Draft of paper published in: P. Materna & B. Jespersen (eds), *Logically Speaking: A Festschrift for Marie Duží*, Tributes Vol. 49, College Publications 2022, pp. 149–186.

Administrative Normal Form and Focusing for Lambda Calculi

David Binder
University of Tübingen
david.binder@uni-tuebingen.de

Thomas Piecha
University of Tübingen
thomas.piecha@uni-tuebingen.de

Abstract

The Curry-Howard correspondence between deductive systems and computational calculi is one of the great unifying ideas. It links purely logical investigations to practical problems in computer science, in particular the design and implementation of programming languages. Many aspects of this correspondence are widely known, such as the correspondence between natural deduction for intuitionistic logic and the simply typed λ-calculus. On the other hand, the importance of the sequent

It is our pleasure to contribute to this *Festschrift* in honor of Marie Duží. Our acquaintance with Marie dates back to her visit to Tübingen in January 2004, where she gave a talk on "Attitudes: Hyperintensional vs. intensional contexts". Our exchange of ideas has continued during the *Second Conference on Proof-Theoretic Semantics* (Tübingen, 8-10 March 2013), where she made "A plea for beta-conversion by value", at the *Third Conference on Proof-Theoretic Semantics* (Tübingen, 27-30 March 2019), where she presented "Natural Language Processing by Natural Deduction in Transparent Intensional Logic" [8], at LOGICA 2011 (Hejnice Monastery, 20-24 June 2011) and LOGICA 2016 (Hejnice Monastery, 20-24 June 2016), as well as during other conferences. This paper relates to Marie's work on call-by-value β-conversion, which she developed in particular in the context of Transparent Intensional Logic (cf. Duží [7], Duží and Jespersen [9, 10, 11], Duží and Kosterec [12] and Jespersen and Duží [18]).

calculus in proof-theoretic investigations is not yet reflected in the study of programming languages, where languages based on the λ-calculus dominate. One of the principal reasons for this is, we think, the lack of introductory material that could serve in helping to translate between logicians and programming language theorists.

Our small contribution in this respect is to introduce and expose the correspondence between two normal forms and their respective normalization procedures: administrative normal form and ANF-transformation on the one hand, and focused normal form and static focusing on the other. Though invented for different purposes, compiler optimizations in the case of the ANF-transformation and proof search in the case of focusing, they are structurally very similar. Both transformations bring proofs into a normal form where functions and constructors are only applied to values and where computations are sequentialized. In this paper we make this similarity explicit.

1 Introduction

In 1935, Gentzen [16] introduced the two most important logical calculi used in proof theory today: natural deduction and the sequent calculus. Natural deduction is used widely in both proof theory and the theory of computation and programming. Its success in the latter is due to the Curry-Howard correspondence (cf. [23]) between natural deduction proofs and programs, or propositions and types. The sequent calculus, on the other hand, did not yet have a comparable impact in the theory of programming languages. Especially in the case of the *classical* sequent calculus, this can be explained by the difficulty to reconcile those of its features that are essential for obtaining classical logic with a good computational interpretation. Such an interpretation was provided when the relationship between classical axioms and control operators was discovered by Griffin [17]. This discovery led to the development of several term systems for encoding sequent calculus proofs. One such system is the $\lambda\mu\tilde{\mu}$-calculus, introduced by Curien and Herbelin [4].

We will use the λ-calculus and the $\lambda\mu\tilde{\mu}$-calculus, which are related by a translation function from λ-terms Λ to $\lambda\mu\tilde{\mu}$-terms $\Lambda_{\mu\tilde{\mu}}$. For the λ-calculus we define the administrative normal form Λ^{ANF}, together with a transformation from Λ to Λ^{ANF}. In distinction to the usual presentation of the ANF-transformation, we divide this transformation into two parts by using an intermediate normal form Λ^{Q} between Λ and Λ^{ANF}. For the $\Lambda_{\mu\tilde{\mu}}$-calculus we define the so-called *focused normal form* $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$ (which corresponds to the subsyntax LKQ of [4]). The focusing transformation from $\Lambda_{\mu\tilde{\mu}}$ to $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$ is adapted from [6]. We define a new normal form $\Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}$ for $\lambda\mu\tilde{\mu}$-terms, which exactly mirrors the syntactic restrictions that characterize the administrative normal form Λ^{ANF} for λ-terms.

As our main result, depicted in Fig. 1, we show how the ANF-transformation on λ-terms corresponds to static focusing of $\lambda\mu\tilde{\mu}$-terms. The first part of the ANF-transformation corresponds precisely to the static focusing transformation. That is, it commutes with focusing via the translation function up to α-equivalence. The second part of the ANF-transformation can be simulated in the $\lambda\mu\tilde{\mu}$-calculus by μ-reductions.

$$\begin{tikzcd}
\Lambda \arrow[r, "\text{translation}"] \arrow[d, "\text{ANF-transformation (1)}"'] & \Lambda_{\mu\tilde{\mu}} \arrow[d, "\text{focusing}"] \\
\Lambda^{\mathrm{Q}} \arrow[r, "\text{translation}"] \arrow[d, "\text{ANF-transformation (2)}"'] & \Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}} \arrow[d, "\mu\text{-reduction}"] \\
\Lambda^{\mathrm{ANF}} \arrow[r, "\text{translation}"] & \Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}
\end{tikzcd}$$

Figure 1: The relationship between the ANF-tranformation of λ-terms and focusing of $\lambda\mu\tilde{\mu}$-terms.

The paper is structured as follows. In Section 2 we present the main idea using an informal example. In Section 3 we formalize the syntax and type assignment rules for the λ-calculus and the $\lambda\mu\tilde{\mu}$-calculus, and in Section 4 we give the translation from the former to the latter. In Section 5 we provide the call-by-value operational semantics for both calculi. We introduce the ANF-transformation in Section 6 and static focusing in Section 7. The main result is presented in Section 8 and summarized in Section 9, which also contains an outlook to future work. The proofs of the main theorems can be found in Appendix A.

2 An informal example

We explain the main idea with an informal example. Consider the following program

$$\pi_2(\pi_1(1,4),3)$$

which consists of natural numbers 1, 4 and 3, pair constructors $(\textvisiblespace,\textvisiblespace)$ and projections $\pi_1\,\textvisiblespace$ and $\pi_2\,\textvisiblespace$ on the first and second element of a pair, respectively.

We expect this program to evaluate to the natural number 3. Using call-by-name we could immediately evaluate this program to its final value 3. However, using call-by-value we first have to evaluate the argument of π_2 to the value $(1,3)$ by evaluating $\pi_1(1,4)$ to 1.

There are different ways to formalize the evaluation of a term within a context. Here we choose the method of evaluation contexts (cf. Felleisen and Hieb [13] and Section 5.1 below). An *evaluation context* $E[-]$ is a term with a placeholder $\square$, which is to be filled with the outermost redex to be evaluated next. We will use the symbol $\doteqdot$ throughout to express syntactic equality up to α-equivalence (i.e., up to the renaming of bound variables).

In our example, this allows us to evaluate the outermost redex $\pi_1(1,4)$ within the context $E[-] \doteqdot \pi_2(\square,3)$ as follows:

$$\text{If } \pi_1(1,4) \triangleright 1, \text{ then } \pi_2(\pi_1(1,4),3) \doteqdot E[\pi_1(1,4)] \triangleright E[1] \doteqdot \pi_2(1,3).$$

The translation (cf. Definition 4.1) of the program $\pi_2(\pi_1(1,4),3)$ into the $\lambda\mu\tilde{\mu}$-calculus (cf. Section 3.3) results in the program

$$\mu\alpha.\langle(\mu\beta.\langle(1,4) \mid \pi_1\,\beta\rangle,3) \mid \pi_2\,\alpha\rangle.$$

We can recognize many familiar constructs from the initial program. We still have natural numbers 1, 4 and 3, the pair constructor $(\textvisiblespace, \textvisiblespace)$ and projections π_1 and π_2, but they are now organized and nested in a very different way with the help of two new constructs.

The first new construct is the *cut* $\langle \textvisiblespace \mid \textvisiblespace \rangle$ which is used to oppose a *proof* (or *proof term*) of a proposition with its *refutation* (or *refutation term*). In our example, we use the cut to oppose a proof $(1, 4)$ of the type $\mathbb{N} \wedge \mathbb{N}$ with a refutation $\pi_1\ \beta$ of the same type, where we assume that the refutation variable β stands for some unknown refutation of type $\mathbb{N}$. The reduction rules of the $\lambda\mu\tilde{\mu}$-calculus always replace a cut by another cut, and in the case of pairs the reduction rule allows to replace $\langle (1, 4) \mid \pi_1\ \beta \rangle$ by the new cut $\langle 1 \mid \beta \rangle$.

The second new construct is the μ-abstraction $\mu\alpha.\textvisiblespace$. We have more to say about this construct in Section 3.3, but for now it suffices to say that we use $\mu\alpha.\langle \textvisiblespace \mid \textvisiblespace \rangle$ to introduce a subcomputation (represented by the cut $\langle \textvisiblespace \mid \textvisiblespace \rangle$) returning to the output named by the variable α. For example, in order to represent the subcomputation $2 + 2$, we use the term $\mu\alpha.\langle 2 + 2 \mid \alpha \rangle$, which evaluates to $\mu\alpha.\langle 4 \mid \alpha \rangle$.

We cannot evaluate the program $\mu\alpha.\langle (\mu\beta.\langle (1, 4) \mid \pi_1\ \beta \rangle, 3) \mid \pi_2\ \alpha \rangle$ directly to its final value, since one can only evaluate cuts $\langle \textvisiblespace \mid \textvisiblespace \rangle$, whereas this program has the form of a μ-abstraction. This can be resolved by introducing a third construct, namely the *toplevel output* Top, which enables us to embed any μ-program in a cut whose second element is Top. Furthermore, a $\tilde{\mu}$-abstraction $\tilde{\mu}x.\langle \textvisiblespace \mid \textvisiblespace \rangle$ has to be used, which binds a value to the variable x in the subcomputation $\langle \textvisiblespace \mid \textvisiblespace \rangle$.

The example program then evaluates in the following way:

$$\langle \mu\alpha.\langle (\mu\beta.\langle (1, 4) \mid \pi_1\ \beta \rangle, 3) \mid \pi_2\ \alpha \rangle \mid \mathsf{Top} \rangle \tag{1}$$
$$\triangleright\ \langle (\mu\beta.\langle (1, 4) \mid \pi_1\ \beta \rangle, 3) \mid \pi_2\ \mathsf{Top} \rangle \tag{2}$$
$$\triangleright\ \langle \mu\beta.\langle (1, 4) \mid \pi_1\ \beta \rangle \mid \tilde{\mu}x.\langle (x, 3) \mid \pi_2\ \mathsf{Top} \rangle \rangle \tag{3}$$
$$\triangleright\ \langle (1, 4) \mid \pi_1(\tilde{\mu}x.\langle (x, 3) \mid \pi_2\ \mathsf{Top} \rangle) \rangle \tag{4}$$
$$\triangleright\ \langle 1 \mid \tilde{\mu}x.\langle (x, 3) \mid \pi_2\ \mathsf{Top} \rangle \rangle \tag{5}$$
$$\triangleright\ \langle (1, 3) \mid \pi_2\ \mathsf{Top} \rangle \tag{6}$$
$$\triangleright\ \langle 3 \mid \mathsf{Top} \rangle \tag{7}$$

Note that in step (5) we project from $(1, 4)$ to 1 without being in an evaluation context. The evaluation within an evaluation context is instead simulated by steps (4) and (6). That is, steps (4) to (6) correspond to the single evaluation step

$$\pi_2(\pi_1(1, 4), 3) \triangleright \pi_2(1, 3).$$

This sort of evaluation within a context, which is present in both the λ-calculus and the $\lambda\mu\tilde{\mu}$-calculus, poses no problem from a theoretical point of view. However, from a practical point of view, it is very inefficient to apply this kind of operational semantics since the search for a redex requires in general to traverse deeply into a term. Moreover, evaluations of this kind render the implementation of various compiler optimizations (cf. [22, 15]) more difficult. These difficulties can be avoided by using certain normal forms, for example, the so-called *administrative normal form* (*A-normal form* or *ANF*)[1] for the λ-calculus, and the *focused normal form* for the $\lambda\mu\tilde{\mu}$-calculus.

The ANF of the first example program

$$\pi_2(\pi_1(1, 4), 3)$$

is

$$\textbf{let}\ x = \pi_1(1, 4)\ \textbf{in}\ (\textbf{let}\ y = \pi_2(x, 3)\ \textbf{in}\ y), \tag{A}$$

whereas the focused normal form of the second program

$$\mu\alpha.\langle(\mu\beta.\langle(1, 4) \mid \pi_1\ \beta\rangle, 3) \mid \pi_2\ \alpha\rangle$$

is

$$\mu\alpha.\langle\mu\beta.\langle(1, 4) \mid \pi_1\ \beta\rangle \mid \tilde{\mu}x.\langle(x, 3) \mid \pi_2\ \alpha\rangle\rangle. \tag{F}$$

Comparing the ANF (A) with the focused normal form (F) makes the structural similarity between the two normal forms apparent: in both cases the subcomputation $\pi_1(1, 4)$ (resp. $\langle(1, 4) \mid \pi_1\ \beta\rangle$) was lifted out and then bound to the variable x in the subsequent computation

[1]While the "A" in "A-normal" originally had no special meaning, it was later given the meaning of "administrative normal form", due to the administrative redexes it introduces.

