexes in the source language. We develop a neat translation such that unnecessary redexes are not produced. This leads us to establish a quite tight reduction relation between normalization and cut-elimination.

We can recover the Hofmann-Streicher-style [9] by considering an intuitionistic decoration of **LKQ** (i.e., an embedding of classical types into intuitionistic types). With the help of this translation, CBV $\lambda\mu_V$ is shown to be simulated by λ -calculus with CBN strategy. This exactly is the Plotkin's CPS simulation theorem [19]. Our CPS-translation is general in the sense that we can also recover Plotkin-style [19] and Fischer-style [6] CPS-translations by considering different intuitionistic decorations of **LKQ**. Furthermore, considering the linear decoration of **LKQ**, one can even use a proof of linear logic (with its cut-elimination procedure) as a target language of CPS-translation.

Since Griffin's pioneering work[8], it is known that there is a connection between the CPS and classical logic. Our work directly relates the classical logic (**CND** and **LKQ**) and the CPS.

Related Works: First, we briefly summarize our previous works. In [12], we show that an intuitionistic decoration of **LKT** (**LKQ**) can be thought of a target language of a Plotkin-style CBN (CBV, respectively) CPS. In [13], we choose Parigot’s $\lambda\mu$ -calculus as a source language; and we show the normalization of $\lambda\mu$ -calculus can be simulated by the t-protocol of **LKT**. In [14], the source language is λ -calculus with various non-local exit operators; and the target is an intuitionistic decoration of **LKQ**.

Curien and Herbelin develops a term calculus for **LKQ** which they call $\bar{\lambda}\bar{\mu}$ -calculus[3]. However, they only establish the isomorphism between the (intuitionistic fragment of) $\bar{\lambda}\bar{\mu}$ -calculus and the CBV λ -calculus with a let-construct (which they call $\lambda\bar{\mu}$ -calculus). Instead, we establish a direct Curry-Howard isomorphism between *full* **LKQ** and λ -calculus with a let-construct ($\bar{\lambda}\mu_{\text{let}}$). What is new here is that we extend the λ -calculus with a let-construct to classically typed (i.e., typed by **LKQ** as a classical logic) language. As far as we know, the correspondence between (intuitionistic decorations of) **LKT** and **LKQ** and the target language of CPS-translation first appeared in [12] and [13].

Building a term calculus on Gentzen’s sequent-style intuitionistic logic (i.e., **LJ**) is investigated by Zucker[25], Pottinger[20], and recently Mints[10]. We extend these to classical case, by using **LKQ** and the q-protocol. In fact, we can present a term calculus on **LK** (using our $\bar{\lambda}\mu_{\text{let}}$) which will be given elsewhere.

As for the relation between classical logic and CPS-translation, Murthy’s pioneering work is also noteworthy[11]. He shows that one can interpret Girard’s **LC**[7] (of which the negative fragment is **LKT**) by means of CPS with “intuitionistic extract” method. We conclude how our approach confronts to the Selinger’s work on co-control category[23] in the last section.

2 Background

In this section, we recall necessary definitions and notations for our presentation. Basically, we follow the notion of *indexed logical system* according to Parigot[18]. It first appeared in Zucker’s pioneering work[25].

2.1 Indexed Logical Systems

In the following, we use the word **derivation**, instead of *proof*, for a tree of derivation rules. **Formulas** are that of second order propositional logic constructed from \rightarrow . We use A, B, C, \dots for formulas and X, Y, \dots for propositional variables.

We use the notion of **Indexed formula**. In order to relate a term and a derivation, we need some way to specify formulas. For this, we change the notion of context. We interpret a context as a set of indexed formulas. An indexed formula is an ordered pair of a formula and an index. We assume there are denumerably many λ -**indices** (resp. μ -**indices**) ranged over by $x, y, z \dots$ (resp. $\alpha, \beta, \gamma, \dots$). We write an indexed formula (A, x) as A^x and (A, α) as A^α .

As we interpret contexts as sets, occurrences of formulas with the same index are automatically contracted. One can interpret this that binary rules are always followed by appropriate explicit contractions which rename the indices to the same name. We also interpret axiom rules contain appropriate weakenings as contexts. Therefore, we say structural rules are implicit in our formulation of classical logic.

Initial index is an index which appears for the first time in whole derivation. We assume all initial indices are distinct unless they are truly related (i.e., subject of further implicit contraction). This is possible, by introducing the “concatenation” on indices on every binary rules. See Zucker[25]. We use this convention because we’d like to skirt off the fruitless discussion about capture avoiding substitution.

2.2 Classical Natural Deduction

As the name implies, **CND** is a natural deduction system (i.e., formation rules take the form of introduction ($\rightarrow\mathcal{I}$) and elimination ($\rightarrow\mathcal{E}$)) but formulated in a Gentzen-style sequent. Sequents of **CND** are of the form: $\Gamma \Rightarrow \Delta, \Xi$, where \Rightarrow is the **entailment sign** of the calculus. Γ is a λ -**context** which is a *set* of λ -indexed formulas. Similarly, Δ is a μ -**context** which is a set of μ -indexed formulas. Ξ denotes exactly one un-indexed formula. Comma means taking union as sets. Thus, the set $\Gamma_0 \cup \Gamma_1$ is denoted by “ Γ_0, Γ_1 ” and $\{A^x\} \cup \Gamma$ by “ A^x, Γ ”.

