Monadic translation of classical sequent calculus

JOSÉ ESPÍRITO SANTO[1] *and RALPH MATTHES*[2] *and*

KOJI NAKAZAWA[3] *and LUÍS PINTO*[1]

[1]*Centro de Matemática, Universidade do Minho, Portugal*[†]
[2]*I.R.I.T. (C.N.R.S. and University of Toulouse), France*[‡]
[3]*Graduate School of Informatics, Kyoto University, Japan*[§]

Received 21 December 2010; Revised 12 January 2012, 8 May 2012

We study monadic translations of the call-by-name (cbn) and the call-by-value (cbv) fragments of the classical sequent calculus $\overline{\lambda}\mu\tilde{\mu}$ by Curien and Herbelin and give modular and syntactic proofs of strong normalization. The target of the translations is a new meta-language for classical logic, named monadic $\lambda\mu$. It is a monadic reworking of Parigot's $\lambda\mu$-calculus, where the monadic binding is confined to commands, thus integrating the monad with the classical features. Also its μ-reduction rule is replaced by one expressing the interaction between monadic binding and μ-abstraction.

Our monadic translations produce very tight simulations of the respective fragments of $\overline{\lambda}\mu\tilde{\mu}$ inside monadic $\lambda\mu$, with reduction steps of $\overline{\lambda}\mu\tilde{\mu}$ being translated in 1-1 fashion, except for β-steps which require two steps. The monad of monadic $\lambda\mu$ can be instantiated to the continuations monad so as to ensure strict simulation of monadic $\lambda\mu$ inside simply-typed λ-calculus with β- and η-reduction. Through strict simulation, strong normalization of simply-typed λ-calculus is inherited to monadic $\lambda\mu$ and then to cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$, thus reproving in an elementary syntactical way strong normalization for these fragments of $\overline{\lambda}\mu\tilde{\mu}$ and establishing it for our new calculus. These results extend to second-order logic, with polymorphic λ-calculus as target, giving new strong normalization results for classical second-order logic in sequent calculus style.

CPS translations of cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$ with the strict simulation property are obtained by composing our monadic translations with the continuations-monad instantiation. In an appendix to the article we investigate several refinements of the continuations-monad instantiation in order to obtain in a modular way improvements of the CPS translations enjoying extra properties like simulation by cbv β-reduction or reduction of administrative redexes at compile time.

[†] The first and fourth author have been financed by FEDER funds through "Programa Operacional Factores de Competitividade – COMPETE" and by Portuguese funds through FCT – "Fundação para a Ciência e a Tecnologia", within the project PEst-C/MAT/UI0013/2011.

[‡] The second author thanks the Centro de Matemática of Universidade do Minho for funding research visits to the first and fourth author.

[§] The third author has been supported by the Kyoto University Foundation for an extended research visit to the second author.

1. Introduction

The $\overline{\lambda}\mu\tilde{\mu}$ calculus (Curien and Herbelin, 2000) is a way to present classical sequent calculus in an operationalized form as an extension of λ-calculus. Such calculus is a prototypical functional language with a control operator μ (introduced in (Parigot, 1992)), but where no deterministic reduction strategy is singled out. It is important thus to consider confluent fragments (where all reduction sequences lead to the same result, if any). Non-confluence of $\overline{\lambda}\mu\tilde{\mu}$ is due to a single critical pair, that can be resolved in two ways, determining the call-by-name (cbn) and call-by-value (cbv) fragments of $\overline{\lambda}\mu\tilde{\mu}$ (Curien and Herbelin, 2000). In addition, it is desirable that, inside each fragment, all reduction sequences starting from typed expressions indeed produce a result (i. e., end in a term that can no longer be reduced). This is the strong normalization property.

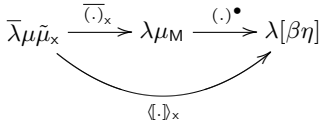
In this article we study embeddings of the cbn and cbv fragments of $\overline{\lambda}\mu\tilde{\mu}$ into the simply-typed λ-calculus. These embeddings are continuation-passing style (CPS) translations and, therefore, a kind of compilation. In addition, through these embeddings, we give a new proof of strong normalization for the mentioned fragments of $\overline{\lambda}\mu\tilde{\mu}$. In fact, the embeddings produce *strict simulations*, that is, each reduction step of the source calculus is mapped to one or more steps of the target calculus, so that strong normalization in the source is reduced to strong normalization in the target, where it holds and has been proven in many different ways.

The interest of this new proof lies, not only in its elementary character, but also in its concepts. The CPS compilations that simulate the fragments of $\overline{\lambda}\mu\tilde{\mu}$ are factored into a respective *monadic* translation and a single *instantiation* mapping, the latter working for both cbn and cbv. The monadic translation is, as advocated in (Moggi, 1991), a semantical interpretation into a *monadic meta-language*, and this, in turn, is a typed calculus with a special type former M, that stands for a *monad*, an ingredient in the categorical semantics originally put forward by Moggi. The monadic translation is thus parameterised by M. Here we consider only the instantiation of M to the so-called *continuations monad*. This corresponds to interpreting M as *double negation*, a type transformation of simple types, and determines a mapping from the meta-language to the simply-typed λ-calculus.

The target of the monadic translation is a *classical* version of Moggi's meta-language, whose definition is a challenge and a major contribution of the present article. This target is a reworking of Parigot's $\lambda\mu$-calculus which we call *monadic $\lambda\mu$-calculus*, and denote $\lambda\mu_{\mathsf{M}}$. It is not a routine amalgamation of $\lambda\mu$ with the monadic meta-language. Monadic $\lambda\mu$ extends the category of commands of $\lambda\mu$-calculus by a monadic bind construct. Co-variables are restricted to "monadic" types, *i.e.*, types of the form MA (otherwise some trivialisation happens, see Section 3.1). Unlike Parigot's calculus, there is no μ-reduction rule corresponding to implication elimination. Instead, the μ-rule now expresses the interaction between bind and μ-abstraction. Nonetheless, the intuitionistic restriction of $\lambda\mu_{\mathsf{M}}$ corresponds to Moggi's monadic meta-language.

Contrary to the original monadic meta-language (Moggi, 1991), but in line with (Hatcliff and Danvy, 1994; Sabry and Wadler, 1997), our classical meta-language is equipped with reduction rules. The cbn and cbv monadic translations produce strict simulations of the respective sources by these reduction rules. On the other hand, the instantiation from $\lambda\mu_{\mathsf{M}}$ into the simply-typed λ-calculus given by the continuations monad is also a strict simulation. In the target, besides the η-reduction rule, we just need Plotkin's cbv β-rule (β_{v}) for the cbv restriction, but the full β-rule for the full calculus. Therefore, both cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$ are strongly normalizing, either by

Fig. 1. Overview



x	n (after name) or v (after value)
$\overline{\lambda}\mu\tilde{\mu}_{\mathsf{x}}$	cbx fragment of $\overline{\lambda}\mu\tilde{\mu}$-calculus
$\lambda\mu_{\mathsf{M}}$	monadic $\lambda\mu$-calculus
$\overline{(.)}_{\mathsf{x}}$	cbx monadic translation
$(.)^{\bullet}$	continuations-monad instantiation
$(\!|.|\!)_{\mathsf{x}}$	cbx CPS translation

observing that $\lambda\mu_{\mathsf{M}}$ was before proved strongly normalizing through strict simulation in simply-typed λ-calculus, or by composing monadic translations with the instantiation mapping, to form direct, CPS compilations into λ-calculus with the strict simulation property. See Fig. 1 for an overview of systems and translations.

All the systems considered in this article can straightforwardly be extended to cover second-order logic, and the simulation results can be extended correspondingly. This demonstrates that our technique uses minimal meta-theoretic strength while it can establish strong normalization in cases where no arithmetic proofs are possible. This is because we are content with a simulation result, inheriting strong normalization from second-order λ-calculus that is considered widely known and established with a multitude of distinct proof strategies.

In an appendix to the paper, we study technical refinements concerning the obtained CPS translations of cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$. The questions of "administrative reductions" and indifference property (Plotkin, 1975) are analyzed. Two variants of the obtained CPS translations are proposed, one that performs administrative reductions at compile time, the other enjoying strict simulation by β_{v} only. The main point is that the modular approach of having decomposition via $\lambda\mu_{\mathsf{M}}$ is kept, as the refinements are confined to the continuations-monad instantiation, and the refined CPS translations are obtained by composition.

Structure of the article. Section 2 recalls $\overline{\lambda}\mu\tilde{\mu}$. Section 3 defines $\lambda\mu_{\mathsf{M}}$ and shows the connection with Moggi's meta-language. Section 4 defines the monadic translations of cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$ into $\lambda\mu_{\mathsf{M}}$ and proves strict simulation. Section 5 defines the continuations-monad instantiation and concludes the proof of strong normalization for cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$. Section 6 extends the results to systems with second-order universal quantification, and Section 7 discusses related and future work. Technical refinements concerning CPS translations are presented in Appendix A.

Notation. Simple types, ranged over by A, B, C, are generated from type variables X by arrow/implication, written $A \supset B$. Monads are denoted M.

Contexts Γ are finite sets of declarations $(x : A)$ with x a variable, while co-contexts Δ are finite sets of declarations $(a : A)$, with a a co-variable. In both cases, there is the usual requirement of consistency, i. e., uniqueness of declaration of the same (co-)variable, which is implicitly enforced in the sense that, e. g., when writing the enlarged context $\Gamma, x : A$, this presupposes that x is not declared in Γ. We write Γ, Γ' for the union of the contexts Γ and Γ', again implicitly assuming that they do not have declarations for some variable in common. If F

is some type operation, then its extension to contexts is

$$F\Gamma = \{(x : FA) | (x : A) \in \Gamma\} \ ,$$

and similarly for co-contexts Δ. An immediate benefit of this notation is its "compositionality": if two operations on types, F and G, are considered, then $F(G\Gamma) = (F \circ G)\Gamma$, for $F \circ G$ the composition of F and G. Trivially, the same holds for co-contexts Δ.

By $\lambda[R_1 \ldots]$ we denote λ-calculus with reduction rules $R_1, \ldots$ Thus, for clarity or emphasis, we may denote ordinary λ-calculus with $\lambda[\beta]$, using the usual β-reduction rule

$$(\beta) \qquad (\lambda x.t)s \to [s/x]t \ .$$

Plotkin's cbv restriction

$$(\beta_{\mathsf{v}}) \qquad (\lambda x.t)V \to [V/x]t \qquad \text{for } V \text{ a value (i. e., not an application)}$$

yields the corresponding cbv λ-calculus $\lambda[\beta_{\mathsf{v}}]$. We sometimes need the even more restricted

$$(\beta_{\mathsf{var}}) \qquad (\lambda x.t)y \to [y/x]t \qquad \text{for } y \notin t \text{ (i. e., } y \text{ not free in } t) \ .$$

The η-reduction rule is

$$(\eta) \qquad \lambda x.tx \to t \qquad \text{for } x \notin t$$

Throughout the paper, when reduction rules are given (the *base* reduction rules), $\to$ stands for the term closure of the base reduction rules, i. e., reduction by $\to$ may happen by applying one of the base reduction rules at arbitrary depth in the expression, including under binders. When $\to$ (possibly with decoration) stands for a reduction relation, $\to^+$ denotes its transitive and $\to^*$ its reflexive and transitive closure.