$\pi_2(x, 3)$ (resp. $\langle (x, 3) \mid \pi_2\, \alpha\rangle$). The difference between (A) and (F) consists in the use of let-constructs in the λ-calculus on the one hand and the use of μ- and $\tilde{\mu}$-constructs in the $\lambda\mu\tilde{\mu}$-calculus on the other hand.

3 Syntax and type assignment

We present the syntax and type-assignment rules of the λ-calculus and of the $\lambda\mu\tilde{\mu}$-calculus. The syntax for types is the same in both calculi.

Definition 3.1 (Types)**.** There are three kinds of *types* τ:

$$\tau ::= X \mid \tau \to \tau \mid \tau \wedge \tau.$$

That is, we have *atomic types* X, *implication types* $\tau \to \tau$ and *conjunction types* $\tau \wedge \tau$.

3.1 The λ-calculus

We use the standard simply typed λ-calculus with conjunction and a let-construct (cf., e.g., [20]). Since we only consider a call-by-value evaluation strategy, the values consist of variables, λ-abstractions and tuples of values.

Definition 3.2. The *syntax* Λ *of the λ-calculus* is defined as follows, where x are *term variables*:

1. *Terms:* $e ::= x \mid \lambda x.e \mid e\, e \mid (e, e) \mid \pi_1\, e \mid \pi_2\, e \mid \textbf{let}\ x = e\ \textbf{in}\ e.$
2. *Values:* $v ::= \lambda x.e \mid (v, v) \mid x.$

A *judgement* is a sequent of the form $\Gamma \vdash e : \tau$, where Γ is a (possibly empty) set of declarations $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$.

Definition 3.3. The *type assignment rules of the λ-calculus* are:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ \textsc{Var} \qquad \frac{\Gamma \vdash e_1 : \sigma \quad \Gamma, x : \sigma \vdash e_2 : \tau}{\Gamma \vdash \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \tau}\ \textsc{Let}$$

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \to \tau}\ \textsc{Abs} \qquad \frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau}\ \textsc{App}$$

$$\frac{\Gamma \vdash e_1 : \sigma \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash (e_1, e_2) : \sigma \wedge \tau}\ \textsc{Pair} \qquad \frac{\Gamma \vdash e : \tau_1 \wedge \tau_2}{\Gamma \vdash \pi_i\, e : \tau_i}\ \textsc{Proj}$$

Note that rule Proj comprises the two cases where either $i = 1$ or $i = 2$.

There are no structural rules since weakening and contraction are implicit. Note that the rule Let is derivable since any term **let** $x = e_1$ **in** e_2 can be replaced by $(\lambda x.e_2)e_1$ without changing the type in the conclusion of a type assignment. However, let-bindings are used to make the evaluation order explicit; we will come back to this point in Section 7.

3.2 Towards the $\lambda\mu\tilde{\mu}$-calculus

The λ-calculus corresponds to natural deduction for the $\{\to, \wedge\}$-fragment of intuitionistic logic. The $\lambda\mu\tilde{\mu}$-calculus [4] was introduced as a system that corresponds to the classical sequent calculus, in which sequents have the form $\Gamma \vdash \Delta$ with (possibly empty) sets Γ, Δ of formulas on either side of the sequent symbol $\vdash$.

The usual interpretation of a valid classical sequent $\Gamma \vdash \Delta$ can be expressed as "If all the formulas in Γ are true, then at least one of the formulas in Δ is true." This interpretation has to be refined in order to understand the correspondence between the $\lambda\mu\tilde{\mu}$-calculus and the classical sequent calculus. The refinement consists in distinguishing three variants of the sequent $\Gamma \vdash \Delta$:

1. $\Gamma \vdash [\varphi], \Delta$

   "If all $\gamma \in \Gamma$ are true and all $\delta \in \Delta$ are false, then φ is true."

2. $\Gamma, [\varphi] \vdash \Delta$

   "If all $\gamma \in \Gamma$ are true and all $\delta \in \Delta$ are false, then φ is false."

3. $\Gamma \vdash \Delta$

   "The assumption that all $\gamma \in \Gamma$ are true and all $\delta \in \Delta$ are false is contradictory."

The formula in square brackets $[\varphi]$ is called the *active* formula of the sequent. There can be at most one active formula in any sequent.

The $\lambda\mu\tilde{\mu}$-calculus has one syntactic category and one judgement form for each of these three interpretations:

1. The active formula φ in the succedent of a sequent $\Gamma \vdash [\varphi], \Delta$ is assigned to a *term* e, and the corresponding *judgement form* is

   $$\Gamma \vdash e : \varphi \mid \Delta.$$

   Here the symbol $\mid$ singles out a formula φ for which the proof e is currently constructed (cf. [4]).

2. The active formula φ in the antecedent of a sequent $\Gamma, [\varphi] \vdash \Delta$ is assigned to a *coterm* s, and the corresponding *judgement form* is

   $$\Gamma \mid s : \varphi \vdash \Delta.$$

   In this case, the symbol $\mid$ singles out a formula φ for which the refutation s is currently constructed.

3. A sequent $\Gamma \vdash \Delta$ with no active formula is interpreted by a *command* c, and the corresponding *judgement form* is

   $$c : (\Gamma \vdash \Delta).$$

   This judgement form can be read as follows: "If all $\gamma \in \Gamma$ are true and all $\delta \in \Delta$ are false, then c is a contradiction and a well-typed command."

3.3 The $\lambda\mu\tilde{\mu}$-calculus

We consider the syntax of the $\lambda\mu\tilde{\mu}$-calculus. We have to partition the set of λ-terms into the three syntactic categories of the $\lambda\mu\tilde{\mu}$-calculus, namely terms, coterms and commands. The basic idea is that the introduction forms $\lambda x.e$ and (e, e) (which correspond to the introduction rules in natural deduction) will remain terms of the $\lambda\mu\tilde{\mu}$-calculus. On the other hand, the elimination forms $\pi_i e$ and $e\,e$ (which correspond to the elimination rules in natural deduction) will become coterms. The terms of the $\lambda\mu\tilde{\mu}$-calculus therefore comprise the introduction forms $\lambda x.t$ and (t, t) of the λ-calculus, whereas the coterms comprise the elimination forms $\pi_i\, s$ and $t \cdot s$.

There are different ways to understand a coterm $t \cdot s$. First, since an implication $\varphi \to \tau$ is false if φ is true and τ is false, one can interpret $t \cdot s$ as a constructive refutation of an implication $\varphi \to \tau$, consisting of a proof t of φ and a refutation s of τ. Alternatively, in a computational context, $t \cdot s$ can be thought of as a stack frame in a call stack with argument t on top and s being the rest of the stack.

There is only one form of command in the $\lambda\mu\tilde{\mu}$-calculus: the *cut* $\langle t \mid s \rangle$, which combines a term with a coterm. The cut rule can be interpreted as a primitive way to construct a contradiction, namely by providing both a proof and a refutation of the same formula.

This leaves us with the two remaining constructs of μ- and $\tilde{\mu}$-abstraction, which, again, can be understood in two different ways. First, from a logical point of view, the μ-construct encodes a form of *reductio ad absurdum* at the level of judgements:

$$\dfrac{\begin{array}{c}[\varphi \text{ is false}]\\ \vdots \\ \text{contradiction}\end{array}}{\varphi \text{ is true}}\ (\mu)$$

This explains why the addition of μ-abstraction makes the logic classical. The $\tilde{\mu}$-construct, on the other hand, encodes the logical inference

$$\dfrac{\begin{array}{c}[\varphi \text{ is true}]\\ \vdots \\ \text{contradiction}\end{array}}{\varphi \text{ is false}}\ (\tilde{\mu})$$

Both inferences are on the level of judgements and do not involve logical constants; neither absurdity $\bot$ nor negation $\neg$ are used.

Second, from an operational point of view we see that $\tilde{\mu}$ behaves very similarly to the let-construct of the λ-calculus. In a command $\langle t \mid \tilde{\mu}x.c\rangle$, the $\tilde{\mu}$-abstraction is used to bind the term t in the remaining computation c. The μ-construct behaves similarly to control operators like call/cc or $\mathcal{C}$ (cf. [2, 17]).

Definition 3.4. The *syntax $\Lambda_{\mu\tilde{\mu}}$ of the $\lambda\mu\tilde{\mu}$-calculus* is defined as follows, where x are *term variables* and α are *coterm variables*:

1. *Terms:* $t ::= x \mid \lambda x.t \mid (t, t) \mid \mu\alpha.c$.
2. *Coterms:* $s ::= \alpha \mid t \cdot s \mid \pi_1\, s \mid \pi_2\, s \mid \tilde{\mu}x.c$.
3. *Commands:* $c ::= \langle t \mid s\rangle$.
4. *Values:* $w ::= \lambda x.t \mid (w, w) \mid x$.

In addition to term variable contexts $\Gamma \doteq \{x_1 : \tau_1, \ldots, x_n : \tau_n\}$, we now have to consider also coterm variable contexts $\Delta \doteq \{\alpha_1 : \tau_1, \ldots, \alpha_n : \tau_n\}$.

Definition 3.5. The *type-assignment rules of the $\lambda\mu\tilde{\mu}$-calculus* for the three judgement forms

1. *term typing:* $\Gamma \vdash t : \tau \mid \Delta$,
2. *coterm typing:* $\Gamma \mid s : \tau \vdash \Delta$, and
3. *command typing:* $c : (\Gamma \vdash \Delta)$

are the following:

Term typing *Coterm typing*

$$\frac{}{\Gamma, x : \tau \vdash x : \tau \mid \Delta}\ \textsc{Var}_x \qquad \frac{}{\Gamma \mid \alpha : \tau \vdash \alpha : \tau, \Delta}\ \textsc{Var}_\alpha$$

$$\frac{\Gamma, x : \sigma \vdash t : \tau \mid \Delta}{\Gamma \vdash \lambda x.t : \sigma \to \tau \mid \Delta}\ \textsc{Abs} \qquad \frac{\Gamma \vdash t : \tau \mid \Delta \quad \Gamma \mid s : \sigma \vdash \Delta}{\Gamma \mid t \cdot s : \tau \to \sigma \vdash \Delta}\ \textsc{App}$$

$$\frac{\Gamma \vdash t_1 : \tau_1 \mid \Delta \quad \Gamma \vdash t_2 : \tau_2 \mid \Delta}{\Gamma \vdash (t_1, t_2) : \tau_1 \wedge \tau_2 \mid \Delta}\ \textsc{Pair} \qquad \frac{\Gamma \mid s : \tau_i \vdash \Delta}{\Gamma \mid \pi_1\ s : \tau_1 \wedge \tau_2 \vdash \Delta}\ \textsc{Proj}$$

$$\frac{c : (\Gamma \vdash \alpha : \tau, \Delta)}{\Gamma \vdash \mu\alpha.c : \tau \mid \Delta}\ \textsc{Mu} \qquad \frac{c : (\Gamma, x : \tau \vdash \Delta)}{\Gamma \mid \tilde{\mu}x.c \vdash \Delta}\ \textsc{Mu}_{\sim}$$

Command typing

$$\frac{\Gamma \vdash t : \tau \mid \Delta \quad \Gamma \mid s : \tau \vdash \Delta}{\langle t \mid s \rangle : (\Gamma \vdash \Delta)}\ \textsc{Cut}$$

4 Translating λ-terms to $\lambda\mu\tilde{\mu}$-terms

We introduce a compositional translation from λ-terms to $\lambda\mu\tilde{\mu}$-terms and show that it preserves typeability.

Definition 4.1. The *translation* $[\![-]\!] : \Lambda \to \Lambda_{\mu\tilde{\mu}}$ is defined as follows:

$$[\![x]\!] :\doteq x \qquad (\mathcal{T}_1)$$
$$[\![\lambda x.e]\!] :\doteq \lambda x.[\![e]\!] \qquad (\mathcal{T}_2)$$
$$[\![(e_1, e_2)]\!] :\doteq ([\![e_1]\!], [\![e_2]\!]) \qquad (\mathcal{T}_3)$$
$$[\![e_1\ e_2]\!] :\doteq \mu\alpha.\langle [\![e_1]\!] \mid [\![e_2]\!] \cdot \alpha \rangle \qquad (\mathcal{T}_4)$$
$$[\![\pi_i\ e]\!] :\doteq \mu\alpha.\langle [\![e]\!] \mid \pi_i\ \alpha \rangle \qquad (\mathcal{T}_5)$$
$$[\![\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2]\!] :\doteq \mu\alpha.\langle [\![e_1]\!] \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha \rangle \rangle. \qquad (\mathcal{T}_6)$$

In the last three clauses, the coterm variable α has to be fresh.

Let e be any expression of the λ-calculus typeable with type τ in a context Γ. Then the translation $[\![e]\!]$ is a term of the $\lambda\mu\tilde{\mu}$-calculus that is typeable with the same type τ (in the same context Γ of term variables and with an empty context of coterm variables).