2.3 Gentzen’s Sequent-Style Constructive Classical Logic

LKQ is a variant of Gentzen’s sequent-style classical logic. That is, formation rules take the form of left and right introduction. Sequents of **LKQ** are of the form: $\Gamma \Rightarrow \Delta; \Pi$, where Π denotes at most one un-indexed formula. The right most place where Π lives is called **stoup**. Roughly speaking, stoup is a place where newly created formula goes. Please pay attention to the fact that application of structural rules are restricted within Γ and Δ . Specifically Π can not be introduced by weakening. We use $\Gamma \Rightarrow \Delta; \emptyset$ to indicate that the stoup is empty.

2.4 Multiplicative Rules

In both **CND** and **LKQ**, we only handle **multiplicative** rules. That is, λ -contexts (μ -contexts) in the conclusion is the union of λ -contexts (μ -contexts respectively) in the premises. For example, in $L\rightarrow$ of **LKQ**:

$$\frac{\Gamma_0 \Rightarrow \Delta_0; A \quad B^y, \Gamma_1 \Rightarrow \Delta_1; \emptyset}{(A \rightarrow B)^z, \Gamma_0, \Gamma_1 \Rightarrow \Delta_0, \Delta_1; \emptyset} L\rightarrow$$

Hereafter, for readability, we only write **active** and **main** formulas, and omit contexts as follows:

$$\frac{\Rightarrow; A \quad B^y \Rightarrow; \emptyset}{(A \rightarrow B)^z \Rightarrow; \emptyset} L\rightarrow$$

In the above, we say A and B^y are **active** formulas, while $(A \rightarrow B)^z$ is a **main** formula.

2.5 Restrictions for Propositional Variables

Usual restrictions for propositional variables apply. For example, in case of introduction of \forall in **CND**:

$$\frac{\Gamma \Rightarrow \Delta, A[X := Y]_{\forall^2 \mathcal{I}^*}}{\Gamma \Rightarrow \Delta, \forall X.A}$$

In the above, the propositional variable Y has no free occurrence in the contexts Γ and Δ . We use $()^*$ to indicate these restrictions.

3 Calculi for Call-By-Value Classical Natural Deduction

3.1 A Call-By-Value Calculus: $\lambda\mu_V$

In this subsection, we shall introduce a Call-By-Value $\lambda\mu$ -calculus, namely $\lambda\mu_V$. The $\lambda\mu_V$ -terms includes two sub-categories, namely **values** and **μ -renames**.

Definition 1 ($\lambda\mu_V$ -terms).

1. We define $\lambda\mu_V$ -**values**, ranged over by v , as follows:

$$\begin{array}{l|l} v := x, y, z, \dots & \lambda\text{-variables} \\ | \lambda x^A.p & \text{abstraction} \\ | \Lambda X.p & \text{universal-abstraction} \end{array}$$

2. We define $\lambda\mu_V$ - **μ -renames**, ranged over by p, q , are defined as follows:

$$p, q := \mu\alpha^A[\beta] M$$

3. We define $\lambda\mu_V$ -**terms**, ranged over by L, M, N , etc., are defined as follows:

$$\begin{array}{l|l} L, M, N := v & \text{value} \\ | p & \mu\text{-rename} \\ | MN & \text{application} \\ | MB & \text{universal-application} \end{array}$$

$\alpha, \beta, \gamma, \dots$ are called as **μ -names** and A, B, \dots are called as **types**. Application associates to left, i.e., we write “ LMN ” instead of “ $(LM)N$ ”. The rules of **term assignment judgment** are displayed in Table 3.1. In the table, λ -variables and μ -names are identified with λ -indices and μ -indices respectively. Moreover types are identified with formulas and type variables are identified with propositional variables. Observe that a body of abstractions must be a μ -rename. This prevents us to include CBV λ -calculus as a sub-calculus of $\lambda\mu_V$. However we have an encoding of CBV λ -calculus into our $\lambda\mu_V$.

$\frac{}{x : A^x \Rightarrow A} \text{Ax}$	$\frac{N : \Rightarrow A, \Delta}{\mu\beta^B.[\alpha] N : \Rightarrow B, ((A^\alpha, \Delta) \setminus B^\beta)} \text{rename}$
$\frac{p : A^x \Rightarrow B}{\lambda x^A.p : \Rightarrow A \rightarrow B} \rightarrow \mathcal{I}$	$\frac{M : \Rightarrow A \rightarrow B \quad N : \Rightarrow A}{MN : \Rightarrow B} \rightarrow \mathcal{E}$
$\frac{p[X := Y] : \Rightarrow A[X := Y]}{\lambda X.p : \Rightarrow \forall X.A} \forall^2 \mathcal{I}^*$	$\frac{M : \Rightarrow \forall X.A}{MB : \Rightarrow A[X := B]} \forall^2 \mathcal{E}$

Table 1. $\lambda\mu_{\mathbf{v}}$ -term Assignment for **CND**

As we show above, we use Church-style typing (i.e., every variable have types as superscripts). We will occasionally abbreviate types because in most case types of variables are clear from the context. The set of **free μ -names** of $\lambda\mu_{\mathbf{v}}$ -term M , denoted by $\text{FN}(M)$, is defined as follows: $\text{FN}(x) = \emptyset$, $\text{FN}(\lambda x.p) = \text{FN}(\lambda X.p) = \text{FN}(p)$, $\text{FN}(\mu\alpha.[\beta] M) = (\text{FN}(M) \cup \{\beta\}) \setminus \{\alpha\}$, $\text{FN}(MB) = \text{FN}(M)$, $\text{FN}(MN) = \text{FN}(M) \cup \text{FN}(N)$. The set of **free λ -variables**, denoted by $\text{FV}(M)$, is defined in the same way with λ -calculus.