2. Background

In this section we recall Curien and Herbelin's $\overline{\lambda}\mu\tilde{\mu}$-calculus (Curien and Herbelin, 2000).

Expressions are values, terms, evaluation contexts, co-terms and commands that are defined by the following grammar:

$$\begin{array}{rclcrclcrcl} V & ::= & x \,|\, \lambda x.t & \quad & E & ::= & a \,|\, u :: e & \quad & c & ::= & \langle t | e \rangle \\ t, u & ::= & V \,|\, \mu a.c & & e & ::= & E \,|\, \tilde{\mu} x.c & & & & \end{array}$$

Expressions are ranged over by T, T'. Variables (resp. co-variables) are ranged over by v, w, x, y, z (resp. a, b). We assume disjoint countably infinite supplies of them and can denote any of them by using decorations of the base symbols. (This will never be made explicit in the rest of the paper.)

There is one kind of sequent per proper syntactic class (terms, co-terms and commands)

$$\Gamma \vdash t : A | \Delta \qquad \Gamma | e : A \vdash \Delta \qquad c : (\Gamma \vdash \Delta)$$

where Γ and Δ are contexts and co-contexts, respectively, as described by the notational conventions in the previous section. Typing rules are given in Fig. 2.

There are 6 substitution operations altogether:

$$[t/x]c \quad [t/x]u \quad [t/x]e \quad [e/a]c \quad [e/a]u \quad [e/a]e'$$

Fig. 2. Typing rules of $\overline{\lambda}\mu\tilde{\mu}$

$$\frac{}{\Gamma, x : A \vdash x : A | \Delta} \qquad \frac{}{\Gamma | a : A \vdash a : A, \Delta} \qquad \frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma | \tilde{\mu}x.c : A \vdash \Delta} \qquad \frac{c : (\Gamma \vdash a : A, \Delta)}{\Gamma \vdash \mu a.c : A | \Delta}$$

$$\frac{\Gamma, x : A \vdash t : B | \Delta}{\Gamma \vdash \lambda x.t : A \supset B | \Delta} \qquad \frac{\Gamma \vdash u : A | \Delta \quad \Gamma | e : B \vdash \Delta}{\Gamma | u :: e : A \supset B \vdash \Delta} \qquad \frac{\Gamma \vdash t : A | \Delta \quad \Gamma | e : A \vdash \Delta}{\langle t | e \rangle : (\Gamma \vdash \Delta)}$$

Fig. 3. Reduction rules of $\overline{\lambda}\mu\tilde{\mu}$

$$\begin{array}{lrcl}
(\beta) & \langle \lambda x.t | u :: e \rangle & \to & \langle u | \tilde{\mu}x.\langle t | e \rangle \rangle \\
(\pi) & \langle \mu a.c | e \rangle & \to & [e/a]c \\
(\sigma) & \langle t | \tilde{\mu}x.c \rangle & \to & [t/x]c \\
(\eta_{\tilde{\mu}}) & \tilde{\mu}x.\langle x | e \rangle & \to & e, \text{ if } x \notin e \\
(\eta_{\mu}) & \mu a.\langle t | a \rangle & \to & t, \text{ if } a \notin t
\end{array}$$

We consider the 5 reduction rules in Fig. 3, where we reuse the name β of λ-calculus (rule names are considered relative to some term system). These are the reductions considered by Polonovski in (Polonovski, 2004), but the β-rule for the subtraction connective is not included. However, throughout the paper, we will only consider fragments where the critical pair rooted in $\langle \mu a.c | \tilde{\mu}x.c' \rangle$ between the rules σ and π is avoided. Following (Curien and Herbelin, 2000), in the cbn fragment $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ of $\overline{\lambda}\mu\tilde{\mu}$, the π rule is replaced by its restriction to evaluation contexts:

$$(\pi_{\mathsf{n}}) \quad \langle \mu a.c | E \rangle \quad \to \quad [E/a]c$$

This is equivalent to saying that σ has priority over π. (Note that this is not the cbn restriction $\overline{\lambda}\mu\tilde{\mu}_T$ of (Curien and Herbelin, 2000) with a proper sub-syntax.)

Dually, in the cbv fragment $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ of $\overline{\lambda}\mu\tilde{\mu}$, the σ rule is replaced by its restriction to values:

$$(\sigma_{\mathsf{v}}) \quad \langle V | \tilde{\mu}x.c \rangle \quad \to \quad [V/x]c$$

Now, π has priority over σ. In both fragments, the only critical pairs are trivial ones involving $\eta_{\tilde{\mu}}$ and η_{μ}, hence $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ are confluent.

3. Monadic $\lambda\mu$-calculus

In this section we define the *monadic $\lambda\mu$-calculus* $\lambda\mu_{\mathsf{M}}$, a monadic reworking of Parigot's $\lambda\mu$-calculus. Its intuitionistic fragment is discussed in Section 3.2.

3.1. *The calculus*

Expressions. Variables (resp. co-variables) are ranged over by v, w, x, y, z (resp. a, b), as for $\overline{\lambda}\mu\tilde{\mu}$. Expressions are given by the following grammar[†]:

$$\begin{array}{rrcl} \text{(values)} & V & ::= & x \mid \lambda x.t \\ \text{(terms)} & r,s,t,u & ::= & V \mid tu \mid \mu a.c \mid \eta t \\ \text{(commands)} & c & ::= & at \mid \mathsf{bind}(t, x.c) \end{array}$$

Notice that a bind, as well as one of its sub-expressions, is a command.

Substitutions $[s/x]t$ and $[s/x]c$ are defined in the obvious way. The following *derived* syntactic classes will be useful:

$$\begin{array}{rrcl} \text{(base contexts)} & L & ::= & a[\,] \mid \mathsf{bind}([\,], x.c) \\ \text{(cbn contexts)} & C & ::= & L \mid \mathsf{bind}(\eta[\,], x.c) \end{array}$$

where $[\,]$ represents the "hole" of the context[‡]. If t is a term, $C[t]$ denotes the command obtained by filling the hole of C with t, and is defined as at (resp. $\mathsf{bind}(t, x.c)$ resp. $\mathsf{bind}(\eta t, x.c)$) if C is $a[\,]$ (resp. $\mathsf{bind}([\,], x.c)$ resp. $\mathsf{bind}(\eta[\,], x.c)$). Notice that every command c has the form $L[t]$, and L and t are uniquely determined by c.

In the sequent calculus $\overline{\lambda}\mu\tilde{\mu}$ we have substitution of co-terms for co-variables. We define, in the natural deduction system $\lambda\mu_{\mathsf{M}}$, substitution of cbn contexts for co-variables in terms of "structural substitution". Structural substitution $[C/a]t$ and $[C/a]c$ is defined by replacing every binding-equivalent subexpression au of t or c, respectively, by $C[u]$, and this has to be done recursively. The most important case is

$$[C/a](at) = C\big[[C/a]t\big] \ .$$

All the other cases are homomorphic and therefore omitted. Notice that $[b[\,]/a]t = [b/a]t$ and $[b[\,]/a]c = [b/a]c$, provided substitution of co-variables for co-variables is defined in the obvious way.

It will be convenient to extend structural substitution of C for a to cbn contexts as well, i. e., to define the cbn context $[C/a]C'$ in the obvious way. In particular,

$$[C/a](a[\,]) = C \ .$$

We assume also to have the definition for the cbn context $[s/x]C$.

Typing rules. Types are given by

$$A, B \ ::= \ X \mid A \supset B \mid MA$$

Types of the form MA are called *monadic types*. Typing rules are given in Fig. 4. There are two kinds of sequents: $\Gamma \vdash t : A|\Delta$ and $c : (\Gamma \vdash \Delta)$. In both cases, Δ is a consistent set of

[†] In the notation of (Moggi, 1991), $\mathsf{bind}(t, x.c)$ and ηt are written $\mathsf{let}\ x = t\ \mathsf{in}\ c$ and $[t]$, respectively.

[‡] The terminology "cbn context" relates to the monadic translations to be introduced below. The form $\mathsf{bind}(\eta[\,], x.c)$ is used in the cbn translation only.

Fig. 4. Typing rules of $\lambda\mu_{\mathsf{M}}$

$$\frac{}{\Gamma, x : A \vdash x : A|\Delta}\ Ax$$

$$\frac{\Gamma, x : A \vdash t : B|\Delta}{\Gamma \vdash \lambda x.t : A \supset B|\Delta}\ Intro \qquad \frac{\Gamma \vdash t : A \supset B|\Delta \quad \Gamma \vdash u : A|\Delta}{\Gamma \vdash tu : B|\Delta}\ Elim$$

$$\frac{\Gamma \vdash t : MA|a : MA, \Delta}{at : (\Gamma \vdash a : MA, \Delta)}\ Pass \qquad \frac{c : (\Gamma \vdash a : MA, \Delta)}{\Gamma \vdash \mu a.c : MA|\Delta}\ Act$$

$$\frac{\Gamma \vdash s : A|\Delta}{\Gamma \vdash \eta s : MA|\Delta}\ Unit \qquad \frac{\Gamma \vdash r : MA|\Delta \quad c : (\Gamma, x : A \vdash \Delta)}{\mathsf{bind}(r, x.c) : (\Gamma \vdash \Delta)}\ Mult$$

declarations $a : MA$, hence with monadic types. Except for the last two rules, these are the rules for Parigot's $\lambda\mu$, however with the restriction of co-variables to monadic types.[§]

The rule for η is just as expected for the unit of a monad, while the typing rule for bind – which is named after monad multiplication – has to be contrasted with the usual rule in the framework of λ-calculus:

$$\frac{\Gamma \vdash r : MA \quad \Gamma, x : A \vdash t : MB}{\Gamma \vdash \mathsf{bind}(r, x.t) : MB}$$

Instead of a term t of monadic type MB, we now have a command c where no type can be assigned. Still, one can recover binding for terms by setting

$$\mathsf{bind}(r, x.t) := \mu a.\mathsf{bind}(r, x.at)$$

for some $a \notin r, t$ and even obtains the expected typing behaviour. For a more detailed analysis of the intuitionistic case, see Section 3.2.