Theorem 4.2. *For all e, τ and Γ: if $\Gamma \vdash e : \tau$, then $\Gamma \vdash [\![e]\!] : \tau \mid \varnothing$.*

Proof. The proof is by induction on the derivation of $\Gamma \vdash e : \tau$ in the λ-calculus. The cases for variables, tuples and λ-abstractions are trivial; we will only discuss the following interesting cases.

The first case is for projections. Assume that the last rule in the typing derivation of e is Proj. Then e has the form $\pi_i\, e$, whose translation is defined as $\mu\alpha.\langle [\![e]\!] \mid \pi_i\, \alpha\rangle$. We replace the λ-calulus derivation by the following $\lambda\mu\tilde{\mu}$-calculus derivation:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\text{IH}\\ \Gamma \vdash [\![e]\!] : \tau_1 \wedge \tau_2 \mid \varnothing\end{array} \qquad \dfrac{\dfrac{}{\varnothing \mid \alpha : \tau_i \vdash \alpha : \tau_i}\,\textsc{Var}_\alpha}{\varnothing \mid \pi_i\, \alpha : \tau_1 \wedge \tau_2 \vdash \alpha : \tau_i}\,\textsc{Proj}}{\langle [\![e]\!] \mid \pi_i\, \alpha\rangle : (\Gamma \vdash \alpha : \tau_i)}\,\textsc{Cut}}{\Gamma \vdash \mu\alpha.\langle [\![e]\!] \mid \pi_i\, \alpha\rangle : \tau_i \mid \varnothing}\,\textsc{Mu}}{}$$

The second interesting case is for function applications. Assume that the last rule in the derivation of $\Gamma \vdash e : \tau$ is App. Then e must have the form $e_1\, e_2$, whose translation is defined as $\mu\alpha.\langle [\![e_1]\!] \mid [\![e_2]\!] \cdot \alpha\rangle$. We replace the original derivation by:

$$\dfrac{\dfrac{\begin{array}{c}\text{IH}\\ \Gamma \vdash [\![e_1]\!] : \sigma \to \tau \mid \varnothing\end{array} \qquad \dfrac{\begin{array}{c}\text{IH}\\ \Gamma \vdash [\![e_2]\!] : \sigma \mid \varnothing\end{array} \quad \dfrac{}{\varnothing \mid \alpha : \tau \vdash \alpha : \tau}\,\textsc{Var}_\alpha}{\varnothing \mid [\![e_2]\!] \cdot \alpha : \sigma \to \tau \vdash \alpha : \tau}\,\textsc{App}}{\langle [\![e_1]\!] \mid [\![e_2]\!] \cdot \alpha\rangle : (\Gamma \vdash \alpha : \tau)}\,\textsc{Cut}}{\Gamma \vdash \mu\alpha.\langle [\![e_1]\!] \mid [\![e_2]\!] \cdot \alpha\rangle : \tau \mid \varnothing}\,\textsc{Mu}$$

The last case is for let-bindings. Assume that the last rule in the derivation of $\Gamma \vdash e : \tau$ is Let. Then e must have the form $\mathbf{let}\, x = e_1\, \mathbf{in}\, e_2$, whose translation is defined as $\mu\alpha.\langle [\![e_1]\!] \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha\rangle\rangle$. We replace the original derivation by:

$$\dfrac{\dfrac{\begin{array}{c}\text{IH}\\ \Gamma \vdash [\![e_1]\!] : \sigma \mid \varnothing\end{array} \qquad \dfrac{\dfrac{\begin{array}{c}\text{IH}\\ \Gamma, x : \sigma \vdash [\![e_2]\!] : \tau \mid \varnothing\end{array} \quad \dfrac{}{\varnothing \mid \alpha : \tau \vdash \alpha : \tau}\,\textsc{Var}_\alpha}{\langle [\![e_2]\!] \mid \alpha\rangle : (\Gamma, x : \sigma \vdash \alpha : \tau)}\,\textsc{Cut}}{\Gamma \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha\rangle \vdash \alpha : \tau}\,\textsc{Mu}_\sim}{\langle [\![e_1]\!] \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha\rangle\rangle : (\Gamma \vdash \alpha : \tau)}\,\textsc{Cut}}{\Gamma \vdash \mu\alpha.\langle [\![e_1]\!] \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha\rangle\rangle : \tau \mid \varnothing}\,\textsc{Mu}$$

□

We will also need the following lemma about the translation of values:

Lemma 4.3 (Translation preserves values)**.** *An expression e is a value of the λ-calculus if, and only if, $[\![e]\!]$ is a value of the $\lambda\mu\tilde{\mu}$-calculus.*

Proof. By inspection of the relevant cases. □

5 Call-by-value operational semantics

We introduce the evaluation rules for the λ-calculus and for the $\lambda\mu\tilde{\mu}$-calculus.

5.1 Evaluation in the λ-calculus

For the λ-calculus we first have to define how to reduce immediate redexes. We do this in Definition 5.1. One can note how all three rules implement the *call-by-value* strategy: a function application $(\lambda x.e_1)e_2$ can only be reduced if e_2 is a value; a projection $\pi_i(e_1, e_2)$ can only be reduced if both e_1 and e_2 are values; and a let-binding $\textbf{let}\, x = e_1 \,\textbf{in}\, e_2$ can only be reduced if e_1 is a value.

Definition 5.1. The *call-by-value evaluation rules for the λ-calculus* are:

$$(\lambda x.e)\, v \triangleright e[v/x] \tag{$\beta_\to$}$$
$$\pi_i(v_1, v_2) \triangleright v_i \tag{$\beta_\wedge$}$$
$$\textbf{let}\, x = v \,\textbf{in}\, e \triangleright e[v/x]. \tag{β_{let}}$$

These rules are not sufficient, since none of the rules are applicable to the term $(\pi_1(v_1, v_2), v_3)$, for example. We therefore need to extend them to allow for the reduction of redexes within a term. Furthermore, since we want evaluations to be deterministic, we must extend Definition 5.1 in such a way that there is always exactly one redex within a term which can be evaluated next. For example, in a tuple (e_1, e_2) we must specify whether we want to evaluate e_1 or e_2 first (and similarly for function applications $e_1\, e_2$).

We specify deterministic evaluations within a context by using the concept of evaluation contexts, introduced in [13]. An evaluation context $E[-]$ is a term with one argument place marked by the symbol □, which indicates where we evaluate the next immediate redex. Deterministic evaluation is ensured by a *unique decomposition lemma*:

Every term e that is not a value can be uniquely decomposed into an evaluation context $E[-]$ and an immediate redex e' such that $e \doteq E[e']$.

Definition 5.2. The syntax of *evaluation contexts* $E[-]$ is defined as follows:

$$E[-] ::= \Box \mid E\, e \mid v\, E \mid (E, e) \mid (v, E) \mid \mathbf{let}\, x = E\, \mathbf{in}\, e \mid \pi_i\, E.$$

Evaluation contexts now allow us to properly define evaluation within a context:

$$e \triangleright e' \implies E[e] \triangleright E[e']. \qquad \text{(Congruence)}$$

5.2 Evaluation in the $\lambda\mu\tilde{\mu}$-calculus

For the $\Lambda_{\mu\tilde{\mu}}$-calculus, we again first introduce the rules for evaluating immediate redexes. The choice of the *call-by-value* evaluation strategy is manifested in the following ways: first, a redex $\langle \lambda x.t \mid e \cdot s\rangle$ can only be reduced if the function argument e is a value; second, a redex $\langle (e_1, e_2) \mid \pi_i\, s\rangle$ can only be reduced if both e_1 and e_2 are values; third, the *critical pair* $\langle \mu\alpha.c_1 \mid \tilde{\mu}x.c_2\rangle$, which could a priori be reduced to either $c_1[\tilde{\mu}x.c_2/\alpha]$ or $c_2[\mu\alpha.c_2/x]$, is resolved by requiring in the rule $(\tilde{\mu})$ that a redex $\langle e \mid \tilde{\mu}x.c\rangle$ can only be reduced if e is a value.

Definition 5.3. The *call-by-value evaluation rules for the $\lambda\mu\tilde{\mu}$-calculus* are:

$$\begin{aligned}
\langle \lambda x.t \mid v \cdot s\rangle &\triangleright \langle t[v/x] \mid s\rangle && (\beta_{\rightarrow})\\
\langle (v_1, v_2) \mid \pi_i\, s\rangle &\triangleright \langle v_i \mid s\rangle && (\beta_{\wedge})\\
\langle \mu\alpha.c \mid s\rangle &\triangleright c[s/\alpha] && (\mu)\\
\langle v \mid \tilde{\mu}x.c\rangle &\triangleright c[v/x]. && (\tilde{\mu})
\end{aligned}$$

These rules are, again, not complete. For example, there is no rule applicable to the cut $\langle (\mu\alpha.c, v) \mid \pi_i\, s\rangle$, since the first element of the tuple is not yet a value. Instead of the evaluation contexts $E[-]$, we will add focusing contexts $F[-]$ and *dynamic focusing rules* ς. The focusing contexts play the role of the evaluation contexts for the λ-calculus, while the ς-rules correspond to the rule (Congruence).

Definition 5.4. The syntax of *focusing contexts* $F[-]$ is defined as follows:

$$F[-] ::= (\Box, t) \mid (w, \Box) \mid \Box \cdot s.$$

Definition 5.5. We extend the evaluation rules of Definition 5.3 by the following *dynamic focusing rules*:

$$\langle F[t] \mid s\rangle \triangleright \langle t \mid \tilde{\mu}x.\langle F[x] \mid s\rangle\rangle \quad \text{(if } t \text{ is not a value)} \qquad (\varsigma_1)$$
$$\langle v \mid F[t]\rangle \triangleright \langle t \mid \tilde{\mu}x.\langle v \mid F[x]\rangle\rangle \quad \text{(if } t \text{ is not a value).} \qquad (\varsigma_2)$$

6 The ANF-transformation

While the evaluation rules presented in Section 5 are sufficient for purely theoretical investigations into the reduction theory of the λ-calculus and the $\lambda\mu\tilde{\mu}$-calculus, they are less ideal for other purposes. In particular, they are not ideal for generating efficient code that can be run on a real computer. For example, consider the congruence rule in Definition 5.1. Its operational meaning implies that we have to *search* for the next redex in the term, and this redex can appear nested at an arbitrary depth within the term. If we implemented this search procedure naively for each reduction step, then the resulting program would be very inefficient indeed.

Various methods to efficiently evaluate terms of the λ-calculus have been proposed, for both the call-by-value and call-by-name evaluation orders. One of these methods is the compilation to *abstract machines*[2], like the SEK, SECD or Krivine machine, which provide much more efficient means of evaluating λ-terms. The evaluation of commands of the $\lambda\mu\tilde{\mu}$-calculus is, in fact, very similar to the evaluation of machine states of an abstract machine. Another class of methods for compiling terms of the ordinary λ-calculus is based on a translation into the so-called *continuation-passing style* (CPS), which was introduced in a seminal paper by Reynolds [21]. These translations have been studied

[2]For an introduction to the theory of abstract machines, cf. [14].

both in logic, where they correspond to double negation translations (cf. [23]), and in the theory of optimizing compilers (cf. [1]).

One important variation of these CPS translations is the so-called *ANF-transformation*, which was introduced by Sabry and Felleisen [22] and later elaborated by Flanagan et al. [15]. We first introduce the syntax of the administrative normal form in Definition 6.1. The ANF-transformation itself is introduced in Definitions 6.3 and 6.7.

Definition 6.1. The *syntax of the administrative normal form* Λ^{ANF} is defined as follows:

1. *Values:* $v ::= \lambda x.e \mid (v, v) \mid x$.
2. *Computations:* $c ::= v \mid v\ v \mid \pi_1\ v \mid \pi_2\ v$.
3. *Terms:* $e ::= c \mid \mathbf{let}\ x = c\ \mathbf{in}\ e$.

The administrative normal form has two characteristic properties. The first is reflected in the syntax of computations c: a projection π_i can only be applied to a value, and, similarly, a function application $v_1\ v_2$ can only be formed between two values. This excludes terms like $\pi_1(x, \pi_2(y, z))$ or $(\pi_1(f, g))(\pi_2(x, y))$. The second property is reflected in the syntax of terms e: a let-expression $\mathbf{let}\ x = c\ \mathbf{in}\ e$ can only bind the result of a computation c to a variable x. Let-expressions cannot bind other let-expressions, that is, expressions like $\mathbf{let}\ x = (\mathbf{let}\ y = e_1\ \mathbf{in}\ e_2)\ \mathbf{in}\ e_3$ are excluded by the second property.