Definition 2 (CBV Singular Evaluation Context). *We define CBV singular evaluation contexts, ranged over by K , as follows: $K := [-]N \mid [-]B \mid v[-]$.*

Definition 3 (CBV Evaluation Context). *We define CBV evaluation contexts, ranged over by E , as follows: $E := [-] \mid EN \mid EB \mid vE$. Note that a CBV evaluation context E has exactly one hole.*

CBV evaluation context E can be defined as a sequence of singular evaluation contexts such as $E = K_0 \circ K_1 \circ \dots \circ K_{n-1}$, where \circ is the context composition which is defined by $(K_0 \circ K_1)[-] = K_0[K_1[-]]$. The composition is associative. Note that one can *parse* every μ -rename in the form of: $\mu\alpha.[\beta] E[N]$, where N is either value: v or μ -rename: p .

Definition 4 ($\lambda\mu_{\mathbf{v}}$ as a reduction system). *The reduction relation $\longrightarrow_{\lambda\mu_{\mathbf{v}}}$ of $\lambda\mu_{\mathbf{v}}$, viewed as a rewrite system, is defined to be the compatible (i.e. contextual) closure of the notion of reduction defined by three redex rules, namely, $\beta_{\mathbf{v}}$, $\zeta_{\mathbf{v}}$ and polymorphic. $\longrightarrow_{\lambda\mu_{\mathbf{v}}}$ is the reflexive, transitive closure of $\longrightarrow_{\lambda\mu_{\mathbf{v}}}$.*

$(\beta_{\mathbf{v}})$	$(\lambda x.L)v$	$\longrightarrow_{\lambda\mu_{\mathbf{v}}} L[x := v]$
$(\zeta_{\mathbf{v}})$	$\mu\alpha.[\beta] E[\mu\gamma.[\delta] M]$	$\longrightarrow_{\lambda\mu_{\mathbf{v}}} \mu\alpha.([\delta] M)[\gamma := [\beta] E[-]]$
(polymorphic)	$(\lambda X.L)B$	$\longrightarrow_{\lambda\mu_{\mathbf{v}}} L[X := B]$

We shall refer to the above as $\lambda\mu_{\mathbf{v}}$ -**redex rules** and terms on the left-hand-side of the redex rules as $\lambda\mu_{\mathbf{v}}$ -**redexes**. Three kinds of substitutions can be

distinguished in $\lambda\mu_V$. The first **λ -substitution** of the form: $L [x := v]$ means the standard substitution as meta-operation. It is the result of substituting v for the free occurrences of x (of the same type as v) in L . The second **μ -substitution** of the form: $M [\gamma := [\beta] E[-]]$ means “in M , replace all subterms of the form $[\gamma] L$ by the term $[\beta] E[L]$ ”. The third **type-substitution** of the form: $M [X := B]$ means “in M , replace all occurrences of the type variable X by the type B ”.

Remark 1 (evaluation context). Traditionally, evaluation contexts are devised so that every non-normal closed term M can be written uniquely as $E[R]$, where R is a redex. It is used to extract a unique redex according to evaluation strategy. Bierman develops operational theory for $\lambda\mu$ -calculus using this idea[2]. Instead, we use the notion of evaluation context to uniquely define a ζ -redex in every μ -rename. In particular we do not specify the order of reduction. Every β -, ζ - and polymorphic redexes can be reduced in any order. Clearly the Church-Rosser property is only meaningful in this setting. Our point here is that the concept of CBV is *not* build on the reduction system as an evaluation order.

3.2 Relation to Ong-Stewart’s CBV $\lambda\mu$ -calculus

Now we demonstrate how our $\lambda\mu_V$ is different from Ong-Stewart’s CBV $\lambda\mu$ -calculus[15, 16]. In a word, we pack $n + 1$ length of “reduction sequence” into single reduction. Consider our general ζ_V -redex: $\mu\alpha.[\beta] E[\mu\gamma.[\delta] M]$. We assume E consists of n -fold singular contexts, i.e., $E = K_0 \circ K_1 \circ \dots \circ K_{n-1}$. In the style of Ong-Stewart’s ζ -reduction rule, the reduction proceeds as follows:

$$\begin{aligned}
& K_{n-1}[\mu\gamma.[\delta] M] \rightarrow \mu\beta_{n-1}.[\delta] M [\gamma := [\beta_{n-1}] K_{n-1}[-]] \\
& K_{n-2}[\mu\beta_{n-1}.[\delta] M [\gamma := [\beta_{n-1}] K_{n-1}[-]]] \rightarrow \\
& \quad \mu\beta_{n-2}.[\delta] M [\gamma := [\beta_{n-2}] (K_{n-2} \circ K_{n-1})[-]] \\
& \quad \vdots \\
& K_0[\mu\beta_1.[\delta] M [\gamma := [\beta_1] (K_1 \circ \dots \circ K_{n-1})[-]]] \rightarrow \mu\beta_0.[\delta] M [\gamma := [\beta_0] E[-]] \\
& \quad \mu\alpha.[\beta] (\mu\beta_0.[\delta] M [\gamma := [\beta_0] E[-]]) \rightarrow \mu\alpha.[\delta] M [\gamma := [\beta] E[-]]
\end{aligned}$$

The last reduction rule is called as μ - β reduction. Observe that each ζ reduction *always* produces another ζ (or μ - β) redex. Hence there always is a $n + 1$ length of **sequential reduction**, where n is the size of E . Because of this one cannot apply the standard Tait-Martin-Löf-Takahashi’s parallel reduction method for Church-Rosser property[24] to Ong-Stewart’s $\lambda\mu$ -calculus. This is simply because the **diamond property** for parallel reduction does not hold in this situation. This phenomenon was first observed by Baba et al.[1] in slightly different settings. Full proof of CR-property for our $\lambda\mu_V$ will be given elsewhere.

