Classically typed λ-calculi
Master thesis

July 31, 2008

With thanks to Herman Geuvers

Freek Verbeek

Radboud University, Nijmegen

Abstract. Intuitionistic natural deduction can be extended to a classical logical system by adding the law of Double Negation or Reductio Ad Absurdum or by allowing multiple conclusions in the sequents. The isomorphic λ-calculi are respectivily λC, $\lambda\Delta$ and $\lambda\mu$. The $\bar{\lambda}\mu$- and the $\bar{\lambda}\mu\tilde{\mu}$-calculus are both isomorphic to classical sequent calculus. These classically typed λ-calculi contain operators that in some way provide control over the current context. The calculi can be compared on three aspects: the typing system, the notion of reduction and the operational semantics. For the systems $\lambda\mu$, $\bar{\lambda}\mu$ and $\bar{\lambda}\mu\tilde{\mu}$ translations have been found that preserve all these aspects. The $\lambda\Delta$-calculus is modified in such a way that such a translation can be found as well.

Research number: 595
Supervisor: Herman Geuvers

Table of Contents

Classically typed λ-calculi . . . 1

1 Introduction . . . 3
2 Natural Deduction and Sequent Calculus . . . 3
3 The λ-calculus . . . 7
3.1 The simply typed λ-calculus . . . 8
3.2 Equivalence of λ-calculi . . . 10
3.3 Subject Reduction . . . 12
4 The λC-calculus . . . 13
4.1 Subject Reduction . . . 13
5 The $\lambda\Delta$-calculus . . . 15
5.1 Notion of reduction and operational semantics . . . 16
5.2 The λC-calculus and the $\lambda\Delta$-calculus . . . 17
5.3 Subject Reduction . . . 17
6 The $\lambda\mu$-calculus . . . 18
6.1 Notion of reduction and operational semantics . . . 22
6.2 The $\lambda\Delta$-calculus and the $\lambda\mu$-calculus . . . 23
6.3 Altering the $\lambda\Delta$-calculus . . . 29
6.4 Conclusion . . . 32
7 The $\bar{\lambda}\mu$-calculus . . . 33
7.1 Notion of reduction and operational semantics . . . 35
7.2 The $\lambda\mu$-calculus and the $\bar{\lambda}\mu$-calculus . . . 37
8 The $\bar{\lambda}\mu\tilde{\mu}$-calculus . . . 40
8.1 Adding $\tilde{\mu}$ . . . 40
8.2 Adding rules to handle $\perp$ . . . 42
8.3 The $\bar{\lambda}\mu$-calculus and the $\bar{\lambda}\mu\tilde{\mu}$-calculus . . . 45
9 Conclusion . . . 47

1 Introduction

This paper concerns the classical logical systems Natural Deduction (N) and Sequent Calculus (L) and the λ-calculi that are isomorphic to them. Natural deduction is based on right introduction and right elimination rules, whereas the Sequent Calculus is based on left and right introduction rules. Furthermore usual Natural Deduction has the restriction that each right hand side must contain exactly one conclusion. Originally it was preferred to use Natural Deduction to study the λ-calculus, using the Curry-Howard isomorphism. In recent studies however, it is argued that the Sequent Calculus is more suitable for this [1]. In [2], Parigot argues that the difficulties in trying to use classical formulas such as $\neg\neg A \rightarrow A$ as types for operators, is not due to classical logic, but to the fact that usual Natural Deduction is restricted to one conclusion, which forbids the most natural transformations of proofs.

In [3] the λ-calculus is extended with $\perp$ and a $\varDelta$-operator which corresponds to the law of Double Negation. The corresponding classical logic is complete, because the operator $\neg$ can be represented and the set $\{\rightarrow, \neg\}$ is ideologically complete in classical logic. This means that $\varDelta$ should suffice as a base operator, powerful enough to represent any other λ-calculus which corresponds to propositional logic. For example, it can represent the C and $\mathcal{A}$ operators of Griffin, introduced in [4] to define a typed λ-calculus that corresponds to classical logic. In this paper, I will study the relation between the $\lambda\varDelta$-calculus of Rehof and Sørensen [3] and both the $\lambda\mu$-calculus of Parigot [2] and the $\bar{\lambda}\mu\tilde{\mu}$-calculus of Curien and Herbelin [1].

2 Natural Deduction and Sequent Calculus

The language of formulas of minimal implicational logic is defined by the following grammar [5]:

$$\text{form} = \text{atom} \mid \text{form} \rightarrow \text{form} \tag{2.1}$$

$$\text{atom} = p \mid \text{atom}' \tag{2.2}$$

Atoms are denoted by $p, q, r, \ldots$ and formulas by $A, B, C, \ldots$. It will be of importance to make the difference between sets of formulas and lists of formulas explicit. This will be done by denoting sets by $\varGamma, \varGamma', \varSigma, \varSigma', \ldots$ and lists by $\varPi, \varPi', \varUpsilon, \varUpsilon', \ldots$. For a list the order of the formulas is fixed and altering the order or the size is done by explicit structural rules. For a set the order is of no importance and adding rules is done implicitly.

A formula is *derivable* in N from the set $\varGamma$, notation $\varGamma \vdash_{\mathsf{N}} A$, if it can be generated from the rules of N in table 2.1. The same definition holds for L.

Definition 2.1. *Two logical systems S and S' are called* equivalent *if*

$$\varPi \vdash_S A \Leftrightarrow \varPi \vdash_{S'} A$$

N

$$\frac{A \in \Gamma}{\Gamma \vdash A}\,\text{axiom}$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\,{\to}\text{E}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,{\to}\text{I}$$

L

$$\frac{A \in \Gamma}{\Gamma \vdash A}\,\text{axiom}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}\,{\to}\text{l}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,{\to}\text{r}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}\,\text{cut}$$

Table 2.1. Rules for Natural Deduction and Sequent Calculus

holds for all formulas A and lists Π.

Theorem 2.2. N *and* L *are equivalent. For the proof, see [5].*

In order to extend N to a natural deduction system for full classical logic, it is sufficient to add ⊥ and the law of Double Negation. This system is denoted by N^{DN} (see table 2.2). In order to extend L to a sequent calculus for full classical logic, the restriction that the right hand side of each sequent must contain exactly one formula must be lifted. This results in the system LK (table 2.3) defined by Gentzen in [6]. Besides the usual logical rules, LK contains structural rules for weakening, contraction and interchange.

N^{DN}

$$\frac{A \in \Gamma}{\Gamma \vdash A}\,\text{axiom}$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\,{\to}\text{E}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\,{\to}\text{I}$$

$$\frac{\Gamma, A \to \bot \vdash \bot}{\Gamma \vdash A}\,\neg\neg\text{E}$$

Table 2.2. Rules for Natural Deduction with Double Negation

The original LK of Gentzen does not contain the rule (⊥E), but takes ¬ as a primitive operator and has added to it the two following inference rules:

$$\frac{\Upsilon \vdash A, \Pi}{\Upsilon, \neg A \vdash \Pi}\,\neg\text{l} \qquad\qquad \frac{\Upsilon, A \vdash \Pi}{\Upsilon \vdash \neg A, \Pi}\,\neg\text{r}$$

The altered version of LK used here is from Prawitz [7] and is equivalent to the original LK. Using this version facilitates the comparison with N^{DN}.

Gentzen originally proved the equivalence of N^{DN} and LK by using a logical system modeled on Hilberts formalism [6]. In this paper, a natural deduction system will be used of which the right hand sides of sequents are not restricted to one conclusion. Such a system is called Classical Natural Deduction (CN) and is defined by Parigot in [2] (see table 2.4). Parigot doesn't define structural rules for CN, but simply states that weakening is managed implicitly. For sake of clarity I will make this explicit and use the right-structural rules of LK for CN as well. Altering the left hand side of the sequents of CN will be done implicitly, since that side consists of sets, not of lists.
The definition of equivalence can't be applied to N^{DN} and CN, since a sequent with multiple conclusions has no meaning in N^{DN}. I have considered two ways to solve this. The first was to define an interpretation in N^{DN} of a CN-sequent with multiple conclusions, using disjunction. This would lead to the following proposition:

$$\Gamma \vdash_{\mathsf{CN}} A_0, A_1, A_2, \ldots \Leftrightarrow \Gamma \vdash_{\mathsf{N}^{\mathsf{DN}}} A_0 \vee A_1 \vee A_2 \ldots \tag{2.3}$$

The second way uses the following lemma:

Lemma 2.3.

$$\Gamma \vdash_{\mathsf{CN}} A, \Pi \Leftrightarrow \Gamma, \neg\Pi \vdash_{\mathsf{CN}} A$$

$$\textit{where } \neg\Pi \overset{def}{=} \{\pi \to \bot \mid \pi \in \Pi\}$$

The second way provides a far easier and more straightforward proof. Let CN^1 denote CN, where each sequent with multiple conclusions is interpreted using lemma 2.3.

Theorem 2.4. N^{DN} *and* CN^1 *are equivalent.*

Proof.
($\Rightarrow$) By induction on the derivations in N^{DN}. A short proof is provided in [2]. For clarity, a proof is given of the soundness of the rule ($\neg\neg$E).

$$\cfrac{\cfrac{\cfrac{\cfrac{\Gamma, A \to \bot \vdash_{\mathsf{CN}} \bot}{\Gamma \vdash_{\mathsf{CN}} (A \to \bot) \to \bot}\;{\to}\mathrm{I} \qquad \cfrac{\cfrac{\cfrac{}{A \vdash_{\mathsf{CN}} A}\;\text{axiom}}{A \vdash_{\mathsf{CN}} \bot, A}\;\text{weak-r}}{\vdash_{\mathsf{CN}} A \to \bot, A}\;{\to}\mathrm{I}}{\Gamma \vdash_{\mathsf{CN}} \bot, A}\;{\to}\mathrm{E}}{\Gamma \vdash_{\mathsf{CN}} A, A}\;\bot\mathrm{E}}{\Gamma \vdash_{\mathsf{CN}} A}\;\text{cont-r}$$

($\Leftarrow$) By induction on the derivations in CN. The proofs of the soundness of ($\to$E) and ($\bot$E) are given:

$$\cfrac{\Gamma, \neg\Pi \vdash_{\mathsf{N}^{\mathsf{DN}}} A \to B \qquad \Gamma', \neg\Pi' \vdash_{\mathsf{N}^{\mathsf{DN}}} A}{\Gamma, \Gamma', \neg\Pi, \neg\Pi' \vdash_{\mathsf{N}^{\mathsf{DN}}} A}\;{\to}\mathrm{E} \qquad\qquad \cfrac{\cfrac{\Gamma, \neg\Pi \vdash_{\mathsf{N}^{\mathsf{DN}}} \bot}{\Gamma, \neg\Pi, A \to \bot \vdash_{\mathsf{N}^{\mathsf{DN}}} \bot}}{\Gamma, \neg\Pi \vdash_{\mathsf{N}^{\mathsf{DN}}} A}\;\neg\neg\mathrm{E}$$

LK

$$\frac{}{A \vdash A}\ \text{axiom} \qquad \frac{\Upsilon \vdash A, \Pi \qquad \Upsilon', A \vdash \Pi'}{\Upsilon, \Upsilon' \vdash \Pi, \Pi'}\ \text{cut}$$

$$\frac{\Upsilon \vdash A, \Pi \qquad \Upsilon', B \vdash \Pi'}{\Upsilon, \Upsilon', A \to B \vdash \Pi, \Pi'}\ \to\text{l} \qquad \frac{\Upsilon, A \vdash B, \Pi}{\Upsilon \vdash A \to B, \Pi}\ \to\text{r}$$

$$\frac{\Upsilon \vdash \bot, \Pi}{\Upsilon \vdash A, \Pi}\ \bot\text{E}$$

$$\frac{\Upsilon \vdash \Pi}{\Upsilon, A \vdash \Pi}\ \text{weak-l} \qquad \frac{\Upsilon \vdash \Pi}{\Upsilon \vdash A, \Pi}\ \text{weak-r}$$

$$\frac{\Upsilon, A, A \vdash \Pi}{\Upsilon, A \vdash \Pi}\ \text{cont-l} \qquad \frac{\Upsilon \vdash A, A, \Pi}{\Upsilon \vdash A, \Pi}\ \text{cont-r}$$

$$\frac{\Upsilon, A, B, \Upsilon' \vdash \Pi}{\Upsilon, B, A, \Upsilon' \vdash \Pi}\ \text{int-l} \qquad \frac{\Upsilon \vdash \Pi, A, B, \Pi'}{\Upsilon \vdash \Pi, B, A, \Pi'}\ \text{int-r}$$

Table 2.3. Rules for Classical Sequent Calculus

CN

$$\frac{}{\Gamma, A \vdash A}\ \text{axiom}$$

$$\frac{\Gamma, A \vdash B, \Pi}{\Gamma \vdash A \to B, \Pi}\ \to\text{I} \qquad \frac{\Gamma \vdash A \to B, \Pi \qquad \Gamma' \vdash A, \Pi'}{\Gamma, \Gamma' \vdash B, \Pi, \Pi'}\ \to\text{E}$$

$$\frac{\Gamma \vdash \bot, \Pi}{\Gamma \vdash A, \Pi}\ \bot\text{E}$$

Table 2.4. Rules for Classical Natural Deduction

□

Lemma 2.5. *In* CN*, the following rule can be derived:*

$$\dfrac{\Gamma, A \vdash_{\mathsf{CN}} \Pi \qquad \Gamma' \vdash_{\mathsf{CN}} A, \Pi'}{\Gamma, \Gamma' \vdash_{\mathsf{CN}} \Pi, \Pi'}\,\text{cut}$$

Proof.
Let n denote the length of Π. For $n = 0$, Π is interpreted as $\top$. For $n \geq 0$, let $\Pi = \sigma_0 \cup \Pi''$.

$$\dfrac{\dfrac{\Gamma, A \vdash_{\mathsf{CN}} \top}{\Gamma \vdash_{\mathsf{CN}} A \to \top}\,{\to}\mathrm{I} \qquad \Gamma' \vdash_{\mathsf{CN}} A, \Pi'}{\Gamma, \Gamma' \vdash_{\mathsf{CN}} \top, \Pi'}\,{\to}\mathrm{E} \qquad\qquad \dfrac{\dfrac{\Gamma, A \vdash_{\mathsf{CN}} \sigma_0, \Pi''}{\Gamma \vdash_{\mathsf{CN}} A \to \sigma_0, \Pi''}\,{\to}\mathrm{I} \qquad \Gamma' \vdash_{\mathsf{CN}} A, \Pi'}{\Gamma, \Gamma' \vdash_{\mathsf{CN}} \Pi, \Pi'}\,{\to}\mathrm{E}$$

Theorem 2.6. CN *and* LK *are equivalent.*

Proof.
(⇒) By induction on the derivations in CN. A proof is given of the soundness of the rule (→E), the other rules are trivial.

$$\dfrac{\dfrac{\Upsilon \vdash_{\mathsf{LK}} A \to B, \Pi \qquad \dfrac{\Upsilon' \vdash_{\mathsf{LK}} A, \Pi' \qquad \dfrac{}{B \vdash_{\mathsf{LK}} B}\,\text{axiom}}{\Upsilon', A \to B \vdash_{\mathsf{LK}} \Pi', B}\,{\to}\mathrm{l}}{\Upsilon, \Upsilon' \vdash_{\mathsf{LK}} \Pi, \Pi', B}\,\text{cut}}{\Upsilon, \Upsilon' \vdash_{\mathsf{LK}} B, \Pi, \Pi'}\,\text{int-r}$$

(⇐) By induction on the derivations in LK. The soundness of the rule (cut) is proven in lemma 2.5, which shows that a cut in LK corresponds to a detour in CN. The soundness of the rule (→l) is proven below and uses lemma 2.5 twice. The other rules are trivial.

$$\dfrac{\dfrac{\dfrac{\Gamma', B \vdash_{\mathsf{CN}} \Pi' \qquad \dfrac{\dfrac{}{A \to B \vdash_{\mathsf{CN}} A \to B}\,\text{axiom} \qquad \dfrac{}{A \vdash_{\mathsf{CN}} A}\,\text{axiom}}{A \to B, A \vdash_{\mathsf{CN}} B}\,{\to}\mathrm{E}}{\Gamma', A \to B, A \vdash_{\mathsf{CN}} \Pi'}\,\text{cut} \qquad \Gamma \vdash_{\mathsf{CN}} A, \Pi}{\Gamma', A \to B, \Gamma \vdash_{\mathsf{CN}} \Pi', \Pi}\,\text{cut}}{\Gamma, \Gamma', A \to B \vdash_{\mathsf{CN}} \Pi, \Pi'}\,\text{int-r}$$

□

Theorem 2.7. N^{DN} *and* LK *are equivalent.*

Proof.
The theorem follows from theorems 2.4 and 2.6.
□

3 The λ-calculus

For each λ-calculus that is considered in this paper, three different aspects will be defined:

Definition 3.1. *Let λS denote an extension of the λ-calculus characterized by an operator S. Let ΛS denote the grammar that defines the set of untyped terms of λS.*

(I) *The* inference system*, notation $\vdash_{\lambda S}$, is defined as the set of typing-rules of λS. It defines a subset of ΛS of well-typed terms, called λS-terms.*

(II) *(a) The* notion of reduction*, notation $\rightarrow_{\lambda S}$, is defined as the union of the reduction rules of λS.*

(b) The one-step reduction*, notation $\rhd_{\lambda S}$, is defined as the compatible closure of $\rightarrow_{\lambda S}$. It defines a reduction tree for each λS-term.*

(III) *The* operational semantics*, notation $\mapsto_{\lambda S}$, is defined as a deterministic reduction strategy for each λS-term. It defines exactly one reduction path in the reduction tree of each λS-term.*

Each calculus has one grammar, one inference system and one notion of reduction, but different kinds of operational semantics can be defined. Variables are denoted by $x, y, z, \ldots$ and λ-terms are denoted by $M, N, \ldots$. Equality of terms, up to renaming of bound variables, is denoted by $\equiv$.

3.1 The simply typed λ-calculus

The set Λ of untyped λ-terms is defined by the following grammar:

$$\Lambda \ni M ::= x \mid \lambda x.M \mid MM' \tag{3.1}$$

The notion of reduction $\rightarrow_\lambda$ is defined by one rule: $(\lambda x.M)N \rightarrow_\beta M[x := N]$. In order to define operational semantics of the λ-calculus, the notion of *evaluation context* is defined [4]. Evaluation contexts specify an order in which a term is evaluated. Informally, if $M = E[N]$, then E represents the rest of the computation that remains to be done after N is evaluated.

Definition 3.2. *Let the operational semantics $\mapsto_{\lambda S}$ be defined by a set of values $V_{\lambda S}$, a set of evaluation contexts $E_{\lambda S}$ and a set of redexes $R_{\lambda S}$. $\mapsto_{\lambda S}$ is called a* well-defined operational semantics *if for all closed terms $M \in \Lambda S$ holds that M is either a value $V_{\lambda S}$ or M can be written in a unique way as $M = E_{\lambda S}[R_{\lambda S}]$.*

Call-by-value operational semantics

Definition 3.3. *The call-by-value (CBV) operational semantics $\overset{\mathsf{v}}{\mapsto}$ is defined by three syntactic classes [8]:*

$$\begin{array}{ll}
\textit{Values} & V_{\mathsf{v}} ::= x \mid \lambda x.M \\
\textit{Evaluation contexts} & E_{\mathsf{v}} ::= [] \mid E_{\mathsf{v}}\ M \mid V_{\mathsf{v}}\ E_{\mathsf{v}} \\
\textit{Redexes} & R\ \ ::= (\lambda x.M)V_{\mathsf{v}}
\end{array}$$

Here $[]$ is a hole and M is a term. If E is an evaluation context, then $E[M]$ is the term that has M placed in the hole of E. A β_v-redex is defined as a term of the form $(\lambda x.M)V$. $\overset{\mathsf{v}}{\mapsto}$ defines a unique reduction path for each λ-term, as is proven in lemma 3.4.