Usual presentations of the ANF-transformation enforce both properties in a single transformation from Λ to Λ^{ANF}. Instead, we present the transformation to administrative normal form as a two-part transformation:

$$\Lambda \xrightarrow{\quad\mathcal{A}\quad} \Lambda^{\mathrm{Q}} \xrightarrow{\quad\mathcal{L}\quad} \Lambda^{\mathrm{ANF}}.$$

The first part consists of a function $\mathcal{A} : \Lambda \to \Lambda^{\mathrm{Q}}$ that enforces only the first of the two characteristic properties described above. A second transformation $\mathcal{L} : \Lambda^{\mathrm{Q}} \to \Lambda^{\mathrm{ANF}}$ then enforces the second property. By presenting the ANF-transformation in this way, we can make the relation to focusing clearer. In Section 8 we will show that the first part of this transformation corresponds to focusing, whereas the second part of the transformation can be simulated by μ-reductions in $\Lambda_{\mu\tilde{\mu}}$.

Definition 6.2. The *syntax of the intermediate normal form* Λ^{Q} is defined as follows:

1. *Values:* $v ::= \lambda x.e \mid (v, v) \mid x$.
2. *Terms:* $e ::= v \mid \mathbf{let}\, x = e\, \mathbf{in}\, e \mid e\, v \mid \pi_1\, e \mid \pi_2\, e$.

Note that Definition 6.2 only guarantees that pairs (v, v) consist of values, and that functions are always applied to values in a function application $e\, v$. The two transformations $\mathcal{A}$ and $\mathcal{L}$ are introduced in turn.

6.1 From Λ to Λ^{Q}

Recall that the first property that we want to enforce is that pairs consist of syntactic values, and that in function applications the function argument is already a value. The transformation $\mathcal{A}$ defined next guarantees the first property by binding any non-value argument which would violate this property to a fresh variable in a let-binding. For example, the term $\pi_1(\pi_2(x, y))$ is transformed by generating a fresh variable z, and binding the computation $\pi_2(x, y)$ to z in the computation $\pi_1\, z$: $\mathcal{A}(\pi_1(\pi_2(x, y))) :\hat{=} \mathbf{let}\, z = \pi_2(x, y)\, \mathbf{in}\, \pi_1\, z$.

Definition 6.3. The *transformation* $\mathcal{A} : \Lambda \to \Lambda^{\mathrm{Q}}$ is defined as follows:

$$\begin{aligned}
\mathcal{A}(x) &:\hat{=} x && (\mathcal{A}_1)\\
\mathcal{A}(\lambda x.e) &:\hat{=} \lambda x.\mathcal{A}(e) && (\mathcal{A}_2)\\
\mathcal{A}(\mathbf{let}\, x = e_1\, \mathbf{in}\, e_2) &:\hat{=} \mathbf{let}\, x = \mathcal{A}(e_1)\, \mathbf{in}\, \mathcal{A}(e_2) && (\mathcal{A}_3)\\
\mathcal{A}(\pi_i\, e) &:\hat{=} \pi_i(\mathcal{A}(e)) && (\mathcal{A}_4)
\end{aligned}$$

$$\begin{aligned}
\mathcal{A}((v_1, v_2)) &:\hat{=} (\mathcal{A}(v_1), \mathcal{A}(v_2)) && (\mathcal{A}_5)\\
\mathcal{A}((v_1, e_2)) &:\hat{=} \mathbf{let}\, x = \mathcal{A}(e_2)\, \mathbf{in}\, (\mathcal{A}(v_1), x) && (\mathcal{A}_6)\\
\mathcal{A}((e_1, v_2)) &:\hat{=} \mathbf{let}\, x = \mathcal{A}(e_1)\, \mathbf{in}\, (x, v_2) && (\mathcal{A}_7)\\
\mathcal{A}((e_1, e_2)) &:\hat{=} \mathbf{let}\, x = \mathcal{A}(e_1)\, \mathbf{in}\, (\mathbf{let}\, y = \mathcal{A}(e_2)\, \mathbf{in}\, (x, y)) && (\mathcal{A}_8)
\end{aligned}$$

$$\mathcal{A}(e_1\, v_2) :\hat{=} \mathcal{A}(e_1)\, \mathcal{A}(v_2) \tag{$\mathcal{A}_9$}$$

$$\mathcal{A}(e_1\, e_2) :\hat{=} \textbf{let}\; x = \mathcal{A}(e_2)\; \textbf{in}\; \mathcal{A}(e_1)\, x. \tag{$\mathcal{A}_{10}$}$$

Remark 6.4. Among the clauses of Definition 6.3, the clauses ($\mathcal{A}_5$) to ($\mathcal{A}_7$) are subsumed by ($\mathcal{A}_8$). Similarly, the clause ($\mathcal{A}_9$) is subsumed by ($\mathcal{A}_{10}$). This redundancy is an optimization which guarantees that the transformation behaves as the identity function on terms that are already in Λ^{Q}.

Example 6.5. The result of the transformation

$$\mathcal{A}(\pi_1(\pi_1(\pi_1(x_1, x_2), x_3), x_4))$$

is the term

$$\textbf{let}\; z_1 = (\textbf{let}\; z_2 = \pi_1(x_1, x_2)\; \textbf{in}\; \pi_1(z_2, x_3))\; \textbf{in}\; \pi_1(z_1, x_4),$$

where z_1 and z_2 are variables that are generated during the transformation. This example shows that the result of $\mathcal{A}$ is, in general, not yet in Λ^{ANF}.

6.2 From Λ^{Q} to Λ^{ANF}

The second property which we want to enforce is that in a let-construct $\textbf{let}\; x = c\; \textbf{in}\; e$ the computation bound to the variable x must be of a restricted form. This will be guaranteed by the transformation $\mathcal{L}$, given by Definition 6.7. In order to define this transformation, we need to define a meta-level operation @ that operates on continuations k and values v from Λ^{ANF}:

Definition 6.6. *Continuations* are defined as follows:

$$k ::= \mathrm{id} \mid \overline{\lambda v}.\textbf{let}\; x = \pi_i\, \overline{v}\; \textbf{in}\; e \mid \overline{\lambda v}.\textbf{let}\; x = \overline{v}\, v\; \textbf{in}\; e \mid \overline{\lambda v}.\textbf{let}\; x = \overline{v}\; \textbf{in}\; e,$$

where e and v range over expressions and values from Λ^{ANF}.

The *meta-level operation* @ takes a continuation k and a value v from Λ^{ANF} and returns an expression of Λ^{ANF}. It is evaluated as follows:

$$\mathrm{id}\;@\;v :\hat{=} v \tag{$@_1$}$$

$$\overline{\lambda v}.\mathbf{let}\ x = \pi_i\ \overline{v}\ \mathbf{in}\ e \mathbin{@} v :\simeq \mathbf{let}\ x = \pi_i\ v\ \mathbf{in}\ e \qquad (@_2)$$

$$\overline{\lambda v}.\mathbf{let}\ x = \overline{v}\ v_2\ \mathbf{in}\ e \mathbin{@} v_1 :\simeq \mathbf{let}\ x = v_1\ v_2\ \mathbf{in}\ e \qquad (@_3)$$

$$\overline{\lambda v}.\mathbf{let}\ x = \overline{v}\ \mathbf{in}\ e \mathbin{@} v :\simeq \mathbf{let}\ x = v\ \mathbf{in}\ e. \qquad (@_4)$$

Using this technical tool we can now define the transformation $\mathcal{L}$.

Definition 6.7. The *transformation* $\mathcal{L} : \Lambda^{\mathrm{Q}} \to \Lambda^{\mathrm{ANF}}$ is given as follows:

Values

$$\mathcal{L}(x) :\simeq x \qquad (\mathcal{L}_1)$$

$$\mathcal{L}(\lambda x.e) :\simeq \lambda x.\mathcal{L}_{\mathrm{id}}(e) \qquad (\mathcal{L}_2)$$

$$\mathcal{L}((v_1, v_2)) :\simeq (\mathcal{L}(v_1), \mathcal{L}(v_2)) \qquad (\mathcal{L}_3)$$

Terms

$$\mathcal{L}(e) :\simeq \mathcal{L}_{\mathrm{id}}(e) \qquad (\mathcal{L}_4)$$

$$\mathcal{L}_k(e_1\ v_2) :\simeq \mathcal{L}_{\overline{\lambda v}.\mathbf{let}\ x=\overline{v}\ \mathcal{L}(v_2)\ \mathbf{in}\ k \mathbin{@} x}(e_1) \qquad (\mathcal{L}_5)$$

$$\mathcal{L}_k(\pi_i\ e) :\simeq \mathcal{L}_{\overline{\lambda v}.\mathbf{let}\ x=\pi_i\ \overline{v}\ \mathbf{in}\ k \mathbin{@} x}(e) \qquad (\mathcal{L}_6)$$

$$\mathcal{L}_k(v) :\simeq k \mathbin{@} \mathcal{L}(v) \qquad (\mathcal{L}_7)$$

$$\mathcal{L}_k(\mathbf{let}\ x = e_1\ \mathbf{in}\ e_2) :\simeq \mathcal{L}_{\overline{\lambda v}.\mathbf{let}\ x=\overline{v}\ \mathbf{in}\ \mathcal{L}_k(e_2)}(e_1). \qquad (\mathcal{L}_8)$$

Example 6.8. As an example of the transformation $\mathcal{L}$, consider the term (from Example 6.5)

$$\mathbf{let}\ z_1 = (\mathbf{let}\ z_2 = \pi_1(x_1, x_2)\ \mathbf{in}\ \pi_1(z_2, x_3))\ \mathbf{in}\ \pi_1(z_1, x_4),$$

which can be transformed into Λ^{ANF} as follows:

$$\begin{aligned}
&\mathcal{L}_{\mathrm{id}}(\mathbf{let}\ z_1 = (\mathbf{let}\ z_2 = \pi_1(x_1, x_2)\ \mathbf{in}\ \pi_1(z_2, x_3))\ \mathbf{in}\ \pi_1(z_1, x_4)) \\
&\quad = \mathcal{L}_{\overline{\lambda v_1}.\mathbf{let}\ z_1=\overline{v_1}\ \mathbf{in}\ \mathcal{L}_{\mathrm{id}}(\pi_1(z_1,x_4))}(\mathbf{let}\ z_2 = \pi_1(x_1, x_2)\ \mathbf{in}\ \pi_1(z_2, x_3)) \\
&\quad = \mathcal{L}_{\overline{\lambda v_1}.\mathbf{let}\ z_1=\overline{v_1}\ \mathbf{in}\ \pi_1(z_1,x_4)}(\mathbf{let}\ z_2 = \pi_1(x_1, x_2)\ \mathbf{in}\ \pi_1(z_2, x_3)) \\
&\quad = \mathcal{L}_{\overline{\lambda v_2}.\mathbf{let}\ z_2=\overline{v_2}\ \mathbf{in}\ \mathcal{L}_{\overline{\lambda v_1}.\mathbf{let}\ z_1=\overline{v_1}\ \mathbf{in}\ \pi_1(z_1,x_4)}(\pi_1(z_2,x_3))}(\pi_1(x_1, x_2))
\end{aligned}$$

$$= \mathcal{L}_{\overline{\lambda v_2}.\mathbf{let}\ z_2=\overline{v_2}\ \mathbf{in}\ (\mathbf{let}\ z_1=\pi_1(z_2,x_3)\ \mathbf{in}\ \pi_1(z_1,x_4))}(\pi_1(x_1,x_2))$$

$$= \mathbf{let}\ z_2 = \pi_1(x_1,x_2)\ \mathbf{in}\ (\mathbf{let}\ z_1 = \pi_1(z_2,x_3)\ \mathbf{in}\ \pi_1(z_1,x_4)).$$

The term was transformed into Λ^{ANF} by (in a certain way) moving the let-binding of z_2 to the outside of the let-binding of z_1.

7 The focusing transformation

In distinction to the dynamic focusing rules of Definition 5.5, we now consider only static focusing. We first introduce the focused subsyntax $\Lambda^{\text{Q}}_{\mu\tilde{\mu}}$ as a subset of $\Lambda_{\mu\tilde{\mu}}$ (Definition 3.4).[3]

Definition 7.1. The *focused subsyntax* $\Lambda^{\text{Q}}_{\mu\tilde{\mu}}$ for the call-by-value evaluation strategy is defined as follows:

1. *Terms:* $t ::= w \mid \mu\alpha.c$.
2. *Coterms:* $s ::= \alpha \mid w \cdot s \mid \pi_1\ s \mid \pi_2\ s \mid \tilde{\mu}x.c$.
3. *Commands:* $c ::= \langle t \mid s \rangle$.
4. *Values:* $w ::= \lambda x.t \mid (w, w) \mid x$.

The focused subsyntax $\Lambda^{\text{Q}}_{\mu\tilde{\mu}}$ differs in two respects from $\Lambda_{\mu\tilde{\mu}}$. First, terms t must now either be values w or abstractions $\mu\alpha.c$. This excludes terms like $(\mu\alpha.c, t)$ and $(t, \mu\alpha.c)$ from the subsyntax $\Lambda^{\text{Q}}_{\mu\tilde{\mu}}$, which are part of the syntax of terms of Definition 3.4. This corresponds precisely to the restriction that constructors can only be applied to values. Second, the syntax of coterms has been changed by requiring the function argument in a coterm $t \cdot s$ to be a value; that is, we require $w \cdot s$. This corresponds to the requirement that functions can syntactically only be applied to values.

Lemma 7.2. *For any term* $e \in \Lambda^{\text{Q}}$*,* $[\![e]\!] \in \Lambda^{\text{Q}}_{\mu\tilde{\mu}}$.