Remark 2. Our $\lambda\mu_V$ does not model η reduction, while Ong-Stewart’s $\lambda\mu$ -calculus does(i.e., it has η and μ - η reductions).

$\frac{}{x: A^x \Rightarrow; A} \text{Ax}$	$\frac{V: \Rightarrow; A}{[\alpha]V: \Rightarrow A^\alpha; \emptyset} \text{D}$
$\frac{V: \Rightarrow; A \quad T: A^x \Rightarrow; \emptyset}{\text{let } x = V \text{ in } T: \Rightarrow; \emptyset} \text{tail}$	$\frac{S: \Rightarrow A^\alpha; \emptyset \quad T: A^x \Rightarrow; \emptyset}{\text{let } x = \bar{\mu}\alpha.S \text{ in } T: \Rightarrow; \emptyset} \text{mid}$
$\frac{V: \Rightarrow; A \quad U: B^y \Rightarrow; \emptyset}{\text{let } y = zV \text{ in } U: (A \rightarrow B)^z \Rightarrow; \emptyset} \text{L}\rightarrow$	$\frac{T: A^x \Rightarrow B^\beta; \emptyset}{\bar{\lambda}x^A.\bar{\mu}\beta^B.T: \Rightarrow; A \rightarrow B} \text{R}\rightarrow$
$\frac{U: (A[X := B])^x \Rightarrow; \emptyset}{\text{let } x = zB \text{ in } U: (\forall X.A)^z \Rightarrow; \emptyset} \text{L}\forall^2$	$\frac{T[X := Y]: \Rightarrow (A[X := Y])^\alpha; \emptyset}{\bar{\Lambda}X.\bar{\mu}\alpha^A.T: \Rightarrow; \forall X.A} \text{R}\forall^{2*}$

Table 2. $\bar{\lambda}\bar{\mu}_{\text{let}}$ -term Assignment for **LKQ**

4 Calculi for Gentzen's sequent-style classical logic: **LKQ**

In this section, we introduce $\bar{\lambda}\bar{\mu}_{\text{let}}$, a variant of λ -calculus with a let-construct, as a term calculus for **LKQ**. The $\bar{\lambda}\bar{\mu}_{\text{let}}$ -terms are classified exclusively in the three categories, namely **values**, **contexts** and $\bar{\mu}$ -**abstractions**.

Definition 5 ($\bar{\lambda}\bar{\mu}_{\text{let}}$ -terms).

1. $\bar{\lambda}\bar{\mu}_{\text{let}}$ -values, ranged over by V , are defined as follows:

$$\begin{array}{ll}
 V := x, y, z, \dots & \lambda\text{-variables} \\
 | \bar{\lambda}x^A.P & \text{right-term} \\
 | \bar{\Lambda}X.P & \text{universal-right-term}
 \end{array}$$

2. $\bar{\lambda}\bar{\mu}_{\text{let}}$ -contexts, ranged over by S, T, U , etc., are defined as follows:

$$\begin{array}{ll}
 S, T, U := [\alpha]V & \text{derelict-term} \\
 | \text{let } x = V \text{ in } U & \text{tail-term} \\
 | \text{let } x = P \text{ in } T & \text{mid-term} \\
 | \text{let } y = zV \text{ in } U & \text{left-term} \\
 | \text{let } x = zB \text{ in } T & \text{universal-left-term}
 \end{array}$$

3. $\bar{\lambda}\bar{\mu}_{\text{let}}$ - $\bar{\mu}$ -abstractions, ranged over by P , are defined as follows:

$$P := \bar{\mu}\alpha.S$$

The rules of **term assignment judgment** are displayed in Table 4. Observe that contexts are assigned to **LKQ**-sequents with empty stoup. On the other hand, values are assigned to sequents which have a formula in the stoup. $\bar{\mu}$ -abstractions are not assigned to any **LKQ**-sequents; they only appear as a sub-terms of values or contexts. In the table, the letter L/R stands for **Left** and **Right** introduction, and D for **Dereliction**.

We have two additional term assignment judgment rules which allows us to express intermediate state between S2-step and L-step of the q-protocol.

$$\frac{V : \Rightarrow ; A \quad U : B^y \Rightarrow ; \emptyset \quad T : A^x \Rightarrow B^\beta ; \emptyset}{\text{let } y = (\overline{\lambda}x^A.\overline{\mu}\beta^B.T)V \text{ in } U : \Rightarrow ; \emptyset} \beta_V$$

$$\frac{T[X := Y] : \Rightarrow (A[X := Y])^\alpha ; \emptyset \quad U : (A[X := B])^x \Rightarrow ; \emptyset}{\text{let } x = (\overline{\lambda}X.\overline{\mu}\alpha^A.T)B \text{ in } U : \Rightarrow ; \emptyset} \beta_{\text{univ}}$$

This idea first appeared in [22] in J.E. Santo's study about intuitionistic fragment of **LKT**.