The following typing rules for structural substitution are admissible where $x \notin C$:

$$\frac{\Gamma \vdash t : B|\Delta, a : MA \qquad C[x] : (\Gamma, x : MA \vdash \Delta)}{\Gamma \vdash [C/a]t : B|\Delta}$$

$$\frac{c : (\Gamma \vdash \Delta, a : MA) \qquad C[x] : (\Gamma, x : MA \vdash \Delta)}{[C/a]c : (\Gamma \vdash \Delta)}$$

Reduction rules. The base reduction rules of $\lambda\mu_{\mathsf{M}}$ are shown in Fig. 5. Thus, as for $\overline{\lambda}\mu\tilde{\mu}$, rule π causes substitution for co-variables whereas σ causes substitution for variables.

Rule π uses the derived syntactic class of base contexts and is therefore a scheme that stands for the following two rules

$$\begin{array}{lrcl} (\pi_{\mathsf{bind}}) & \mathsf{bind}(\mu a.c, x.c') & \to & [\mathsf{bind}([\,], x.c')/a]c \\ (\pi_{\mathsf{covar}}) & b(\mu a.c) & \to & [b/a]c \end{array}$$

[§] If the restriction on the type of co-variables would not be imposed, and accordingly the typing rules $Pass$ and Act could act with any type, instead of monadic types only, then, from any term t of type MA, we could build the term $\mu a.\mathsf{bind}(t, x.ax)$ of type A. This would represent a trivialisation of the system as a monadic language. We thank Dan Licata for this remark.

Fig. 5. Reduction rules of $\lambda\mu_{\mathsf{M}}$

$$\begin{array}{rrcl}
(\beta) & (\lambda x.t)s & \to & [s/x]t \\
(\sigma) & \mathsf{bind}(\eta s, x.c) & \to & [s/x]c \\
(\pi) & L[\mu a.c] & \to & [L/a]c \\
(\eta_\mu) & \mu a.at & \to & t \quad (a \notin t) \\
(\eta_{\mathsf{bind}}) & \mathsf{bind}(t, x.a(\eta x)) & \to & at
\end{array}$$

Counting these two rules separately, we can see that three rules are inherited from ordinary $\lambda\mu$ (β, "renaming" π_{covar} and η_μ), and one rule from ordinary monadic meta-language (σ). But two rules are original: π_{bind} and η_{bind}. Rule π_{bind} expresses the interaction of bind with μ-abstraction; notice that the left-hand side of π_{bind} fits well with the restriction of co-variables to monadic type: if $\mu a.c$ is well-typed, then with a monadic type, which is needed for the principal (first) argument of bind in order to type the whole expression.

A particular case of π_{bind} is

$$\mathsf{bind}(\mathsf{bind}(r, x.t), y.c) = \mathsf{bind}(\mu a.\mathsf{bind}(r, x.at), y.c) \quad \to \quad \mathsf{bind}(r, x.\mathsf{bind}(t, y.c))$$

for $a \notin r, t$. This is an "associativity" rule, formally similar to the "associativity" rule for bind found in the framework of λ-calculus, and recalled in Section 3.2 below.

Rule η_{bind} will be needed for the simulation of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ by the cbv monadic translation.

In the target of the monadic translations or in the source of some continuations-monad instantiations to be introduced below, the reduction rules of $\lambda\mu_{\mathsf{M}}$ are used in a variety of restricted forms. There is the restriction of β to variable renaming:

$$(\beta_{\mathsf{var}}) \quad (\lambda x.t)y \quad \to \quad [y/x]t \quad (y \notin t)$$

There are the cbv restrictions of the rules β, σ, and η_μ:

$$\begin{array}{rrcl}
(\beta_{\mathsf{v}}) & (\lambda x.t)V & \to & [V/x]t \\
(\sigma_{\mathsf{v}}) & \mathsf{bind}(\eta V, x.c) & \to & [V/x]c \\
(\eta_{\mu\mathsf{v}}) & \mu a.a(\eta V) & \to & \eta V \quad (a \notin V)
\end{array}$$

There are also cbn versions of the same rules, whose definition uses a $\lambda\mu_{\mathsf{M}}$-term N that is not an application :

$$\begin{array}{rrcl}
(\beta_{\mathsf{n}}) & (\lambda x.t)N & \to & [N/x]t \\
(\sigma_{\mathsf{n}}) & \mathsf{bind}(\eta N, x.c) & \to & [N/x]c \\
(\eta_{\mu\mathsf{n}}) & \mu a.aN & \to & N \quad (a \notin N)
\end{array}$$

Note that the cbn versions properly contain the respective cbv versions. All of the seven restricted versions of $\lambda\mu_{\mathsf{M}}$ reduction rules obviously fail closure under term substitution, i. e., we do *not* have that $T \to_\rho T'$ implies $[t/x]T \to_\rho [t/x]T'$, where $\to_\rho$ stands for any of the above restricted reductions. This is because variables, values and non-applications are evidently not closed under term substitution. However, all the rules of Fig. 5 satisfy closure under term substitution, as well as a final restriction of σ that we will consider:

$$(\sigma_{\mathsf{C}}) \qquad \mathsf{bind}(\eta s, x.C[x]) \quad \to \quad C[s] \quad (x \notin C)$$

It goes without saying that $\lambda\mu_{\mathsf{M}}$ enjoys subject reduction: types of terms and commands are preserved under $\rightarrow$ (the term closure of the base reduction rules that would more precisely be called the "closure under all expression constructors"). There are five critical pairs between the reduction rules, not surprisingly always in connection with one of the rules η_μ and η_{bind}. Two such pairs involve π_{covar} and η_μ (in one the root of the term is a π_{covar}-redex and in the other the root of the term is a η_μ-redex), but they are both trivial. The critical pair between π_{bind} and η_μ and the one between η_{bind} and σ are also trivial. The remaining critical pair is between η_{bind} and π_{bind}: $\mathsf{bind}(\mu a.c, x.b(\eta x))$ reduces both to (i) $[\mathsf{bind}([\,], x.b(\eta x))/a]c$ and to (ii) $b(\mu a.c)$, but these two terms both reduce to $[b/a]c$; (ii) by one π_{covar}-step and (i) with zero or more η_{bind}-steps (a result proved together with its analogue for terms). Since all critical pairs are joinable, $\lambda\mu_{\mathsf{M}}$ enjoys local confluence.

3.2. *Intuitionistic subsystem and relation with Moggi's meta-language*

We identify the *intuitionistic fragment* and an *intuitionistic subsystem* of monadic $\lambda\mu$, and show that the latter is essentially Moggi's meta-language, with a difference just in the reduction rule for the associativity of binds.

We start with two isomorphic presentations of the intuitionistic fragment. Let $*$ be a fixed co-variable. The intuitionistic terms and commands are generated by the grammar

$$\begin{array}{lrcl} \text{(Terms)} & r,s,t,u & ::= & x \mid \lambda x.t \mid tu \mid \mu * .c \mid \eta t \\ \text{(Commands)} & c & ::= & *t \mid \mathsf{bind}(t, x.c) \end{array}$$

Terms have no free occurrences of co-variables and each command has exactly one free occurrence of $*$. Sequents are restricted to have exactly one formula on the RHS. The typing rules and the reduction rules of the intuitionistic fragment are the expected restrictions to the typing and reduction rules of $\lambda\mu_{\mathsf{M}}$. We do not spell them out. Instead, we develop an isomorphic variant of the intuitionistic fragment, where $*$ and the μ-binder are fully avoided, and replaced by two *coercion* constructs, one from commands to terms and the other from terms to commands. The grammar of expressions becomes:

$$\begin{array}{lrcl} \text{(Terms)} & r,s,t,u & ::= & x \mid \lambda x.t \mid tu \mid \{c\} \mid \eta t \\ \text{(Commands)} & c & ::= & \ulcorner t \urcorner \mid \mathsf{bind}(t, x.c) \end{array}$$

The two forms of judgements are $\Gamma \vdash t : A$, and $c : (\Gamma \vdash MA)$. Note that these simplified judgement forms reflect both the restriction to only one formula on RHS's and the complete absence of co-variables.

The typing rules $Pass$ and Act are now:

$$\frac{\Gamma \vdash t : MA}{\ulcorner t \urcorner : (\Gamma \vdash MA)}\ Pass \qquad \frac{c : (\Gamma \vdash MA)}{\Gamma \vdash \{c\} : MA}\ Act$$

We omit writing the other typing rules. Reduction rules β and σ read as for $\lambda\mu_{\mathsf{M}}$, and the other rules read as follows:

$$\begin{array}{lrcl} (\pi_{\mathsf{bind}}) & \mathsf{bind}(\{c\}, x.c') & \rightarrow & (c@x.c') \\ (\pi_{\ulcorner . \urcorner}) & \ulcorner \{c\} \urcorner & \rightarrow & c \\ (\eta_{\{.\}}) & \{\ulcorner t \urcorner\} & \rightarrow & t \\ (\eta_{\mathsf{bind}}) & \mathsf{bind}(t, x.\ulcorner \eta x \urcorner) & \rightarrow & \ulcorner t \urcorner \end{array}$$

where the operation @ is the isomorphic counterpart of substitution of base contexts in the bind form for $*$, and is given by

$$\begin{array}{rcl}(\mathsf{bind}(t, y.c')@x.c) & = & \mathsf{bind}(t, y.(c'@x.c)) \\ (\ulcorner t \urcorner @x.c) & = & \mathsf{bind}(t, x.c)\end{array}$$

Now, we consider a simplification of this isomorphic variant of the intuitionistic fragment of $\lambda\mu_{\mathsf{M}}$. We call it the *intuitionistic subsystem* of $\lambda\mu_{\mathsf{M}}$. If we do not write the coercions $\{.\}$ and $\ulcorner . \urcorner$ in the isomorphic fragment, we can merge terms and commands into the grammar

$$\text{(Terms)} \qquad r, s, t, u \quad ::= \quad x \mid \lambda x.t \mid tu \mid \mathsf{bind}(t, x.u) \mid \eta t$$

and have only the sequent form $\Gamma \vdash t : A$. This causes rules $Pass$ and Act to collapse, gives the usual rule for typing bind in the framework of λ-calculus (see Section 3.1), and leaves the other typing rules unchanged.