Proof. By induction on e.

[3] $\Lambda^{\text{Q}}_{\mu\tilde{\mu}}$ corresponds to the subsyntax LKQ defined in [4].

1. Case $e \doteq \mathbf{let}\, x = e_1 \,\mathbf{in}\, e_2$: the translation of e is $\mu\alpha.\langle [\![e_1]\!] \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha\rangle\rangle$. Using the induction hypothesis for e_1 and e_2, this term is in the subsyntax $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$.
2. If e is of the form $e_1\, v_2$, then the translation of e is $\mu\alpha.\langle [\![e_1]\!] \mid [\![v_2]\!] \cdot \alpha\rangle$. By Lemma 4.3, $[\![v_2]\!]$ is a value, and by the induction hypothesis both $[\![e_1]\!]$ and $[\![v_2]\!]$ are in the subsyntax $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$, so the resulting term is in the subsyntax $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$.
3. If e is of the form $\pi_i\, e_1$, then $[\![\pi_i\, e_1]\!]$ is $\mu\alpha.\langle [\![e_1]\!] \mid \pi_i\, \alpha\rangle$. Using the induction hypothesis for e_1, $[\![e_1]\!]$ is in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$. Therefore $[\![\pi_i\, e_1]\!]$ is also in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$.
4. If $e \doteq v$, then we have to distinguish the following cases:
   (a) If $v \doteq x$, then $[\![x]\!] \doteq x$, which is in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$.
   (b) If $v \doteq (v_1, v_2)$, then $[\![(v_1, v_2)]\!] \doteq ([\![v_1]\!], [\![v_2]\!])$. By Lemma 4.3, both $[\![v_i]\!]$ are values, and by the induction hypothesis they are in the subsyntax $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$. Therefore $[\![v]\!]$ is also in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$.
   (c) If $v \doteq \lambda x.e$, then $[\![\lambda x.e]\!] \doteq \lambda x.[\![e]\!]$. By the induction hypothesis $[\![e]\!]$ is in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$, therefore $[\![v]\!]$ is also in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$. □

The subsyntax does not restrict the set of derivable sequents, since any term, coterm or command in the unrestricted syntax can be translated into the focused subsyntax $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$ by using the following *static* focusing transformation.

Definition 7.3. The *static focusing transformation* $\mathcal{F} : \Lambda_{\mu\tilde{\mu}} \to \Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$ is defined as follows:

Terms

$$\mathcal{F}(x) :\doteq x \qquad (\mathcal{F}_1)$$
$$\mathcal{F}(\mu\alpha.c) :\doteq \mu\alpha.\mathcal{F}(c) \qquad (\mathcal{F}_2)$$
$$\mathcal{F}(\lambda x.e) :\doteq \lambda x.\mathcal{F}(e) \qquad (\mathcal{F}_3)$$
$$\mathcal{F}((w_1, w_2)) :\doteq (\mathcal{F}(w_1), \mathcal{F}(w_2)) \qquad (\mathcal{F}_4)$$
$$\mathcal{F}((w_1, t_2)) :\doteq \mu\alpha.\langle \mathcal{F}(t_2) \mid \tilde{\mu}x.\langle (\mathcal{F}(w_1), x) \mid \alpha\rangle\rangle \qquad (\mathcal{F}_5)$$

$$\mathcal{F}((t_1, w_2)) :\doteq \mu\alpha.\langle \mathcal{F}(t_1) \mid \tilde{\mu}x.\langle (x, \mathcal{F}(w_2)) \mid \alpha\rangle\rangle \qquad (\mathcal{F}_6)$$
$$\mathcal{F}((t_1, t_2)) :\doteq \mu\alpha.\langle \mathcal{F}(t_1) \mid \tilde{\mu}x.\langle \mu\beta.\langle \mathcal{F}(t_2) \mid \tilde{\mu}y.\langle (x, y) \mid \beta\rangle\rangle \mid \alpha\rangle\rangle \qquad (\mathcal{F}_7)$$

Coterms

$$\mathcal{F}(\alpha) :\doteq \alpha \qquad (\mathcal{F}_8)$$
$$\mathcal{F}(\tilde{\mu}x.c) :\doteq \tilde{\mu}x.\mathcal{F}(c) \qquad (\mathcal{F}_9)$$
$$\mathcal{F}(\pi_i\, s) :\doteq \pi_i\, \mathcal{F}(s) \qquad (\mathcal{F}_{10})$$
$$\mathcal{F}(w \cdot s) :\doteq \mathcal{F}(w) \cdot \mathcal{F}(s) \qquad (\mathcal{F}_{11})$$
$$\mathcal{F}(t \cdot s) :\doteq \tilde{\mu}x.\langle \mathcal{F}(t) \mid \tilde{\mu}y.\langle x \mid y \cdot \mathcal{F}(s)\rangle\rangle \qquad (\mathcal{F}_{12})$$

Commands

$$\mathcal{F}(\langle t \mid s\rangle) :\doteq \langle \mathcal{F}(t) \mid \mathcal{F}(s)\rangle \qquad (\mathcal{F}_{13})$$
$$\mathcal{F}(\langle t_1 \mid t_2 \cdot s\rangle) :\doteq \langle \mathcal{F}(t_2) \mid \tilde{\mu}x.\langle \mu\alpha.\langle \mathcal{F}(t_1) \mid x \cdot \alpha\rangle \mid \mathcal{F}(s)\rangle\rangle. \qquad (\mathcal{F}_{14})$$

In general, when several clauses are applicable, the most specific clause should be applied. The clauses ($\mathcal{F}_4$), ($\mathcal{F}_5$) and ($\mathcal{F}_6$) are subsumed by the more general clause ($\mathcal{F}_7$), and ($\mathcal{F}_{11}$) is subsumed by the clause ($\mathcal{F}_{12}$). The presence of these additional clauses guarantees that $\mathcal{F}$ behaves as the identity function when it is applied to a term, coterm or command that is already in the subsyntax $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$. With these optimizations, our definition corresponds to the one given in [6, Fig. 18], with the exception of the clause ($\mathcal{F}_{14}$). The additional clause ($\mathcal{F}_{14}$) is necessary to guarantee that the functions $[\![-]\!]$, $\mathcal{A}$ and $\mathcal{F}$ commute up to α-equivalence, as shown by Theorem 8.1. Without the clause ($\mathcal{F}_{14}$), Theorem 8.1 has to be slightly weakened to Theorem 8.2.

Lemma 7.4 ($\mathcal{F}$ preserves typeability)**.** *For all terms t, coterms s and commands c:*

1. *If* $\Gamma \vdash t : \tau \mid \Delta$*, then* $\Gamma \vdash \mathcal{F}(t) : \tau \mid \Delta$.
2. *If* $\Gamma \mid s : \tau \vdash \Delta$*, then* $\Gamma \mid \mathcal{F}(s) : \tau \vdash \Delta$.
3. *If* $c : (\Gamma \vdash \Delta)$*, then* $\mathcal{F}(c) : (\Gamma \vdash \Delta)$.

Proof. By simultaneous structural induction on t, s and c, respectively. □

8 The main result

As explained in Section 6, the ANF-transformation can be split into a purely local transformation $\mathcal{A}$ and a global transformation $\mathcal{L}$. We show what these two parts correspond to in the $\lambda\mu\tilde{\mu}$-calculus, and prove how the ANF-transformation on λ-terms relates to static focusing of $\lambda\mu\tilde{\mu}$-terms.

8.1 The correspondence between $\mathcal{A}$ and $\mathcal{F}$

Theorem 8.1 (Focusing reflects the ANF-transformation)**.** *For all λ-terms e, we have $\mathcal{F}(\llbracket e \rrbracket) \doteq \llbracket \mathcal{A}(e) \rrbracket$.*

Proof. See Appendix A. □

If we omit the focusing rule ($\mathcal{F}_{14}$) from Definition 7.3, then Theorem 8.1 no longer holds up to syntactic equality ($\doteq$). Instead, the following weaker result (Theorem 8.2) holds for $\eta\mu$-equality $\equiv$, which includes η-equivalence

$$\tilde{\mu}x.\langle x \mid s \rangle \equiv_\eta s \qquad \text{(for } x \text{ not free in } s\text{)}.$$

Theorem 8.2 (Focusing reflects the ANF-transformation; case $\equiv$)**.** *For all λ-terms e, we have $\mathcal{F}(\llbracket e \rrbracket) \equiv \llbracket \mathcal{A}(e) \rrbracket$.*

Proof. See Appendix A. □

8.2 Simulating $\mathcal{L}$ in the $\lambda\mu\tilde{\mu}$-calculus

Our main contention in this section is that a special purpose transformation like $\mathcal{L}$ is not necessary in $\Lambda_{\mu\tilde{\mu}}$. In order to transform from $\Lambda_{\mu\tilde{\mu}}^{\mathrm{Q}}$ to $\Lambda_{\mu\tilde{\mu}}^{\mathrm{ANF}}$ we only have to apply μ-reductions and $\tilde{\mu}$-expansions. More concretely, the effect that $\mathcal{L}$ has on a term, namely to globally reorganize the ordering of let-bindings, can be simulated by simply reducing μ-redexes in the image of the translation. In order to illustrate this central point, let us come back to Examples 6.5 and 6.8. Recall that we showed in Example 6.8 that $\mathcal{L}$ has the effect of changing the order of the two let-bindings of z_1 and z_2:

$$\begin{aligned}&\mathcal{L}(\mathbf{let}\, z_1 = (\mathbf{let}\, z_2 = \pi_1(x_1, x_2)\,\mathbf{in}\,\pi_1(z_2, x_3))\,\mathbf{in}\,\pi_1(z_1, x_4))\\ &\qquad\quad \doteq \mathbf{let}\, z_2 = \pi_1(x_1, x_2)\,\mathbf{in}\,(\mathbf{let}\, z_1 = \pi_1(z_2, x_3)\,\mathbf{in}\,\pi_1(z_1, x_4)).\end{aligned}$$

This can be simulated as follows:

$$\begin{aligned}&[\![\mathbf{let}\, z_1 = (\mathbf{let}\, z_2 = \pi_1(x_1, x_2)\,\mathbf{in}\,\pi_1(z_2, x_3))\,\mathbf{in}\,\pi_1(z_1, x_4)]\!]\\ &\doteq \mu\alpha.\underline{\langle \mu\beta.\langle [\![\pi_1(x_1, x_2)]\!] \mid \tilde{\mu}z_2.\langle [\![\pi_1(z_2, x_3)]\!] \mid \beta\rangle\rangle \mid \tilde{\mu}z_1.\langle [\![\pi_1(z_1, x_4)]\!] \mid \alpha\rangle\rangle}\\ &\triangleright \mu\alpha.\langle [\![\pi_1(x_1, x_2)]\!] \mid \tilde{\mu}z_2.\langle [\![\pi_1(z_2, x_3)]\!] \mid \tilde{\mu}z_1.\langle [\![\pi_1(z_1, x_4)]\!] \mid \alpha\rangle\rangle\rangle\\ &\triangleleft \mu\alpha.\langle [\![\pi_1(x_1, x_2)]\!] \mid \tilde{\mu}z_2.\underline{\langle \mu\beta.\langle [\![\pi_1(z_2, x_3)]\!] \mid \tilde{\mu}z_1.\langle [\![\pi_1(z_1, x_4)]\!] \mid \beta\rangle\rangle \mid \alpha\rangle}\rangle\\ &\doteq [\![\mathbf{let}\, z_2 = \pi_1(x_1, x_2)\,\mathbf{in}\,(\mathbf{let}\, z_1 = \pi_1(z_2, x_3)\,\mathbf{in}\,\pi_1(z_1, x_4))]\!].\end{aligned}$$

Next, we define the subsyntax $\Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}$, which differs from $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$ (Definition 7.1) in two aspects. First, commands are now required to consist of a *value* and a coterm instead of a term and a coterm, i.e., they do not contain any μ-redexes. Second, the coterms for projections and function applications are required to give an explicit name to the value they bind in the coterm they contain, i.e., they are $\tilde{\mu}$-expanded.

Definition 8.3. The *focused subsyntax* $\Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}$ for the call-by-value strategy is defined as follows:

1. *Terms:* $t ::= w \mid \mu\alpha.c$.
2. *Coterms:* $s ::= \alpha \mid w \cdot \tilde{\mu}x.c \mid \pi_1\,\tilde{\mu}x.c \mid \pi_2\,\tilde{\mu}x.c \mid \tilde{\mu}x.c$
3. *Commands:* $c ::= \langle w \mid s\rangle$.
4. *Values:* $w ::= \lambda x.t \mid (w, w) \mid x$.

We have to refine Definition 4.1.