Definition 6 ($\overline{\lambda\mu}_{\text{let}}$ as a reduction system). *The reduction relation $\longrightarrow_{\overline{\lambda\mu}_{\text{let}}}$ of $\overline{\lambda\mu}_{\text{let}}$, viewed as a rewrite system, is defined to be the compatible (i.e. contextual) closure of the notion of reduction defined by four redex rules, namely, S1, S2, L \rightarrow and L \forall . $\longrightarrow_{\overline{\lambda\mu}_{\text{let}}}$ is the reflexive, transitive closure of $\longrightarrow_{\overline{\lambda\mu}_{\text{let}}}$. We use $M \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} N$ to mean that $M \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} L \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} N$ holds for some L .*

$$\begin{aligned} \text{(S1)} \quad & \text{let } x = \overline{\mu}\alpha.S \text{ in } T \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} S [\alpha := (\text{let } x = _ \text{ in } T)] \\ \text{(S2)} \quad & \text{let } x = V \text{ in } S \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} S [x := V] \\ \text{(L}\rightarrow\text{)} \quad & (\overline{\lambda}x^A.\overline{\mu}\beta^B.T)V \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \overline{\mu}\beta^B.(\text{let } x = V \text{ in } T) \\ \text{(L}\forall\text{)} \quad & (\overline{\lambda}X.\overline{\mu}\alpha^A.T)B \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \overline{\mu}\alpha^A.T [X := B] \end{aligned}$$

These redex rules are set to be compatible to the reduction step of the q-protocol (i.e., S1-step, S2-step and L-step). Three kinds of substitutions can be distinguished in $\overline{\lambda\mu}_{\text{let}}$. $\overline{\lambda}$ -**substitution** of the form: $T [x := V]$ means the standard substitution as meta-operation. Note that one can only substitute λ -variable for $\overline{\lambda\mu}_{\text{let}}$ -value, like β_V of $\lambda\mu_V$. $\overline{\mu}$ -**substitution** of the form: $U [\alpha := (\text{let } x = _ \text{ in } T)]$ means “in U , replace all subterms of the form $[\alpha] V$ by the term $(\text{let } x = V \text{ in } T)$ ”. The third **type-substitution** of the form: $M [X := B]$ means the standard one.

Remark 3 (**LKQ**). We refer to [5] for “technical terms” in this remark. Strictly speaking, our presentation of **LKQ** is a “q-fragment of **LK** q ” where all formulas are coloured q . That is, all formulas in the stoup of **LKQ** have “flat meta-interspaces”. This constraint can be rephrased as follows: the main formula introduced in the stoup by Ax or L \rightarrow must be an active formula of the previous derivation rule. Of course, by the “stability lemma”, this property is preserved under the q-protocol. This definition is slightly different from the one presented in earlier literature[4].

5 Translation

5.1 Simulation of $\lambda\mu_V$ by $\overline{\lambda\mu}_{\text{let}}$

First, we define the translation from $\lambda\mu_V$ -terms to $\overline{\lambda\mu}_{\text{let}}$ -terms. Clearly, an endsequent of **CND**: $\Gamma \Rightarrow B, \Delta$ corresponds to an endsequent of **LKQ**: $\Gamma \Rightarrow B, \Delta; \emptyset$. The latter is not a proper **LKQ** sequent. It is introduced by the extra non-logical derivation rule, namely $\overline{\mu}$ -abstraction. At the same time, $\lambda\mu_V$ -term must be μ -rename in order to specify the μ -name in $\overline{\mu}$ -abstraction. So the last derivation rules must be μ -rename and $\overline{\mu}$ -abstraction respectively. That is, one can only define the translation from $\lambda\mu_V$ - μ -renames to $\overline{\lambda\mu}_{\text{let}}$ - $\overline{\mu}$ -abstractions. We can assume this without loss of generality, since we always have $\mu\alpha.[\alpha] M$ ($\alpha \notin \text{FN}(M)$) for arbitrary $\lambda\mu_V$ -term M . The situation is illustrated as follows:

$$\frac{N: \Rightarrow A, \Delta}{\mu\beta^B.[\alpha] N: \Rightarrow B, ((A^\alpha, \Delta) \setminus B^\beta)} \text{rename} \quad \frac{S: \Gamma \Rightarrow B^\beta, \Delta; \emptyset}{\overline{\mu}\beta^B.S: \Gamma \Rightarrow B, \Delta; \emptyset} \overline{\mu}\text{-abstraction}$$

Definition 7 (Translation from $\lambda\mu_V$ to $\overline{\lambda\mu}_{\text{let}}$).

1. The translation $\overline{(-)}$, $\lambda\mu_V$ - μ -renames $\mapsto \overline{\lambda\mu}_{\text{let}}$ - $\overline{\mu}$ -abstractions is defined as follows:

$$\overline{\mu\beta^B.[\alpha] N} = \overline{\mu}\beta^B.(N : \text{let } x = _ \text{ in } [\alpha] x)$$

2. The infix operator $;$, $\lambda\mu_V$ -terms $\times \overline{\lambda\mu}_{\text{let}}$ -contexts $\mapsto \overline{\lambda\mu}_{\text{let}}$ -contexts is defined as follows:

$$\begin{aligned} v : \text{let } y = _ \text{ in } S &= S [y := \Psi(v)] \\ \mu\alpha.[\beta] M : \text{let } y = _ \text{ in } S &= \text{let } y = \mu\alpha.[\beta] M \text{ in } S \\ MN : \text{let } y = _ \text{ in } S &= M : \text{let } z = _ \text{ in } (N : \text{let } x = _ \text{ in } (\text{let } y = zx \text{ in } S)) \\ MB : \text{let } y = _ \text{ in } S &= M : \text{let } z = _ \text{ in } (\text{let } y = zB \text{ in } S) \end{aligned}$$