These terms and typing rules correspond to those of Moggi's monadic meta-language (Moggi, 1991). With this term grammar, rules $\pi_{\ulcorner . \urcorner}$ and $\eta_{\{.\}}$ become identities, just as the case $c = \ulcorner t \urcorner$ of π_{bind}, and we are left with the following rules:

$$\begin{array}{rrcl}(\beta) & (\lambda x.t)s & \to & [s/x]t \\ (\sigma) & \mathsf{bind}(\eta s, x.t) & \to & [s/x]t \\ (\pi_{\mathsf{bind}}) & \mathsf{bind}(\mathsf{bind}(t, x.u), y.s) & \to & \mathsf{bind}(t, x.(u@y.s)) \\ (\eta_{\mathsf{bind}}) & \mathsf{bind}(t, x.\eta x) & \to & t\end{array}$$

where @ is as for the isomorphic variant (recall that commands became terms):

$$\begin{array}{rcll}(\mathsf{bind}(t, y.s)@x.u) & = & \mathsf{bind}(t, y.(s@x.u)) & \\ (t@x.u) & = & \mathsf{bind}(t, x.u) & \text{otherwise}\end{array}$$

The difference of these rules to the usual reduction rules for Moggi's monadic meta-language, as in (Hatcliff and Danvy, 1994; Sabry and Wadler, 1997), is rule π_{bind}. There the rule used reads:

$$(\mathsf{assoc}) \quad \mathsf{bind}(\mathsf{bind}(t, x.u), y.s) \to \mathsf{bind}(t, x.\mathsf{bind}(u, y.s))$$

But $t \to_{\pi_{\mathsf{bind}}} u$ implies $t \to^{+}_{\mathsf{assoc}} u$ since $\mathsf{bind}(u, y.s) \to^{*}_{\mathsf{assoc}} (u@y.s)$. Thus, this intuitionistic subsystem of $\lambda\mu_{\mathsf{M}}$ corresponds to Moggi's monadic meta-language with an eager version of assoc.

4. Monadic translation

Two translations of $\overline{\lambda}\mu\tilde{\mu}$ into $\lambda\mu_{\mathsf{M}}$ are given. The first one, denoted $\overline{(.)}_{\mathsf{n}}$, allows to simulate every reduction step of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ by at least one reduction step of $\lambda\mu_{\mathsf{M}}$ (thus, $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ is strictly simulated by $\lambda\mu_{\mathsf{M}}$); the second one, denoted $\overline{(.)}_{\mathsf{v}}$, gives strict simulation of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ within $\lambda\mu_{\mathsf{M}}$. Thus, they are monadic cbn and cbv translations of $\overline{\lambda}\mu\tilde{\mu}$, respectively.

4.1. *Call-by-name translation*

In this section we define and study translation $\overline{(.)}_{\mathsf{n}}$. To keep the notation light, the subscript $_{\mathsf{n}}$ is omitted throughout the section, including for the auxiliary notion $(.)^{\dagger}_{\mathsf{n}}$.

Fig. 6. Admissible typing rules for monadic cbn translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\frac{\Gamma \vdash t : A|\Delta}{\overline{\Gamma} \vdash \overline{t} : \overline{A}|\overline{\Delta}} \qquad \frac{c : (\Gamma \vdash \Delta)}{\overline{c} : (\overline{\Gamma} \vdash \overline{\Delta})} \qquad \frac{\Gamma|e : A \vdash \Delta}{\overline{e}[y] : (\overline{\Gamma}, y : \overline{A} \vdash \overline{\Delta})}$$

Fig. 7. Monadic cbn translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\begin{array}{rcl} \overline{y} & = & y \\ \overline{\lambda y.t} & = & \eta(\lambda y.\overline{t}) \\ \overline{\mu a.c} & = & \mu a.\overline{c} \\ \overline{a} & = & a[\,] \\ \overline{u :: e} & = & \mathsf{bind}([\,], f.\mathsf{bind}(\eta\overline{u}, z.\overline{e}[fz])) \\ \overline{\tilde{\mu} y.c} & = & \mathsf{bind}(\eta[\,], y.\overline{c}) \end{array} \qquad \overline{\langle t|e\rangle} \;=\; \overline{e}[\overline{t}]$$

A type A of $\overline{\lambda}\mu\tilde{\mu}$ is translated to $\overline{A}$ of $\lambda\mu_{\mathsf{M}}$, defined by recursion on A:

$$\overline{X} = MX \qquad \text{and} \qquad \overline{A \supset B} = M(\overline{A} \supset \overline{B})$$

Denote by $A^\dagger$ the type $\overline{A}$ without the outermost application of M, i. e., we have

$$X^\dagger = X \qquad \text{and} \qquad (A \supset B)^\dagger = \overline{A} \supset \overline{B}\ ,$$

and then $\overline{A} = MA^\dagger$.

Any term t of $\overline{\lambda}\mu\tilde{\mu}$ is translated into a term $\overline{t}$ of $\lambda\mu_{\mathsf{M}}$, any command c of $\overline{\lambda}\mu\tilde{\mu}$ into a command $\overline{c}$ and any co-term e of $\overline{\lambda}\mu\tilde{\mu}$ into a cbn context $\overline{e}$ of $\lambda\mu_{\mathsf{M}}$. This is done so that the typing rules in Fig. 6 are admissible, where $\overline{\Gamma}$ and $\overline{\Delta}$ follow the notational conventions of Section 1, with type operation $F := \overline{(.)}$ (notice that $\overline{A}$ is a monadic type, as required for co-contexts in $\lambda\mu_{\mathsf{M}}$).

The definitions are in Fig. 7, where it is understood that f and z are fresh variable names (we assume henceforth that also f denotes a variable of $\lambda\mu_{\mathsf{M}}$). We prefer to denote all variables from the source calculus $\overline{\lambda}\mu\tilde{\mu}$ as y (which will be translated into variables of type $\overline{A}$ for some A). Admissibility of the rules of Fig. 6 is routine and makes – through $\overline{e}[\overline{t}] = [\overline{t}/y](\overline{e}[y])$ for $y \notin e$ – use of the following admissible rule for term substitution in commands in $\lambda\mu_{\mathsf{M}}$:

$$\frac{c : (\Gamma, x : A \vdash \Delta) \qquad \Gamma \vdash t : A|\Delta}{[t/x]c : (\Gamma \vdash \Delta)}$$

Notice that $\overline{E}$ is a base context, which will be important for simulation of π. The σ-redex in $\overline{u :: e}$ is needed for the simulation of β, which is a rule that generates but does not execute a substitution.

We immediately observe that the free variables and the free co-variables agree between T and $\overline{T}$ for any expression T (the hole $[\,]$ in $\overline{e}$ does not count as a variable). In general, t is a subterm of $\overline{e}[t]$ that does not occur below a binder.

Lemma 1. The translation satisfies:
1. $\overline{[t/y]T} = [\overline{t}/y]\overline{T}$.
2. $\overline{[e/a]T} = [\overline{e}/a]\overline{T}$.

Proof. 1. By induction on T.

2. By induction on T. Case $T = a$. $\overline{[e/a]a} = \overline{e} = [\overline{e}/a](a[\,]) = [\overline{e}/a]\overline{a}$. The second equation is by definition of structural substitution on the context $a[\,]$. □

Theorem 2 (Strict simulation).

1. If $T \to T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, then $\overline{T} \to^{+} \overline{T'}$ in $\lambda\mu_{\mathsf{M}}$, where T, T' are either two terms or two commands.

2. If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, then $\overline{e}[\overline{t}] \to^{+} \overline{e'}[\overline{t}]$ in $\lambda\mu_{\mathsf{M}}$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

For this simulation, we do not need rule η_{bind} in $\lambda\mu_{\mathsf{M}}$, the rules β and η_{μ} could have been restricted to the forms β_{var} and $\eta_{\mu\mathsf{n}}$, respectively, and we do not need full σ, but just the restrictions σ_{n} and σ_{C}.

Proof. Statement 2 is strengthened so that $\overline{e}[u] \to^{+} \overline{e'}[u]$ for any $u \in \lambda\mu_{\mathsf{M}}$. This is needed because in the definition of $\overline{u :: e}$ a term outside the range of $\overline{(.)}$ is filled into the hole of $\overline{e}$. Statement 1 and strengthened statement 2 are proved by simultaneous induction on the appropriate $T \to T'$. The base cases are shown in detail. The term closure is then evident since t is a subterm of $\overline{e}[t]$. For the justification of the restriction of η_{μ}-steps to their cbn form in the proof, as well as for the restriction to σ_{n}-steps for the simulation of σ, notice that $\overline{t}$ is not an application, for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

Case β: $\langle \lambda y.t | u :: e\rangle \to \langle u | \tilde{\mu} y.\langle t | e\rangle\rangle$.

$$\begin{array}{rcl}
\overline{LHS} & = & \mathsf{bind}(\eta(\lambda y.\overline{t}), f.\mathsf{bind}(\eta\overline{u}, z.\overline{e}[fz])) \\
& \to_{\sigma_{\mathsf{v}}} & \mathsf{bind}(\eta\overline{u}, z.\overline{e}[(\lambda y.\overline{t})z]) \\
& \to_{\beta_{\mathsf{var}}} & \mathsf{bind}(\eta\overline{u}, z.\overline{e}[[z/y]\overline{t}]) \\
& = & \mathsf{bind}(\eta\overline{u}, y.\overline{e}[\overline{t}]) \\
& = & \overline{RHS}
\end{array}$$

Case σ: $\langle t | \tilde{\mu} y.c\rangle \to [t/y]c$.

$$\overline{LHS} = \mathsf{bind}(\eta\overline{t}, y.\overline{c}) \to_{\sigma_{\mathsf{n}}} [\overline{t}/y]\overline{c} \stackrel{*}{=} \overline{[t/y]c} = \overline{RHS}$$

where the marked equality comes from Lemma 1.1.¶

Case π_{n}: $\langle \mu a.c | E\rangle \to [E/a]c$.

$$\overline{LHS} = \overline{E}[\mu a.\overline{c}] \to_{\pi} [\overline{E}/a]\overline{c} \stackrel{*}{=} \overline{[E/a]c} = \overline{RHS}$$

where the marked equality comes from Lemma 1.2. Recall that $\overline{E}$ is a base context. Otherwise, the π rule of $\lambda\mu_{\mathsf{M}}$ would not have been applicable.

Case $\eta_{\tilde{\mu}}$: $\tilde{\mu} y.\langle y | e\rangle \to e,$ with $y \notin e$.

$$\begin{array}{rcl}
\overline{LHS}[u] & = & \mathsf{bind}(\eta u, y.\overline{e}[y]) \\
& \to_{\sigma_{\mathsf{C}}} & [u/y](\overline{e}[y]) \stackrel{*}{=} \overline{e}[u] = \overline{RHS}[u]
\end{array}$$

where the σ_{C}-step and the marked equality use the fact $y \notin \overline{e}$.

Case η_{μ}: $\mu a.\langle t | a\rangle \to t,$ with $a \notin t$, hence also $a \notin \overline{t}$: $\overline{LHS} = \mu a.a\overline{t} \to_{\eta_{\mu\mathsf{n}}} \overline{t} = \overline{RHS}$. □

¶ Lemma 1.1 refers to the item 1. in the statement of Lemma 1. This kind of reference will be used throughout the paper.