Lemma 3.4. *The operational semantics $\overset{\mathsf{v}}{\mapsto}$ is a well-defined operational semantics.*

Proof. A closed term is either an abstraction or an application. In the first case the term is a value. In the second case, we have a closed term of the form: $P = (\lambda x.M)N_1 \dots N_n$ where $N_1 \dots N_n$ are closed terms ($n \geq 1$). The proof is by induction on P. There are two possibilities:

1. N_1 is a value. Then $P = RN_2 \dots N_n$, thus $P = E[R]$. If $n = 1$, then $E = [\,]$.
2. N_1 is not a value. Then the induction hypothesis states that $N_1 = E'[R]$ for some context E' and some β_v-redex R. This means $P = (\lambda x.M)(E'[R])N_2 \dots N_n = E[R]$, where $E = (\lambda x.M)E'[\,]N_2 \dots N_n$.

Call-by-name operational semantics

Definition 3.5. *The call-by-name (CBN) operational semantics $\overset{\mathsf{n}}{\mapsto}$ is defined by three syntactic classes:*

$$\begin{array}{ll} \textit{Values} & V_{\mathsf{n}} ::= x \mid \lambda x.M \\ \textit{Evaluation contexts} & E_{\mathsf{n}} ::= [] \mid E_{\mathsf{n}}\ M \\ \textit{Redexes} & R_{\mathsf{n}} ::= (\lambda x.M)N \end{array}$$

A β_n-redex is defined as a term of the form $(\lambda x.M)N$. The above definition states that a reducible λ_n-term is either a redex or the redex occurs in the left hand side of an application. There are no reductions under abstraction [9]. CBN thus reduces the leftmost redex that is not under abstraction. $\overset{\mathsf{n}}{\mapsto}$ defines a unique reduction path for each λ-term, as is proven in lemma 3.6.

Lemma 3.6. *The operational semantics $\overset{\mathsf{n}}{\mapsto}$ is a well-defined operational semantics.*

Proof. A closed term is either an abstraction or an application. In the first case, M is a value. In the second case $M = (\lambda x.M')N_1N_2 \dots N_n$. Then $M = E[(\lambda x.M')N_1]$ where $E = []N_2 \dots N_n$. If $n = 1$, then $E = []$.

If evaluation contexts are used in order to define operational semantics, a hole is always filled with a redex. However, sometimes a term M will be written as $E[N]$ with N not nessecarily a redex. In this case, M can either be written in a unique way as $E'[R']$ or it is a value.

Definition 3.7. *Let $\mapsto_{\lambda S}$ be operational semantics defined by evaluation contexts $E_{\lambda S}$.*

$M \doteq E_{\lambda S}[R_{\lambda S}] \stackrel{def}{=}$ *"M can be written in a unique way as $E_{\lambda S}[R_{\lambda S}]$, with $R_{\lambda S}$ the redex that is first to be contracted according to the operational semantics $\mapsto_{\lambda S}$ "*

$M = E_{\lambda S}[N] \stackrel{def}{=}$ *"M is the context $E_{\lambda S}$ with its hole filled with N"*

3.2 Equivalence of λ-calculi

A commonly accepted notion of equivalence for functional languages is Morris style conceptual equivalence [8]. Bierman defines a notion of equivalence based on transitions in an abstract machine in [8]. These notions often require complex proofs. In this paper, I define three different sorts of equivalence:

1. *Type-equivalence* is determined by the inference systems of the two calculi.
2. *Reductional equivalence* is determined by the one-step reductions of the two calculi.
3. *Computational equivalence* is determined by the operational semantics of the two calculi.

Definition 3.8.
$\mathcal{I} : \Lambda S \to \Lambda S'$ is a type-preserving translation *from ΛS to $\Lambda S'$ iff for all M*

$$\Gamma \vdash_{\lambda S} M : A \Leftrightarrow \Gamma \vdash_{\lambda S'} \mathcal{I}(M) : A$$

holds for all Γ and A.

Definition 3.9.

1. *A type A is Γ-*inhabited *in λS if there exists a term M such that*
$$\Gamma \vdash_{\lambda S} M : A$$
2. *λS is* type-embedded into *$\lambda S'$, notation $\lambda S \leq_t \lambda S'$, iff for all types $A \in \Lambda S$ and contexts Γ holds that if A is Γ-inhabited in λS, then A is Γ-inhabited in $\lambda S'$.*
3. *λS and $\lambda S'$ are called* type-equivalent*, notation $\lambda S \equiv_t \lambda S'$, iff λS is type-embedded into $\lambda S'$ and $\lambda S'$ is type-embedded into λS.*

Definition 3.10.

1. *$\mathcal{I} : \Lambda S \to \Lambda S'$ is a* reductional embedding *of λS into $\lambda S'$ if it is a type-preserving translation such that*
$$M \rhd_{\lambda S} M' \implies \mathcal{I}(M) \unrhd\!\!\rhd_{\lambda S'} \mathcal{I}(M')$$
holds for all $M, M' \in \Lambda S$.
2. *λS can be* reductionally embedded in *$\lambda S'$, notation $(\rhd_{\lambda S}) \leq_r (\rhd_{\lambda S'})$, if there exists a reductional embedding from λS to $\lambda S'$.*
3. *λS is* reductionally equivalent to *$\lambda S'$, notation $(\rhd_{\lambda S}) \equiv_r (\rhd_{\lambda S'})$, if λS can be reductionally embedded into $\lambda S'$ and $\lambda S'$ can be reductionally embedded into λS.*

Definition 3.11.

1. *$\mathcal{I} : \Lambda S \to \Lambda S'$ is a* computational embedding *of $\mapsto_{\lambda S}$ into $\mapsto_{\lambda S'}$ if it is a type-preserving translation such that*
$$M \mapsto_{\lambda S} M' \implies \mathcal{I}(M) \mapsto\!\!\!\!\to_{\lambda S'} \mathcal{I}(M')$$
holds for all $M, M' \in \Lambda S$.

2. $\mapsto_{\lambda S}$ *can be* computationally embedded in $\mapsto_{\lambda S'}$*, notation* $(\mapsto_{\lambda S}) \leq_c (\mapsto)_{\lambda S'}$*), if there exists a computational embedding from* $\mapsto_{\lambda S}$ *to* $\mapsto_{\lambda S'}$*.*
3. $\mapsto_{\lambda S}$ *is* computationally equivalent to $\mapsto_{\lambda S'}$*, notation* $(\mapsto_{\lambda S}) \equiv_c (\mapsto_{\lambda S'})$*, if* $\mapsto_{\lambda S}$ *can be computationally embedded into* $\mapsto_{\lambda S'}$ *and* $\mapsto_{\lambda S'}$ *can be computationally embedded into* $\mapsto_{\lambda S}$*.*

Remark 3.12. If λS and $\lambda S'$ are reductionally equivalent, they aren't necessarily equivalent with respect to their notions of reduction, that is:

$$(\rhd_{\lambda S}) \equiv_r (\rhd_{\lambda S'}) \not\Rightarrow (\rightarrow_{\lambda S}) \equiv_{r'} (\rightarrow_{\lambda S'})$$

where $\equiv_{r'}$ is defined analogously to definition 3.10. A counter-example will be given in the proof of lemma 6.24.

Definition 3.13. *Given a type-preserving translation* $\mathcal{I} : \Lambda S \rightarrow \Lambda S'$*, the* CBN context-translation $\mathcal{I}^{\mathsf{n}}$ *is defined as:*

$$\begin{aligned} \mathcal{I}^{\mathsf{n}}([])[] &= [] \\ \mathcal{I}^{\mathsf{n}}(E_{\lambda S}\ M)[] &= \mathcal{I}^{\mathsf{n}}(E_{\lambda S})[]\ \mathcal{I}(M) \end{aligned}$$

The CBV context-translation $\mathcal{I}^{\mathsf{v}}$ *is defined as:*

$$\begin{aligned} \mathcal{I}^{\mathsf{v}}([])[] &= [] \\ \mathcal{I}^{\mathsf{v}}(E_{\lambda S}\ M)[] &= \mathcal{I}^{\mathsf{v}}(E_{\lambda S})[]\ \mathcal{I}(M) \\ \mathcal{I}^{\mathsf{v}}(V\ E_{\lambda S})[] &= \mathcal{I}(V)\ \mathcal{I}^{\mathsf{v}}(E_{\lambda S})[] \end{aligned}$$

Definition 3.14. *Let* $\mathcal{I} : \Lambda S \rightarrow \Lambda S'$ *be a type-preserving translation. Let* $^{\mathsf{E}}$ *denote operational semantics defined by the evaluation contexts* $E_{\lambda S}$ *(*$^{\mathsf{E}}$ *is for example* $^{\mathsf{n}}$ *or* $^{\mathsf{v}}$*). The translation* $\mathcal{I}$ *is a* context-preserving translation*, if*

(I) for all terms $M \in \Lambda S$ *holds*

$$M = E_{\lambda S}[N] \implies \mathcal{I}(M) = \mathcal{I}^{\mathsf{E}}(E_{\lambda S})[\mathcal{I}(N)]$$

(II) for all terms $M \in \Lambda S$ *holds*

$$M \doteq E_{\lambda S}[R_{\lambda S}] \implies \mathcal{I}(M) \doteq \mathcal{I}^{\mathsf{E}}(E_{\lambda S})[\mathcal{I}(R_{\lambda S})]$$

Remark 3.15. Requirement (II) implies that $\mathcal{I}^{\mathsf{E}}(E_{\lambda S})[]$ is required to be a $\lambda S'$-context and $\mathcal{I}(R_{\lambda S})$ is required to be a $\lambda S'$-redex.

Lemma 3.16. *Let* λS *and* $\lambda S'$ *have well-defined CBN operational semantics* $\overset{\mathsf{n}}{\mapsto}_{\lambda S}$ *and* $\overset{\mathsf{n}}{\mapsto}_{\lambda S'}$*. Let* $\mathcal{I} : \Lambda S \rightarrow \Lambda S'$ *be a translation such that it is*

1. *an embedding of the notion of reduction of* λS *into the notion of reduction of* $\lambda S'$*;*
2. *a CBN context-preserving translation;*

Then $\mathcal{I}$ *is a computational embedding of* $\overset{\mathsf{n}}{\mapsto}_{\lambda S}$ *into* $\overset{\mathsf{n}}{\mapsto}_{\lambda S'}$ *as well.*

Proof.
It must be proven that if $M \overset{\mathsf{n}}{\mapsto}_{\lambda S} M'$ this implies $\mathcal{I}(M) \overset{\mathsf{n}}{\mapsto\!\!\!\!\twoheadrightarrow}_{\lambda S'} \mathcal{I}(M')$. We have that $M \stackrel{\circ}{=} E_{\lambda S}[R_{\lambda S}]$ (since $\overset{\mathsf{n}}{\mapsto}_{\lambda S}$ is well-defined) and thus the proof is by induction on the evaluation context $E_{\lambda S}$. The base case is $E_{\lambda S} = []$.

$$\begin{aligned} M = R_{\lambda S} &\overset{\mathsf{n}}{\mapsto}_{\lambda S} M', \text{ with } R_{\lambda S} \rightarrow_{\lambda S} M' \\ \mathcal{I}(M) = \mathcal{I}(R_{\lambda S}) &\overset{\mathsf{n}}{\mapsto\!\!\!\!\twoheadrightarrow}_{\lambda S'} \mathcal{I}(M'), \text{ since by assumption 1: } \mathcal{I}(R_{\lambda S}) \twoheadrightarrow_{\lambda S'} \mathcal{I}(M') \end{aligned}$$

Note that $N \twoheadrightarrow N'$ implies $N \overset{\mathsf{n}}{\mapsto\!\!\!\!\twoheadrightarrow} N'$, but that $N \triangleright\!\!\triangleright N'$ doesn't imply this. This is why $\mathcal{I}$ must embed the notion of reduction and not the one-step reduction. For the induction step, let $E_{\lambda S} = E'\ P$.

$$\begin{aligned} M &\stackrel{\circ}{=} E_{\lambda S}[R_{\lambda S}] \\ &\overset{\mathsf{n}}{\mapsto}_{\lambda S} E_{\lambda S}[M'] \end{aligned}$$

$$\begin{aligned} \mathcal{I}(M) &= \mathcal{I}(E_{\lambda S}[R_{\lambda S}]) \\ &\stackrel{\circ}{=} \mathcal{I}^{\mathsf{n}}(E_{\lambda S})[\mathcal{I}(R_{\lambda S})], \text{ by assumption 2.(II)} \\ &\overset{\mathsf{n}}{\mapsto\!\!\!\!\twoheadrightarrow}_{\lambda S'} \mathcal{I}^{\mathsf{n}}(E_{\lambda S})[\mathcal{I}(M')], \text{ by definition 3.7, since } \mathcal{I}(R_{\lambda S}) \twoheadrightarrow \mathcal{I}(M') \\ &= \mathcal{I}(E[M']), \text{ by assumption 2.(I)} \end{aligned}$$

□

3.3 Subject Reduction

Definition 3.17. *Let λS be a λ-calculus. The notion of reduction $\rightarrow_{\lambda S}$ has* Subject Reduction *(SR) iff*

$$\varGamma \vdash_{\lambda S} M : A \text{ and } M \rightarrow_{\lambda S} M' \implies \varGamma \vdash_{\lambda S} M' : A$$

holds for all terms $M, M' \in \Lambda S$.

Lemma 3.18. *Let λS and $\lambda S'$ be two λ-calculi and $\rightarrow_{\lambda S}$ and $\rightarrow_{\lambda S'}$ be their notions of reductions. If $\rightarrow_{\lambda S}$ has SR and $\rightarrow_{\lambda S'}$ hasn't, then $\lambda S'$ can't be reductionally embedded into λS.*

Proof.
Since $\rightarrow_{\lambda S'}$ doesn't have SR, there exists terms $M, M' \in \Lambda S'$ such that $M \rightarrow_{\lambda S'} M'$, but M and M' don't have the same type. Now assume that $\lambda S' \leq_r \lambda S$. Then there exists a type-preserving translation $\mathcal{I}$, such that $\mathcal{I}(M) \rightarrow_{\lambda S} \mathcal{I}(M')$. But then the fact that $\mathcal{I}(M)$ and $\mathcal{I}(M')$ don't have the same type and the fact that $\rightarrow_{\lambda S}$ has SR contradict. Thus $\lambda S' \nleq_r \lambda S$.
□

4 The λC-calculus

In [4], Griffin extended the CBV λ-calculus with the $\mathcal{A}$ and C-operator, respecitivily called `abort` and `control`. A type assignment of a type A to a term M is derivable from context Γ in the λC-calculus, notation $\Gamma \vdash_{\lambda C} M : A$, if it can be generated by the rules of table 4.1. In the resulting λC-calculus, the operator `abort` is actually defined using `control` by $\mathcal{A}(M) \overset{\text{def}}{=} C(\lambda d.M)$, where d is a dummy variable, not free in M.

λC
$\dfrac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\ \text{axiom}$
$\dfrac{\Gamma \vdash P : A \to B \qquad \Gamma \vdash Q : A}{\Gamma \vdash P\,Q : B}\ \to\text{E}$
$\dfrac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x.P : A \to B}\ \to\text{I}$
$\dfrac{\Gamma \vdash M : (A \to \bot) \to \bot}{\Gamma \vdash C(M) : A}\ \neg\neg\text{E}$

Table 4.1. Rules for $\lambda\Delta$-calculus

For the λC-calculus, a clear separation between the notion of reduction and the operational semantics doesn't exist, since the behaviour of the operators is only defined using evaluation contexts.

Definition 4.1. *The operational semantics $\mapsto_{Cu}$ is defined as the union of the following rules:*

$$\begin{aligned} E[(\lambda x.M)V] &\mapsto_\beta E[M[x := V]] \\ E[C(M)] &\mapsto_C M\ \lambda z.\mathcal{A}(E[z]) \\ E[\mathcal{A}(M)] &\mapsto_{\mathcal{A}} M \end{aligned}$$

Here E is a CBV evaluation context from definition 3.3.

Theorem 4.2. *A formula A is derivable in N^{DN} iff there exists a closed λC-term M such that $\vdash_{\lambda C} M : A$ is derivable.*

4.1 Subject Reduction

The operational semantics of C reveal a problem in the typing of C: on the right hand side of $\mapsto_C$ the expression $\mathcal{A}(E[z])$ forces E to be of type $\bot$, since $\mathcal{A}$ has type $\bot \to A$. This means that the type of E (the top-level-type) is forced to be $\bot$, but since there are no closed terms of that type, the rule is useless. Griffin noted this same problem in [4]. The problem is caused by the fact that the `abort` in the operational semantics of C doesn't abort to a user defined continuation, but to the top-level-continuation. Griffins solution was to reduce $C(\lambda k.k\ M)$ instead of M [10], where k represents the top-level-continuation [4].

Definition 4.3. *A closed λC-term M* evaluates with top-level-access to *N, notation $M \mapsto_{\mathsf{Ctop}} N$, if*

$$C(\lambda k.k\ M) \mapsto_{Ct} N$$

where the operational semantics $\mapsto_{Ct}$ is defined as the union of the rules:

$$\begin{aligned} C(\lambda k.E[(\lambda x.M)V]) &\mapsto_{t\beta} C(\lambda k.E[M[x := V]]) \\ C(\lambda k.E[C(M)]) &\mapsto_{tC} C(\lambda k.M\ \lambda z.\mathcal{A}(E[z])) \\ C(\lambda k.E[\mathcal{A}(M)]) &\mapsto_{t\mathcal{A}} C(\lambda k.M) \\ C(\lambda k.kV) &\mapsto_{tCc} V, \text{ where } k \notin FV(V) \end{aligned}$$

Example 4.4.
Let $M = \lambda t.C(\lambda j.j\ \mathcal{A}(j\ t))$ and let V be some value of type α. M represents the identity function and is typed as follows:

$$\underbrace{\lambda \overbrace{t}^{\alpha}.\ C(\lambda \overbrace{j}^{\neg\alpha}.\ j\ \overbrace{\mathcal{A}(\overbrace{j\ t}^{\bot})}^{\alpha})}_{\alpha\to\alpha}$$

$\mapsto_{Cu}$	$\mapsto_{Ct}$
$M\ V$	$C(\lambda k.k\ (M\ V))$
$\equiv E[M\ V]$, where $E = []$	$\equiv C(\lambda k.E[M\ V])$, where $E = k[]$
$\mapsto_{\beta} C(\lambda j.j\ \mathcal{A}(j\ V))$	$\mapsto_{t\beta} C(\lambda k.E[C(\lambda j.j\ \mathcal{A}(j\ V))])$
$\mapsto_{C} (\lambda z.\mathcal{A}(E[z]))\ \mathcal{A}((\lambda z.\mathcal{A}(E[z]))\ V)$	$\mapsto_{tC} C(\lambda k.(\lambda z.\mathcal{A}(E[z]))\ \mathcal{A}((\lambda z.\mathcal{A}(E[z]))\ V))$
$\mapsto_{\beta} \mathcal{A}(E[\mathcal{A}((\lambda z.\mathcal{A}(E[z]))\ V)])$	$\mapsto_{t\beta} C(\lambda k.\mathcal{A}(E[\mathcal{A}((\lambda z.\mathcal{A}(E[z]))\ V)]))$
$\mapsto_{\mathcal{A}} E[\mathcal{A}((\lambda z.\mathcal{A}(E[z]))\ V)]$	$\mapsto_{t\mathcal{A}} C(\lambda k.E[\mathcal{A}((\lambda z.\mathcal{A}(E[z]))\ V)])$
$\mapsto_{\mathcal{A}} (\lambda z.\mathcal{A}(E[z]))\ V$	$\mapsto_{t\mathcal{A}} C(\lambda k.(\lambda z.\mathcal{A}(E[z]))\ V)$
$\mapsto_{\beta} \mathcal{A}(E[V])$	$\mapsto_{t\beta} C(\lambda k.\mathcal{A}(E[V]))$
$\mapsto_{\mathcal{A}} E[V]$	$\mapsto_{t\mathcal{A}} C(\lambda k.E[V])$
$\equiv V$	$\equiv C(\lambda k.k\ V)$
	$\mapsto_{tCc} V$

This example shows that $\mapsto_{Cu}$ doesn't possess the SR property. The top-level-type (the type of E) is α, but the last reduction forces E to be of type $\bot$. The type is thus only preserved when α is $\bot$. The operational semantics $\mapsto_{Ct}$ do preserve the type in this example: both the entire term and the subterm $E[M\ V]$ have type α.