Definition 8.4. The *refined translation* $[\![-]\!]^* : \Lambda^{\mathrm{ANF}} \to \Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}$ is defined as the first function in the following set of mutually defined recursive functions:

First function (on expressions)

$$*[\![e]\!]^* :\doteq \mu\alpha.[\![e]\!]^*_\alpha \qquad (\mathcal{T}_1^*)$$

Second function (on expressions)

$$*[\![\mathbf{let}\; x = c \;\mathbf{in}\; e]\!]^*_s :\doteq [\![c]\!]^*_{\tilde{\mu}x.[\![e]\!]^*_s} \qquad (\mathcal{T}_2^*)$$

$$[\![v_1\; v_2]\!]^*_s :\doteq \langle [\![v_1]\!]^* \mid [\![v_2]\!]^* \cdot s\rangle \qquad (\mathcal{T}_3^*)$$

$$[\![\pi_i\; v]\!]^*_s :\doteq \langle [\![v]\!]^* \mid \pi_i\; s\rangle \qquad (\mathcal{T}_4^*)$$

$$[\![v]\!]^*_s :\doteq \langle [\![v]\!]^* \mid s\rangle \qquad (\mathcal{T}_5^*)$$

Third function (on values)

$$*[\![x]\!]^* :\doteq x \qquad (\mathcal{T}_6^*)$$

$$[\![(v_1, v_2)]\!]^* :\doteq ([\![v_1]\!]^*, [\![v_2]\!]^*) \qquad (\mathcal{T}_7^*)$$

$$[\![\lambda x.e]\!]^* :\doteq \lambda x.[\![e]\!]^*. \qquad (\mathcal{T}_8^*)$$

Lemma 8.5. *For all terms $e \in \Lambda^{\mathrm{ANF}}$, $[\![e]\!]^* \in \Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}$.*

Proof. By induction on terms e. □

Theorem 8.6. *For all terms $e \in \Lambda^{\mathrm{Q}}$, $[\![\mathcal{L}(e)]\!]^* \equiv_\mu [\![e]\!]$.*

Proof. See Appendix A. □

9 Summary and outlook

We can summarize our results in the following diagram, where both the lower and the upper part are commutative.

$$\begin{tikzcd}
\Lambda \text{ (Def. 3.2)} \arrow[r, "[\![-]\!]"] \arrow[d, "\mathcal{A} \text{ (Def. 6.3)}"'] \arrow[dr, phantom, "\text{(Theorem 8.1)}"] & \Lambda_{\mu\tilde{\mu}} \text{ (Def. 3.4)} \arrow[d, "\mathcal{F} \text{ (Def. 7.3)}"] \\
\Lambda^{\mathrm{Q}} \text{ (Def. 6.2)} \arrow[r, "[\![-]\!]"] \arrow[d, "\mathcal{L} \text{ (Def. 6.7)}"'] \arrow[dr, phantom, "\text{(Theorem 8.6)}"] & \Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}} \text{ (Def. 7.1)} \arrow[d, "\mu\text{-reduction}"] \\
\Lambda^{\mathrm{ANF}} \text{ (Def. 6.1)} \arrow[r, "[\![-]\!]^*"] & \Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}} \text{ (Def. 8.3)}
\end{tikzcd}$$

These results are embedded in a wider conceptual context. By the Curry-Howard correspondence, natural deduction for intuitionistic logic (more precisely, the $\{\rightarrow, \wedge\}$-fragment) corresponds to the λ-calculus on the one side, and the sequent calculus for classical logic corresponds to the $\lambda\mu\tilde{\mu}$-calculus on the other side. Our results thus establish a bridge between natural deduction for intuitionistic logic with its computational interpretation on the one side and the classical sequent calculus with its computational interpretation on the other side.

We would like to extend this work in two directions. The first concerns the asymmetry of the translation $[\![-]\!]$, which is only a mapping from Λ to $\Lambda_{\mu\tilde{\mu}}$, but not vice versa. In order to provide a translation in the opposite direction, from $\Lambda_{\mu\tilde{\mu}}$ to Λ, we will have to extend the λ-calculus with control operators. The seond extension concerns the treatment of evaluation orders other than call-by-value. While the treatment of call-by-name seems to be straightforward, the study of call-by-need (cf. [3, 19]) and its dual call-by-co-need (cf. [5]) in both the λ-calculus and the $\lambda\mu\tilde{\mu}$-calculus seems to be promising.

Acknowledgements

We would like to thank Jonathan Brachthäuser for discussions, and Philipp Schuster and Paul Downen for helpful remarks on an earlier version of this paper. This work was carried out within the DFG project "Constructive Semantics and the Completeness Problem", DFG grant PI 1174/1-1.

References

[1] Andrew W. Appel. *Compiling with Continuations.* Cambridge University Press, 1992.

[2] Zena M. Ariola and Hugo Herbelin. Minimal classical logic and control operators. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, pages 871–885. Springer, 2003.

[3] Zena M. Ariola, John Maraist, Martin Odersky, Matthias Felleisen, and Philip Wadler. A call-by-need lambda calculus. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 233–246, New York, NY, USA, 1995. Association for Computing Machinery. doi: 10.1145/199448.199507.

[4] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*, ICFP '00, pages 233–243, New York, NY, USA, 2000. Association for Computing Machinery.

[5] Paul Downen and Zena M. Ariola. Beyond polarity: Towards a multi-discipline intermediate language with sharing. In *27th EACSL Annual Conference on Computer Science Logic (CSL 2018).* Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

[6] Paul Downen and Zena M. Ariola. A tutorial on computational classical logic and the sequent calculus. *Journal of Functional Programming*, 28:e3, 2018. doi: 10.1017/S0956796818000023.

[7] Marie Duží. Structural isomorphism of meaning and synonymy. *Computación y Sistemas*, 18(3):439–453, 2014.

[8] Marie Duží. Natural Language Processing by Natural Deduction in Transparent Intensional Logic. In Thomas Piecha and Peter Schroeder-Heister, editors, *Proof-Theoretic Semantics: Assessment and Future Perspectives. Proceedings of the Third Tübingen Conference on Proof-Theoretic Semantics, 27-30 March 2019*, pages 41, 685–706. University of Tübingen, 2019. doi: 10.15496/publikation-35319.

[9] Marie Duží and Bjørn Jespersen. Procedural isomorphism, analytic information and β-conversion by value. *Logic Journal of the IGPL*, 21(2):291–308, 2012. doi: 10.1093/jigpal/jzs044.

[10] Marie Duží and Bjørn Jespersen. Transparent quantification into hyperpropositional contexts *de re*. *Logique et Analyse*, 55(220): 513–554, 2012.

[11] Marie Duží and Bjørn Jespersen. Transparent quantification into hyperintensional objectual attitudes. *Synthese*, 192:635–677, 2015.

[12] Marie Duží and Miloš Kosterec. A valid rule of β-conversion for the logic of partial functions. *Organon F*, 24(1):10–36, 2017.

[13] Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.

[14] Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex.* MIT Press, 2009.

[15] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, PLDI ’93, pages 237–247, New York, NY, USA, 1993. Association for Computing Machinery.

[16] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[17] Timothy G. Griffin. A formulae-as-type notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 47–58, New York, NY, USA, 1989. Association for Computing Machinery. doi: 10.1145/96709.96714.

[18] Bjørn Jespersen and Marie Duží. Transparent quantification into hyperpropositional attitudes de dicto. *Linguistics and Philosophy*, 45:1119–1164, 2022. doi: 10.1007/s10988-021-09344-9.

[19] John Launchbury. A natural semantics for lazy evaluation. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93, pages 144–154, New York, NY, USA, 1993. Association for Computing Machinery. doi: 10.1145/158511.158618.

[20] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, 2002.

[21] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of the ACM Annual Conference – Volume 2*, ACM '72, pages 717–740, New York, NY, USA, 1972. Association for Computing Machinery. ISBN 9781450374927. doi: 10.1145/800194.805852.

[22] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In *Proceedings of the 1992 ACM Conference on LISP and Functional Programming*, LFP '92, pages 288–298, New York, NY, USA, 1992. Association for Computing Machinery.

[23] Morten Heine Sørensen and Paweł Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, 2006.

A Proofs of the main theorems

Theorem 8.1 (Focusing reflects the ANF-transformation)**.** *For all λ-terms e, we have $\mathcal{F}(\llbracket e \rrbracket) \doteq \llbracket \mathcal{A}(e) \rrbracket$.*

Proof. By induction on the structure of e.

1. Case $e \doteq x$: $\mathcal{F}(\llbracket x \rrbracket) \doteq x \doteq \llbracket \mathcal{A}(x) \rrbracket$.

2. Case $e \doteq \lambda x.e_1$:

$$\mathcal{F}(\llbracket \lambda x.e_1 \rrbracket) \doteq \lambda x.\mathcal{F}(\llbracket e_1 \rrbracket) \overset{\text{IH}}{\doteq} \lambda x.\llbracket \mathcal{A}(e_1) \rrbracket \doteq \llbracket \mathcal{A}(\lambda x.e_1) \rrbracket.$$

3. Case $e \doteq \pi_i\, e_1$:

$$\begin{aligned}
\mathcal{F}(\llbracket \pi_i\, e_1 \rrbracket) &\doteq \mathcal{F}(\mu\alpha.\langle \llbracket e_1 \rrbracket \mid \pi_i\, \alpha \rangle) && (\mathcal{T}_5)\\
&\doteq \mu\alpha.\langle \mathcal{F}(\llbracket e_1 \rrbracket) \mid \pi_i\, \alpha \rangle && (\mathcal{F}_2,\, \mathcal{F}_{13},\, \mathcal{F}_{10},\, \mathcal{F}_8)\\
&\doteq \mu\alpha.\langle \llbracket \mathcal{A}(e_1) \rrbracket \mid \pi_i\, \alpha \rangle && (\text{IH})\\
&\doteq \llbracket \pi_i\, \mathcal{A}(e_1) \rrbracket && (\mathcal{T}_5)\\
&\doteq \llbracket \mathcal{A}(\pi_i\, e_1) \rrbracket. && (\mathcal{A}_4)
\end{aligned}$$

4. In case $e \doteq e_1\, e_2$, we have to distinguish two subcases:

   (i) Subcase $e \doteq e_1\, v_2$:

$$\begin{aligned}
\mathcal{F}(\llbracket e_1\, v_2 \rrbracket) &\doteq \mathcal{F}(\mu\alpha.\langle \llbracket e_1 \rrbracket \mid \llbracket v_2 \rrbracket \cdot \alpha \rangle) && (\mathcal{T}_4)\\
&\doteq \mu\alpha.\langle \mathcal{F}(\llbracket e_1 \rrbracket) \mid \mathcal{F}(\llbracket v_2 \rrbracket) \cdot \alpha \rangle && (\mathcal{F}_2,\, \mathcal{F}_{13},\, \mathcal{F}_{11},\, \mathcal{F}_8)\\
&\doteq \mu\alpha.\langle \llbracket \mathcal{A}(e_1) \rrbracket \mid \llbracket \mathcal{A}(v_2) \rrbracket \cdot \alpha \rangle && (\text{IH})\\
&\doteq \llbracket \mathcal{A}(e_1)\, \mathcal{A}(v_2) \rrbracket && (\mathcal{T}_4)\\
&\doteq \llbracket \mathcal{A}(e_1\, v_2) \rrbracket. && (\mathcal{A}_9)
\end{aligned}$$

   (ii) Subcase $e \doteq e_1\, e_2$:

$$\begin{aligned}
\mathcal{F}(\llbracket (e_1\, e_2) \rrbracket) &\doteq \mathcal{F}(\mu\alpha.\langle \llbracket e_1 \rrbracket \mid \llbracket e_2 \rrbracket \cdot \alpha \rangle) && (\mathcal{T}_4)\\
&\doteq \mu\alpha.\langle \mathcal{F}(\llbracket e_2 \rrbracket) \mid \tilde{\mu}x.\langle \mu\beta.\langle \mathcal{F}(\llbracket e_1 \rrbracket) \mid x \cdot \beta \rangle \mid \alpha \rangle \rangle\\
& && (\mathcal{F}_{14},\, \mathcal{F}_2,\, \mathcal{F}_8)
\end{aligned}$$

$$\begin{aligned}
&\doteq \mu\alpha.\langle [\![\mathcal{A}(e_2)]\!] \,|\, \tilde{\mu}x.\langle \mu\beta.\langle [\![\mathcal{A}(e_1)]\!] \,|\, x \cdot \beta\rangle \,|\, \alpha\rangle\rangle && \text{(IH)}\\
&\doteq [\![\textbf{let } x = \mathcal{A}(e_2) \textbf{ in } \mathcal{A}(e_1)\ x]\!] && (\mathcal{T}_4,\ \mathcal{T}_6)\\
&\doteq [\![\mathcal{A}(e_1\ e_2)]\!]. && (\mathcal{A}_{10})
\end{aligned}$$

5. In case $e \doteq (e_1, e_2)$, we have to distinguish four subcases:

(i) Subcase $e \doteq (v_1, v_2)$:

$$\begin{aligned}
\mathcal{F}([\![(v_1, v_2)]\!]) &\doteq \mathcal{F}([\![v_1]\!], [\![v_2]\!]) && (\mathcal{T}_3)\\
&\doteq (\mathcal{F}([\![v_1]\!]), \mathcal{F}([\![v_2]\!])) && (\mathcal{F}_4)\\
&\doteq ([\![\mathcal{A}(v_1)]\!], [\![\mathcal{A}(v_2)]\!]) && \text{(IH)}\\
&\doteq [\![(\mathcal{A}(v_1), \mathcal{A}(v_2))]\!] && (\mathcal{T}_3)\\
&\doteq [\![\mathcal{A}((v_1, v_2))]\!]. && (\mathcal{A}_5)
\end{aligned}$$