Remark 3. Notice the structural "tightness" of the simulation. Every reduction step of the forms σ, π_{n}, η_μ, $\eta_{\tilde{\mu}}$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ corresponds to exactly one step in $\lambda\mu_{\mathsf{M}}$ of the forms σ_{n}, π, $\eta_{\mu\mathsf{n}}$, σ_{C}, respectively; only β-steps of $\overline{\lambda}\mu\tilde{\mu}$ are decomposed into two steps of $\lambda\mu_{\mathsf{M}}$, which are of the restricted forms σ_{v} and β_{var}.[‖]

Remark 4. Strict simulation is satisfied because the monadic translation never erases subexpressions. More precisely, the translation satisfies the following *Subexpression Property*: (i) for T' a term or command: if T' is a subexpression of T, then $\overline{T'}$ is a subexpression of $\overline{T}$ (and of $\overline{T}[t]$, for any t, in case T is a co-term); (ii) if co-term e is a subexpression of T, then, for some t', $\overline{e}[t']$ is a subexpression of $\overline{T}$ (and of $\overline{T}[t]$, for any t, in case T is a co-term).

4.2. *Call-by-value translation*

In this section we define and study translation $\overline{(.)}_{\mathsf{v}}$. The subscript $_{\mathsf{v}}$ is omitted throughout the section, in order to have a light notation.

The cbv translation on types is the same as the cbn one except for the implication:

$$\overline{(A \supset B)} = M(A^\dagger \supset \overline{B})$$

Then $(A \supset B)^\dagger$ is defined as $(A^\dagger \supset \overline{B})$. The monadic cbv translation on expressions is defined in Fig. 8 so that the typing rules in Fig. 9 are admissible, where $\Gamma^\dagger$ and $\overline{\Delta}$ again follow the notational pattern set up in Section 1.

Notice that $\overline{e}$ is always a base context of $\lambda\mu_{\mathsf{M}}$, and that $V^\dagger$ is a value.

We see that there are only minimal differences between the monadic translations for cbn (in the previous section) and cbv for the part that is not already dictated by the λ-calculus part inside $\overline{\lambda}\mu\tilde{\mu}$. That part, namely the rules for types, values and typing of terms are the standard ones in monadic translations. The new elements are also treated mostly in the same way: μ-abstraction is translated homomorphically, commands by plugging the term translation into a context obtained from co-term translation, and the co-variables are translated in the most obvious way. The remaining clauses for $u :: e$ and $\tilde{\mu}$-abstraction are identical for both translations, except for the extra uses of the unit η of the monad which is, however, not applied throughout, so that the cbn translation of $u :: e$ remains still a base context. (Evidently, this is already dictated by typing considerations, where the type translation leaves no room as soon as values have been treated in the standard way.)

We prefer to denote all variables from the source calculus $\overline{\lambda}\mu\tilde{\mu}$ as v (which will be translated into variables of type $A^\dagger$ for some A).

Lemma 5. The translation satisfies:
1. $\overline{[V/v]T} = [V^\dagger/v]\overline{T}$.
2. $\overline{[e/a]T} = [\overline{e}/a]\overline{T}$.

Proof. Induction on T. □

[‖] In the intuitionistic case of (Espírito Santo et al., 2009b), the β-rule of the monadic calculus enters in the simulation of every step, and π-steps of the source are decomposed into several reduction steps in the target.

Fig. 8. Monadic cbv translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\begin{array}{rclcrcl}
\overline{V} & = & \eta V^\dagger & & v^\dagger & = & v \\
\overline{\mu a.c} & = & \mu a.\overline{c} & & (\lambda v.t)^\dagger & = & \lambda v.\overline{t} \\
\overline{a} & = & a[\,] & & \overline{\langle t|e\rangle} & = & \overline{e}[\overline{t}] \\
\overline{u :: e} & = & \mathsf{bind}([\,], f.\mathsf{bind}(\overline{u}, w.\overline{e}[fw])) & & & & \\
\overline{\tilde{\mu} v.c} & = & \mathsf{bind}([\,], v.\overline{c}) & & & &
\end{array}$$

Fig. 9. Admissible typing rules for monadic cbv translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\frac{\Gamma \vdash t : A|\Delta}{\Gamma^\dagger \vdash \overline{t} : \overline{A}|\overline{\Delta}} \qquad \frac{\Gamma \vdash V : A|\Delta}{\Gamma^\dagger \vdash V^\dagger : A^\dagger|\overline{\Delta}} \qquad \frac{c : (\Gamma \vdash \Delta)}{\overline{c} : (\Gamma^\dagger \vdash \overline{\Delta})} \qquad \frac{\Gamma|e : A \vdash \Delta}{\overline{e}[y] : (\Gamma^\dagger, y : \overline{A} \vdash \overline{\Delta})}$$

Theorem 6 (Strict simulation). 1. If $T \to T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $\overline{T} \to^+ \overline{T'}$ in $\lambda\mu_{\mathsf{M}}$, where T, T' are either two terms or two commands.

2. If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $\overline{e}[\overline{t}] \to^+ \overline{e'}[\overline{t}]$ in $\lambda\mu_{\mathsf{M}}$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

The reductions in $\lambda\mu_{\mathsf{M}}$ only use β, σ, and η_μ in their restricted forms β_{var}, σ_{v}, and $\eta_{\mu\mathsf{v}}$, respectively.

Proof. The proof is similar to the cbn case. Statement 2 is again strengthened, so that $\overline{e}[u] \to^+ \overline{e'}[u]$ for any $u \in \lambda\mu_{\mathsf{M}}$. Term closure is again evident since t is a subterm of $\overline{e}[t]$. We show the base cases.

Case β: $\langle \lambda v.t | u :: e\rangle \to \langle u|\tilde{\mu}v.\langle t|e\rangle\rangle$.

$$\begin{array}{rcl}
\overline{LHS} & = & \mathsf{bind}(\eta(\lambda v.\overline{t}), f.\mathsf{bind}(\overline{u}, w.\overline{e}[fw])) \\
 & \to_{\sigma_{\mathsf{v}}} & \mathsf{bind}(\overline{u}, w.\overline{e}[(\lambda v.\overline{t})w]) \\
 & \to_{\beta_{\mathsf{var}}} & \mathsf{bind}(\overline{u}, w.\overline{e}[[w/v]\overline{t}]) \\
 & = & \mathsf{bind}(\overline{u}, v.\overline{e}[\overline{t}]) \\
 & = & \overline{RHS}
\end{array}$$

Case π: $\langle \mu a.c|e\rangle \to [e/a]c$.

$$\overline{LHS} = \overline{e}[\mu a.\overline{c}] \to_\pi [\overline{e}/a]\overline{c} = \overline{RHS}$$

where the last equality comes from Lemma 5.2.

Case σ_{v}: $\langle V|\tilde{\mu}v.c\rangle \to [V/v]c$.

$$\overline{LHS} = \mathsf{bind}(\eta V^\dagger, v.\overline{c}) \to_{\sigma_{\mathsf{v}}} [V^\dagger/v]\overline{c} = \overline{RHS}$$

where the last equality comes from Lemma 5.1.

Case η_μ: $\mu a.\langle t|a\rangle \to t$, with $a \notin t$.

$$\overline{LHS} = \mu a.a\overline{t} \to_{\eta_\mu} \overline{RHS}$$

We now argue that the restriction of η_μ to $\eta_{\mu\mathsf{v}}$ suffices in the target system. If t is a value V, then $\overline{t} = \eta V^\dagger$, and $V^\dagger$ is again a value, and so the displayed reduction is a $\eta_{\mu\mathsf{v}}$-reduction step. If

$t = \mu b.c$, then the η_μ-reduction $\mu a.\langle t|a\rangle \to t$ is also a π-reduction (this is one of the two trivial critical pairs between η_μ and π) and can be considered as such for the purpose of this proof.

Case $\eta_{\tilde{\mu}}$: $\tilde{\mu}v.\langle v|e\rangle \to e$, with $v \notin e$. We have $\overline{LHS}[u] = \mathsf{bind}(u, v.\overline{e}[\eta v])$. If e is a co-variable a, we have

$$\mathsf{bind}(u, v.\overline{e}[\eta v]) = \mathsf{bind}(u, v.a(\eta v)) \to_{\eta_{\mathsf{bind}}} au = \overline{RHS}[u]$$

Otherwise, $\overline{e}$ is of the form $\mathsf{bind}([], w.c)$, so we have

$$\begin{array}{rcl} \mathsf{bind}(u, v.\overline{e}[\eta v]) & = & \mathsf{bind}(u, v.\mathsf{bind}(\eta v, w.c)) \\ & \to_{\sigma_{\mathsf{v}}} & \mathsf{bind}(u, w.c) = \overline{e}[u] = \overline{RHS}[u] \end{array}$$

□

Remark 7. Rule η_{bind} is now required. As for cbn, the simulation is quite "tight". Every reduction step of the forms σ_{v}, π, η_μ, $\eta_{\tilde{\mu}}$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ corresponds to exactly one step in $\lambda\mu_{\mathsf{M}}$ of the forms σ_{v}, π, $\eta_{\mu\mathsf{v}}$ or π, η_{bind} or σ_{v}, respectively; again, only β-steps of $\overline{\lambda}\mu\tilde{\mu}$ are decomposed into two steps of $\lambda\mu_{\mathsf{M}}$ of the restricted forms σ_{v} and β_{var}.

Remark 8. The cbv monadic translation satisfies the same Subexpression Property as the cbn one.

5. Continuations-monad instantiation

The monad operation M of $\lambda\mu_{\mathsf{M}}$ can be instantiated to be double negation that yields the well-known continuations monad. This defines a translation into the λ-calculus with the strict simulation property. Given that the monadic translations of Section 4 also enjoy strict simulation, strong normalization for cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$ will follow. Composition of instantiation with the monadic translations will yield cbn and cbv CPS translations of $\overline{\lambda}\mu\tilde{\mu}$, whose recursive definition is calculated at the end of the present section.

5.1. *Instantiation and strong normalization*

We define a translation from $\lambda\mu_{\mathsf{M}}$ into $\lambda[\beta\eta]$ - recall from Section 1 the meaning of this notation. Also recall from the same section the definition of simple types, and write $A \supset B$ also for function types in $\lambda[\beta\eta]$, and, as usual, write $\neg A$ for $A \supset \bot$ for some dedicated type variable $\bot$ that will never be instantiated. Recall finally that λ-calculus has the grammar $t ::= x \mid \lambda x.t \mid tu$ and that its only typing rules are Ax, $Intro$ and $Elim$ from Fig. 4, without the Δ parts.