Lemma 4.5. *The operational semantics $\mapsto_{Ct}$ has SR.*

Proof.
The only non-trivial case is $\mapsto_{tC}$. Assume $C(\lambda k.E[C(M)])$ has type A. This implies that M has type $\neg\neg B$ for some B.

$$C(\underbrace{\lambda \overbrace{\ k.\ }^{\neg A} \overbrace{E[C(M)]}^{\perp}}_{\neg\neg A}) \implies C(\lambda k. \overbrace{M\ \underbrace{\lambda z.\mathcal{A}(E[z])}_{\neg B}}^{\perp}) : A$$

The typing thus implies that the $\mathcal{A}$ on the right hand side of $\mapsto_{tC}$ has type $\perp \to \perp$.
□

5 The $\lambda\Delta$-calculus

The set $\Lambda\Delta$ of terms from the $\lambda\Delta$-calculus is defined by the grammar [11]:

$$\Lambda\Delta \ni M ::= x \mid \lambda x.M \mid MM' \mid \Delta x.M \tag{5.1}$$

A type assignment of a type A to a term M is derivable from context Γ in the $\lambda\Delta$-calculus, notation $\Gamma \vdash_{\lambda\Delta} M : A$, if it can be generated by the rules of table 5.1. The term $\nabla(M)$ is defined as $\Delta d.M$, where $d \notin FV(M)$. So for M of type $\perp$, the term $\nabla(M)$ has any arbitrary type A.

$\lambda\Delta$
$\dfrac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$ axiom
$\dfrac{\Gamma \vdash P : A \to B \qquad \Gamma \vdash Q : A}{\Gamma \vdash P\,Q : B}$ →E
$\dfrac{\Gamma, x : A \vdash P : B}{\Gamma \vdash \lambda x.P : A \to B}$ →I
$\dfrac{\Gamma, x : A \to \perp \vdash M : \perp}{\Gamma \vdash \Delta x.M : A}$ ¬¬E

Table 5.1. Rules for $\lambda\Delta$-calculus

The $\lambda\Delta$-calculus is an extension of $\lambda\!\to$. Whereas $\lambda\!\to$ is isomorphic to N, $\lambda\Delta$ is isomorphic to N^{DN}. This leads to theorem 5.1.

Theorem 5.1. *A formula A is derivable in* N^{DN} *iff there exists a closed* $\lambda\Delta$*-term M such that* $\vdash_{\lambda\Delta} M : A$ *is derivable.*

Example 5.2. As an example of a derivation in the $\lambda\Delta$-calculus, an inhabitant of Peirce's Law is derived:

$$\dfrac{\dfrac{}{\varGamma \vdash_{\lambda\varDelta} y : \neg A}\,\text{ax} \qquad \dfrac{\dfrac{}{\varGamma \vdash_{\lambda\varDelta} x : P}\,\text{ax} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\varGamma, z : A, w : \neg B \vdash_{\lambda\varDelta} y : \neg A}\,\text{ax} \qquad \dfrac{}{\varGamma, z : A, w : \neg B \vdash_{\lambda\varDelta} z : A}\,\text{ax}}{\varGamma, z : A, w : \neg B \vdash_{\lambda\varDelta} y\ z : \bot}\,\to\text{E}}{\varGamma, z : A \vdash_{\lambda\varDelta} \varDelta w.y\ z : B}\,\neg\neg\text{E}}{\varGamma \vdash_{\lambda\varDelta} \lambda z.\varDelta w.y\ z : A \to B}\,\to\text{I}}{\varGamma \vdash_{\lambda\varDelta} x\ (\lambda z.\varDelta w.y\ z) : A}\,\to\text{E}}{\dfrac{\dfrac{\varGamma \vdash_{\lambda\varDelta} y\ (x\ \lambda z.\varDelta w.y\ z) : \bot}{x : P \vdash_{\lambda\varDelta} \varDelta y.y\ (x\ \lambda z.\varDelta w.y\ z) : A}\,\neg\neg\text{E}}{\vdash_{\lambda\varDelta} \lambda x.\varDelta y.y\ (x\ \lambda z.\nabla(y\ z)) : ((A \to B) \to A) \to A}\,\to\text{I}}\,\to\text{E}$$

Where $\neg A = A \to \bot$, $\varGamma = x : P, y : \neg A$ and $P = (A \to B) \to A$.

5.1 Notion of reduction and operational semantics

Rehof and Sørensen defined the notion of reduction of the $\lambda\varDelta$-calculus.

Definition 5.3. *The notion of reduction $\to_{\lambda\varDelta}$ is defined as the union of the rules:*

$$\begin{aligned}
(\lambda x.M)N &\to_{\beta} M[x := N] \\
(\varDelta x.M)N &\to_{\varDelta 1} \varDelta z.M[x := \lambda y.z(yN)] \\
(\varDelta x.xM) &\to_{\varDelta 2} M, \textit{ where } x \notin FV(M) \\
\varDelta x.x\ \nabla(xM) &\to_{\varDelta 3} M, \textit{where } x \notin FV(M)
\end{aligned}$$

Remark 5.4. The notion of reduction $\to_{\lambda\varDelta}$ is confluent [3], has subject reduction (see theorem 5.9) and strong normalization [3].

In [3], it is written that these rules are meant to be applied CBN. However, the operational semantics of $\varDelta$ isn't expressed using evaluation contexts. This is done in definition 5.5.

Definition 5.5. *The CBN operational semantics $\overset{\mathsf{n}}{\mapsto}_{\lambda\varDelta}$ is defined by three syntactic classes:*

$$\begin{array}{ll}
\textit{Values} & V_{\varDelta} ::= x \mid \lambda x.M \mid \varDelta x.M \textit{ if } \varDelta x.M \notin R_{\varDelta} \\
\textit{Evaluation contexts} & E_{\varDelta} ::= [] \mid E_{\varDelta}M \\
\textit{Redexes} & R_{\varDelta} ::= (\lambda x.M)N \mid (\varDelta x.M)N \mid \\
 & \qquad \varDelta x.x\ M \textit{ with } x \notin FV(M) \mid \\
 & \qquad \varDelta x.x\ \nabla(x\ M) \textit{ with } x \notin FV(M)
\end{array}$$

Lemma 5.6 shows that these operational semantics define a unique reduction path for each $\lambda\varDelta$-term.

Lemma 5.6. *The operational semantics $\overset{\mathsf{n}}{\mapsto}_{\lambda\varDelta}$ is a well-defined operational semantics.*

Proof. M is either an abstraction, an application or a $\varDelta$-abstraction. In the first case M is a value. In the second case, there are two possibilities:

1. $M = (\lambda x.M)N_1N_2 \ldots N_n$, in which case it can be written in a unique way as $E[R]$ (see lemma 3.6).

2. $M = (\varDelta x.M')N_1 N_2 \ldots N_n$, in which case either $\varDelta x.M'$ is a redex and thus $M = E[R]$ and $E = []N_1 \ldots N_N$ or it isn't a redex and $M = E[R]$, where $R = (\varDelta x.M')N_1$ and $E = []N_2 \ldots N_n$.

In the third case, $M = \varDelta x.M'$ for some M'. There are two possibilities:

1. $M' = x\ N$. When $x \notin FV(N)$ or when $N = \nabla(x\ N')$ with $x \notin FV(N')$, then M can be written as $E[R]$, where $R = M$ and $E = []$. In all other cases M is a value.
2. Otherwise M is a value.

5.2 The $\lambda\mathcal{C}$-calculus and the $\lambda\varDelta$-calculus

In the paper where Rehof and Sørensen introduced the $\varDelta$-operator [3], they compare it to Felleisens control operator $\mathcal{F}$ from [12], which is the same as Griffins $\mathcal{C}$. Furthermore they show that $\varDelta$ can express a `catch`/`throw`-mechanism. In comparing the $\lambda\varDelta$-calculus with the $\lambda\mathcal{C}$-calculus, it isn't possible to compare notions of reduction, since for $\lambda\mathcal{C}$ the notion of reduction and the operational semantics aren't clearly separated. They will thus be compared by their operational semantics. A translation between the two calculi can be accomplished by expressing the $\varDelta$-operator in $\mathcal{C}$:

$$\varDelta x.M \equiv \mathcal{C}(\lambda x.M) \tag{5.2}$$

The soundness of this definition is proven by the following derivation:

$$\dfrac{\dfrac{\Gamma, x : A \to \bot \vdash_{\lambda C} M : \bot}{\Gamma \vdash_{\lambda C} \lambda x.M : (A \to \bot) \to \bot}\,\to\mathrm{I}}{\Gamma \vdash_{\lambda C} \mathcal{C}(\lambda x.M) : A}\,\mathcal{C}$$

It follows that $\nabla(M)$ translates to $\mathcal{A}(M)$. This translation however doesn't provide a computational embedding, as is shown by example 5.7.

Example 5.7. Let $M = (\lambda x.y)\ \mathcal{A}(z)$. Using the operational semantics $\overset{\mathsf{n}}{\mapsto}_{\lambda\varDelta}$, this term reduces to y. Using the operational semantics from $\mathcal{C}$, this term reduces to z. This is caused by the fact that $\varDelta$ reduces CBN, whereas $\mathcal{C}$ reduces CBV.

5.3 Subject Reduction

Example 5.8. Let $M = \lambda t.\varDelta j.j\ \nabla(j\ t)$ and let N be some term of type α, like in example 4.4.

$$M\ N \rhd_{\beta} \varDelta j.j\ \nabla(j\ N) \rhd_{\varDelta 3} N$$

This example shows that for the term MN the type is preserved under reduction.

Theorem 5.9. *The notion of reduction $\to_{\lambda\varDelta}$ has SR.*

Proof.
($\to_{\Delta 1}$)
Let the type of $(\Delta x.M)N$ be B and $\Delta x.M$ of type $A \to B$. This implies that $\Delta z.M[x := \lambda y.z(y\ N)]$ has type B. Since x must have type $(A \to B) \to \perp$, y has type $A \to B$.

$$\underbrace{(\Delta x.\overbrace{\ M\ }^{\perp})\overbrace{\ N\ }^{A}}_{B} : B \implies \Delta z.\overbrace{M[x := \lambda y.\overbrace{\ z\ }^{B\to\perp}(\overbrace{\ y\,N\ }^{B})]}^{\perp} : B$$

($\to_{\Delta 2}$)
Let the type of $\Delta x.x\ M$ be A. This implies that M has type A.

$$\underbrace{\Delta\overbrace{\ x.\ }^{A\to\perp}\overbrace{x\,M}^{\perp}}_{A} : A \implies M : A$$

($\to_{\Delta 3}$)
Let the type of $\Delta x.x\ \nabla(x\ M)$ be A. This implies that M has type A.

$$\underbrace{\Delta\overbrace{\ x.\ }^{A\to\perp}x\,\nabla(\overbrace{\,x\,M\,}^{\perp})}_{A} : A \implies M : A$$

□

Theorem 5.10. *λC and $\lambda\Delta$ are type-equivalent, but aren't computationally equivalent with respect to the operational semantics $\overset{\mathsf{n}}{\mapsto}_{\lambda\Delta}$ and $\mapsto_{Cu}$.*

Proof.

$\lambda C \equiv_t \lambda\Delta$	This follows from theorems 4.2 and 5.1.
$(\mapsto_{\lambda\Delta}) \not\equiv_c (\mapsto_{Cu})$	This follows from lemma 3.18, example 4.4 and theorem 5.9.

□

Furthermore, example 5.7 shows that the straightforward type-preserving translation given by equation 5.2 isn't a computational embedding of $\mapsto_{\mathsf{Ctop}}$ into $\mapsto_{\lambda\Delta}$.

6 The $\lambda\mu$-calculus

The set $\Lambda\mu$ of terms from the $\lambda\mu$-calculus is defined by the grammar [13]:

$$\Lambda\mu \ni M ::= x \mid \lambda x.M \mid MM' \mid \mu\alpha.M \mid [\alpha]M \tag{6.1}$$

Besides the set of λ-variables, a distinct alphabet of μ-variables is used, which are denoted by α, β, γ, ... A type consists of a left hand side of formulas indexed with λ-variables, followed by the sign $\vdash_{\lambda\mu}$, followed by a right hand side that consists of at most one so called *active* unindexed formula, followed by a set of so called *passive* formulas indexed with μ-variables. So, an example of a type in the $\lambda\mu$-calculus is:

$$x : A \to B, y : A \vdash_{\lambda\mu} B; B \to A^{\alpha}, B^{\beta}$$

A set of formulas indexed by λ-variables is denoted by $\Gamma, \Gamma', \ldots$ and a set of formulas indexed by μ-variables is denoted by $\Sigma, \Sigma', \ldots$. A type assignment of a term M to a type $\Gamma \vdash_{\lambda\mu} A; \Sigma$ is derivable in the $\lambda\mu$-calculus, notation $M : \Gamma \vdash_{\lambda\mu} A; \Sigma$, if it can be generated by the rules of table 6.1. Here A is an active formula. A type without an active formula has the form $\Gamma \vdash_{\lambda\mu} ; \Sigma$.

$\lambda\mu$	
$\dfrac{}{x : \Gamma, A^x \vdash A; \Sigma}$ axiom	
$\dfrac{M : \Gamma, A^x \vdash B; \Sigma}{\lambda x.M : \Gamma \vdash A \to B; \Sigma}$ →I	$\dfrac{M : \Gamma \vdash A \to B; \Sigma \qquad N : \Gamma' \vdash A; \Sigma'}{(M\ N) : \Gamma, \Gamma' \vdash B; \Sigma, \Sigma'}$ →E
$\dfrac{M : \Gamma \vdash A; \Sigma}{[\alpha]M : \Gamma \vdash ; A^\alpha, \Sigma}$ Passivate	$\dfrac{M : \Gamma \vdash ; A^\alpha, \Sigma}{\mu\alpha.M : \Gamma \vdash A; \Sigma}$ Activate

Table 6.1. Rules for the $\lambda\mu$-calculus

Since Σ represents a set of passive formulas, no structural rules are needed to structurally alter this set. Right-contraction of an active formula A with a passive formula $A^\alpha \in \Sigma$ consists of passivating A. It then implicitly merges in the set with A^α. Right-weakening consists of activating a type with no active formula. Right-interchange only happens implicitly in Σ, thus only between passive formulas. Structural alterations on the left hand side occur implicitly. See example 6.1.

Parigot states in [2] that the $\lambda\mu$-calculus can be considered as an extension of the λ-calculus where one has the possibility to name arbitrary subterms by μ-variables and to abstract on these names. One can thus apply operations directly on a subterm, instead of only to the term as a whole. This is why the Passivate and Activate originally where called naming rules. Another interpretation is given by Ariola and Herbelin in [10]. A μ-variable α represents a context with a hole. By invoking α with the *command* $[\alpha]M$, the current context is abandoned and M is computed in the context associated with α. A term $\mu\alpha.M$ stores the current continuation for later use and resumes computing M.

Example 6.1. As an example of a derivation, Peirces Law is derived:

$$\dfrac{\dfrac{}{y : (A \to B) \to A^y \vdash_{\lambda\mu} (A \to B) \to A;}\ \text{axiom} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{x : A^x \vdash_{\lambda\mu} A;}\ \text{axiom}}{[\alpha]x : A^x \vdash_{\lambda\mu} ;A^\alpha}\ \text{Passivate}}{\mu\beta.[\alpha]x : A^x \vdash_{\lambda\mu} B;A^\alpha}\ \text{Activate (weak-r)}}{\lambda x.\mu\beta.[\alpha]x :\ \vdash_{\lambda\mu} A \to B;A^\alpha}\ \to\text{I}}{\dfrac{y\ (\lambda x.\mu\beta.[\alpha]x) : (A \to B) \to A^y \vdash_{\lambda\mu} A;A^\alpha}{\dfrac{[\alpha](y\ (\lambda x.\mu\beta.[\alpha]x)) : (A \to B) \to A^y \vdash_{\lambda\mu} ;A^\alpha}{\dfrac{\mu\alpha.[\alpha](y\ (\lambda x.\mu\beta.[\alpha]x)) : (A \to B) \to A^y \vdash_{\lambda\mu} A;}{\lambda y.\mu\alpha.[\alpha](y\ (\lambda x.\mu\beta.[\alpha]x)) :\ \vdash_{\lambda\mu} ((A \to B) \to A) \to A;}\ \to\text{I}}\ \text{Activate}}\ \text{Passivate (cont-r)}}\ \to\text{E}$$

This example shows that right weakening is managed by the Activate rule and right contraction is managed by the Passivate rule.

This calculus is isomorphic to CN without the rule (⊥E). Rules to handle ⊥ still need to be added. Parigot defined rules for this in [2] as in table 6.2.

$\dfrac{x : \Gamma \vdash \bot;\Sigma}{[\gamma]M : \Gamma \vdash ;\Sigma}$ ⊥E (Parigot)	$\dfrac{x : \Gamma \vdash ;\Sigma}{\mu\delta.M \vdash \bot;\Sigma}$ ⊥I (Parigot)

Table 6.2. Rules for handling ⊥ by Parigot

These rules however lead to a problem, for example in the following type assignment, from [14]:

$$\lambda y.\mu\alpha.[\gamma](y\ (\lambda x.\mu\beta.[\alpha]x)) :\ \vdash_{\lambda\mu} ((A \to \bot) \to \bot) \to A$$

In this closed term, γ is still free, like a useless hypothesis that is not discarded. The derivation step that caused the free μ-variable corresponds to an EFQ derivation step in the logical system. Using Parigots rules for handling ⊥, it is in fact impossible to create a closed $\lambda\mu$-term, thus without any open λ- or μ-variables of type $\vdash_{\lambda\mu} \bot \to A;$. This leads to theorem 6.2.

Theorem 6.2. *A sequent $\Gamma \vdash A;\Sigma$ is derivable in* CN \ (⊥*E) (minimal classical natural deduction) iff there exists a closed $\lambda\mu$-term M such that $\vdash_{\lambda\mu_{Par}} M : A;\Sigma$ is derivable, where $\vdash_{\lambda\mu_{Par}}$ is defined as the set of derivation rules of table 6.1.*

Ariola and Herbelin provided a solution [10]: instead of associating a command whose body M has type ⊥ with a fresh μ-variable, they associated it with a fixed continuation constant top. De Groote provided another solution in [14], by replacing Parigots rules by the rules in table 6.4.

Theorem 6.3. *A sequent $\Gamma \vdash A;\Sigma$ is derivable in* CN *iff there exists a closed $\lambda\mu$-term M such that $M : \Gamma \vdash_S A;\Sigma$ is derivable, where S is defined as:*

$\dfrac{M : \Gamma \vdash \bot ; \Sigma}{[\mathsf{top}]M : \Gamma \vdash\ ; \Sigma}\ \bot\mathrm{E}$ (Ariola/Herbelin)

Table 6.3. Rules for handling $\bot$ by Ariola & Herbelin

$\dfrac{M : \Gamma, A^x \vdash\ ; \Sigma}{\lambda x.M : \Gamma \vdash A \to \bot ; \Sigma}\ \neg\mathrm{I}$	$\dfrac{M : \Gamma \vdash A \to \bot ; \Sigma}{Mx : \Gamma, A^x \vdash\ ; \Sigma}\ \neg\mathrm{E}$
$\dfrac{M : \Gamma \vdash \bot ; A^\alpha, \Sigma}{\mu\alpha.M : \Gamma \vdash A ; \Sigma}\ \bot\mathrm{E}$	

Table 6.4. Rules for handling $\bot$ by De Groote

(I) $S = \lambda\mu_{\mathsf{top}}$*: the union of the derivation rules of table 6.1 and 6.3*

(II) $S = \lambda\mu$*: the union of the derivation rules of table 6.1 and 6.4*

See for proofs of these theorems respectively [10] and [14]. Solution (II) simply identifies any sequent of the form $\Gamma \vdash_{\lambda\mu}\ ; \Sigma$ with $\Gamma \vdash_{\lambda\mu} \bot ; \Sigma$. In table 6.5 a $\lambda\mu$-term with type $\vdash \bot \to A;$ is derived for the three mentioned systems.