(ii) Subcase $e \doteq (v_1, e_2)$:

$$\begin{aligned}
\mathcal{F}([\![(v_1, e_2)]\!]) &\doteq \mathcal{F}(([\![v_1]\!], [\![e_2]\!])) && (\mathcal{T}_3)\\
&\doteq \mu\alpha.\langle \mathcal{F}([\![e_2]\!]) \mid \tilde{\mu}x.\langle (\mathcal{F}([\![v_1]\!]), x) \mid \alpha\rangle\rangle && (\mathcal{F}_5)\\
&\doteq \mu\alpha.\langle [\![\mathcal{A}(e_2)]\!] \mid \tilde{\mu}x.\langle ([\![\mathcal{A}(v_1)]\!], x) \mid \alpha\rangle\rangle && \text{(IH)}\\
&\doteq [\![\textbf{let } x = \mathcal{A}(e_2) \textbf{ in } (\mathcal{A}(v_1), x)]\!] && (\mathcal{T}_3,\ \mathcal{T}_6)\\
&\doteq [\![\mathcal{A}((v_1, e_2))]\!]. && (\mathcal{A}_6)
\end{aligned}$$

(iii) Subcase $e \doteq (e_1, v_2)$:

$$\begin{aligned}
\mathcal{F}([\![(e_1, v_2)]\!]) &\doteq \mathcal{F}(([\![e_1]\!], [\![v_2]\!])) && (\mathcal{T}_3)\\
&\doteq \mu\alpha.\langle \mathcal{F}([\![e_1]\!]) \mid \tilde{\mu}x.\langle (x, \mathcal{F}([\![v_2]\!])) \mid \alpha\rangle\rangle && (\mathcal{F}_6)\\
&\doteq \mu\alpha.\langle [\![\mathcal{A}(e_1)]\!] \mid \tilde{\mu}x.\langle (x, [\![\mathcal{A}(v_2)]\!]) \mid \alpha\rangle\rangle && \text{(IH)}\\
&\doteq [\![\textbf{let } x = \mathcal{A}(e_1) \textbf{ in } (x, \mathcal{A}(v_2))]\!] && (\mathcal{T}_3,\ \mathcal{T}_6)\\
&\doteq [\![\mathcal{A}((e_1, v_2))]\!]. && (\mathcal{A}_7)
\end{aligned}$$

(iv) Subcase $e \doteq (e_1, e_2)$:

$$\begin{aligned}
&\mathcal{F}([\![(e_1, e_2)]\!])\\
&\ \doteq \mathcal{F}(([\![e_1]\!], [\![e_2]\!])) && (\mathcal{T}_3)\\
&\ \doteq \mu\alpha.\langle \mathcal{F}([\![e_1]\!]) \mid \tilde{\mu}x.\langle \mu\beta.\langle \mathcal{F}([\![e_2]\!]) \mid \tilde{\mu}y.\langle (x, y) \mid \beta\rangle\rangle \mid \alpha\rangle\rangle && (\mathcal{F}_7)
\end{aligned}$$

$$\begin{aligned}
&\doteq \mu\alpha.\langle [\![\mathcal{A}(e_1)]\!] \mid \tilde{\mu}x.\langle \mu\beta.\langle [\![\mathcal{A}(e_2)]\!] \mid \tilde{\mu}y.\langle (x,y) \mid \beta\rangle\rangle \mid \alpha\rangle\rangle && \text{(IH)}\\
&\doteq [\![\mathbf{let}\, x = \mathcal{A}(e_1)\,\mathbf{in}\,(\mathbf{let}\, y = \mathcal{A}(e_2)\,\mathbf{in}\,(x,y))]\!] && (\mathcal{T}_3,\, \mathcal{T}_6)\\
&\doteq [\![\mathcal{A}((e_1,e_2))]\!]. && (\mathcal{A}_8)
\end{aligned}$$

6. In case $e \doteq \mathbf{let}\, x = e_1\,\mathbf{in}\, e_2$ we have:

$$\begin{aligned}
\mathcal{F}([\![\mathbf{let}\, x = e_1\,\mathbf{in}\, e_2]\!]) &\doteq \mathcal{F}(\mu\alpha.\langle [\![e_1]\!] \mid \tilde{\mu}x.\langle [\![e_2]\!] \mid \alpha\rangle\rangle) && (\mathcal{T}_6)\\
&\doteq \mu\alpha.\langle \mathcal{F}([\![e_1]\!]) \mid \tilde{\mu}x.\langle \mathcal{F}([\![e_2]\!]) \mid \alpha\rangle\rangle\\
& && (\mathcal{F}_2,\, \mathcal{F}_{13},\, \mathcal{F}_9,\, \mathcal{F}_8)\\
&\doteq \mu\alpha.\langle [\![\mathcal{A}(e_1)]\!] \mid \tilde{\mu}x.\langle [\![\mathcal{A}(e_2)]\!] \mid \alpha\rangle\rangle && \text{(IH)}\\
&\doteq [\![\mathbf{let}\, x = \mathcal{A}(e_1)\,\mathbf{in}\,\mathcal{A}(e_2)]\!] && (\mathcal{T}_6)\\
&\doteq [\![\mathcal{A}(\mathbf{let}\, x = e_1\,\mathbf{in}\, e_2)]\!]. && (\mathcal{A}_3)
\end{aligned}$$

$\square$

Theorem 8.2 (Focusing reflects the ANF-transformation; case $\equiv$)**.** *For all λ-terms e, we have $\mathcal{F}([\![e]\!]) \equiv [\![\mathcal{A}(e)]\!]$.*

Proof. We only have to modify subcase 4(ii) in the proof of Theorem 8.1. The modified proof is as follows (evaluated redexes are underlined).

4. In case $e \doteq e_1\, e_2$, we have to distinguish two subcases:

   (i) Subcase $e \doteq e_1\, v_2$: identical to the proof of Theorem 8.1.

   (ii) Subcase $e \doteq e_1\, e_2$:

$$\begin{aligned}
\mathcal{F}([\![(e_1\, e_2)]\!]) &\doteq \mathcal{F}(\mu\alpha.\langle [\![e_1]\!] \mid [\![e_2]\!] \cdot \alpha\rangle) && (\mathcal{T}_4)\\
&\doteq \mu\alpha.\langle \underline{\mathcal{F}([\![e_1]\!]) \mid \tilde{\mu}y.\langle \mathcal{F}([\![e_2]\!]) \mid \tilde{\mu}x.\langle y \mid x\cdot\alpha\rangle\rangle}\rangle\\
& && (\mathcal{F}_2,\mathcal{F}_{13},\, \mathcal{F}_9,\, \mathcal{F}_1,\, \mathcal{F}_8,\, \mathcal{F}_{11})\\
&\rhd \mu\alpha.\langle \mathcal{F}([\![e_2]\!]) \mid \tilde{\mu}x.\langle \mathcal{F}([\![e_1]\!]) \mid x\cdot\alpha\rangle\rangle\\
&\overset{\text{IH}}{\equiv} \mu\alpha.\langle [\![\mathcal{A}(e_2)]\!] \mid \tilde{\mu}x.\langle [\![\mathcal{A}(e_1)]\!] \mid x\cdot\alpha\rangle\rangle\\
&\lhd \mu\alpha.\langle [\![\mathcal{A}(e_2)]\!] \mid \tilde{\mu}x.\langle \underline{\mu\beta.\langle [\![\mathcal{A}(e_1)]\!] \mid x\cdot\beta\rangle \mid \alpha}\rangle\rangle\\
&\doteq [\![\mathbf{let}\, x = \mathcal{A}(e_2)\,\mathbf{in}\,\mathcal{A}(e_1)\, x]\!] && (\mathcal{T}_4 + \mathcal{T}_6)\\
&\doteq [\![\mathcal{A}(e_1\, e_2)]\!]. && (\mathcal{A}_{10})
\end{aligned}$$

$\square$

In order to prove that for all $e \in \Lambda^{\mathrm{Q}}$, $[\![\mathcal{L}(e)]\!]^* =_\mu [\![e]\!]$ (Theorem 8.6), we introduce a transformation $\mathcal{M}$ that simulates the effect of applying $\mathcal{L}$ on a term from Λ^{Q} on its translation in $\Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$.

Definition A.1. The *μ-normalization operation* $\mathcal{M} : \Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}} \to \Lambda^{\mathrm{ANF}}_{\mu\tilde{\mu}}$ is defined by the following clauses:

Values

$$\mathcal{M}(x) :\doteq x \qquad (\mathcal{M}_1)$$

$$\mathcal{M}(\lambda x.e) :\doteq \lambda x.\mathcal{M}(e) \qquad (\mathcal{M}_2)$$

$$\mathcal{M}((w_1, w_2)) :\doteq (\mathcal{M}(w_1), \mathcal{M}(w_2)) \qquad (\mathcal{M}_3)$$

Terms

$$\mathcal{M}(e) :\doteq \mu\alpha.\mathcal{M}_\alpha(e) \qquad (\mathcal{M}_4)$$

$$\mathcal{M}_s(w) :\doteq \langle \mathcal{M}(w) \mid s \rangle \qquad (\mathcal{M}_5)$$

$$\mathcal{M}_s(\mu\alpha.c) :\doteq \mathcal{M}(c[s/\alpha]) \qquad (\mathcal{M}_6)$$

Coterms

$$\mathcal{M}(\alpha) :\doteq \alpha \qquad (\mathcal{M}_7)$$

$$\mathcal{M}(\pi_i\, s) :\doteq \pi_i\, \tilde{\mu}x.\langle x \mid s \rangle \qquad (\mathcal{M}_8)$$

$$\mathcal{M}(w \cdot s) :\doteq w \cdot \tilde{\mu}x.\langle x \mid s \rangle \qquad (\mathcal{M}_9)$$

$$\mathcal{M}(\tilde{\mu}x.\langle e \mid s \rangle) :\doteq \tilde{\mu}x.\mathcal{M}_s(e) \qquad (\mathcal{M}_{10})$$

Computations

$$\mathcal{M}(\langle e \mid s \rangle) :\doteq \mathcal{M}_{\mathcal{M}(s)}(e). \qquad (\mathcal{M}_{11})$$

Definition A.2. We define the operation $\sqcup ; \sqcup$ which takes a continuation k (cf. Definition 6.6) and a coterm s, and returns a coterm $k; s$:

$$\mathrm{id}; s :\doteq s \qquad (\mathcal{C}_1)$$

$$\overline{\lambda v}.\mathbf{let}\ x = \pi_i\, \overline{v}\ \mathbf{in}\ e; s :\doteq \pi_i\, \tilde{\mu}x.[\![e]\!]^*_s \qquad (\mathcal{C}_2)$$

$$\overline{\lambda v}.\mathbf{let}\ x = \overline{v}\, v'\ \mathbf{in}\ e; s :\doteq v' \cdot \tilde{\mu}x.[\![e]\!]^*_s \qquad (\mathcal{C}_3)$$

$$\overline{\lambda v}.\mathbf{let}\ x = \overline{v}\ \mathbf{in}\ e; s :\doteq \tilde{\mu}x.[\![v]\!]^*_s. \qquad (\mathcal{C}_4)$$

Lemma A.3. *We have* $[\![k \mathbin{@} v]\!]^*_s \doteq \langle [\![v]\!]^* \mid k; s\rangle$.