A translation of the terms and commands of $\lambda\mu_{\mathsf{M}}$ into terms of λ-calculus necessarily has to associate both variables and co-variables of $\lambda\mu_{\mathsf{M}}$ with variables of the λ-calculus. The obvious and usual choice for a variable x of $\lambda\mu_{\mathsf{M}}$ is to associate it with the same variable in λ-calculus, hence assuming that the variables of $\lambda\mu_{\mathsf{M}}$ are included in the variable supply of the λ-calculus. For the co-variables, the traditional way would be to associate a "fresh" variable k_a of λ-calculus with every co-variable a. Given an expression T of $\lambda\mu_{\mathsf{M}}$, there would always be enough "fresh" variables when defining the translation of T, but the notion k_a rather suggests to have one fixed association that works for all source expressions. We adopt this uniform choice, but we go one step further: we assume that the co-variables of $\lambda\mu_{\mathsf{M}}$ (that are those of $\overline{\lambda}\mu\tilde{\mu}$) are also included

Fig. 10. Continuations-monad instantiation

$$
\begin{array}{rlrl}
x^\bullet &= x & (L[t])^\bullet &= L^\bullet[t^\bullet] \\
(\lambda x.t)^\bullet &= \lambda x.t^\bullet & & \\
(tu)^\bullet &= t^\bullet u^\bullet & (a[\,])^\bullet &= [\,]a \\
(\mu a.c)^\bullet &= \lambda a.c^\bullet & \mathsf{bind}([\,],x.c)^\bullet &= [\,](\lambda x.c^\bullet) \\
(\eta t)^\bullet &= \lambda k.kt^\bullet & &
\end{array}
$$

in the variable supply of the λ-calculus, so that we can associate the co-variable a with the λ-calculus variable a. Such an assumption simplifies notation, in particular in the extension of type operations to co-contexts Δ (see Section 1, where extension works the same on Γ and Δ), and because k_a is just a.[††]

The translation on types is defined as follows:

$$X^\bullet = X \qquad (A \supset B)^\bullet = A^\bullet \supset B^\bullet \qquad (MA)^\bullet = \neg\neg A^\bullet$$

The translation of expressions is defined in Fig. 10. Notice the mapping of co-variables a in the source calculus $\lambda\mu_{\mathsf{M}}$ into ordinary variables of λ-calculus that is silently done in the cases for $\mu a.c$ and $a[\,]$. All expressions (terms and commands) of $\lambda\mu_{\mathsf{M}}$ are translated into λ-terms. The definition of $c^\bullet$ is well-formed since every command c can be uniquely presented as $L[t]$.

Define $L^{\bullet -}$ to be the argument to $[\,]$ of the context $L^\bullet$, i. e.,

$$
\begin{array}{rcl}
(a[\,])^{\bullet -} & = & a \\
\mathsf{bind}([\,],x.c)^{\bullet -} & = & \lambda x.c^\bullet \ ,
\end{array}
$$

so that $L^\bullet = [\,]L^{\bullet -}$ and $L^{\bullet -}$ is a value.

Consider the following two operators:

$$\mathsf{Eta}(t) = \lambda k.kt \qquad \mathsf{Bind}(t,x.u) = \overline{\lambda}k.t(\lambda x.uk),$$

where $\overline{\lambda}$ denotes the static λ-abstraction in the two-level λ-calculus of (Danvy and Filinski, 1992), that is, redexes of the form $(\overline{\lambda}x.t)u$ are supposed to be reduced in the translation. Then the two monad-related clauses of the definition of $(\cdot)^\bullet$ can be turned into

$$(\eta t)^\bullet = \mathsf{Eta}(t^\bullet) \qquad \mathsf{bind}(t,x.L[u])^\bullet = \mathsf{Bind}(t^\bullet,x.u^\bullet)L^{\bullet -}.$$

For the use in cbn translations, also define

$$\mathsf{bind}(\eta[\,],x.c)^\bullet = \mathsf{Eta}([\,])(\lambda x.c^\bullet) \ .$$

This immediately implies

$$(C[t])^\bullet = C^\bullet[t^\bullet] \tag{1}$$

[††] Viewed from the λ-calculus, there is no difference between these two variable supplies guaranteed by our assumptions. E. g., the letter x in rule β given in Section 1 still denotes any variable of λ-calculus, and we will *not* use a to denote an arbitrary variable of λ-calculus. If a appears in a translation, it stands for an arbitrary co-variable of $\lambda\mu_{\mathsf{M}}$ or $\overline{\lambda}\mu\tilde{\mu}$, and this co-variable is then also a variable of λ-calculus and can therefore appear in terms in the range of the translation.

Fig. 11. Admissible typing rules for continuations-monad instantiation

$$\frac{\Gamma \vdash t : A|\Delta}{\Gamma^\bullet, \Delta^{\bullet -} \vdash t^\bullet : A^\bullet} \qquad \frac{c : (\Gamma \vdash \Delta)}{\Gamma^\bullet, \Delta^{\bullet -} \vdash c^\bullet : \bot} \qquad \frac{C[x] : (\Gamma, x : MA \vdash \Delta)}{\Gamma^\bullet, x : \neg\neg A^\bullet, \Delta^{\bullet -} \vdash (C[x])^\bullet : \bot}\ x \notin C$$

for all cbn contexts C.

This translation also satisfies a Subexpression Property: if T' is a subexpression of T, then the term $T'^\bullet$ is a subterm of $T^\bullet$. The best way to see this in the case $T = c$ is to unfold the two cases of the definition of $(L[t])^\bullet$.

We can easily check that the rules in Fig. 11 are admissible (the third rule is a special case of the second one, displayed for later proofs), where $\Delta^{\bullet -}$ follows the usual pattern, with the type operation $(.)^{\bullet -}$ with

$$(MB)^{\bullet -} := \neg B^\bullet \tag{2}$$

(recall $a : A \in \Delta$ implies $A = MB$, for some B, so the apparent partiality of this operation is no problem when forming $\Delta^{\bullet -}$). The minus sign is a warning that $(MB)^{\bullet -}$ has a negation less than $(MB)^\bullet$. In addition, this notation is coherent with the notation $L^{\bullet -}$ introduced above, in the sense that the following rule is admissible:

$$\frac{L[x] : (\Gamma, x : MA \vdash \Delta)}{\Gamma^\bullet, \Delta^{\bullet -} \vdash L^{\bullet -} : (MA)^{\bullet -}}\ x \notin L$$

So the type of $L^{\bullet -}$ has one negation less than the type of the hole of $L^\bullet$.

We now show that the instantiation is a strict simulation of $\lambda\mu_{\mathsf{M}}$ in $\lambda[\beta\eta]$.

Lemma 9. The translation satisfies:
1. $([u/x]T)^\bullet = [u^\bullet/x]T^\bullet$.
2. $([L/a]T)^\bullet = [L^{\bullet -}/a]T^\bullet$.

Proof. 1. Induction on T.

2. Induction on T. The case of $T = at$ is the only non-trivial case, and it is proved as follows:

$$\begin{array}{rcll} ([L/a](at))^\bullet & = & (L[[L/a]t])^\bullet & \text{(by def. of struct. subst.)} \\ & = & ([L/a]t)^\bullet L^{\bullet -} & \text{(by def of } (.)^\bullet \text{ and } L^\bullet[u^\bullet] = u^\bullet L^{\bullet -}) \\ & = & [L^{\bullet -}/a]t^\bullet L^{\bullet -} & \text{(by IH)} \\ & = & [L^{\bullet -}/a](t^\bullet a) & \text{(by def. of subst. in the } \lambda\text{-calculus)} \\ & = & [L^{\bullet -}/a](at)^\bullet & \text{(by def. of } (.)^\bullet) \end{array}$$

□

Proposition 10 (Instantiation). 1. If $T \to T'$ in $\lambda\mu_{\mathsf{M}}$, then $T^\bullet \to^+ T'^\bullet$ in $\lambda[\beta\eta]$.

2. If the reduction rules η_μ and η_{bind} are omitted from the source, then reduction rule η can be omitted from the target. If the reduction rules β and σ in the source are restricted to the forms β_{v} and σ_{v}, respectively, then reduction rule β in the target can be restricted to β_{v}.

Proof. 1. Induction on $T \to T'$. We just show the base cases, as term closure is evident by the Subexpression Property.

Case β: $(\lambda x.t)s \to [s/x]t$.

$$\begin{aligned} LHS^\bullet &= (\lambda x.t^\bullet)s^\bullet \\ &\to_\beta [s^\bullet/x]t^\bullet \\ &= RHS^\bullet \qquad \text{(by Lemma 9.1)} \end{aligned}$$

Case σ: $\mathsf{bind}(\eta s, x.c) \to [s/x]c$.

$$\begin{aligned} LHS^\bullet &= (\lambda k.ks^\bullet)(\lambda x.c^\bullet) \\ &\to_{\beta_\mathsf{v}} (\lambda x.c^\bullet)s^\bullet \\ &\to_\beta [s^\bullet/x]c^\bullet \\ &= RHS^\bullet \qquad \text{(by Lemma 9.1)} \end{aligned}$$

Case π: $L[\mu a.c] \to [L/a]c$.

$$\begin{aligned} LHS^\bullet &= (\lambda a.c^\bullet)L^{\bullet -} \\ &\to_{\beta_\mathsf{v}} [L^{\bullet -}/a]c^\bullet \\ &= RHS^\bullet \qquad \text{(by Lemma 9.2)} \end{aligned}$$

Case η_μ: $\mu a.at \to t$, with $a \notin t$ (hence $a \notin t^\bullet$).

$$LHS^\bullet = \lambda a.t^\bullet a \to_\eta t^\bullet = RHS^\bullet$$

Case η_{bind}: $\mathsf{bind}(t, x.a(\eta x)) \to at$.

$$\begin{aligned} LHS^\bullet &= t^\bullet(\lambda x.(\lambda k.kx)a) \\ &\to_{\beta_{\mathsf{var}}} t^\bullet(\lambda x.ax) \\ &\to_\eta t^\bullet a \\ &= RHS^\bullet. \end{aligned}$$

2. Observe that $V^\bullet$ is always a value. Therefore, the β steps in the cases β and σ of 1. turn into β_v steps for β_v and σ_v. □

In Appendix A.3, we will see that we can obtain refined continuations-monad instantiations which only need $\lambda[\beta_\mathsf{v}]$ as target. They only work for subsystems of $\lambda\mu_\mathsf{M}$, that however cover the images of the monadic translations.

Corollary 11. 1. Every typable expression of $\lambda\mu_\mathsf{M}$ is strongly normalizable.
2. The system $\lambda\mu_\mathsf{M}$ is confluent for typable expressions.

Proof. 1. By the previous proposition, strong normalization of $\lambda[\beta\eta]$, and that typability is preserved by the instantiation, shown in Fig. 11.
2. By strong normalizability and local confluence of $\lambda\mu_\mathsf{M}$ (using Newman's Lemma). □

Corollary 12. The systems $\overline{\lambda}\mu\tilde{\mu}_\mathsf{n}$ and $\overline{\lambda}\mu\tilde{\mu}_\mathsf{v}$ are strongly normalizing.