$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \bot^x \vdash \bot ; A^\alpha}\ \text{axiom}}{[\gamma]x : \bot^x \vdash\ ; A^\alpha}\ \bot\mathrm{E}}{\mu\alpha.[\gamma]x : \bot^x \vdash A;}\ \text{Activate}}{\lambda x.\mu\alpha.[\gamma]x :\ \vdash \bot \to A;}\ \to\mathrm{I}$	$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \bot^x \vdash \bot ; A^\alpha}\ \text{axiom}}{[\mathsf{top}]x : \bot^x \vdash\ ; A^\alpha}\ \bot\mathrm{E}}{\mu\alpha.[\mathsf{top}]x : \bot^x \vdash A;}\ \text{Activate}}{\lambda x.\mu\alpha.[\mathsf{top}]x :\ \vdash \bot \to A;}\ \to\mathrm{I}$
Parigot	Ariola & Herbelin
$\dfrac{\dfrac{\dfrac{}{y : \bot^y \vdash \bot ; A^\alpha}\ \text{axiom}}{\mu\alpha.y : \bot^y \vdash A;}\ \bot\mathrm{E}}{\lambda y.\mu\alpha.y :\ \vdash \bot \to A;}\ \to\mathrm{I}$	
De Groote	

Table 6.5. Deriving $\lambda\mu$-terms with type EFQ

From now on, the $\lambda\mu$-calculus is defined as the first-order calculus defined by the rules of tables 6.1 and 6.4.

6.1 Notion of reduction and operational semantics

Parigots interpretation states that an operation can be applied directly to named subterms of a term. This interpretation is captured by *structural reduction*:

$$(\mu\alpha.M)N \rightarrow_{\mu s} \mu\alpha.M[[\alpha]P ::== [\alpha](PN)]$$

The argument N of the whole term is passed to all subterms P named α. This is done by structurally substituting $[\alpha](P\ N)$ for $[\alpha]P$. This substitution is inductively defined as follows ($(M)^*$ denotes $M[[\alpha]P ::== [\alpha](P\ N)]$):

$$\begin{aligned}
(x)^* &= x\\
(\lambda x.M)^* &= \lambda x.(M)^*\\
(M\ M')^* &= (M)^*\ (M')^*\\
(\mu\alpha.M)^* &= \mu\alpha.(M)^*\\
([\alpha]M)^* &= [\alpha]((M)^*\ N)\\
([\beta]M)^* &= [\beta](M)^*,\ \text{if } \alpha \neq \beta
\end{aligned}$$

Definition 6.4. *The notion of reduction $\rightarrow_{\lambda\mu}$ is defined as the union of the rules [10]:*

$$\begin{aligned}
(\lambda x.M)N &\rightarrow_{\beta} M[x := N]\\
(\mu\alpha.M)N &\rightarrow_{\mu s} \mu\alpha.M[[\alpha]P ::== [\alpha](PN)]\\
\mu\alpha.[\beta]\mu\gamma.M &\rightarrow_{\mu r} \mu\alpha.M[\gamma := \beta]\\
\mu\alpha.[\alpha]M &\rightarrow_{\mu t} M,\ \textit{if}\ \alpha \notin \mu FV(M)
\end{aligned}$$

Remark 6.5. The notion of reduction $\rightarrow_{\lambda\mu}$ is confluent, has subject reduction and strong normalization [15].

Remark 6.6. In [15], another reduction rule is suggested, which is called *elimination of absurd weakening*:

$$\mu\alpha.\mu\beta.M \rightarrow_{\mu\epsilon} \mu\alpha.M[\beta := \lambda^{\diamond} f.f]$$

Here, $\lambda^{\diamond}$, is defined as a λ-abstraction that spontaneously β-reduces. In this paper, this rule will not be used, since the reduction concerns only the elimination of declarations of variables of type $\bot \rightarrow \bot$. These declarations are useless and do not occur in this paper.

A CBN operational semantics for the $\lambda\mu$-calculus is defined:

Definition 6.7. *The CBN operational semantics $\overset{n}{\mapsto}_{\lambda\mu}$ is defined by three syntactic classes:*

$$\begin{array}{lll}
\textit{Values} & V_{\mu} & ::= x \mid \lambda x.M \mid \mu\alpha.M\ \textit{if}\ \mu\alpha.M \notin R_{\mu}\\
\textit{Evaluation contexts} & E_{\mu} & ::= [\,] \mid E_{\mu}M\\
\textit{Redexes} & R_{\mu} & ::= (\lambda x.M)N \mid (\mu\alpha.M)N \mid\\
 & & \qquad \mu\alpha.[\beta]\mu\gamma.M \mid \mu\alpha.[\alpha]M\ \textit{with}\ \alpha \notin \mu FV(M)
\end{array}$$

Lemma 6.8 shows that these operational semantics define a unique reduction path for each $\lambda\mu$-term.

Lemma 6.8. *The operational semantics $\overset{\mathsf{n}}{\mapsto}_{\lambda\mu}$ is a well-defined operational semantics.*

This can be proven analogously to the proof of lemma 5.6.

Bierman provides other operational semantics for the $\lambda\mu$-calculus in [8] that captures the command-interpretation. The set of redexes of the λ_n-calculus is extended with $[\alpha]M$ and $\mu\alpha.M$. Evaluation is written as $(E[R], \mathcal{E}) \mapsto (M', \mathcal{E}')$, where $\mathcal{E}$ is a function from μ-variables to evaluation contexts.

Definition 6.9. *The operational semantics $\mapsto_{\mu\mathcal{E}}$ is defined as the union of the rules:*

$$\begin{aligned}(E[(\lambda x.M)N], \mathcal{E}) &\mapsto_{\beta} (E[M[x := N]], \mathcal{E})\\ (E[\mu\alpha.M], \mathcal{E}) &\mapsto_{\mu 1} (M, \mathcal{E} \cup (\alpha, E[]))\\ (E[[\alpha]M, \mathcal{E} \cup (\alpha, E'[])) &\mapsto_{\mu 2} (E'[M], \mathcal{E} \cup (\alpha, E'[]))\end{aligned}$$

Here $\mathcal{E} \cup (\alpha, E)$ denotes the extension of the function $\mathcal{E}$ with the mapping $\alpha \mapsto E$.

These two operational semantics are not equivalent. Parigots semantics are strictly CBN and thus no reductions inside of μ-abstractions occur. Biermans semantics do allow reductions inside μ-abstractions.

Example 6.10. Let $M = \mu\alpha.(\lambda x.x)\ ([\alpha]N)$. Using Parigots semantics $\overset{\mathsf{n}}{\mapsto}_{\lambda\mu}$ this term is a value. Using Biermans semantics $\mapsto_{\mu\mathcal{E}}$ this term reduces to N.

6.2 The $\lambda\Delta$-calculus and the $\lambda\mu$-calculus

Analogously to the definition of equivalence of N^{DN} and CN, the definition of type-equivalence can't hold in the straightforward way for the $\lambda\Delta$-calculus and the $\lambda\mu$-calculus, since a type assignment with multiple types has no meaning in $\lambda\Delta$. Definition 6.11 provides an interpretation.

Definition 6.11. *The $\lambda\Delta$-calculus and the $\lambda\mu$-calculus are called type-equivalent if*

$$\begin{aligned}M : \Gamma \vdash_{\lambda\mu} A; \Sigma &\Leftrightarrow \Gamma, \neg\Sigma \vdash_{\lambda\Delta} M' : A\\ M : \Gamma \vdash_{\lambda\mu} ; \Sigma &\Leftrightarrow \Gamma, \neg\Sigma \vdash_{\lambda\Delta} M' : \bot\end{aligned}$$

holds for all $\lambda\mu$-types. Here $\neg\Sigma$ is defined as $\{\alpha : \sigma \to \bot \mid \sigma^{\alpha} \in \Sigma\}$.

Translating Δ into μ

In order to translate Δ into μ, the following derivation is used:

$$\dfrac{\dfrac{\dfrac{M : A \to \bot^{x} \vdash_{\lambda\mu} \bot;}{\lambda x.M : \;\vdash_{\lambda\mu} (A \to \bot) \to \bot;}\to\text{I} \qquad \dfrac{\dfrac{y : A^{y} \vdash_{\lambda\mu} A;}{[\alpha]y : A^{y} \vdash_{\lambda\mu} ; A^{\alpha}}\text{Passivate}}{\lambda y.[\alpha]y : \;\vdash_{\lambda\mu} A \to \bot; A^{\alpha}}\to\text{I}}{(\lambda x.M)\ (\lambda y.[\alpha]y) : \;\vdash_{\lambda\mu} \bot; A^{\alpha}}\to\text{E}}{\mu\alpha.(\lambda x.M)\ (\lambda y.[\alpha]y) : \;\vdash_{\lambda\mu} A;}\bot\text{E}$$

This derivation is isomorphic to the derivation used in theorem 2.4. De Groote uses a similar derivation in [14] to translate an altered variant of C, which is CBN and has altered reduction rules, into μ. The derivation leads to the following translation:

Definition 6.12. *The translation* $\overset{\wedge}{\rightarrow}$ *from a term* $M \in \Lambda\Delta$ *to a term* $\overset{\wedge}{\overrightarrow{M}} \in \Lambda\mu$ *is defined as follows:*

$$\begin{aligned}
\overset{\wedge}{\overrightarrow{x}} &= x\\
\overset{\wedge}{\overrightarrow{\lambda x.M)}} &= \lambda x.\overset{\wedge}{\overrightarrow{M}}\\
\overset{\wedge}{\overrightarrow{M\,N}} &= \overset{\wedge}{\overrightarrow{M}}\,\overset{\wedge}{\overrightarrow{N}}\\
\overset{\wedge}{\overrightarrow{\Delta x.M}} &\equiv \mu\alpha.\overset{\wedge}{\overrightarrow{M}}[x := \lambda y.[\alpha]y]
\end{aligned}$$

The translation of De Groote for λC-terms doesn't preserve reduction [14] and thus it isn't a reductional embedding. This similar translation for $\lambda\Delta$-terms isn't a reductional embedding of Δ into μ either, since reduction $\rhd_{\Delta 1}$ is not preserved:

$$\begin{aligned}
\overset{\wedge}{\overrightarrow{(\Delta x.M)N}} &= (\mu\alpha.\overset{\wedge}{\overrightarrow{M}}[x := \lambda y.[\alpha]y])\overset{\wedge}{\overrightarrow{N}}\\
&\rhd_{\mu s} \mu\alpha.\overset{\wedge}{\overrightarrow{M}}[x := \lambda y.[\alpha](y\,\overset{\wedge}{\overrightarrow{N}})] \qquad (6.2)\\
\overset{\wedge}{\overrightarrow{\Delta z.M[x := \lambda y.z(y\,N)]}} &= \mu\alpha.\overset{\wedge}{\overrightarrow{M}}[x := \lambda y.(\lambda q.[\alpha]q)(y\,\overset{\wedge}{\overrightarrow{N}})] \qquad (6.3)
\end{aligned}$$

Term (6.3) β-reduces to term (6.2), but in order to preserve reduction, term (6.2) must reduce to term (6.3). This is impossible, since it requires inverse β-reduction. The translation does preserve β-equality. In definition 6.13 a more specific equality for $\lambda\mu$-terms is defined that is preserved.

Definition 6.13. *Let the equality* $\hat{=}$ *for* $\lambda\mu$*-terms be defined as follows:*

$$\begin{aligned}
x &\hat{=} x\\
\lambda x.M &\hat{=} \lambda x.M', \text{ if } M \hat{=} M'\\
M\,N &\hat{=} M'\,N', \text{ if } M \hat{=} M' \text{ and } N \hat{=} N'\\
\mu\alpha.M &\hat{=} \mu\alpha.M', \text{ if } M \hat{=} M'\\
[\alpha]M &\hat{=} N, \text{ if either} \begin{cases} N \equiv [\alpha]P \text{ and } M \hat{=} P\\ N \equiv (\lambda q.N')P, \text{ with } [\alpha]q \hat{=} N' \text{ and } M \hat{=} P\end{cases}
\end{aligned}$$

Lemma 6.14. *The translation* $\overset{\wedge}{\rightarrow}$ *from definition 6.12 is a reductional embedding of* $\lambda\Delta$ *into* $\lambda\mu$ *with respect to* $\hat{=}$*, that is:*

$$M \rhd_{\lambda\Delta} N \implies \overset{\wedge}{\overrightarrow{M}} \unrhd\!\!\rhd_{\lambda\mu} N', \text{ with } N' \hat{=} \overset{\wedge}{\overrightarrow{N}}$$

Proof.
The proof is similar to the proof of lemma 6.18.
□

Another translation that is considered is created by inductively applying equation 6.4.

$$\varDelta x.M \equiv \mu\alpha.M[x := [\alpha]] \tag{6.4}$$

This translation would suffice if x occurred in M solely as the left hand side of an application. However, if it occurs as right hand side of an application, as in the type assignment $\vdash_{\lambda\varDelta} \lambda y.\varDelta x.y\ x : ((A \to \bot) \to \bot) \to A$, this translation would lead to syntacticly incorrect terms. The translation of $\lambda y.\varDelta x.y\ x$ would be $\lambda y.\mu\alpha.y\ [\alpha]$ which is not a well-formed term.

A solution is to have the translation of a term $\varDelta x.M$ depend on whether it is the left hand side of an application or not. In the first case, the translation is done by applying translation 6.12 and then by proceeding with translating M. In the second case the translation of a term $\varDelta x.M$ is done in two steps: the first step is applying translation 6.12 and proceeding with M just as in the first case. The second step is cleaning up the resulting term $N \in \Lambda\mu$ to a term $\llbracket N \rrbracket^{\alpha} \in \Lambda\mu$ by inductively substituting all occurrences of subterms of the form $(\lambda y.[\alpha]P)P'$ in N by $[\alpha]P[y := P']$ until no such subterms occur.

Definition 6.15. *The translation* $\to$ *from a term* $M \in \Lambda\varDelta$ *to a term* $\overrightarrow{M} \in \Lambda\mu$ *is defined as follows:*

$$\begin{array}{ll}
\overrightarrow{x} & = x \\
\overrightarrow{(\varDelta x.M)N} & = (\mu\alpha.\overrightarrow{M}[x := \lambda y.[\alpha]y])\ \overrightarrow{N} \\
\overrightarrow{\varDelta x.M} & = \mu\alpha.\llbracket \overrightarrow{M}[x := \lambda y.[\alpha]y] \rrbracket^{\alpha}_{[]} \\
\overrightarrow{M\ N} & = \overrightarrow{M}\ \overrightarrow{N} \\
\overrightarrow{\lambda x.M} & = \lambda x.\overrightarrow{M}
\end{array}$$

$$\begin{array}{ll}
\llbracket x \rrbracket^{\alpha}_{L} & = \mathsf{abstr}(x, L) \\
\llbracket (\lambda x.M)N \rrbracket^{\alpha}_{L} & = \llbracket M \rrbracket^{\alpha}_{[(x,N),L]} \\
\llbracket x\ N \rrbracket^{\alpha}_{L} & = \llbracket x \rrbracket^{\alpha}_{L}\ \llbracket N \rrbracket^{\alpha}_{L} \\
\llbracket \lambda x.M \rrbracket^{\alpha}_{L} & = \lambda x.\llbracket M \rrbracket^{\alpha}_{L} \quad \textit{if } \lambda x.M \textit{ is not the left hand side of an application} \\
\llbracket \mu\alpha.M \rrbracket^{\alpha}_{L} & = \mu\alpha.\llbracket M \rrbracket^{\alpha}_{L} \\
\llbracket [\alpha]M \rrbracket^{\alpha}_{L} & = [\alpha]\llbracket \mathsf{repl}(M, L) \rrbracket^{\alpha}_{[]} \\
\llbracket [\beta]M \rrbracket^{\alpha}_{L} & = \mathsf{abstr}([\beta]\llbracket M \rrbracket^{\alpha}_{[]}, L) \quad \textit{if } \beta \neq \alpha
\end{array}$$

$$\begin{array}{ll}
\mathsf{abstr}(x, [(x_1, N_1), L]) & = (\lambda x_1.\mathsf{abstr}(x, L))\ N_1 \\
\mathsf{abstr}(x, []) & = x \\
\mathsf{repl}(M, [(x_1, N_1), \ldots, (x_n, N_n)]) & = M[x_1 := N_1, \ldots, x_n := N_n]
\end{array}$$

Example 6.16.
The term in the type assignment $\vdash_{\lambda\varDelta} \lambda x.\varDelta y.y\ (x\ \lambda z.\nabla(y\ z)) : ((A \to B) \to A) \to A$, see example 5.2, translates in two steps, since the $\varDelta$ and ∇-operators don't occur in the left hand side of an application.

$$\begin{aligned}
\overrightarrow{\lambda x.\varDelta y.y\ (x\ \lambda z.\nabla(y\ z))} &\overset{1}{=} \lambda x.\mu\alpha.\llbracket (\lambda q.[\alpha]q)\ (x\ \lambda z.\mu\beta.(\lambda q.[\alpha]q)\ z) \rrbracket^{\alpha}_{[]} \\
&\overset{2}{=} \lambda x.\mu\alpha.[\alpha](x\ (\lambda z.\mu\beta.[\alpha]z))
\end{aligned}$$

The result is the term from example 6.1.

The term in the type assignment $\vdash_{\lambda\varDelta} \lambda y.\varDelta x.y\ x : ((A \to \bot) \to \bot) \to A$ is translated in two steps as well:

$$\begin{aligned}\overrightarrow{\lambda y.\varDelta x.y\ x} &\overset{1}{=} \lambda y.\mu\alpha.[\![y\ (\lambda q.[\alpha]q)]\!]^{\alpha}_{[]}\\ &\overset{2}{=} \lambda y.\mu\alpha.y\ (\lambda q.[\alpha]q)\end{aligned}$$

In this translation, the second step doesn't alter the term.

Lemma 6.17. $\longrightarrow$ *is a type-preserving translation.*

Proof.
The only non-trivial cases are the translations of $(\varDelta x.M)N$ and $\varDelta x.M$. For the first case, let $\varDelta x.M : A \to B$ and $N : A$ for arbitrary A and B. Then $x : (A \to B) \to \bot$ and $M : \bot$. This implies that $y : A \to B$. Thus in the command $[\alpha]y$ an active type $A \to B$ is passivated, which implies that α is the index of a passive type $A \to B$. This, with the fact that $M : \bot$, implies that the term $\mu\alpha.M$ has type $A \to B$.