3. An auxiliary function Ψ , $\lambda\mu_V$ -values $\mapsto \overline{\lambda\mu}_{\text{let}}$ -values, is defined as follows:

$$\Psi(x) = x; \quad \Psi(\lambda x.p) = \overline{\lambda}x.\overline{p}; \quad \Psi(\Lambda X.p) = \overline{\Lambda}X.\overline{p}$$

Proof theoretically, this translation is based on Prawitz's observation to simulate natural deduction-style by Gentzen's sequent style logic[21]. For example, the application pq can be written in **LKQ** as follows:

$$\frac{\frac{\frac{x: A^x \Rightarrow; A \quad [\beta] y: B^y \Rightarrow B^\beta; \emptyset}{s: \Rightarrow A^\alpha; \emptyset \text{ let } y = zx \text{ in } [\beta] y: A^x, (A \rightarrow B)^z \Rightarrow B^\beta; \emptyset} \text{L}\rightarrow}{u: \Rightarrow (A \rightarrow B)^\gamma; \emptyset \text{ let } x = \overline{\mu}\alpha^A.S \text{ in } (\text{let } y = zx \text{ in } [\beta] y): (A \rightarrow B)^z \Rightarrow B^\beta; \emptyset} \text{mid}}{\text{let } z = \overline{\mu}\gamma^{A \rightarrow B}.U \text{ in } (\text{let } x = \overline{\mu}\alpha^A.S \text{ in } (\text{let } y = zx \text{ in } [\beta] y)): \Rightarrow B^\beta; \emptyset} \text{mid}$$

where $\overline{p} = \overline{\mu}\gamma.U$ and $\overline{q} = \overline{\mu}\alpha.S$.

Remark 4. A $\overline{\lambda\mu}_{\text{let}}$ - $\overline{\mu}$ -abstraction: \overline{p} (for some $\lambda\mu_V$ - μ -rename p) contains no S2 redex. Instead it contains $\text{L}\rightarrow$, and/or $\text{L}\vee$ redexes.

Remark 5. In the above derivation, the order of two mid-cuts does matter. This situation is called the “**q/t dilemma**” in [5]; implication is the dilemmatic logical operator in the q-protocol. To say that the order matters is just to say we have already made a choice. Of course, another choice is possible. See subsection 5.2.

Our main theorem below can be seen as a proof theoretical explanation for Plotkin’s CPS simulation theorem.

Theorem 1. *If $p \longrightarrow_{\lambda\mu\nu} q$ then $\bar{p} \longrightarrow_{\lambda\mu\bar{\nu}} \bar{q}$*

We devote the rest of the subsection for this proof.

Proposition 1 (λ -substitution and $\bar{\lambda}$ -substitution).

$$\bar{p}[x := \Psi(v)] = \overline{p[x := v]}$$

Proof. by induction on L .

Proposition 2 (An Image of an Evaluation Context). *An image of μ -rename: $\mu\alpha.[\beta]E[M]$ always has the form of: $\bar{\mu}\alpha.(M : \text{let } y = _ \text{ in } S_{[\beta]E})$.*

Proof. By induction on the construction of evaluation context.

Proposition 3 (μ -substitution and $\bar{\mu}$ -substitution). *One only uses S1 in the following reduction relation.*

- (1) $\bar{p}[\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}] \longrightarrow_{\lambda\mu\bar{\nu}} \overline{p[\gamma := [\beta]E[-]]}$
- (2) $(M : \text{let } y = _ \text{ in } S) [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}]$
 $\longrightarrow_{\lambda\mu\bar{\nu}} (M [\gamma := [\beta]E[-]] : (\text{let } y = _ \text{ in } S [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}]))$

Proof. By mutual induction on p and M .

(1) Assume $p = \mu\alpha.[\gamma]N$.

$$\begin{aligned} & \overline{\mu\alpha.[\gamma]N} [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}] \\ &= \bar{\mu}\alpha.(N : \text{let } y'' = _ \text{ in } [\gamma]y'') [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}] \\ &\longrightarrow_{\lambda\mu\bar{\nu}} \bar{\mu}\alpha.(N [\gamma := [\beta]E[-]] : (\text{let } y'' = _ \text{ in } [\gamma]y'') [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}]) \\ &\longrightarrow_{\lambda\mu\bar{\nu}} \bar{\mu}\alpha.(N [\gamma := [\beta]E[-]] : \text{let } y' = _ \text{ in } S_{[\beta]E}) \\ &= \overline{\mu\alpha.([\beta]E[N [\gamma := [\beta]E[-]])} \\ &= \overline{(\mu\alpha.[\gamma]N) [\gamma := [\beta]E[-]}} \end{aligned}$$

We use (2) from second to third, S1 from third to fourth.

(2) We only consider the base case: $M = \mu\alpha.[\eta]N$.

$$\begin{aligned} & (\mu\alpha.[\eta]N : \text{let } y = _ \text{ in } S) [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}] \\ &= (\text{let } y = \overline{\mu\alpha.[\eta]N} \text{ in } S) [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}] \\ &\longrightarrow_{\lambda\mu\bar{\nu}} \text{let } y = \overline{\mu\alpha.[\eta]N} [\gamma := [\beta]E[-]] \text{ in } (S [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}]) \\ &= (\mu\alpha.[\eta]N) [\gamma := [\beta]E[-]] : (\text{let } y = _ \text{ in } S [\gamma := \text{let } y' = _ \text{ in } S_{[\beta]E}]) \end{aligned}$$

We use (1) from second to third.

In the proofs below, we use the abbreviation $\overline{[\delta] L} = L : \text{let } y = _ \text{ in } [\delta] y$. With this notion, $\overline{\mu\alpha.[\delta] L} = \overline{\bar{\mu}\alpha.[\delta] L}$.