Proof. By case analysis on k:

1. Case $k \doteq \mathrm{id}$:

$$\begin{aligned}
[\![\mathrm{id} \mathbin{@} v]\!]^*_s &\doteq [\![v]\!]^*_s && (@_1)\\
&\doteq \langle [\![v]\!]^* \mid s\rangle && (\mathcal{T}^*_5)\\
&\doteq \langle [\![v]\!]^* \mid \mathrm{id}; s\rangle. && (\mathcal{C}_1)
\end{aligned}$$

2. Case $k \doteq \overline{\lambda v}.\mathbf{let}\ x = \pi_i\, \overline{v}\ \mathbf{in}\ e$:

$$\begin{aligned}
[\![\overline{\lambda v}.\mathbf{let}\ x = \pi_i\, \overline{v}\ \mathbf{in}\ e \mathbin{@} v]\!]^*_s &\doteq [\![\mathbf{let}\ x = \pi_i\, v\ \mathbf{in}\ e]\!]^*_s && (@_2)\\
&\doteq [\![\pi_i\, v]\!]^*_{\tilde\mu x.[\![e]\!]^*_s} && (\mathcal{T}^*_2)\\
&\doteq \langle [\![v]\!]^* \mid \pi_i\, \tilde\mu x.[\![e]\!]^*_s\rangle && (\mathcal{T}^*_4)\\
&\doteq \langle [\![v]\!]^* \mid \overline{\lambda v}.\mathbf{let}\ x = \pi_i\, \overline{v}\ \mathbf{in}\ e; s\rangle. && (\mathcal{C}_2)
\end{aligned}$$

3. Case $k \doteq \overline{\lambda v}.\mathbf{let}\ x = \overline{v}\, v'\ \mathbf{in}\ e$:

$$\begin{aligned}
[\![\overline{\lambda v}.\mathbf{let}\ x = \overline{v}\, v'\ \mathbf{in}\ e \mathbin{@} v]\!]^*_s &\doteq [\![\mathbf{let}\ x = v\, v'\ \mathbf{in}\ e]\!]^*_s && (@_3)\\
&\doteq [\![v\, v']\!]^*_{\tilde\mu x.[\![e]\!]^*_s} && (\mathcal{T}^*_2)\\
&\doteq \langle [\![v]\!]^* \mid [\![v']\!]^* \cdot \tilde\mu x.[\![e]\!]^*_s\rangle && (\mathcal{T}^*_3)\\
&\doteq \langle [\![v]\!]^* \mid \overline{\lambda v}.\mathbf{let}\ x = \overline{v}\, v'\ \mathbf{in}\ e; s\rangle. && (\mathcal{C}_3)
\end{aligned}$$

4. Case $k \doteq \overline{\lambda v}.\mathbf{let}\ x = \overline{v}\ \mathbf{in}\ e$:

$$\begin{aligned}
[\![\overline{\lambda v}.\mathbf{let}\ x = \overline{v}\ \mathbf{in}\ e \mathbin{@} v]\!]^*_s &\doteq [\![\mathbf{let}\ x = v\ \mathbf{in}\ e]\!]^*_s && (@_4)\\
&\doteq [\![v]\!]^*_{\tilde\mu x.[\![e]\!]^*_s} && (\mathcal{T}^*_2)\\
&\doteq \langle [\![v]\!]^* \mid \tilde\mu x.[\![e]\!]^*_s\rangle && (\mathcal{T}^*_5)\\
&\doteq \langle [\![v]\!]^* \mid \overline{\lambda v}.\mathbf{let}\ x = \overline{v}\ \mathbf{in}\ e; s\rangle. && (\mathcal{C}_4)
\end{aligned}$$

□

The μ-normalization operation $\mathcal{M}$ corresponds precisely to the transformation $\mathcal{L}$:

Lemma A.4. *The following statements hold:*

1. *For all values $v \in \Lambda^{\mathrm{Q}}$: $[\![\mathcal{L}(v)]\!]^* \doteq \mathcal{M}([\![v]\!])$.*
2. *For all expressions $e \in \Lambda^{\mathrm{Q}}$: $[\![\mathcal{L}(e)]\!]^* \doteq \mathcal{M}([\![e]\!])$.*
3. *For all $e \in \Lambda^{\mathrm{Q}}$, continuations k and coterms s:*

$$[\![\mathcal{L}_k(e)]\!]^*_s \doteq \mathcal{M}_{k;s}([\![e]\!]).$$

Proof. We prove these three statements by simultaneous induction. For the first statement, Lemma A.4(1), we use induction on v:

1. Case $v \doteq x$:

$$\begin{aligned} [\![\mathcal{L}(x)]\!]^* &\doteq [\![x]\!]^* && (\mathcal{L}_1)\\ &\doteq x && (\mathcal{T}_6^*)\\ &\doteq \mathcal{M}(x) && (\mathcal{M}_1)\\ &\doteq \mathcal{M}([\![x]\!]). && (\mathcal{T}_1)\end{aligned}$$

2. Case $v \doteq (v_1, v_2)$:

$$\begin{aligned} [\![\mathcal{L}((v_1,v_2))]\!]^* &\doteq [\![(\mathcal{L}(v_1),\mathcal{L}(v_2))]\!]^* && (\mathcal{L}_3)\\ &\doteq ([\![\mathcal{L}(v_1)]\!]^*, [\![\mathcal{L}(v_2)]\!]^*) && (\mathcal{T}_7^*)\\ &\doteq (\mathcal{M}([\![v_1]\!]), \mathcal{M}([\![v_2]\!])) && (\text{IH for Lemma A.4(1)})\\ &\doteq \mathcal{M}(([\![v_1]\!],[\![v_2]\!])) && (\mathcal{M}_3)\\ &\doteq \mathcal{M}([\![(v_1,v_2)]\!]). && (\mathcal{T}_3)\end{aligned}$$

3. Case $v \doteq \lambda x.e$:

$$\begin{aligned} [\![\mathcal{L}(\lambda x.e)]\!]^* &\doteq [\![\lambda x.\mathcal{L}(e)]\!]^* && (\mathcal{L}_2)\\ &\doteq \lambda x.[\![\mathcal{L}(e)]\!]^* && (\mathcal{T}_8^*)\\ &\doteq \lambda x.\mathcal{M}([\![e]\!]) && (\text{IH for Lemma A.4(2)})\\ &\doteq \mathcal{M}(\lambda x.[\![e]\!]) && (\mathcal{M}_2)\\ &\doteq \mathcal{M}([\![\lambda x.e]\!]). && (\mathcal{T}_2)\end{aligned}$$

For Lemma A.4(2), we show the following:

$$\begin{aligned}
[\![\mathcal{L}(e)]\!]^{*} &\mathrel{\hat{=}} \mu\alpha.[\![\mathcal{L}(e)]\!]^{*}_{\alpha} && (\mathcal{T}^{*}_{1})\\
&\mathrel{\hat{=}} \mu\alpha.[\![\mathcal{L}_{\mathrm{id}}(e)]\!]^{*}_{\alpha} && (\mathcal{L}_{4})\\
&\mathrel{\hat{=}} \mu\alpha.\mathcal{M}_{\mathrm{id};\alpha}([\![e]\!]) && (\text{IH for Lemma A.4(3)})\\
&\mathrel{\hat{=}} \mu\alpha.\mathcal{M}_{\alpha}([\![e]\!]) && (\mathcal{C}_{1})\\
&\mathrel{\hat{=}} \mathcal{M}([\![e]\!]). && (\mathcal{M}_{4})
\end{aligned}$$

For Lemma A.4(3), we perform induction on e:

1. Case $e \mathrel{\hat{=}} v$:

$$\begin{aligned}
[\![\mathcal{L}_{k}(v)]\!]^{*}_{s} &\mathrel{\hat{=}} [\![k \mathbin{@} \mathcal{L}(v)]\!]^{*}_{s} && (\mathcal{L}_{7})\\
&\mathrel{\hat{=}} \langle [\![\mathcal{L}(v)]\!]^{*} \mid k; s\rangle && (\text{Lemma A.3})\\
&\mathrel{\hat{=}} \langle \mathcal{M}([\![v]\!]) \mid k; s\rangle && (\text{IH for 1})\\
&\mathrel{\hat{=}} \mathcal{M}_{k;s}([\![v]\!]). && (\mathcal{M}_{5})
\end{aligned}$$

2. Case $e \mathrel{\hat{=}} \mathbf{let}\ x = e_{1}\ \mathbf{in}\ e_{2}$:

$$\begin{aligned}
[\![\mathcal{L}_{k}(\mathbf{let}\ x = e_{1}\ \mathbf{in}\ e_{2})]\!]^{*}_{s} &\mathrel{\hat{=}} [\![\mathcal{L}_{\overline{\lambda v}.\mathbf{let}\ x=\overline{v}\ \mathbf{in}\ \mathcal{L}_{k}(e_{2})}(e_{1})]\!]^{*}_{s} && (\mathcal{L}_{8})\\
&\mathrel{\hat{=}} \mathcal{M}_{\overline{\lambda v}.\mathbf{let}\ x=\overline{v}\ \mathbf{in}\ \mathcal{L}_{k}(e_{2});s}([\![e_{1}]\!]) && (\text{IH})\\
&\mathrel{\hat{=}} \mathcal{M}_{\tilde{\mu}x.[\![\mathcal{L}_{k}(e_{2})]\!]^{*}_{s}}([\![e_{1}]\!]) && (\mathcal{C}_{4})\\
&\mathrel{\hat{=}} \mathcal{M}_{\tilde{\mu}x.\mathcal{M}_{k;s}([\![e_{2}]\!])}([\![e_{1}]\!]) && (\text{IH})\\
&\mathrel{\hat{=}} \mathcal{M}_{\mathcal{M}(\tilde{\mu}x.\langle [\![e_{2}]\!] \mid k;s\rangle)}([\![e_{1}]\!]) && (\mathcal{M}_{10})\\
&\mathrel{\hat{=}} \mathcal{M}(\langle [\![e_{1}]\!] \mid \tilde{\mu}x.\langle [\![e_{2}]\!] \mid k; s\rangle\rangle) && (\mathcal{M}_{11})\\
&\mathrel{\hat{=}} \mathcal{M}_{k;s}(\mu\alpha.\langle [\![e_{1}]\!] \mid \tilde{\mu}x.\langle [\![e_{2}]\!] \mid \alpha\rangle\rangle) && (\mathcal{M}_{6})\\
&\mathrel{\hat{=}} \mathcal{M}_{k;s}([\![\mathbf{let}\ x = e_{1}\ \mathbf{in}\ e_{2}]\!]). && (\mathcal{T}_{6})
\end{aligned}$$

3. Case $e \mathrel{\hat{=}} e\, v$:

$$\begin{aligned}
[\![\mathcal{L}_{k}(e\, v)]\!]^{*}_{s} &\mathrel{\hat{=}} [\![\mathcal{L}_{\overline{\lambda v'}.\mathbf{let}\ x=v'\ v\ \mathbf{in}\ k \mathbin{@} x}(e)]\!]^{*}_{s} && (\mathcal{L}_{5})\\
&\mathrel{\hat{=}} \mathcal{M}_{\overline{\lambda v'}.\mathbf{let}\ x=v'\ v\ \mathbf{in}\ k \mathbin{@} x;s}([\![e]\!]) && (\text{IH})\\
&\mathrel{\hat{=}} \mathcal{M}_{[\![v]\!]\cdot\tilde{\mu}x.[\![k \mathbin{@} x]\!]^{*}_{s}}([\![e]\!]) && (\mathcal{C}_{3})
\end{aligned}$$

$$\begin{aligned}
&\doteq \mathcal{M}_{[\![v]\!]\cdot\tilde{\mu}x.\langle x|s\rangle}([\![e]\!]) && \text{(Lemma A.3)}\\
&\doteq \mathcal{M}_{\mathcal{M}([\![v]\!]\cdot k;s)}([\![e]\!]) && (\mathcal{M}_9)\\
&\doteq \mathcal{M}(\langle [\![e]\!] \mid [\![v]\!] \cdot k; s\rangle) && (\mathcal{M}_{11})\\
&\doteq \mathcal{M}_{k;s}(\mu\alpha.\langle [\![e]\!] \mid [\![v]\!] \cdot \alpha\rangle) && (\mathcal{M}_6)\\
&\doteq \mathcal{M}_{k;s}([\![e\ v]\!]). && (\mathcal{T}_4)
\end{aligned}$$

4. Case $e \doteq \pi_i\ e$:

$$\begin{aligned}
[\![\mathcal{L}_k(\pi_i\ e)]\!]^*_s &\doteq [\![\mathcal{L}_{\overline{\lambda v}.\mathbf{let}\ x=\pi_i\ \overline{v}\ \mathbf{in}\ k\ @\ x}(e)]\!]^*_s && (\mathcal{L}_6)\\
&\doteq \mathcal{M}_{\overline{\lambda v}.\mathbf{let}\ x=\pi_i\ \overline{v}\ \mathbf{in}\ k\ @\ x;s}([\![e]\!]) && \text{(IH)}\\
&\doteq \mathcal{M}_{\pi_i\ \tilde{\mu}x.[\![k\ @\ x]\!]^*_s}([\![e]\!]) && (\mathcal{C}_2)\\
&\doteq \mathcal{M}_{\pi_i\ \tilde{\mu}x.\langle x|k;s\rangle}([\![e]\!]) && \text{(Lemma A.3)}\\
&\doteq \mathcal{M}_{\mathcal{M}(\pi_i\ (k;s))}([\![e]\!]) && (\mathcal{M}_8)\\
&\doteq \mathcal{M}(\langle [\![e]\!] \mid \pi_i\ (k;s)\rangle) && (\mathcal{M}_{11})\\
&\doteq \mathcal{M}_{k;s}(\mu\alpha.\langle [\![e]\!] \mid \pi_i\ \alpha\rangle) && (\mathcal{M}_6)\\
&\doteq \mathcal{M}_{k;s}([\![\pi_i\ e]\!]). && (\mathcal{T}_5)
\end{aligned}$$

□

The effect of the application of $\mathcal{M}$ can be achieved by applying μ-reductions in commands and η-expansions in coterms:

Lemma A.5. *For all terms, coterms and commands $t \in \Lambda^{\mathrm{Q}}_{\mu\tilde{\mu}}$ we have $t \equiv_\mu \mathcal{M}(t)$.*

Proof. By inspection of the relevant clauses in Definition A.1. The combination of the clauses ($\mathcal{M}_6$) and ($\mathcal{M}_{11}$) corresponds to the reduction of μ-redexes. Coterms are η-expanded in the clauses ($\mathcal{M}_8$) and ($\mathcal{M}_9$). □

Theorem 8.6. *For all terms $e \in \Lambda^{\mathrm{Q}}$, $[\![\mathcal{L}(e)]\!]^* \equiv_\mu [\![e]\!]$.*

Proof. By combining Lemmas A.4 and A.5. □