Proof. Use the previous corollary, the strict simulation results from Section 4, and preservation of typability, shown in Fig. 6 and in Fig. 9, respectively. □

We have thus reproved in a completely syntactic way strong normalization of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ from that of $\lambda[\beta\eta]$.

5.2. *CPS translations through instantiation of monadic translations*

Our proof of strong normalization for $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ gives syntactic embeddings of these systems into $\lambda[\beta\eta]$, obtained by composing cbn and cbv monadic translation, respectively, with the continuations-monad instantiation. The result is continuation-passing style (CPS) transformations of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ into $\lambda[\beta\eta]$.

We already know that both CPS translations yield strict simulations, being the composition of mappings with the strict simulation property. In the following we make this precise, obtaining a direct inheritance of strong normalization from the λ-calculus (rather than a two-step inheritance via $\lambda\mu_{\mathsf{M}}$ as done before). Similarly, we already know that both CPS translations enjoy type soundness, being the composition of type sound mappings. In the following we make explicit the typing rules they obey. Finally we discover the recursive structure of the CPS translations.

Define, for $\mathsf{x} \in \{\mathsf{n}, \mathsf{v}\}$:

$$A^*_{\mathsf{x}} := (A^\dagger_{\mathsf{x}})^\bullet \tag{3}$$
$$\langle\![A]\!\rangle_{\mathsf{x}} := (\overline{A}_{\mathsf{x}})^\bullet \tag{4}$$
$$\langle\![A]\!\rangle^-_{\mathsf{x}} := (\overline{A}_{\mathsf{x}})^{\bullet-} \tag{5}$$
$$\langle\![T]\!\rangle_{\mathsf{x}} := (\overline{T}_{\mathsf{x}})^\bullet \ . \tag{6}$$

In the cbn case, the last equation is well-defined because the definition of $(.)^\bullet$ was extended to any cbn context C. For the cbn case, we set

$$\langle\![E]\!\rangle^-_{\mathsf{n}} := (\overline{E}_{\mathsf{n}})^{\bullet-} \ . \tag{7}$$

For the cbv case, we set

$$V^* := (V^\dagger)^\bullet \tag{8}$$
$$\langle\!\langle e]\!\rangle^-_{\mathsf{v}} := (\overline{e}_{\mathsf{v}})^{\bullet-} \ . \tag{9}$$

Notice that there is no index v in $V^\dagger$, and consequently neither in V^*. This seems justified since there is simply no such concept in the cbn translations.

An easy calculation shows that $\langle\!\langle A]\!\rangle_{\mathsf{x}} = \neg\neg A^*_{\mathsf{x}}$, hence $\langle\!\langle A]\!\rangle^-_{\mathsf{x}} = \neg A^*_{\mathsf{x}}$. (Again, the minus sign warns that $\langle\!\langle A]\!\rangle^-_{\mathsf{x}}$ has one negation less than $\langle\!\langle A]\!\rangle_{\mathsf{x}}$.) Obviously, $X^*_{\mathsf{x}} = X$.

5.2.1. *Call-by-name CPS translation ($\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}} \longrightarrow \lambda[\beta\eta]$)* The translations of types (3) and (4) satisfy in the cbn case:

$$(A \supset B)^*_{\mathsf{n}} = \langle\!\langle A]\!\rangle_{\mathsf{n}} \supset \langle\!\langle B]\!\rangle_{\mathsf{n}}$$

Corollary 13 (Typing). The typing rules of Fig. 12 are admissible.

Proof. We "compose" the rules in Fig. 6 for $\overline{(.)}_{\mathsf{n}}$ with those in Fig. 11 for $(.)^\bullet$. We just show the typing rules for co-terms, the others being analogous but simpler.

Fig. 12. Admissible typing rules for cbn CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\frac{\Gamma \vdash t : A|\Delta}{(\!|\Gamma|\!)_{\mathsf{n}}, (\!|\Delta|\!)^{-}_{\mathsf{n}} \vdash (\!|t|\!)_{\mathsf{n}} : (\!|A|\!)_{\mathsf{n}}} \qquad \frac{\Gamma|e : A \vdash \Delta}{(\!|\Gamma|\!)_{\mathsf{n}}, y : (\!|A|\!)_{\mathsf{n}}, (\!|\Delta|\!)^{-}_{\mathsf{n}} \vdash (\!|e|\!)_{\mathsf{n}}[y] : \bot} \qquad \frac{c : (\Gamma \vdash \Delta)}{(\!|\Gamma|\!)_{\mathsf{n}}, (\!|\Delta|\!)^{-}_{\mathsf{n}} \vdash (\!|c|\!)_{\mathsf{n}} : \bot}$$

Fig. 13. Cbn CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\begin{array}{rcl}
(\!|y|\!)_{\mathsf{n}} & = & y \\
(\!|\lambda y.t|\!)_{\mathsf{n}} & = & \mathsf{Eta}(\lambda y.(\!|t|\!)_{\mathsf{n}}) \\
(\!|\mu a.c|\!)_{\mathsf{n}} & = & \lambda a.(\!|c|\!)_{\mathsf{n}} \\
(\!|a|\!)_{\mathsf{n}} & = & [\,]a \\
(\!|u :: e|\!)_{\mathsf{n}} & = & [\,](\lambda f.\mathsf{Eta}((\!|u|\!)_{\mathsf{n}})(\lambda z.(\!|e|\!)_{\mathsf{n}}[fz])) \\
(\!|\tilde{\mu} y.c|\!)_{\mathsf{n}} & = & \mathsf{Eta}([\,])(\lambda y.(\!|c|\!)_{\mathsf{n}}) \\
(\!|\langle t|e\rangle|\!)_{\mathsf{n}} & = & (\!|e|\!)_{\mathsf{n}}[\,(\!|t|\!)_{\mathsf{n}}\,]
\end{array}$$

$$\dfrac{\dfrac{\dfrac{\Gamma|e : A \vdash \Delta}{\overline{e}_{\mathsf{n}}[y] : (\overline{\Gamma}_{\mathsf{n}}, y : \overline{A}_{\mathsf{n}} \vdash \overline{\Delta}_{\mathsf{n}})}\ (a)}{(\overline{\Gamma}_{\mathsf{n}})^{\bullet}, y : \neg\neg(A^{\dagger}_{\mathsf{n}})^{\bullet}, (\overline{\Delta}_{\mathsf{n}})^{\bullet^{-}} \vdash (\overline{e}_{\mathsf{n}}[y])^{\bullet} : \bot}\ (b)}{(\!|\Gamma|\!)_{\mathsf{n}}, y : (\!|A|\!)_{\mathsf{n}}, (\!|\Delta|\!)^{-}_{\mathsf{n}} \vdash (\!|e|\!)_{\mathsf{n}}[y] : \bot}\ (c)$$

Justifications:

(a) By the third typing rule in Fig. 6.
(b) By the third typing rule in Fig. 11 and $\overline{A}_{\mathsf{n}} = MA^{\dagger}_{\mathsf{n}}$.
(c) Thanks to compositionality of the extension of type operations to (co-)contexts (see Section 1), we get $(\!|\Gamma|\!)_{\mathsf{n}} = (\overline{\Gamma}_{\mathsf{n}})^{\bullet}$ and $(\!|\Delta|\!)^{-}_{\mathsf{n}} = (\overline{\Delta}_{\mathsf{n}})^{\bullet^{-}}$ from (4) and (5), respectively. Moreover, $\neg\neg(A^{\dagger}_{\mathsf{n}})^{\bullet} = (\overline{A}_{\mathsf{n}})^{\bullet} = (\!|A|\!)_{\mathsf{n}}$, using (4); and $(\overline{e}_{\mathsf{n}}[y])^{\bullet} = (\overline{e}_{\mathsf{n}})^{\bullet}[y^{\bullet}] = (\!|e|\!)_{\mathsf{n}}[y]$ using (1) and (6).

□

Corollary 14 (Strict simulation). 1. If $T \to T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, then $(\!|T|\!)_{\mathsf{n}} \to^{+} (\!|T'|\!)_{\mathsf{n}}$ in $\lambda[\beta\eta]$, where T, T' are either two terms or two commands.
2. If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, then $(\!|\langle t|e\rangle|\!)_{\mathsf{n}} \to^{+} (\!|\langle t|e'\rangle|\!)_{\mathsf{n}}$ in $\lambda[\beta\eta]$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

Proof. The method of proof is to "compose" strict simulation for $\overline{(.)}_{\mathsf{n}}$ (Theorem 2) with strict simulation for $(.)^{\bullet}$ (Proposition 10). More precisely:
1. Let $T \to T'$. From Theorem 2 we get $\overline{T}_{\mathsf{n}} \to^{+} \overline{T'}_{\mathsf{n}}$ in $\lambda\mu_{\mathsf{M}}$, whence we get $(\overline{T}_{\mathsf{n}})^{\bullet} \to^{+} (\overline{T'}_{\mathsf{n}})^{\bullet}$ in $\lambda[\beta\eta]$, by Proposition 10. Now apply the definition of $(\!|T|\!)_{\mathsf{n}}$ in (6).
2. Let $e \to e'$ and $t \in \overline{\lambda}\mu\tilde{\mu}$. Then $\langle t|e\rangle \to \langle t|e'\rangle$. Now apply 1. □

Proposition 15 (Recursive characterization). $(\!|T|\!)_{\mathsf{n}}$ satisfies the equations in Fig. 13.

Fig. 14. Admissible typing rules for cbv CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$\frac{\Gamma \vdash t : A|\Delta}{\Gamma_{\mathsf{v}}^*, (\!|\Delta|\!)_{\mathsf{v}}^- \vdash (\!|t|\!)_{\mathsf{v}} : (\!|A|\!)_{\mathsf{v}}} \qquad \frac{\Gamma \vdash V : A|\Delta}{\Gamma_{\mathsf{v}}^*, (\!|\Delta|\!)_{\mathsf{v}}^- \vdash V^* : A_{\mathsf{v}}^*}$$

$$\frac{\Gamma|e : A \vdash \Delta}{\Gamma_{\mathsf{v}}^*, y : (\!|A|\!)_{\mathsf{v}}, (\!|\Delta|\!)_{\mathsf{v}}^- \vdash (\!|e|\!)_{\mathsf{v}}[y] : \bot} \qquad \frac{c : (\Gamma \vdash \Delta)}{\Gamma_{\mathsf{v}}^*, (\!|\Delta|\!)_{\mathsf{v}}^- \vdash (\!|c|\!)_{\mathsf{v}} : \bot}$$

Proof. For the sake of the proof, take the recursive characterization as the definition of $(\!|T|\!)_{\mathsf{n}}$. Prove (i) $(\overline{t}_{\mathsf{n}})^\bullet = (\!|t|\!)_{\mathsf{n}}$; (ii) $(\overline{c}_{\mathsf{n}})^\bullet = (\!|c|\!)_{\mathsf{n}}$; (iii) $(\overline{e}_{\mathsf{n}})^\bullet = (\!|e|\!)_{\mathsf{n}}$ by simultaneous induction on t, c and e. The case $c = \langle t|e\rangle$ makes use of $(C[t])^\bullet = C^\bullet[t^\bullet]$. □

With this recursive characterization, one could give direct proofs of the typing rules and of strict simulation for $(\!|.|\!)_{\mathsf{n}}$. But such proofs would not be as modular as the ones given above.