For the second case the same proof holds, under assumption that the second translation step is type-preserving. To prove this assumption, it is sufficient to prove that for all N and P, the terms $(\lambda y.[\alpha]P)\ N$ and $[\alpha]P[y := N]$ have the same type. This is true, because this is like a β-reduction.
□

Lemma 6.18. $\longrightarrow$ *is a reductional embedding of $\lambda\varDelta$ into $\lambda\mu$, with exception of β-reduction.*

Proof.

$$\begin{aligned}\overrightarrow{\varDelta x.x\ M} &= \mu\alpha.[\![(x\ \overrightarrow{M})[x := \lambda y.[\alpha]y]]\!]^{\alpha}_{[]}\\ &= \mu\alpha.[\![(\lambda y.[\alpha]y)\ \overrightarrow{M}]\!]^{\alpha}_{[]}, \text{ since } x \notin FV(M) \implies x \notin FV(\overrightarrow{M})\\ &= \mu\alpha.[\alpha][\![\overrightarrow{M}]\!]^{\alpha}_{[]}\\ &\rhd_{\mu t} [\![\overrightarrow{M}]\!]^{\alpha}_{[]}, \text{ since } x \notin FV(M) \implies \alpha \notin \mu FV([\![\overrightarrow{M}]\!]^{\alpha}_{[]})\\ &= \overrightarrow{M}, \text{ since } \alpha \notin \mu FV(\overrightarrow{M}) \implies [\![\overrightarrow{M}]\!]^{\alpha}_{[]} = \overrightarrow{M}\end{aligned}$$

$$\begin{aligned}
\overrightarrow{\varDelta x.x\ \nabla(x\ M)} &= \mu\alpha.\llbracket(\overrightarrow{x\ \nabla(x\ M)})[x := \lambda y.[\alpha]y]\rrbracket^{\alpha}_{[]}\\
&= \mu\alpha.\llbracket(x\ \mu\beta.\llbracket x\ \overrightarrow{M}\rrbracket^{\alpha}_{[]})[x := \lambda y.[\alpha]y]\rrbracket^{\alpha}_{[]}\\
&= \mu\alpha.\llbracket(x\ \mu\beta.x\ \overrightarrow{M})[x := \lambda y.[\alpha]y]\rrbracket^{\alpha}_{[]}\\
&= \mu\alpha.\llbracket(\lambda y.[\alpha]y)\ \mu\beta.(\lambda y.[\alpha]y)\ \overrightarrow{M}\rrbracket^{\alpha}_{[]}, \text{since } x \notin FV(M) \implies x \notin FV(\overrightarrow{M})\\
&= \mu\alpha.[\alpha]\mu\beta.[\alpha]\llbracket\overrightarrow{M}\rrbracket^{\alpha}_{[]}\\
&\rhd_{\mu r} \mu\alpha.[\alpha]\llbracket\overrightarrow{M}\rrbracket^{\alpha}_{[]} \text{ since } \beta \notin \mu FV(\llbracket\overrightarrow{M}\rrbracket^{\alpha}_{[]})\\
&\rhd_{\mu t} \llbracket\overrightarrow{M}\rrbracket^{\alpha}_{[]}, \text{since } x \notin FV(M) \implies \alpha \notin \mu FV(\llbracket\overrightarrow{M}\rrbracket^{\alpha}_{[]})\\
&= \overrightarrow{M}, \text{since } \alpha \notin \mu FV(\overrightarrow{M}) \implies \llbracket\overrightarrow{M}\rrbracket^{\alpha}_{[]} = \overrightarrow{M}\\
\overrightarrow{(\varDelta x.M)N} &= (\mu\alpha.\overrightarrow{M}[x := \lambda y.[\alpha]y])\ \overrightarrow{N}\\
&\rhd_{\mu s} \mu\alpha.\overrightarrow{M}[x := \lambda y.[\alpha](y\ \overrightarrow{N})]\\
&= \mu\alpha.\llbracket\overrightarrow{M}[x := \lambda y.(\lambda q.[\alpha]q)\ (y\ \overrightarrow{N})]\rrbracket^{\alpha}_{[]}\\
&= \mu\alpha.\llbracket\overrightarrow{M}[x := \lambda y.z(y\ \overrightarrow{N})][z := \lambda q.[\alpha]q]\rrbracket^{\alpha}_{[]}\\
&= \mu\alpha.\llbracket\overrightarrow{M[x := \lambda y.z(y\ N)]}[z := \lambda q.[\alpha]q]\rrbracket^{\alpha}_{[]}\\
&= \overrightarrow{\varDelta z.M[x := \lambda y.z(y\ N)]}
\end{aligned}$$

However, since the translation isn't compositional, β-reduction isn't preserved. Let $I = (\lambda x.x)$ in the following counterexample:

$$\begin{aligned}
(\lambda y.y\ I)(\varDelta x.x\ I) &\rhd_{\beta} (\varDelta x.x\ I)\ I\\
\overrightarrow{(\lambda y.y\ I)(\varDelta x.x\ I)} &= (\lambda y.y\ I)(\mu\alpha.[\alpha]I)\\
&\rhd_{\beta} (\mu\alpha.[\alpha]I)\ I
\end{aligned}$$

Since $\overrightarrow{(\varDelta x.x\ I)\ I} = (\mu\alpha.(\lambda q.[\alpha]q)\ I)\ I$ and this translation is not equal to $(\mu\alpha.[\alpha]I)\ I$ nor does it reduce to this term, this term is a counterexample.

□

Lemma 6.19. $\longrightarrow$ *isn't a computational embedding of $\lambda\varDelta$ into $\lambda\mu$.*

Proof.
A counterexample is given by by the term $(\varDelta x.x\ M)N$:

$$\begin{aligned}
(\varDelta x.x\ M)N &\overset{\mathsf{n}}{\mapsto}_{\lambda\varDelta} M\ N\\
\overrightarrow{(\varDelta x.x\ M)N} &= (\mu\alpha.(\lambda y.[\alpha]y)\ \overrightarrow{M})\overrightarrow{N}\\
&\overset{\mathsf{n}}{\mapsto}_{\lambda\mu} \mu\alpha.(\lambda y.[\alpha](y\ \overrightarrow{N}))\ \overrightarrow{M}
\end{aligned}$$

□

Remark 6.20. With exception of the β-rule, the translation $\longrightarrow$ is an embedding of the notion of reduction of $\lambda\varDelta$ into the notion of reduction of $\lambda\mu$ as well, since in the proof of lemma 6.18 all occurrences of $\rhd$ can be replaced by $\rightarrow$. Still lemma 3.16 can't be used to prove that this translation is a computational embedding as well, since it doesn't preserve CBN context.

Translating μ into Δ

In order to translate from μ to $\varDelta$, definition 6.11 is used to interpret the Activate and Passivate steps:

μ derivation step	Interpretation in $\varDelta$
$\dfrac{M : \varGamma \vdash_{\lambda\mu} ; A^{\alpha}, \varSigma}{\mu\alpha.M : \varGamma \vdash_{\lambda\mu} A; \varSigma}\ \text{Activate}$	$\dfrac{\varGamma, \neg\varSigma, \alpha : \neg A \vdash_{\lambda\varDelta} M : \bot}{\varGamma, \neg\varSigma \vdash_{\lambda\varDelta} \varDelta\alpha.M : A}\ \neg\neg\,\text{E}$
$\dfrac{M : \varGamma \vdash_{\lambda\mu} A; \varSigma}{[\alpha]M : \varGamma \vdash_{\lambda\mu} ; A^{\alpha}, \varSigma}\ \text{Passivate}$	$\dfrac{\varGamma, \neg\varSigma \vdash_{\lambda\varDelta} M : A}{\varGamma, \neg\varSigma, \alpha : \neg A \vdash_{\lambda\varDelta} \alpha\ M : \bot}\ {\to}\text{E}$

This leads to a straightforward translation that consists of simply substituting μ for $\varDelta$ and $[\alpha]$ for α. These interpretations are isomorphic to the derivations used in the second part of the proof of theorem 2.4.

Definition 6.21. *The translation* $\leftarrow$ *from a term* $M \in \Lambda\mu$ *to a term* $\overleftarrow{M} \in \Lambda\varDelta$ *is inductively defined as follows:*

$\overleftarrow{x} = x$
$\overleftarrow{M\,N} = \overleftarrow{M}\,\overleftarrow{N}$
$\overleftarrow{\lambda x.M} = \lambda x.\overleftarrow{M}$
$\overleftarrow{\mu\alpha.M} = \varDelta\alpha.\overleftarrow{M}$
$\overleftarrow{[\alpha]M} = \alpha\ \overleftarrow{M}$

Lemma 6.22. $\leftarrow$ *is a type-preserving translation.*

Proof.
The proof is given by the interpretations of the Activate and Passivate derivation steps.
□

Lemma 6.23. $\leftarrow$ *is a reductional embedding of* $\lambda\mu$ *into* $\lambda\varDelta$*, with exception of the renaming rule* $\to_{\mu r}$*.*

Proof.

$$\begin{array}{lcl}
\overleftarrow{M[x := N]} & = & \overleftarrow{M}[x := \overleftarrow{N}], \text{(trivial)} \\
\overleftarrow{\mu\alpha.[\alpha]M} & = & \varDelta\alpha.\alpha\ \overleftarrow{M} \\
 & \rhd_{\varDelta 2} & \overleftarrow{M} \text{ since } \alpha \notin \mu FV(\overleftarrow{M}) \\
\overleftarrow{(\mu\alpha.M)N} & = & (\varDelta\alpha.\overleftarrow{M})\ \overleftarrow{N} \\
 & \rhd_{\varDelta 1} & \varDelta z.\overleftarrow{M}[\alpha := \lambda y.z(y\ \overleftarrow{N})] \\
 & = & \varDelta z.\overleftarrow{M}[\alpha\ P \mathrel{::==} (\lambda y.z(y\ \overleftarrow{N}))\ P], \text{ since } \alpha \text{ occurs only in an application} \\
 & \rhd\!\!\!\rhd_{\beta} & \varDelta z.\overleftarrow{M}[\alpha\ P \mathrel{::==} z(P\ \overleftarrow{N})] \\
 & = & \overleftarrow{\varDelta\alpha.M[[\alpha]P \mathrel{::==} [\alpha](P\ N)]} \\
 & = & \overleftarrow{\mu\alpha.M[[\alpha]P \mathrel{::==} [\alpha](P\ N)]}
\end{array}$$

However, the renaming rule isn't preserved. Let V be some closed value. Then a closed well-typed term that is a counterexample is:

$$\mu\beta.[\beta](\mu\alpha.[\beta](\mu\gamma.[\alpha]V)) \rhd_{\mu r} \mu\beta.[\beta](\mu\alpha.[\alpha]V)$$

This term reduces further to V. The translation of this term, $\varDelta\beta.\beta\ (\varDelta\alpha.\beta\ (\varDelta\gamma.\alpha\ \overleftarrow{V}))$ is in normal form. Only when α, β and γ are equal, the translation reduces to V as well.

□

Lemma 6.24. *$\leftarrow$ isn't a computational embedding of $\overset{\mathsf{n}}{\longmapsto}_{\lambda\mu}$ into $\overset{\mathsf{n}}{\longmapsto}_{\lambda\varDelta}$.*

Proof.
A counterexample is given by the following term:

$$\begin{array}{rcl}
(\mu\alpha.[\alpha](I\ (\mu\beta.[\alpha]I))\ I & \overset{\mathsf{n}}{\longmapsto}_{\lambda\mu} & \mu\alpha.[\alpha](I\ (\mu\beta.[\alpha](I\ I))\ I) \qquad (6.5) \\
\overline{(\mu\alpha.[\alpha](I\ (\mu\beta.[\alpha]I)))\ I} & = & (\varDelta\alpha.\alpha\ (I\ \varDelta\beta.\alpha\ I))\ I \\
 & \overset{\mathsf{n}}{\longmapsto}_{\lambda\varDelta} & \varDelta z.((\lambda y.z(y\ I))\ (I\ \varDelta\beta.(\lambda y.z(y\ I))))\ I \qquad (6.6)
\end{array}$$

Both terms (6.5) and (6.6) are values and don't reduce further under CBN operational semantics. Since (6.5) doesn't translate into (6.6), this term is a counterexample.

□

Remark 6.25. Although the translation $\leftarrow$ is compositional and it is a reductional embedding, lemma 3.16 can't be used to prove that it is a computational embedding as well, since $\leftarrow$ isn't an embedding of the notion of reduction of $\lambda\mu$ into the notion of reduction of $\lambda\varDelta$. The problem lies in the multiple β-reductions that need to occur inside a $\varDelta$-abstraction in order to simulate reduction $\rightarrow_{\mu s}$.

6.3 Altering the $\lambda\varDelta$-calculus

In this section, the $\lambda\varDelta$-calculus will be modified in such a way that it is equivalent to the $\lambda\mu$-calculus on both the reductional and the computational level.

Variating the CBN operational semantics of $\lambda\Delta$

With exception of the renaming reduction rule, it is possible to embed the CBN operational semantics of $\lambda\mu$ in the $\lambda\Delta$-calculus. This is done in definition 6.26.

Definition 6.26. *The operational semantics $\overset{n'}{\mapsto}_{\lambda\Delta}$ is defined by the following algorithm:*

Let M be the $\lambda\Delta$-term that is to be reduced:

1. *If M has no occurrences of $\underline{\lambda}$, then apply the CBN reduction $\overset{\mathsf{n}}{\mapsto}_{\lambda\Delta}$ as usual, with exception that reduction rule $\rightarrow_{\Delta 1}$ is executed as follows:*

$$(\Delta x.M)N \rightarrow_{\underline{\Delta 1}} \Delta z.M[x := \underline{\lambda} y.z(y\ N)]$$

2. *If M has occurrences of $(\underline{\lambda} x.P)P')$, then apply the reduction:*

$$M \rightarrow_{\underline{\beta}} M[(\underline{\lambda} x.P)P' ::== P[x := P']]$$

Lemma 6.27. $\leftarrow$ *is a computational embedding of $\overset{\mathsf{n}}{\mapsto}_{\lambda\mu}$ into $\overset{n'}{\mapsto}_{\lambda\Delta}$, with exception of the renaming rule $\rightarrow_{\mu r}$.*

Proof.
Since the translation preserves CBN context, the proof is exactly like the proof of lemma 6.23, with every occurrence of $\rhd$ replaced by $\overset{n'}{\mapsto}_{\lambda\Delta}$.
□

Variating the notion of reduction of $\lambda\Delta$

The idea behind the previous variation is to reduce the redexes that are created by reduction $\rightarrow_{\Delta 1}$ as soon as they are created. This idea can be accomplished as well by altering this reduction rule to a reduction rule that spontaneously reduces the created redexes. Effectively this means that the new reduction rule performs structural substitution.

Definition 6.28. *The reduction rule $\rightarrow_{\underline{\Delta 1}}$ is defined as:*

$$(\Delta x.M)N \rightarrow_{\underline{\Delta 1}} (\Delta z.M[x := \underline{\lambda} y.z(y\ N)])^*$$

Here $(M)^*$ is defined as:

$$\begin{aligned}
(x)^* &= x\\
(\lambda x.M)^* &= \lambda x.(M)^*\\
(\underline{\lambda} x.M)^* &= \lambda x.(M)^*\\
(\mu\alpha.M)^* &= \mu\alpha.(M)^*\\
(M\ N)^* &= (M)^*\ (N)^*, \textit{ if } M \not\equiv \underline{\lambda} x.P\\
((\underline{\lambda} x.P)N)^* &= (P[x := N])^*
\end{aligned}$$

In order to get reductional equivalence for renaming as well, the renaming reduction rule of $\lambda\varDelta$ must be altered. The new reduction rule $\rightarrow_{\underline{\varDelta 3}}$ is a more general instance of the old version. The resulting new notion of reduction $\rightarrow_{\underline{\lambda}\varDelta}$ is given in table 6.6.

Definition 6.29. *The reduction rule $\rightarrow_{\underline{\varDelta 3}}$ is defined as:*

$$\varDelta x.y\ \varDelta z.M \rightarrow_{\underline{\varDelta 3}} \varDelta x.M[z := y],\ \textit{with } y \textit{ a } \varDelta\textit{-variable}$$

Definition 6.30. *The translation $\overline{}^{*}$ from a term $M \in \Lambda\varDelta$ to a term $\overline{M}^{*} \in \Lambda\mu$ is defined as:*

$$\begin{aligned}
\overline{x}^{*} &= x\\
\overline{\lambda x.M}^{*} &= \lambda x.\overline{M}^{*}\\
\overline{M\,N}^{*} &= \overline{M}^{*}\ \overline{N}^{*}\\
\overline{\varDelta x.M}^{*} &= (\mu\alpha.\overline{M}^{*}[x := \underline{\lambda}y.[\alpha]y])^{*}
\end{aligned}$$

Here $(M)^{*}$ is defined as in definition 6.28.

$\rightarrow_{\underline{\lambda}\varDelta}$	$\rightarrow_{\lambda\mu}$
$(\lambda x.M)N \rightarrow_{\beta} M[x := N]$ $(\varDelta x.M)N \rightarrow_{\varDelta 1} (\varDelta z.M[x := \underline{\lambda}y.z(y\ N)])*$ $\varDelta x.y\ \varDelta z.M \rightarrow_{\varDelta 3} \varDelta x.M[z := y]$ $(\varDelta x.xM) \rightarrow_{\varDelta 2} M$, if $x \notin FV(M)$	$(\lambda x.M)N \rightarrow_{\beta} M[x := N]$ $(\mu\alpha.M)N \rightarrow_{\mu s} \mu\alpha.M[[\alpha]P ::== [\alpha](PN)]$ $\mu\alpha.[\beta]\mu\gamma.M \rightarrow_{\mu r} \mu\alpha.M[\gamma := \beta]$ $\mu\alpha.[\alpha]M \rightarrow_{\mu t} M$, if $\alpha \notin \mu FV(M)$

Table 6.6. Notion of reduction of $\underline{\lambda}\varDelta$ and $\lambda\mu$

Lemma 6.31. $\overline{}^{*}$ *is a reductional embedding of $\underline{\lambda}\varDelta$ into $\lambda\mu$.*

Proof.
The proof is straightforward, except for the rule $\rightarrow_{\underline{\varDelta 1}}$:

$$\begin{aligned}
\overline{(\varDelta x.M)N}^{*} \quad &= \ (\mu\alpha.\overline{M}^{*}[x := \underline{\lambda}y.[\alpha]y])^{*}\ \overline{N}^{*}\\
&= \ (\mu\alpha.\overline{M}^{*}[x\ \_ := [\alpha](\_)][x := \lambda y.[\alpha]y])\ \overline{N}^{*}\\
&\rhd_{\mu s}\ \mu\alpha.\overline{M}^{*}[x\ \_ := [\alpha](\_\ \overline{N}^{*})][x := \lambda y.[\alpha](y\ \overline{N}^{*})]\\
&= \ \overline{\varDelta z.M[x\ \_ := z(\_\ N)][x := \lambda y.z(y\ N)]}^{*}\\
&= \ \overline{(\varDelta z.M[x := \lambda y.[\alpha](y\ N])*}^{*}
\end{aligned}$$

□

Lemma 6.32. $\overline{}^{*}$ *is a computational embedding of $\overset{\mathsf{n}}{\mapsto}_{\underline{\lambda}\varDelta}$ into $\overset{\mathsf{n}}{\mapsto}_{\lambda\mu}$.*

Proof.
Since the translation preserves CBN context, the proof is exactly like the proof of lemma 6.31, with every occurrence of $\rhd$ replaced by $\overset{\mathsf{n}}{\mapsto}_{\underline{\lambda\Delta}}$.
□

Lemma 6.33. $\overleftarrow{}$ *is a reductional embedding of $\lambda\mu$ into $\underline{\lambda\Delta}$.*

Proof.
The proof is straightforward.
□

Lemma 6.34. $\overleftarrow{}$ *is a computational embedding of $\overset{\mathsf{n}}{\mapsto}_{\lambda\mu}$ into $\overset{\mathsf{n}}{\mapsto}_{\underline{\lambda\Delta}}$.*

Proof.
Since the translation preserves CBN context, the proof is straightforward.
□

6.4 Conclusion

Theorem 6.35. *The $\lambda\Delta$-calculus and the $\lambda\mu$-calculus are type-equivalent. The modified $\underline{\lambda\Delta}$ and $\lambda\mu$ are reductionally and computationally equivalent.*

Proof.
$\lambda\Delta \equiv_t \lambda\mu$ This follows from theorems 2.4, 5.1 and 6.3.
$\underline{\lambda\Delta} \equiv_r \lambda\mu$ This follows from lemmas 6.31 and 6.33.
$\underline{\lambda\Delta} \equiv_c \lambda\mu$ This follows from lemma 6.32 and 6.34.
□

It seems that $\lambda\mu$ with included the notion of renaming is reductionally stronger than $\lambda\Delta$, and that without this reduction rule, it is weaker. In order to prove this hypothesis completely, it must either be proven that renaming is necessary to embed $\lambda\Delta$ into $\lambda\mu$, that is, it must be proven that $\lambda\Delta \nleq_r \lambda\mu_{\backslash\mu r}$, or it must be proven that the renaming reduction rule of $\lambda\mu$ can never be embedded in $\lambda\Delta$, that is, $\lambda\mu \nleq_c \lambda\Delta$. In order to prove this hypothesis wrong, it must be proven that renaming can be embedded in $\lambda\Delta$ and that thus the two calculi are reductionally equivalent, that is, it must be proven that $\lambda\mu \leq_r \lambda\Delta$.

It also seems that $\lambda\Delta$ and $\lambda\mu$ are computationally inequivalent. Intuitively this hypothesis seems true: an application of a Δ-abstraction creates λ-abstractions inside a value, which cause the creation of extra β-redexes. These redexes don't reduce further, since they are inside a value. An application of a μ-abstraction structurally substitutes inside the μ-abstraction. This structural substitution can be seen as simultaneously creating and contracting β-redexes inside a value.

7 The $\bar{\lambda}\mu$-calculus

The set of $\bar{\lambda}\mu$-terms is defined by three syntactic categories [1]:

Commands	$c ::= \langle v \mid E \rangle$
Contexts	$E ::= \alpha \mid v \cdot E$
Terms	$v ::= x \mid \mu\beta.c \mid \lambda x.v$

A *term* represents a program. Note the difference between $\bar{\lambda}\mu$-terms, denoted by $M, N, \ldots$ and the syntactic category terms denoted by $v, v', \ldots$. A *context* represents the environment of a program. A *command* represents a closed system containing both the program and the environment [1]. The command $\langle v \mid \alpha \rangle$ is the same as the command $[\alpha]v$ of the $\lambda\mu$-calculus. The command $\langle v \mid E \rangle$ can be read as $E[v]$. In section 7.1 the intuition behind the $\lambda\mu$-calculus will be extended to the $\bar{\lambda}\mu$-calculus.