Proposition 4. *If $p \longrightarrow_{\lambda\mu\nu} q$ by β_V then $\bar{p} \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \bar{q}$. The two $\longrightarrow_{\overline{\lambda\mu}_{\text{let}}}$ are L_{\rightarrow} and $S2$ respectively.*

Proof. Assume the β_V under consideration being $(\lambda x.\mu\gamma.[\delta] L)v$ and it appears within context $\text{let } y = _ \text{ in } S$.

$$\begin{aligned}
& ((\lambda x.\mu\gamma.[\delta] L)v) : \text{let } y = _ \text{ in } S \\
&= \text{let } y = (\lambda x.\overline{\mu\gamma.[\delta] L})\Psi(v) \text{ in } S && \text{translation} \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \text{let } y = \overline{\bar{\mu}\gamma}.\overline{(\text{let } x = \Psi(v) \text{ in } [\delta] L)} \text{ in } S && L_{\rightarrow} \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \text{let } y = \overline{\mu\gamma.[\delta] L} [x := \Psi(v)] \text{ in } S && S2 \\
&= \text{let } y = \overline{\mu\gamma.[\delta] L} [x := v] \text{ in } S && \text{proposition 1} \\
&= \overline{\mu\gamma.[\delta] L} [x := v] : \text{let } y = _ \text{ in } S && \text{translation}
\end{aligned}$$

Proposition 5. *If $p \longrightarrow_{\lambda\mu\nu} q$ by ζ_V then $\bar{p} \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \bar{q}$. One only uses $S1$ in these reduction relations.*

Proof. If $p = \mu\alpha.[\beta] E[\mu\gamma.[\delta] L]$, then

$$\begin{aligned}
& \overline{\mu\alpha.[\beta] E[\mu\gamma.[\delta] L]} \\
&= \overline{\bar{\mu}\alpha}.\overline{(\mu\gamma.[\delta] L : \text{let } y' = _ \text{ in } S_{[\beta]E})} && \text{proposition 2} \\
&= \overline{\bar{\mu}\alpha}.\overline{(\text{let } y' = \overline{\bar{\mu}\gamma}.\overline{[\delta] L} \text{ in } S_{[\beta]E})} && \text{translation} \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \overline{\bar{\mu}\alpha}.\overline{([\delta] L} [\gamma := (\text{let } y' = _ \text{ in } S_{[\beta]E})])} && S1 \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \overline{\mu\alpha.[\delta] L} [\gamma := [\beta] E[-]] && \text{proposition 3 (2)}
\end{aligned}$$

Proposition 6. *If $p \longrightarrow_{\lambda\mu\nu} q$ by polymorphic then $\bar{p} \longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \bar{q}$ by L_{\forall} .*

This proof is easy, and concludes the proof of the simulation theorem.

Corollary 1. *$\lambda\mu\nu$ is Strongly Normalizable.*

Proof. Simulation theorem says that if there is an infinite reduction sequence in $\lambda\mu\nu$, then there also is in $\overline{\lambda\mu}_{\text{let}}$. This contradicts the SN property of **LKQ**.

Please note that p being normal does not mean \bar{p} being normal. Consider the normal $\lambda\mu\nu$ -term $(\lambda x.p)(zw)$. Then

$$\begin{aligned}
(\lambda x.p)(zw) : \text{let } y = _ \text{ in } S &= (\text{let } x' = zw \text{ in } (\text{let } y = (\overline{\lambda x}.\bar{p})x' \text{ in } S)) \\
&\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} (\text{let } x' = zw \text{ in } (\text{let } y = \bar{p} [x := x'] \text{ in } S))
\end{aligned}$$

That is, we can extract “hidden” redexes by translating a $\lambda\mu\nu$ - μ -rename into a $\overline{\lambda\mu}_{\text{let}}$ - $\bar{\mu}$ -abstraction. The familiar trick to avoid this obstacle was to extend the syntax of λ -calculus to include a let-construct. What is new here is that we revise and extend this syntax to the classically typed language(i.e., it is typed by **LKQ** sequents).

Claim. A familiar λ -calculus with a let-construct, as a sub-calculus of $\overline{\lambda\mu}_{\text{let}}$, is a target language of CBV CPS-translation. Complex reduction rules related to a let-construct (e.g., see [3]) can be unified into single, simple S1 reduction rule. It also is isomorphic to (a sub-calculus of) λ -calculus which is a target language of Hofmann-Streicher-style CPS-translation.

Remark 6. Prawitz's conversion sends **CND** normal derivations to cut-free **LK** derivations. However the conversion from **LK** into **LKQ**, in general, does not send cut-free **LK** derivations to cut-free **LKQ** derivations. That is why \overline{p} being non-normal in case p being normal.

5.2 There are two ways to map **CND** into **LKQ**

We choose the ζ_V -redex in the application from left-to-right(LR) order. Of course, the opposite order should be studied in its own right. This phenomena is known in the previous study of CPS; the CBV right-to-left(RL) evaluation method. This kind of CPS-translation was shown, for example, by Murthy[11]. One can adopt the RL evaluation method in our $\lambda\mu_V$. For this, we first modify the evaluation context as follows:

$$E := [-] \mid ME \mid Ev$$

This modification leads us the RL version of our $\lambda\mu_V$. Then, we modify the translation (in order to keep the simulation theorem) as follows:

$$MN:\text{let } y = _ \text{ in } S = N:\text{let } x = _ \text{ in } (M:\text{let } z = _ \text{ in } (\text{let } y = zx \text{ in } S))$$

Danos-Joinet-Schellinx's theory give a proof theoretical explanation to this phenomenon – they say there are two ways to map **LK** derivations to **LKQ** derivations. However, Selinger seems to overlook this in his paper[23].