Each syntactic category comes with its own type. All types are sequents. They can either have one active formula on the right hand side, one active formula on the left hand side, or no active formula at all. A command is typed by a sequent with on both sides a set of passive formulas and thus without an active formula. A context is typed by a sequent with one active type on the left. A term is, as in the $\lambda\mu$-calculus, typed by a sequent with an active formula on the right. The typing assignments are denoted as follows:

$$\begin{aligned} c \quad &: \quad (\Gamma \vdash_{\bar{\lambda}\mu} \Sigma) \\ \Gamma \mid E : A \ &\vdash_{\bar{\lambda}\mu} \ \Sigma \\ \Gamma \ &\vdash_{\bar{\lambda}\mu} \ v : A \mid \Sigma \end{aligned}$$

An active type is thus separated from the passive types by a $|$. The inference system is given in table 7.1. Structural rules of weakening and contraction are done just as in the $\lambda\mu$-calculus.

$\bar{\lambda}\mu$

$$\frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Sigma}\ \text{axiom-l} \qquad\qquad \frac{}{\Gamma, x : A \vdash x : A \mid \Sigma}\ \text{axiom-r}$$

$$\frac{\Gamma \vdash v : A \mid \Sigma \qquad \Gamma \mid E : B \vdash \Sigma}{\Gamma \mid v \cdot E : A \to B \vdash \Sigma}\ \to\text{l} \qquad\qquad \frac{\Gamma, x : A \vdash v : B \mid \Sigma}{\Gamma \vdash \lambda x.v : A \to B \mid \Sigma}\ \to\text{r}$$

$$\frac{c : (\Gamma \vdash \beta : B, \Sigma)}{\Gamma \vdash \mu\beta.c : B \mid \Sigma}\ \text{Activate-r}$$

$$\frac{\Gamma \vdash v : A \mid \Sigma \qquad \Gamma \mid E : A \vdash \Sigma}{\langle v \mid E \rangle : (\Gamma \vdash \Sigma)}\ \text{cut}$$

Table 7.1. Rules for the $\bar{\lambda}\mu$-calculus

Example 7.1. As an example of a derivation, Peirces Law is derived:

$$\dfrac{\dfrac{}{\Gamma \vdash \Gamma \mid \alpha : A}\,\text{ax-r} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, y : A \vdash y : A \mid \beta : B, \alpha : A}\,\text{ax-r} \quad \dfrac{}{\Gamma, y : A \mid \alpha : A \vdash \beta : B, \alpha : A}\,\text{ax-l}}{\langle y \mid \alpha \rangle : (\Gamma, y : A \vdash \beta : B, \alpha : A)}\,\text{cut}}{\Gamma, y : A \vdash \mu\beta.\langle y \mid \alpha \rangle : B \mid \alpha : A}\,\text{Activate-r}}{\Gamma \vdash \lambda y.\mu\beta.\langle y \mid \alpha \rangle : A \to B \mid \alpha : A}\,\to\text{r} \qquad \dfrac{}{\Gamma \mid \alpha : A \vdash \alpha : A}\,\text{ax-l}}{\Gamma \mid (\lambda y.\mu\beta.\langle y \mid \alpha \rangle) \cdot \alpha : (A \to B) \to A \vdash \alpha : A}\,\to\text{l}}{\dfrac{\dfrac{\langle x \mid (\lambda y.\mu\beta.\langle y \mid \alpha \rangle) \cdot \alpha \rangle : (\Gamma \vdash \alpha : A)}{\Gamma \vdash \mu\alpha.\langle x \mid (\lambda y.\mu\beta.\langle y \mid \alpha \rangle) \cdot \alpha \rangle : A \mid}\,\text{Activate-r}}{\vdash \lambda x.\mu\alpha.\langle x \mid (\lambda y.\mu\beta.\langle y \mid \alpha \rangle) \cdot \alpha \rangle : ((A \to B) \to A) \to A \mid}\,\to\text{r}}\,\text{cut}$$

Here $\Gamma = x : (A \to B) \to A$.

Since a command $\langle y \mid \alpha \rangle$ can be read as $[\alpha]y$ and the command $\langle x \mid N \cdot \alpha \rangle$ can be read as $[\alpha](x\ N)$, the term from this example corresponds exactly to the term of example 6.1. In section 7.2 a formal translation between $\lambda\mu$ and $\bar{\lambda}\mu$-terms will be defined.

In [16], an interpretation of $\bar{\lambda}\mu$-terms is defined in order to render $\bar{\lambda}\mu$-terms in natural logical language. In this paper an adaption of this interpretation provided by Mamane will be used. Mamane added a stack that is used to remember the formulae that can be proven.

Definition 7.2. *Let π denote a stack of formulae and let $\diamond$ denote an empty stack. Adding a formula A to π is denoted by $A \cdot \pi$. Let $\vee(\pi)$ be defined as the disjunction of the formulae from π. The Sacerdoti-Mamane-interpretation of a $\bar{\lambda}\mu$-term M is defined as $[\![M]\!]_\diamond$, where $[\![M]\!]_\pi$ is defined as:*

$$\begin{aligned}
[\![\langle v \mid E \rangle]\!]_\pi &\overset{def}{=} [\![v]\!]_\pi\ [\![E]\!]_\pi \\
[\![x]\!]_\pi &\overset{def}{=} \texttt{by}\ x \\
[\![\lambda x : A.M]\!]_\pi &\overset{def}{=} \texttt{suppose}\ A\ (x) \\
&\qquad [\![M]\!]_\pi \\
[\![\mu\alpha : A.\langle v \mid \beta : B \rangle]\!]_\pi &\overset{def}{=} \texttt{we need to prove}\ \vee(A \cdot \pi) \\
&\qquad \texttt{we now prove}\ B \\
&\qquad [\![v]\!]_{A \cdot \pi} \\
[\![\mu\alpha : A.\langle v \mid E \cdot (\beta : B) \rangle]\!]_\pi &\overset{def}{=} \texttt{we need to prove}\ \vee(A \cdot \pi) \\
&\qquad \texttt{we now prove}\ B \\
&\qquad [\![v]\!]_{A \cdot \pi}\ \texttt{and}\ [\![E]\!]_{A \cdot \pi}
\end{aligned}$$

The Sacerdoti-interpretation of the term from example 7.1 is as follows:

`suppose` $(A \to B) \to A$ (x)
`we need to prove` A
    `we now prove` A

```
by $x$ and suppose $A$ ($y$)
we need to prove $B \vee A$
    we now prove $A$
    by $y$
```

Theorem 7.3. *A sequent $\Gamma \vdash A; \Sigma$ is derivable in $\mathsf{LK} \setminus (\bot E)$ (minimal classical sequent calculus) iff there exists a closed $\bar{\lambda}\mu$-term M such that $\Gamma \vdash_{\bar{\lambda}\mu} M : A \mid \Sigma$ is derivable.*

Originally there were no rules of the $\bar{\lambda}\mu$-calculus to handle $\bot$. For the $\bar{\lambda}\mu\tilde{\mu}$-calculus, these rules will be discussed in section 8.1.

7.1 Notion of reduction and operational semantics

Definition 7.4. *The notion of reduction $\rightarrow_{\bar{\lambda}\mu}$ is defined as the union of the rules [1]:*

$$\langle \lambda x.v \mid v' \cdot E \rangle \rightarrow_\beta \langle v[x := v'] \mid E \rangle$$
$$\langle \mu\alpha.c \mid E \rangle \rightarrow_\mu c[\alpha := E]$$

The intuition behind the $\lambda\mu$-calculus is that a term $[\alpha]M$ can be read as binding the name α to the subterm M. If a μ-abstraction $\mu\alpha.M$ is given an argument N, then this argument is passed to every subterm named α. The same intuition holds for the $\bar{\lambda}\mu$-calculus. A command has always the following form: $\langle v \mid v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot \alpha \rangle$, with α some μ-variable. A command whose context ends with α can be seen as a command named α. Now consider a command $\langle \mu\alpha.c \mid E \rangle$. This command passes the terms from E as arguments to the μ-abstraction. This means that inside the body c of the μ-abstraction, every occurrence of α is replaced by the terms from E. In other words, the arguments that are passed to the μ-abstraction are actually passed to every subcommand named α in its body c.

In a command $\langle v \mid E \rangle$, the $\bar{\lambda}\mu$-context E defines the environment of the term of a command. Thereby it defines a CBN reduction path for the command. A $\bar{\lambda}\mu$-command is actually an extended variant of a so called *Krivine abstract machine*. Definition 7.5 defines these machines for simply typed λ-calculus.

Definition 7.5. *Let M denote a simply typed λ-term and let π denote a stack of these terms. Adding a term N to a stack π is denoted by $N \cdot \pi$. A Krivine abstract machine contains a term and a stack and is denoted by $(M\ ,\ \pi)$. Evaluation is defined as the union of the following rules:*

$$(MN\ ,\ \pi) \overset{push}{\rightarrow} (M\ ,\ N \cdot \pi)$$
$$(\lambda x.M\ ,\ N \cdot \pi) \overset{grab}{\rightarrow} (M[x := N]\ ,\ \pi)$$

A $\bar{\lambda}\mu$-command thus specifies its own operational semantics. An application $M\ N$ is encoded by a μ-abstraction: $\mu\alpha.\langle M \mid N \cdot \alpha \rangle$. However, a $\bar{\lambda}\mu$-term can contain more than one $\bar{\lambda}\mu$-command, in which case reduction is nondeterministic, as in example 7.6.

Example 7.6. The $\lambda\mu$-term $M = (\lambda x.(\lambda y.y)\ N)\ P$, with $x \notin FV(P)$ has two possible reduction paths. Let $N_{\bar{\lambda}\mu}$ and $P_{\bar{\lambda}\mu}$ be $\bar{\lambda}\mu$-terms corresponding to respectively N and P. The $\bar{\lambda}\mu$-term $M_{\bar{\lambda}\mu}$ corresponding to M is $\mu\alpha.\langle\lambda x.\mu\beta.\langle\lambda y.y \mid N_{\bar{\lambda}\mu}\cdot\beta\rangle \mid P_{\bar{\lambda}\mu}\cdot\alpha\rangle$. This term has more than two reduction paths, but only the two analogous reduction paths are given. The first reduction path that is given, is CBN.

$$\begin{aligned}(\lambda x.(\lambda y.y)\ N)\ P &\rhd (\lambda y.y)N \rhd N && \text{(I)}\\ &\rhd (\lambda x.N)N' \rhd N && \text{(II)}\end{aligned}$$

$$\begin{aligned}\mu\alpha.\langle\lambda x.\mu\beta.\langle\lambda y.y \mid N_{\bar{\lambda}\mu}\cdot\beta\rangle \mid P_{\bar{\lambda}\mu}\cdot\alpha\rangle &\rhd \mu\alpha.\langle\mu\beta.\langle\lambda y.y \mid N_{\bar{\lambda}\mu}\cdot\beta\rangle \mid \alpha\rangle && \text{(I)}\\ &\rhd \mu\alpha.\langle\lambda y.y \mid N_{\bar{\lambda}\mu}\cdot\alpha\rangle\\ &\rhd \mu\alpha.\langle N_{\bar{\lambda}\mu} \mid \alpha\rangle\\ \mu\alpha.\langle\lambda x.\mu\beta.\langle\lambda y.y \mid N_{\bar{\lambda}\mu}\cdot\beta\rangle \mid P_{\bar{\lambda}\mu}\cdot\alpha\rangle &\rhd \mu\alpha.\langle\lambda x.\mu\beta.\langle N_{\bar{\lambda}\mu} \mid \beta\rangle \mid N'_{\bar{\lambda}\mu}\cdot\alpha\rangle && \text{(II)}\\ &\rhd \mu\alpha.\langle\mu\beta.\langle N_{\bar{\lambda}\mu} \mid \beta\rangle \mid \alpha\rangle\\ &\rhd \mu\alpha.\langle N_{\bar{\lambda}\mu} \mid \alpha\rangle\end{aligned}$$

In [1], there is no operational semantics defined for $\bar{\lambda}\mu$-terms. A CBN operational semantics is easily defined, since the CBN strategy simply takes the left most application and never reduces inside a λ-abstraction or in an inner μ-abstraction. A $\bar{\lambda}\mu$-term M is thus only CBN-reducible, if it has the form $\mu\alpha.R$, with $R = \langle v \mid E\cdot\alpha\rangle$.

Definition 7.7. *The CBN operational semantics $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu}$ is defined by three syntactic classes:*

Values	$\mathbf{V}_{\bar{\lambda}\mu} ::= x \mid \lambda x.M \mid \mu\alpha.M \textit{ if } M \notin \mathbf{R}_{\bar{\lambda}\mu}$
Evaluation contexts	$\mathbf{E}_{\bar{\lambda}\mu} ::= \mu\alpha.[]$
Redexes	$\mathbf{R}_{\bar{\lambda}\mu} ::= \langle\lambda x.M \mid N\cdot E\rangle \mid \langle\mu\alpha.c \mid E\rangle$

Lemma 7.8. *The operational semantics $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu}$ is a well-defined operational semantics.*

Proof.
A closed $\bar{\lambda}\mu$-term is a term. It can't be a context, since every context has the form $\ldots\cdot\alpha$, with α unbound. It can't be a command, since thus the context in every command contains an unbound variable. A closed $\bar{\lambda}\mu$-term is thus either a λ-abstraction, in which case it is a value, or a μ-abstraction. In the second case, let it be $\mu\alpha.\langle M \mid E\rangle$. There are two possibilities:

1. The command $\langle M \mid E\rangle$ is a redex and $E = N_1\cdot\ldots\cdot N_n\cdot\alpha$, in which case the $\bar{\lambda}\mu$-term can be written as $\mathbf{E}_{\bar{\lambda}\mu}[R]$, with $\mathbf{E}_{\bar{\lambda}\mu} = \mu\alpha.[]$ and $R = \langle M \mid E\rangle$;
2. otherwise, in which case the $\bar{\lambda}\mu$-term is a value.

□

7.2 The $\lambda\mu$-calculus and the $\bar{\lambda}\mu$-calculus

In [1], the following translation is defined:

Definition 7.9. *The translation $^{\mathcal{N}}$ from a term $M \in \Lambda\mu$ to a term $M^{\mathcal{N}} \in \bar{\lambda}\mu$ is defined as follows:*

$$\begin{array}{ll}
x^{\mathcal{N}} & = x \\
(\lambda x.M)^{\mathcal{N}} & = \lambda x.M^{\mathcal{N}} \\
(M\ N)^{\mathcal{N}} & = \mu\alpha.(M\ N)^{\mathcal{N}}_{\alpha} \textit{ for } \alpha \textit{ fresh} \\
(\mu\beta.c)^{\mathcal{N}} & = \mu\beta.c^{\mathcal{N}} \\
([\alpha]M)^{\mathcal{N}} & = M^{\mathcal{N}}_{\alpha} \\
 & \\
(M\ N)^{\mathcal{N}}_{E} & = M^{\mathcal{N}}_{N^{\mathcal{N}}\cdot E} \\
V^{\mathcal{N}}_{E} & = \langle V^{\mathcal{N}} \mid E\rangle \textit{ where } V = x \mid \lambda x.M \mid \mu\alpha.M
\end{array}$$

This translation can be inversed, which leads to the following translation:

Definition 7.10. *The translation $^{\mathcal{N}^{-1}}$ from a term $M \in \overline{\Lambda}\mu$ to a term $M^{\mathcal{N}^{-1}} \in \Lambda\mu$ is defined as follows:*

$$\begin{array}{ll}
x^{\mathcal{N}^{-1}} & = x \\
(\lambda x.M)^{\mathcal{N}^{-1}} & = \lambda x.M^{\mathcal{N}^{-1}} \\
(\mu\alpha.\langle v \mid \beta\rangle)^{\mathcal{N}^{-1}} & = \mu\alpha.[\beta]v^{\mathcal{N}^{-1}} \\
(\mu\alpha.\langle v \mid v_1 \cdot \ldots \cdot v_n \cdot \alpha\rangle)^{\mathcal{N}^{-1}} & = v^{\mathcal{N}^{-1}}\ v_1^{\mathcal{N}^{-1}} \ldots v_n^{\mathcal{N}^{-1}} \\
(\mu\alpha.\langle v \mid v_1 \cdot \ldots \cdot v_n \cdot \beta\rangle)^{\mathcal{N}^{-1}} & = \mu\alpha.[\beta](v^{\mathcal{N}^{-1}}\ v_1^{\mathcal{N}^{-1}} \ldots v_n^{\mathcal{N}^{-1}}) \textit{, with } \beta \neq \alpha
\end{array}$$

Example 7.11. The terms from examples 6.1 and 7.1 translate into each other.

Definition 7.12. *Let the equality $=_{\mu}$ for $\bar{\lambda}\mu$-terms be defined as follows:*

$$M =_{\mu} N \Leftrightarrow \begin{cases} M \equiv N \\ M \equiv \mu\beta.\langle N \mid \beta\rangle, \textit{ with } M =_{\mu} N \textit{ and } \beta \textit{ fresh} \end{cases}$$

Let the equality $=_{\mu}$ for $\lambda\mu$-terms be defined as follows:

$$M =_{\mu} N \Leftrightarrow \begin{cases} M \equiv N \\ M \equiv \mu\beta.[\beta]N, \textit{ with } M =_{\mu} N \textit{ and } \beta \textit{ fresh} \end{cases}$$

Lemma 7.13. *For all terms $M \in \Lambda\mu$ and $N \in \overline{\Lambda}\mu$ holds:*

$$\left((M)^{\mathcal{N}}\right)^{\mathcal{N}^{-1}} =_{\mu} M$$

$$\left((N)^{\mathcal{N}^{-1}}\right)^{\mathcal{N}} =_{\mu} N$$

Proof.
The proof is straightforward. It is done by induction on the structure of respectively M and N.
□

The reduction step $\rightarrow_\mu$ of the $\bar{\lambda}\mu$-calculus passes an entire context to the μ-abstraction in one step. The reduction step $\rightarrow_{\mu s}$ of the $\lambda\mu$-calculus passes one term per step to the μ-abstraction. Therefor, in order to achieve reductional and computational equivalence between the two calculi, the $\rightarrow_\mu$ rule is decomposed in smaller steps.

Definition 7.14. *The notion of reduction $\rightarrow_{\bar{\lambda}\mu'}$ is defined as the union of the rules [1]:*

$$\begin{aligned} \langle \lambda x.v \mid v' \cdot E \rangle &\rightarrow_\beta \langle v[x := v'] \mid E \rangle \\ \langle \mu\alpha.c \mid v \cdot E \rangle &\rightarrow_{\mu 1} \langle \mu\alpha.(c[\alpha := v \cdot \alpha]) \mid E \rangle \\ \langle \mu\alpha.c \mid \beta \rangle &\rightarrow_{\mu 2} c[\alpha := \beta] \end{aligned}$$

The operational semantics $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu'}$ is defined as in definition 7.7, using the notion of reduction $\rightarrow_{\bar{\lambda}\mu'}$.

Lemma 7.15. *Let $\lambda\mu-$ denote the $\lambda\mu$-calculus without rules to handle $\perp$. The translation ${}^{\mathcal{N}}$ is a reductional embedding, both ways, between $\lambda\mu-$ and $\bar{\lambda}\mu$, with respect to $\rightarrow_{\bar{\lambda}\mu'}$.*

Proof.
In [1] it is proven that ${}^{\mathcal{N}}$ is a bijection that preserves this notion of reduction step by step both ways.
□

Lemma 7.16. *The translation ${}^{\mathcal{N}}$ is a computational embedding of the CBN operational semantics of $\lambda\mu-$ into $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu'}$, with respect to the equality $=_\mu$. The inverse translation is a computational embedding the other way around.*

Proof.
In this proof $(M)^{\mathcal{N}}$ will be denoted by $\overrightarrow{M}$ and $(M)^{\mathcal{N}^{-1}}$ by $\overleftarrow{M}$, for the sake of clarity.

$$(\Longrightarrow):\ M \overset{\mathsf{n}}{\mapsto}_{\lambda\mu} N \implies \overrightarrow{M} \overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu} \overrightarrow{N}$$

$$\begin{array}{lcl}
\overrightarrow{(\lambda x.M)N} & = & \mu\beta.\langle \lambda x.\overrightarrow{M} \mid \overrightarrow{N} \cdot \beta \rangle \\
 & \overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu'} & \mu\beta.\langle \lambda x.\overrightarrow{M}[x := \overrightarrow{N}] \mid \beta \rangle \\
 & =_\mu & \overrightarrow{\lambda x.M[x := N]} \\
\overrightarrow{\mu\alpha.[\alpha]M} & = & \mu\alpha.\langle \overrightarrow{M} \mid \alpha \rangle \\
 & =_\mu & \overrightarrow{M} \\
\overrightarrow{(\mu\alpha.M)N} & = & \mu\beta.\langle \mu\alpha.\overrightarrow{M} \mid \overrightarrow{N} \cdot \beta \rangle \\
 & \overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu'} & \mu\beta.\langle \mu\alpha.(\overrightarrow{M}[\alpha := \overrightarrow{N} \cdot \alpha]) \mid \beta \rangle \\
 & =_\mu & \overrightarrow{\mu\alpha.M[[\alpha]P ::== [\alpha](P\ N)]}, \text{ by lemma 7.17} \\
\overrightarrow{\mu\alpha.[\gamma]\mu\beta.M} & = & \mu\alpha.\langle \mu\beta.\overrightarrow{M} \mid \gamma \rangle \\
 & \overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu'} & \overrightarrow{\mu\alpha.M[\beta := \gamma]}
\end{array}$$

$(\Longleftarrow):\ M \overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu} N \implies \overleftarrow{M} \overset{\mathsf{n}}{\mapsto}_{\lambda\mu} \overleftarrow{N}$

The CBN operational semantics always reduce the outermost command of a closed term. The CBN redex therefor always has the form $\mu\alpha.\langle v \mid v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot \alpha\rangle$. The row of arguments $v_2 \cdot \ldots \cdot v_n$ will be denoted by E.

$$\begin{array}{lll}
\overleftarrow{\mu\alpha.\langle\lambda x.v \mid v_1 \cdot E \cdot \alpha\rangle} & = & (\lambda x.\overleftarrow{v})\overleftarrow{v_1}\ \overleftarrow{E} \\
 & \overset{\mathsf{n}}{\mapsto}_{\lambda\mu} & \overleftarrow{v}[x := \overleftarrow{v_1}]\ \overleftarrow{E} \\
 & = & \overleftarrow{\mu\alpha.\langle v[x := v_1] \mid E \cdot \alpha\rangle} \\
\overleftarrow{\mu\alpha.\langle\mu\beta.c \mid v_1 \cdot E \cdot \alpha\rangle} & = & (\mu\beta.\overleftarrow{c})\overleftarrow{v_1}\ \overleftarrow{E} \\
 & \overset{\mathsf{n}}{\mapsto}_{\lambda\mu} & (\mu\beta.\overleftarrow{c}[[\beta]P ::== [\beta](P\ v_1)])\ \overleftarrow{E} \\
 & = & (\mu\beta.\overleftarrow{c[\langle P \mid \ldots \cdot \beta\rangle ::== \langle P \mid \ldots \cdot v_1 \cdot \beta\rangle]})\ \overleftarrow{E} \\
 & = & (\mu\beta.\overleftarrow{c[\beta := v_1 \cdot \beta]})\ \overleftarrow{E} \\
 & = & \overleftarrow{\mu\alpha.\langle\mu\beta.c[\beta := v_1 \cdot \beta] \mid E \cdot \alpha\rangle} \\
\overleftarrow{\mu\alpha.\langle\mu\beta.c \mid \alpha\rangle} & = & \mu\alpha.[\alpha]\mu\beta.\overleftarrow{c} \\
 & \overset{\mathsf{n}}{\mapsto}_{\lambda\mu} & \mu\alpha.\overleftarrow{c}[\beta := \alpha] \\
 & = & \overleftarrow{\mu\alpha.c[\beta := \alpha]}
\end{array}$$

□

Lemma 7.17. *Let M^* denote $M[[\alpha]P ::== [\alpha](P\ N)]$. Then for all terms M and N:*

$$(M^*)^{\mathcal{N}} = M^{\mathcal{N}}[\alpha := N^{\mathcal{N}} \cdot \alpha]$$

Proof.
The proof is by induction on the structure of M. The only non-trivial case is $M = [\beta]P$ for some μ-variable β. Two different cases arise:

$M = [\alpha]P$:

$$\begin{aligned}
(M^*)^{\mathcal{N}} &= ([\alpha](P^*\ N))^{\mathcal{N}} \\
&= \langle (P^*)^{\mathcal{N}} \mid N^{\mathcal{N}} \cdot \alpha\rangle \\
&= \langle P^{\mathcal{N}}[\alpha := N^{\mathcal{N}} \cdot \alpha] \mid N^{\mathcal{N}} \cdot \alpha\rangle, \text{ by IH} \\
&= \langle P^{\mathcal{N}} \mid \alpha\rangle[\alpha := N^{\mathcal{N}} \cdot \alpha] \\
&= M^{\mathcal{N}}[\alpha := N^{\mathcal{N}} \cdot \alpha]
\end{aligned}$$

$M = [\beta]P$:

$$\begin{aligned}
(M^*)^{\mathcal{N}} &= ([\beta]P^*)^{\mathcal{N}} \\
&= \langle (P^*)^{\mathcal{N}} \mid \beta\rangle \\
&= \langle P^{\mathcal{N}}[\alpha := N^{\mathcal{N}} \cdot \alpha] \mid N^{\mathcal{N}} \cdot \beta\rangle, \text{ by IH} \\
&= \langle P^{\mathcal{N}} \mid \beta\rangle[\alpha := N^{\mathcal{N}} \cdot \alpha] \\
&= M^{\mathcal{N}}[\alpha := N^{\mathcal{N}} \cdot \alpha]
\end{aligned}$$

□

Theorem 7.18. *Let $\lambda\mu-$ denote the minimal part of the $\lambda\mu$-calculus. $\lambda\mu-$ and $\bar{\lambda}\mu$ are type-equivalent, reductionally equivalent and computationally equivalent.*

Proof.

$\lambda\mu- \equiv_t \bar{\lambda}\mu$	This follows from theorems 6.2 and 7.3.
$\lambda\mu- \equiv_r \bar{\lambda}\mu$	This follows from lemma 7.15.
$(\overset{n}{\mapsto}_{\lambda\mu}) \equiv_c (\overset{n}{\mapsto}_{\bar{\lambda}\mu})$	This follows lemma 7.16

□

8 The $\bar{\lambda}\mu\tilde{\mu}$-calculus

In this section, two notions will be added to $\bar{\lambda}\mu$, which will result in the complete $\bar{\lambda}\mu\tilde{\mu}$-calculus. In section 8.2 a complete overview of the syntactic categories and the typing system will be given and in section 8.1 the complete notion of reduction will be given.

8.1 Adding $\tilde{\mu}$

So far, a command only reduces its term CBN. In order to add CBV reduction, the construct $\tilde{\mu}$ is added. Suppose the term $M\ N$ must be computed. A CBV discipline says to first reduce N to a value, before resuming with M. A common notation for such behaviour is let $x = N$ in $M\ x$. If N is replaced by a hole [], such a construct is a context for the term N, since first N must be reduced to V and then computation must continue with let $x = V$ in $M\ x$. The operator $\tilde{\mu}$ builds such a context. The context $\tilde{\mu}x.\langle M \mid E\rangle$ can be read as let $x =$ [] in M, evaluated in the context E.

In table 8.1 the typing rule for $\tilde{\mu}$ is added. Furthermore, a new reduction rule is added and the notion of β-reduction is altered. The new notion of reduction is:

$$\begin{aligned}
\langle \lambda x.v \mid v' \cdot e\rangle &\rightarrow_{\beta'} \langle v' \mid \tilde{\mu}x.\langle v \mid e\rangle\rangle \\
\langle \mu\alpha.c \mid E\rangle &\rightarrow_{\mu} c[\alpha := E] \\
\langle v \mid \tilde{\mu}x.c\rangle &\rightarrow_{\tilde{\mu}} c[x := v]
\end{aligned}$$

This means that a command no longer has a unique reduction path. Figure 8.1 gives an example of a $\bar{\lambda}\mu\tilde{\mu}$-term that contains a command that has two possible reduction paths. However CBV and CBN operational semantics for $\bar{\lambda}\mu\tilde{\mu}$-commands is easily defined.

Definition 8.1. *The CBV operational semantics $\overset{v}{\mapsto}_{\bar{\lambda}\mu\tilde{\mu}}$ consists in giving priority to the μ-redexes, whereas the CBN operational semantics $\overset{n}{\mapsto}_{\bar{\lambda}\mu\tilde{\mu}}$ consists in giving priority to the $\tilde{\mu}$-redexes.*

$$\begin{tikzcd}[column sep=tiny] & \mu\alpha.\langle\mu\beta.\langle M \mid \alpha\rangle \mid \tilde{\mu}x.\langle N \mid \alpha\rangle\rangle \arrow[dl, "\text{CBV}"'] \arrow[dr, "\text{CBN}"] & \\ \mu\alpha.\langle M' \mid \alpha\rangle & & \mu\alpha.\langle N' \mid \alpha\rangle \\ M' = M[\beta := \tilde{\mu}.\langle N \mid \alpha\rangle] & & N' = N[x := \mu\beta.\langle M \mid \alpha\rangle] \end{tikzcd}$$

Fig. 8.1. Reduction of the critical pair

Example 8.2. Let $M = (\lambda x.x)\ ((\lambda q.y)z)$. The CBV reduction path is $M \overset{\mathsf{v}}{\mapsto} (\lambda x.x)\ y \overset{\mathsf{v}}{\mapsto} y$. The CBN reduction path is $M \overset{\mathsf{n}}{\mapsto} (\lambda q.y)z \overset{\mathsf{n}}{\mapsto} y$. The corresponding $\bar{\lambda}\mu\tilde{\mu}$-term is $\mu\alpha.\langle\lambda x.x \mid \mu\beta.\langle\lambda q.y \mid z \cdot \beta\rangle \cdot \alpha\rangle$. The term reduces as follows:

$$\begin{aligned} &\mu\alpha.\langle\lambda x.x \mid \mu\beta.\langle\lambda q.y \mid z \cdot \beta\rangle \cdot \alpha\rangle \\ \rhd\ &\mu\alpha.\langle\lambda x.x \mid \mu\beta.\langle z \mid \tilde{\mu}q.\langle y \mid \beta\rangle\rangle \cdot \alpha\rangle \\ \rhd\ &\mu\alpha.\langle\mu\beta.\langle z \mid \tilde{\mu}q.\langle y \mid \beta\rangle\rangle \mid \tilde{\mu}x.\langle x \mid \alpha\rangle\rangle \end{aligned}$$

$$\begin{array}{ll} \overset{\mathsf{v}}{\mapsto} \mu\alpha.\langle z \mid \tilde{\mu}q.\langle y \mid \tilde{\mu}x.\langle x \mid \alpha\rangle\rangle\rangle & \overset{\mathsf{n}}{\mapsto} \mu\alpha.\langle\mu\beta.\langle z \mid \tilde{\mu}q.\langle y \mid \beta\rangle\rangle \mid \alpha\rangle \\ \overset{\mathsf{v}}{\mapsto} \mu\alpha.\langle y \mid \tilde{\mu}x.\langle x \mid \alpha\rangle\rangle & \overset{\mathsf{n}}{\mapsto} \mu\alpha.\langle z \mid \tilde{\mu}q.\langle y \mid \alpha\rangle \\ \overset{\mathsf{v}}{\mapsto} \mu\alpha.\langle y \mid \alpha\rangle & \overset{\mathsf{n}}{\mapsto} \mu\alpha.\langle y \mid \alpha\rangle \end{array}$$

In order to make the computational meaning of this example more clear, definition 8.3 provides an interpretation of $\bar{\lambda}\mu\tilde{\mu}$-terms. The interpretation is only defined for a small set of $\bar{\lambda}\mu\tilde{\mu}$-terms, but it is sufficient for the example.

Definition 8.3. *The interpretation* **–** *is defined as follows:*

$$\begin{aligned} \overline{x} &\overset{def}{=} x \\ \overline{\lambda x.v} &\overset{def}{=} \lambda x.\overline{v} \\ \overline{\mu\alpha.\langle v_1 \mid v_2 \cdot \alpha\rangle} &\overset{def}{=} \mathsf{app}(\overline{v_1}, \overline{v_2}) \\ \overline{\mu\alpha.\langle v \mid \alpha\rangle} &\overset{def}{=} [\overline{v}],\ \textit{if}\ \alpha \notin \mu FV(v) \\ \overline{\mu\alpha.\langle v_1 \mid \tilde{\mu}x.\langle v_2 \mid e\rangle\rangle} &\overset{def}{=} \begin{cases} \mathsf{let}\ x := \overline{v_1}\ \mathsf{in}\ \overline{v_2} & \textit{if}\ e = \alpha \\ \mathsf{let}\ x := \overline{v_1}\ \mathsf{in}\ \overline{\mu\alpha.\langle v_2 \mid e\rangle} & \textit{if}\ e = \tilde{\mu}y.\langle v_3 \mid e'\rangle \end{cases} \end{aligned}$$

Definition 8.4. *The notion of reduction* $\rightarrow_{\overline{\beta}}$ *is defined as:*

$$\mathsf{app}(\lambda x.v_1, v_2) \rightarrow_{\overline{\beta}} \mathsf{let}\ x := v_2\ \mathsf{in}\ v_1$$

Remark 8.5. $\overline{\beta}$-reduction holds for all v_1 and v_2. Unlike usual β-reduction, this reduction doesn't substitute: substitution is done by the reductions of μ and $\tilde{\mu}$.

Example 8.6. After interpretation, the reduction behaviour of the term of example 8.2 is depicted as follows:

$$\begin{aligned} &\mathsf{app}(\lambda x.x, \mathsf{app}(\lambda q.y, z)) \\ \rhd\ &\mathsf{app}(\lambda x.x, \mathsf{let}\ q := z\ \mathsf{in}\ y) \\ \rhd\ &\mathsf{let}\ x := (\mathsf{let}\ q := z\ \mathsf{in}\ y)\ \mathsf{in}\ x \end{aligned}$$

Up to this point, no real computation is done, merely the contraction of $\overline{\beta}$-redexes. At this point however, the critical pair is reached, where CBV operational semantics gives priority to computing $\mathsf{let}\, q := z \,\mathsf{in}\, y$ and CBN gives priority to computing the entire term:

$$\begin{array}{ll}
\overset{\mathsf{v}}{\mapsto} \mathsf{let}\, q := z \,\mathsf{in}\, (\mathsf{let}\, x := y \,\mathsf{in}\, x) & \overset{\mathsf{n}}{\mapsto} [\mathsf{let}\, q := z \,\mathsf{in}\, y] \\
\overset{\mathsf{v}}{\mapsto} \mathsf{let}\, x := y \,\mathsf{in}\, x & \overset{\mathsf{n}}{\mapsto} \mathsf{let}\, q := z \,\mathsf{in}\, y \\
\overset{\mathsf{v}}{\mapsto} [y] & \overset{\mathsf{n}}{\mapsto} [y]
\end{array}$$

On a logical level, the semantics of μ is that of a top-down proof, whereas the semantics of $\tilde{\mu}$ is that of a bottom-up proof [16]. Example 8.7, provided by [16], gives a sequent that can be proven bottom-up and a top-down.

Example 8.7.
Let x, y and z be variables, respectivily of type A, $A \to B$ and $B \to C$. With disregard for structural rules, the following proofs of LK correspond to the following terms:

Bottom-up

$$\dfrac{A \vdash A \qquad \dfrac{B \vdash B \qquad C \vdash C}{B, B \to C, \vdash C}\to\mathrm{l}}{A, A \to B, B \to C \vdash C}\to\mathrm{l}$$

$$\begin{array}{c} \mu\beta : C.\langle y : (A \to B) \mid \\ x : A \cdot \tilde{\mu}w : B.\langle z : (B \to C) \mid w \cdot \beta\rangle\rangle \end{array}$$

```
we need to prove $C$
   we now prove $C$
      by y and by $x$ we proved $B$ $(w)$
   by $z$ and by $w$
```

Top-down

$$\dfrac{\dfrac{A \vdash A \qquad B \vdash B}{A, A \to B \vdash B}\to\mathrm{l} \qquad C \vdash C}{A, A \to B, B \to C \vdash C}\to\mathrm{l}$$

$$\begin{array}{c} \mu\beta : C.\langle z : (B \to C) \mid \\ \mu\gamma.\langle y : (A \to B) \mid \mu\delta.\langle x : A \mid \delta\rangle \cdot \gamma\rangle \cdot \beta\rangle \end{array}$$

```
we need to prove $C$
   we now prove $C$
   by z and we need to prove $B \vee C$
      we now prove $B$
      by $y$ and we need to prove $A \vee B \vee C$
         we now prove $A$
         by $x$
```

Theorem 8.8. *A sequent $\Gamma \vdash A; \Sigma$ is derivable in $\mathsf{LK} \setminus (\bot E)$ (minimal classical sequent calculus) iff there exists a closed $\bar{\lambda}\mu\tilde{\mu}$-term M such that $\Gamma \vdash_{\bar{\lambda}\mu\tilde{\mu}} M : A \mid \Sigma$ is derivable.*

8.2 Adding rules to handle ⊥

Rules to handle ⊥ for the $\bar{\lambda}\mu\tilde{\mu}$-calculus are discussed in [17]. In that paper, $\bar{\lambda}\mu\tilde{\mu}$ is defined by two extra syntactic categories, with extra typing rules as well. The first new category is *values*, which are the same as usual: a value is either a variable or an λ-abstraction. The second one is *linear contexts*, which are contexts that aren't derived using contraction [17]. A constant, similar to the `top`-constant for $\lambda\mu$ is introduced, denoted by $\rtimes$. In table 8.1 new rules are added to handle these new categories of $\bar{\lambda}\mu\tilde{\mu}$-terms and to handle $\rtimes$.

Remark 8.9. It is unclear to me whether the new syntatic categories are actually nesseccary to define rules to handle ⊥. In [17] these categories are used in the definition of other new operators, corresponding to ¬, ∨ and ∧. In [1] these new categories are used to define two subcalculi of $\bar{\lambda}\mu\tilde{\mu}$, that contain the CBN and the CBV reducts of the translation of any $\lambda\mu$-term.