6 Conclusions and Further Directions

We formulate second-order Call-By-Value $\lambda\mu$ -calculus as a yet another term calculus for Parigot's Classical Natural Deduction. We show it is Church-Rosser and Strongly Normalizable. We also show that the translation from $\lambda\mu_V$ to $\overline{\lambda\mu}_{\text{let}}$ can be thought of a proof theoretical counterpart of a familiar CPS-translation.

Our proof theoretical approach confronts semantical work in some point. Actually there are some advantages of using proof theory as a syntax of the calculus. Recall that the SN-property is proved as a corollary of SN-property of **LKQ**. We also know the class of functions representable in our second-order $\lambda\mu_V$; it exactly is the class of provably total functions in second-order Peano Arithmetic **PA**₂(i.e., Π_2^0 statements). Since our $\lambda\mu_V$ can encode Girard's system F, so it includes, at least, all provably total functions in **PA**₂. At the same time, the functions representable in second-order **LKQ** are exactly the provably total functions in **PA**₂. This fact can also be understood from the fact that the intuitionistic decoration of **LKQ** can be simulated by system F.

In Selinger's co-control category, the two inputs of the application map are connected via a pretensor \otimes . This amounts to say that the order of compositions of morphisms does matter. Composition of morphisms corresponds to cut-elimination in proof theory. Hence this means the order of two-cuts matters in implication elimination. If we made a choice of morphisms in co-control category such that every \otimes are bifunctorial, one gets a sub co-control category called the “center” of the category. On the other hand, we have made a choice (LR or RL) in order to map **CND** into **LKQ**. This observation of close resemblance between category theory and proof theory deserves further study.

Our simulation theorem says that **CND** and **LKQ** share denotation under specific reduction rules and translation. Also **LKQ** has its own denotational semantics which is invariant under the q-protocol. Specifically **LKQ** inherits the denotation in linear logic's coherent space semantics. It is shown by considering linear decoration method. Moreover, through intuitionistic decoration, we also know that one can map a center of co-control category to (sub) cartesian-closed category. Obviously, the relation between the semantics of **LKQ** and the center of the co-control category should be investigated in future work. Our conjecture is that **LKQ** is an internal language of the center of the co-control category.

References

1. K. Baba, S. Hirokawa, and K. Fujita. Parallel reduction in type-free $\lambda\mu$ -calculus. *Electronic Notes in Theoretical Computer Science*, 42, 2001.
2. G.M. Bierman. A computational interpretation of the $\lambda\mu$ -calculus. In *Proceedings of Symposium on Mathematical Foundations of Computer Science 98*, pages 336–345. Springer-Verlag LNCS 1450, August 1998.
3. P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of ICFP*. World Scientific, September 2000.
4. Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Sequent calculi for second order logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 211–224. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.
5. Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic: linear logic. *Journal of Symbolic Logic*, 62(3), September 1997.
6. Michael J. Fischer. Lambda-calculus schemata. *Lisp and Symbolic Computation*, 6(3/4):259–287, November 1993.
7. Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991.
8. Timothy Griffin. A formulae-as-types notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Francisco, California, January 1990.
9. Martin Hofmann and Thomas Streicher. Continuation models are universal for $\lambda\mu$ -calculus. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*, June 1997.
10. Grigori Mints. Normal forms for sequent derivations. In Piergiorgio Odifreddi, editor, *Kreiseliana – About and Around George Kreisel*. A K Peters Ltd., March 1996.

11. Chetan R. Murthy. A computational analysis of Girard's translation and LC. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 90–101, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press.
12. Ichiro Ogata. Cut elimination for classical proofs as continuation passing style computation. In *Proceedings of the Asian Computing Science Conference 98*, pages 61–78, Manila, Philippines, December 1998. Springer-Verlag LNCS 1538.
13. Ichiro Ogata. Gentzen-style classical proofs as $\lambda\mu$ -terms. In *Proceedings of the Asian Computing Science Conference 99*, pages 266–280, Phuket, Thailand, December 1999. Springer-Verlag LNCS 1742.
14. Ichiro Ogata. Constructive classical logic as cps-calculus. *International Journal of Foundations of Computer Science*, 11(1):89–112, March 2000.
15. C.-H. L. Ong. A semantic view of classical proofs: type-theoretic, categorical, and denotational characterizations (preliminary extended abstract). In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 230–241, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.
16. C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 215–227, Paris, France, 15–17 January 1997.
17. Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *Proc. of LPAR'92*, pages 190–201. Springer-Verlag LNCS 624, 1992.
18. Michel Parigot. Strong normalization for second order classical natural deduction. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 39–46, Montreal, Canada, 19–23 June 1993. IEEE Computer Society Press.
19. G. D. Plotkin. Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1(2):125–159, December 1975.
20. G. Pottinger. Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357, 1977.
21. D. Prawitz. *Natural Deduction, a Proof-Theoretical Study*. Almqvist and Wiksell, Stockholm, 1965.
22. José Espírito Santo. Revisiting the correspondence between cut elimination and normalization. In *Proc. of ICALP 2000*, pages 600–611. Springer-Verlag LNCS 1853, 2000.
23. Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11:207–260, 2001.
24. Masako Takahashi. Parallel reductions in λ -calculus. *Information and Computation*, 118(1):120–127, April 1995.
25. J. I. Zucker. Correspondence between cut-elimination and normalization, part i and ii. *Annals of Mathematical Logic*, 7:1–156, 1974.