Name	Denotation	Syntax	Type assignment
Commands	c ::=	$\langle v \mid e\rangle$	$c : (\Gamma \vdash \Sigma)$
Values	V ::=	$x \mid \lambda x.v$	$\Gamma \vdash V : A; \Sigma$
Terms	v ::=	$\mu\beta.c \mid V$	$\Gamma \vdash v : A \mid \Sigma$
Linear Contexts	E ::=	$\alpha \mid v \cdot e \mid \rtimes$	$\Gamma; E : A \vdash \Sigma$
Contexts	e ::=	$\tilde{\mu}x.c \mid E$	$\Gamma \mid e : A \vdash \Sigma$

$\bar{\lambda}\mu\tilde{\mu}$

$$\frac{}{\Gamma \mid \alpha : A \vdash \alpha : A, \Sigma}\ \text{axiom-l} \qquad \frac{}{\Gamma, x : A \vdash x : A \mid \Sigma}\ \text{axiom-r}$$

$$\frac{\Gamma \vdash v : A \mid \Sigma \qquad \Gamma \mid e : B \vdash \Sigma}{\Gamma \mid v \cdot e : A \to B \vdash \Sigma}\ \to\text{l} \qquad \frac{\Gamma, x : A \vdash v : B \mid \Sigma}{\Gamma \vdash \lambda x.v : A \to B \mid \Sigma}\ \to\text{r}$$

$$\frac{c : (\Gamma, x : A \vdash \Sigma)}{\Gamma \mid \tilde{\mu}x.c : A \vdash \Sigma}\ \text{Activate-l} \qquad \frac{c : (\Gamma \vdash \beta : B, \Sigma)}{\Gamma \vdash \mu\beta.c : B \mid \Sigma}\ \text{Activate-r}$$

$$\frac{\Gamma; E : A \vdash \Sigma}{\Gamma \mid E : A \vdash \Sigma}\ \text{Der-l} \qquad \frac{\Gamma \vdash V : A; \Sigma}{\Gamma \vdash V : A \mid \Sigma}\ \text{Der-r}$$

$$\frac{}{\Gamma \mid \rtimes : \bot \vdash \Sigma}\ \bot\text{l}$$

$$\frac{\Gamma \vdash M : A \mid \Sigma \qquad \Gamma \mid E : A \vdash \Sigma}{\langle M \mid E\rangle : (\Gamma \vdash \Sigma)}\ \text{cut}$$

Table 8.1. Rules for the $\bar{\lambda}\mu\tilde{\mu}$-calculus

Theorem 8.10. *A sequent $\Gamma \vdash A; \Sigma$ is derivable in* LK *iff there exists a closed $\bar{\lambda}\mu\tilde{\mu}$-term M such that $\Gamma \vdash_{\bar{\lambda}\mu\tilde{\mu}} M : A \mid \Sigma$ is derivable.*

Example 8.11. As example, inhabitants of $\bot \to A$ and $((A \to \bot) \to \bot) \to A$ are given.

$$\frac{\dfrac{\dfrac{\dfrac{}{x : \bot \vdash x : \bot \mid \alpha : A}\ \text{Ax-r} \qquad \dfrac{}{x : \bot \mid \rtimes : \bot \vdash \alpha : A}\ \bot\text{l}}{\langle x \mid \rtimes\rangle : (x : \bot \vdash \alpha : A)}\ \text{cut}}{x : \bot \vdash \mu\alpha.\langle x \mid \rtimes\rangle : A \mid}\ \text{Activate-r}}{\vdash \lambda x.\mu\alpha.\langle x \mid \rtimes\rangle : \bot \to A \mid}\ \to\text{r}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y:\neg\neg A, x:A \vdash x:A \mid \alpha:A,\beta:\bot}\,\text{Ax-r} \qquad \dfrac{}{y:\neg\neg A, x:A \mid \alpha:A \vdash \alpha:A,\beta:\bot}\,\text{Ax-l}}{\langle x \mid \alpha\rangle : (y:\neg\neg A, x:A \vdash \alpha:A,\beta:\bot)}\,\text{cut}}{y:\neg\neg A, x:A \vdash \mu\beta.\langle x \mid \alpha\rangle : \bot \mid \alpha:A}\,\text{Activate-r}}{y:\neg\neg A \vdash \lambda x.\mu\beta.\langle x \mid \alpha\rangle : A \to \bot \mid \alpha:A}\,\to\text{r}}{\vdots}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{y:\neg\neg A \vdash y:\neg\neg A \mid \alpha:A}\,\text{Ax-r} \qquad \dfrac{\vdots \qquad \dfrac{}{y:\neg\neg A \mid \rtimes:\bot \vdash \alpha:A}\,\bot\text{l}}{\mid (\lambda x.\mu\beta.\langle x \mid \alpha\rangle)\cdot\rtimes : \neg\neg A \vdash \alpha:A}\,\to\text{l}}{\langle y \mid (\lambda x.\mu\beta.\langle x \mid \alpha\rangle)\cdot\rtimes\rangle : (y:\neg\neg A \vdash \alpha:A)}\,\text{cut}}{y:\neg\neg A \vdash \mu\alpha.\langle y \mid (\lambda x.\mu\beta.\langle x \mid \alpha\rangle)\cdot\rtimes\rangle : A \mid}\,\text{Activate-r}}{\vdash \lambda y.\mu\alpha.\langle y \mid (\lambda x.\mu\beta.\langle x \mid \alpha\rangle)\cdot\rtimes\rangle : \neg\neg A \to A \mid}\,\to\text{r}$$

In order to give computational meaning to these terms, let $\mathsf{eval}(E)[] \overset{\text{def}}{=} \lambda x.\mu\mathsf{cc}.\langle x \mid E\rangle$, with cc a μ-variable. The informal meaning of the function $\mathsf{eval}(E)[]$ is that it expects a term and computes that term in the context E. The current context can be accessed by the μ-variable cc.

$$\begin{aligned}\mathcal{A} &\overset{\text{def}}{=} \mathsf{eval}(\rtimes)[]\\ C &\overset{\text{def}}{=} \lambda y.\mu\mathsf{cc}.\langle y \mid \mathsf{eval}(\mathsf{cc})[]\cdot\rtimes\rangle\\ \mathcal{K} &\overset{\text{def}}{=} \lambda y.\mu\mathsf{cc}.\langle y \mid \mathsf{eval}(\mathsf{cc})[]\cdot\mathsf{cc}\rangle\end{aligned}$$

The terms $\mathcal{A}$ and C are the terms of example 8.11. The term $\mathcal{K}$ is the term from example 7.1. The term $\mathcal{A}$ computes the expected term in the top-level context (represented by $\rtimes$), thereby abandoning the current context. The term C abandons the current context as well, but keeps it stored and provides the possibility to return to it. $\mathcal{K}$ also stores the current context, but computation resumes in the current context (no aborting behaviour). Their behaviour is exactly like the behaviour of the operators of the λC-calculus and the call/cc operator from [18].

$\bar{\lambda}\mu\tilde{\mu}$-calculus	λC-calculus
$\mu\alpha.\langle\mathcal{A} \mid M\cdot E\rangle \twoheadrightarrow \mu\alpha.\langle M \mid \rtimes\rangle$	$E[\mathcal{A}(M)] \to M$
$\mu\alpha.\langle C \mid M\cdot E\rangle \twoheadrightarrow \mu\alpha.\langle M \mid \mathsf{eval}(E)[]\cdot\rtimes\rangle$	$E[C(M)] \to M\ \lambda y.\mathcal{A}(E[y])$
$\mu\alpha.\langle\mathcal{K} \mid M\cdot E\rangle \twoheadrightarrow \mu\alpha.\langle M \mid \mathsf{eval}(E)[]\cdot E\rangle$	$E[\mathcal{K}(M)] \to E[M\ \lambda y.\mathcal{A}(E[y])]$

The terms $\mathcal{A}$ and C can be interpreted using Sacerdoti, if his interpretation is extended with:

$$\begin{aligned}[\![\mu\alpha : A.\langle v \mid \rtimes\rangle]\!]_{\pi} := {}&\texttt{we need to prove } \vee(A\cdot\pi)\\ &\texttt{we now prove } \bot\\ &[\![v]\!]_{A\cdot\pi}\end{aligned}$$

$$\begin{aligned}[\![\mu\alpha : A.\langle v \mid E\cdot\rtimes\rangle]\!]_{\pi} := {}&\texttt{we need to prove } \vee(A\cdot\pi)\\ &\texttt{we now prove } \bot\\ &[\![v]\!]_{A\cdot\pi} \text{ and } [\![E]\!]_{A\cdot\pi}\end{aligned}$$

In words: instead of proving a formula from the list, one can always choose to prove $\perp$. The interpretations of $\mathcal{A}$ and C are as follows:

$\mathcal{A} : \perp \to A$	$C : ((A \to \perp) \to \perp) \to A$

`suppose` $\perp$ (x)
`we need to prove` A
    `we now prove` $\perp$
    `by` x
	`suppose` $(A \to \perp) \to \perp$ (y)
	`we need to prove` A
	    `we now prove` $\perp$
	    `by` y `and` `suppose` A (x)
	            `we need to prove` $\perp \vee A$
	            `we now prove` A
	            `by` x

Remark 8.12. The construct "`we now prove`" chooses which formula is to be proven. If it switches to another formula, this is isomorphic to switching to another context. If in a proof this construct is only used to choose the current left-most formula (so it isn't used to alter the current formula that is to be proven to either another formula from the list or to $\perp$), than the proof is intuitionistically true, since than one gets a sequent calculus with sequents with one conclusion, which is intuitionistic [6].

8.3 The $\bar{\lambda}\mu$-calculus and the $\bar{\lambda}\mu\tilde{\mu}$-calculus

Lemma 8.13. *The identity function is a reductional embedding of* $\rhd_{\bar{\lambda}\mu}$ *into* $\rhd_{\bar{\lambda}\mu\tilde{\mu}}$.

Proof.

$$\begin{array}{ll}
\langle \lambda x.v_1 \mid v_2 \cdot E\rangle & \rhd_{\bar{\lambda}\mu\tilde{\mu}} \langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid E\rangle\rangle \\
& \rhd_{\bar{\lambda}\mu\tilde{\mu}} \langle v_1[x := v_2] \mid E\rangle \\
\langle \mu\beta.c \mid E\rangle & \rhd_{\bar{\lambda}\mu\tilde{\mu}} c[\beta := E]
\end{array}$$

□

Definition 8.14. *The translation* $\overset{\sim}{\rightarrow}$ *from a term* $M \in \overline{\Lambda}\mu\tilde{\mu}$ *to a term* $\overset{\sim}{\overrightarrow{M}} \in \overline{\Lambda}\mu$ *is defined as follows:*

$$\begin{aligned}
\overset{\sim}{\overrightarrow{x}} &= x \\
\overset{\sim}{\overrightarrow{\lambda x.v}} &= \lambda x.\overset{\sim}{\overrightarrow{v}} \\
\overset{\sim}{\overrightarrow{\mu\alpha.c}} &= \mu\alpha.\overset{\sim}{\overrightarrow{c}} \\
\overset{\sim}{\overrightarrow{\langle v \mid e\rangle}} &= \langle \overset{\sim}{\overrightarrow{v}} \mid \overset{\sim}{\overrightarrow{e}}\rangle, \textit{ if e doesn't end with } \tilde{\mu}x.c \\
\overset{\sim}{\overrightarrow{\langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid e\rangle\rangle}} &= \overset{\sim}{\overrightarrow{\langle v_1[x := v_2] \mid e\rangle}} \\
\overset{\sim}{\overrightarrow{\langle \lambda y.v \mid v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot \tilde{\mu}x.\langle v_m \mid e\rangle\rangle}} &= \overset{\sim}{\overrightarrow{\langle v[y := v_1] \mid v_2 \cdot \ldots v_n \cdot \tilde{\mu}x.\langle v_m \mid e\rangle\rangle}} \\
\overset{\sim}{\overrightarrow{\langle \mu\alpha.c \mid v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot \tilde{\mu}x.\langle v_m \mid e\rangle\rangle}} &= \overset{\sim}{\overrightarrow{c[\alpha := v_1 \cdot v_2 \cdot \ldots \cdot v_n \cdot \tilde{\mu}x.\langle v_m \mid e\rangle]}}
\end{aligned}$$

Lemma 8.15. *The translation* $\overset{\sim}{\rightarrow}$ *is a reductional embedding of* $\rhd_{\bar{\lambda}\mu\tilde{\mu}}$ *into* $\rhd_{\bar{\lambda}\mu}$.

Proof.

The only non-trivial reduction is $\rightarrow_{\beta'}$. It must be proven that $\overset{\sim}{\overrightarrow{\langle \lambda x.v_1 \mid v_2 \cdot e\rangle}} \ \rhd_{\bar{\lambda}\mu} \ \overset{\sim}{\overrightarrow{\langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid e\rangle\rangle}}$. The proof is by induction on e.

$e = \alpha$**:**

$$\begin{aligned}
\overset{\sim}{\overrightarrow{\langle \lambda x.v_1 \mid v_2 \cdot e\rangle}} \quad & = \ \langle \lambda x.\ \overset{\sim}{\overrightarrow{v_1}} \mid \overset{\sim}{\overrightarrow{v_2}} \cdot \alpha\rangle \\
& \rhd_{\bar{\lambda}\mu} \ \langle \overset{\sim}{\overrightarrow{v_1}} \ [x := \overset{\sim}{\overrightarrow{v_2}}] \mid \alpha\rangle \\
& = \ \overset{\sim}{\overrightarrow{\langle v_1[x := v_2] \mid \alpha\rangle}} \\
& = \ \overset{\sim}{\overrightarrow{\langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid \alpha\rangle\rangle}} \\
& = \ \overset{\sim}{\overrightarrow{\langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid e\rangle\rangle}}
\end{aligned}$$

$e = \tilde{\mu}y.\langle v_3 \mid e'\rangle$**:**

$$\begin{aligned}
\overset{\sim}{\overrightarrow{\langle \lambda x.v_1 \mid v_2 \cdot e\rangle}} \quad & = \overset{\sim}{\overrightarrow{\langle \lambda x.v_1 \mid v_2 \cdot \tilde{\mu}y.\langle v_3 \mid e'\rangle\rangle}} \\
& = \overset{\sim}{\overrightarrow{\langle v_1[x := v_2] \mid \tilde{\mu}y.\langle v_3 \mid e'\rangle\rangle}} \\
& = \overset{\sim}{\overrightarrow{\langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid \tilde{\mu}y.\langle v_3 \mid e'\rangle\rangle\rangle}} \\
& = \overset{\sim}{\overrightarrow{\langle v_2 \mid \tilde{\mu}x.\langle v_1 \mid e\rangle\rangle}}
\end{aligned}$$

□

Lemma 8.16. *The identity function is a computational embedding of* $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu}$ *into* $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu\tilde{\mu}}$.

Proof.
The proof is exactly like the proof of lemma 8.13, with every occurrence of $\rhd_{\bar{\lambda}\mu\tilde{\mu}}$ replaced with $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu\tilde{\mu}}$.
□

Lemma 8.17. *The translation* $\overset{\sim}{\rightarrow}$ *is a computational embedding of* $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu\tilde{\mu}}$ *into* $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu}$.

Proof.
The proof is exactly like the proof of lemma 8.15, with every occurrence of $\rhd_{\bar{\lambda}\mu}$ replaced with $\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu}$.
□

Theorem 8.18. $\bar{\lambda}\mu$ *and* $\bar{\lambda}\mu\tilde{\mu}$ *are type-equivalent, reductionally equivalent and computationally equivalent.*

Proof.

$\bar{\lambda}\mu \equiv_t \bar{\lambda}\mu\tilde{\mu}$	This follows from theorems 7.3 and 8.8.
$\bar{\lambda}\mu \equiv_r \bar{\lambda}\mu\tilde{\mu}$	This follows from lemmas 8.13 and 8.15.
$(\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu}) \ \equiv_c \ (\overset{\mathsf{n}}{\mapsto}_{\bar{\lambda}\mu\tilde{\mu}})$	This follows from lemmas 8.16 and 8.17.

□

9 Conclusion

In this paper, different λ-calculi for classical logic have been compared on three aspects: their typing system, their notion of reduction and their operational semantics. Classically typed λ-calculi have operators which in some way possess control over the current context. The relations that have been studied in this paper are graphically summarized in figure 9.1.

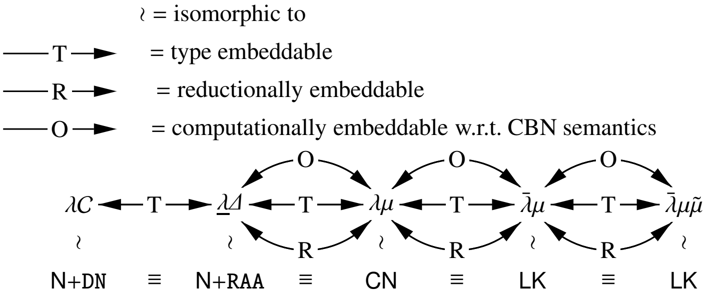
≀ = isomorphic to
—T→ = type embeddable
—R→ = reductionally embeddable
—O→ = computationally embeddable w.r.t. CBN semantics
λC
T
λΔ
λμ
λ̄μ
λ̄μμ̃
O
R
N+DN
N+RAA
CN
LK
LK
≡

Fig. 9.1. Conclusion

Four different classical logical systems have been considered. Natural deduction with added either Double Negation, Reductio Ad Absurdum or multiple conclusions and classical sequent calculus. These four systems are respectively isomorphic to λC, $\lambda\Delta$, $\lambda\mu$ and $\bar{\lambda}\mu$. The classical logical systems are all equivalent. The isomorphisms imply that all the λ-calculi are type-equivalent. However, type-equivalence does not imply reductional equivalence. Furthermore, even if two λ-calculi are equivalent on a reductional level, this doesn't imply that their CBN operational semantics are equivalent.

The λC-calculus is based on CBV operational semantics and doesn't make a distinction between the notion of reduction and operational semantics. It is therefore incomparable to the other λ-calculi.

The $\lambda\Delta$-calculus and the $\lambda\mu$-calculus are both meant to be reduced CBN and are considered as such. A problem of the $\lambda\Delta$-calculus lies in the fact that its notion of renaming has an implicit restriction. If this restriction is lifted, as it is in the $\lambda\mu$-calculus, then the only real difference between the two calculi lies in the reduction of an application of a Δ- or μ-abstraction. The first reduction creates λ-abstractions and it thus creates new β-redexes, which is behaviour for which no encoding has been found in the $\lambda\mu$-calculus. The second reduction performs structural substitution, which is behaviour that can be encoded in the $\lambda\Delta$-calculus. With some changes in the notion of reduction of the $\lambda\Delta$-calculus (which results in the $\underline{\lambda\Delta}$-calculus), the equivalence to $\lambda\mu$ is proven.

For the minimal parts of $\lambda\mu$, $\bar{\lambda}\mu$ and $\bar{\lambda}\mu\tilde{\mu}$, that is the calculi without rules to handle $\perp$, it is proven that they are equal on every level, including CBN operational semantics. The addition of the construct $\tilde{\mu}$ allows for both CBV and CBN operational semantics to be defined easily. The main difference between $\lambda\mu$ on the one hand and $\bar{\lambda}\mu$ and $\bar{\lambda}\mu\tilde{\mu}$ on the other, lies in the fact that the latter reveal a clear duality between building terms (proofs) and building contexts (counterproofs).

References

1. Pierre-Louis Curien and Hugo Herbelin. The duality of computation. *ACM SIGPLAN Notices*, 35(9):233–243, 2000.
2. Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *LPAR '92: Proceedings of the International Conference on Logic Programming and Automated Reasoning*, pages 190–201, London, UK, 1992. Springer-Verlag.
3. N. Rehof and M. Sorensen. The λ_Δ-calculus. In *Lecture Notes of Computer Science*, volume 789, pages 516–542. Springer-Verlag, 1994.
4. Timothy G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan 1990*, pages 47–57. ACM Press, New York, 1990.
5. Henk Barendregt and Silvia Ghilezan. Lambda terms for natural deduction, sequent calculus and cut elimination.
6. M. E. Szabo, editor. *The collected papers of Gerhard Gentzen*. North-Holland Publishing Company, 1969.
7. Dag Prawitz. *Natural Deduction, A Proof-Theoretical Study*. Almqvist & Wiksell, 1965.
8. G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proceedings 23rd Int. Symp. on Math. Found. of Comp. Science, MFCS'98, Brno, Czech Rep., 24–28 Aug. 1998*, volume 1450, pages 336–345. Springer-Verlag, Berlin, 1998.
9. Zena M. Ariola and Mathhias Felleisen. The call-by-need lambda calculus. In *Journal of Functional Programming*, volume 7, pages 265–301. Cambridge University Press, may 1997.
10. Zena M. Ariola and Hugo Herbelin. Minimal classical logic and control operators. *Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP '03)*, 2719:128–136, 2003.
11. Morten Heine Sørensen and Paweł Urzyczyn. Lectures on the curry-howard isomorphism. Technical report, University of Copenhagen, 1998.
12. Matthias Felleisen. *The calculi of λ_v-CS conversion: a syntactic theory of control and state in imperative higher-order programming languages*. PhD thesis, Indianapolis, IN, USA, 1987.
13. Philippe de Groote. A CPS-translation of the $\lambda\mu$-calculus. In S. Tison, editor, *Proceedings 19th Intl. Coll. on Trees in Algebra and Programming, CAAP'94, Edinburgh, UK, 11–13 Apr 1994*, volume 787, pages 85–99. Springer-Verlag, Berlin, 1994.
14. Philippe de Groote. On the relation between the lambda-calculus and the syntactic theory of sequential control.
15. Philippe de Groote. An environment machine for the lambda-mu-calculus. *Mathematical Structures in Computer Science*, 8(6):637–669, 1998.
16. Claudio Sacerdoti Coen. Explanation in natural language of $\bar{\lambda}\mu\tilde{\mu}$-terms. In *Lecture Notes in Artificial Intelligence (LNAI)*, volume 3863, pages 234–249, 2006.
17. Hugo Herbelin et al. C'est maintenant qu'on calcule: au coeur de la dualité, mémoire d'habilitation. http://pauillac.inria.fr/˜herbelin/publis/index.html, 2005.
18. Dorai Sitaram and Matthias Felleisen. Reasoning with equations ii: Full abstraction for models of control. In *Proc. 1990 ACM Conference on Lisp and Functional Programming*, pages 161–175, 1990.