Remark 16. Given the recursive characterization, statement 2 in Corollary 14 reads

$$\text{If } e \to e' \text{ in } \overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}, \text{ then } (\!|e|\!)_{\mathsf{n}}[(\!|t|\!)_{\mathsf{n}}] \to^+ (\!|e'|\!)_{\mathsf{n}}[(\!|t|\!)_{\mathsf{n}}] \text{ in } \lambda[\beta\eta] \text{ for any } t \in \overline{\lambda}\mu\tilde{\mu}.$$

This statement can easily be generalized so that $(\!|e|\!)_{\mathsf{n}}[u] \to^+ (\!|e'|\!)_{\mathsf{n}}[u]$ holds for any λ-term u. The case $u = y$ is a particular case of the statement already proved, since $y = (\!|y|\!)_{\mathsf{n}}$. The case of u an arbitrary λ-term then follows from this particular case, since $(\!|e|\!)_{\mathsf{n}}[u] = [u/y]((\!|e|\!)_{\mathsf{n}}[y])$ if y is fresh and since the reduction rules of $\lambda[\beta\eta]$ are closed under substitution.

5.2.2. *Call-by-value CPS translation ($\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}} \longrightarrow \lambda[\beta_{\mathsf{v}}\eta]$)* The translations of types (3) and (4) satisfy in the cbv case:

$$(A \supset B)_{\mathsf{v}}^* = A_{\mathsf{v}}^* \supset (\!|B|\!)_{\mathsf{v}}$$

Corollary 17 (Typing). The typing rules in Fig. 14 are admissible.

Proof. Similar to the cbn case (Corollary 13), "composing" the rules in Fig. 9 for $\overline{(.)}_{\mathsf{v}}$ with those in Fig. 11 for $(.)^\bullet$. This time, we use that $\Gamma_{\mathsf{v}}^* = (\Gamma_{\mathsf{v}}^\dagger)^\bullet$ and $(\!|\Delta|\!)_{\mathsf{v}}^- = (\overline{\Delta}_{\mathsf{v}})^{\bullet -}$ which follow from (3) and (5), respectively. □

Corollary 18 (Strict simulation). 1. If $T \to T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $(\!|T|\!)_{\mathsf{v}} \to^+ (\!|T'|\!)_{\mathsf{v}}$ in $\lambda[\beta_{\mathsf{v}}\eta]$, where T, T' are two terms or two commands.

2. If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $(\!|\langle t|e\rangle|\!)_{\mathsf{v}} \to^+ (\!|\langle t|e'\rangle|\!)_{\mathsf{v}}$ in $\lambda[\beta_{\mathsf{v}}\eta]$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

Proof. Similar to the cbn case (Corollary 14), "composing" strict simulation for $\overline{(.)}_{\mathsf{v}}$ (Theorem 6) with strict simulation for $(.)^\bullet$ (Proposition 10). As observed in Theorem 6, $\overline{(.)}_{\mathsf{v}}$ only requires $\beta_{\mathsf{var}} \subset \beta_{\mathsf{v}}$ and σ_{v} from the target $\lambda\mu_{\mathsf{M}}$ in place of β and σ. So, statement 2 of Proposition 10 applies, and β_{v} instead of β is sufficient in the λ-calculus. □

Proposition 19 (Recursive characterization). $(\!|T|\!)_{\mathsf{v}}$ satisfies the equations in Fig. 15.

Proof. For the sake of the proof, take the recursive characterization as the definition of $(\!|T|\!)_{\mathsf{v}}$ and V^*. One proves: (i) $(V^\dagger)^\bullet = V^*$; (ii) $(\overline{t}_{\mathsf{v}})^\bullet = (\!|t|\!)_{\mathsf{v}}$; (iii) $(\overline{c}_{\mathsf{v}})^\bullet = (\!|c|\!)_{\mathsf{v}}$; (iv) $(\overline{e}_{\mathsf{v}})^\bullet = (\!|e|\!)_{\mathsf{v}}$ by simultaneous induction on V, t, c, and e. □

Fig. 15. Cbv CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$
\begin{array}{rcl@{\qquad}rcl}
\langle\!|V|\!\rangle_{\mathsf{v}} & = & \mathsf{Eta}(V^*) & v^* & = & v \\
\langle\!|\mu a.c|\!\rangle_{\mathsf{v}} & = & \lambda a.\langle\!|c|\!\rangle_{\mathsf{v}} & (\lambda v.t)^* & = & \lambda v.\langle\!|t|\!\rangle_{\mathsf{v}} \\
\langle\!|a|\!\rangle_{\mathsf{v}} & = & [\,]a & \langle\!|\langle t|e\rangle|\!\rangle_{\mathsf{v}} & = & \langle\!|e|\!\rangle_{\mathsf{v}}[\,\langle\!|t|\!\rangle_{\mathsf{v}}\,] \\
\langle\!|u :: e|\!\rangle_{\mathsf{v}} & = & [\,](\lambda f.\langle\!|u|\!\rangle_{\mathsf{v}}(\lambda w.\langle\!|e|\!\rangle_{\mathsf{v}}[fw])) & & & \\
\langle\!|\tilde{\mu}v.c|\!\rangle_{\mathsf{v}} & = & [\,](\lambda v.\langle\!|c|\!\rangle_{\mathsf{v}}) & & &
\end{array}
$$

Remark 20. Given the recursive characterization, statement 2 of Corollary 18 reads:

If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $\langle\!|e|\!\rangle_{\mathsf{v}}[\langle\!|t|\!\rangle_{\mathsf{v}}] \to^+ \langle\!|e'|\!\rangle_{\mathsf{v}}[\langle\!|t|\!\rangle_{\mathsf{v}}]$ in $\lambda[\beta_{\mathsf{v}}\eta]$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

This statement can be generalized so that $\langle\!|e|\!\rangle_{\mathsf{v}}[u] \to^+ \langle\!|e'|\!\rangle_{\mathsf{v}}[u]$ for arbitrary λ-terms u. But the argument used in Remark 16 cannot be repeated with $\langle\!|.|\!\rangle_{\mathsf{v}}$, since $v \neq \langle\!|v|\!\rangle_{\mathsf{v}}$ and since rule β_{v} is not closed under substitution. The generalization requires a new induction; however, since Corollary 18 is already proved, it suffices to prove the generalized statement 2 together with some trivial statement for terms and commands: if $T \to T'$ then true (this amounts to saying that we are only interested in the base reduction rules acting on co-terms – this is only $\eta_{\tilde{\mu}}$ – and the clauses of the term closure that justify a reduction of co-terms). Inductive cases are routine (either by induction hypothesis – in the single case $t :: e \to t :: e'$ due to $e \to e'$ – or by appeal to Corollary 18.1). We only treat the single base case of generalized statement 2.

Case $\eta_{\tilde{\mu}}$: $\tilde{\mu}v.\langle v|e\rangle \to e$, with $v \notin e$, hence $v \notin \langle\!|e|\!\rangle^-_{\mathsf{v}}$.

$$
\begin{array}{rcll}
\langle\!|LHS|\!\rangle_{\mathsf{v}}[u] & = & u(\lambda v.\langle\!|v|\!\rangle_{\mathsf{v}}\,\langle\!|e|\!\rangle^-_{\mathsf{v}}) & \\
& \to_{\beta_{\mathsf{v}}} & u(\lambda v.\langle\!|e|\!\rangle^-_{\mathsf{v}}\,v) & (\langle\!|e|\!\rangle^-_{\mathsf{v}} \text{ is a value}) \\
& \to_{\eta} & u\langle\!|e|\!\rangle^-_{\mathsf{v}} = \langle\!|RHS|\!\rangle_{\mathsf{v}}[u] &
\end{array}
$$

Further analysis of the CPS translations can be found in the appendix to this article.

6. Extension to second-order logic

All the systems considered in this article can straightforwardly be extended to cover second-order logic, and the main simulation results can be extended correspondingly. These, in turn, produce new strong normalization results for classical second-order logic in sequent calculus.

6.1. *Extension of systems*

We present the systems $\overline{\lambda}2\mu\tilde{\mu}$, $\lambda 2\mu_{\mathsf{M}}$ and $\lambda 2$, our second-order versions of $\overline{\lambda}\mu\tilde{\mu}$, $\lambda\mu_{\mathsf{M}}$ and the λ-calculus, respectively. We will not be overly formal here and often only describe the new inductive clauses for some syntactic class. It is understood that all notions in the rules (e. g., the notion of type in the new grammar for terms and the notions of types and terms in the old and the new typing rules) refer to the extended notions and thus that all former definitions (such as substitution and translation) and results are to be interpreted over these larger domains. This *reinterpretation* never adds new cases to the proofs that were just done by structural induction. However, the new grammatical elements have to be treated as such. Only where this leads to non-trivial new cases, it will be mentioned.

The grammar of types is an extension of the grammar of the corresponding first-order system thus:

$$A, B ::= \cdots \mid \forall X.A$$

The type variable X is considered bound in $\forall X.A$. In an obvious way, one can define type substitution $[A/X]B$ denoting the result of capture-avoiding substitution of all free occurrences of type variable X in type B by type A. As an example in $\lambda\mu_{\mathsf{M}}$, $[A/X](MB) = M([A/X]B)$.

The grammar of expressions of $\overline{\lambda}2\mu\tilde{\mu}$ is:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} V & ::= & x \mid \lambda x.t \mid \Lambda X.t & E & ::= & a \mid u :: e \mid A :: e & c & ::= & \langle t|e\rangle \\ t, u & ::= & V \mid \mu a.c & e & ::= & E \mid \tilde{\mu}x.c & & & \end{array}$$

where type variable X is bound in the new term $\Lambda X.t$.

We consider $\Lambda X.t$ as a value for any term t, following the call-by-value $\lambda\mu$-calculus of (Fujita, 1999). Note that, as discussed in (Asada, 2008) for example, regarding Λ-abstractions as values may be incompatible with the second-order η-rule, which is expressed as

$$\Lambda X.tX \to t \qquad (X \notin t),$$

in natural-deduction style. However such an η-rule is not considered in the second-order calculi in this article, and these calculi (or the cbn and cbv fragments in the case of $\overline{\lambda}2\mu\tilde{\mu}$) have good properties as reduction systems, as we will see: subject reduction, strong normalization, and confluence. Moreover, regarding Λ-abstractions as values preserves the duality between values and evaluation contexts, and leads us to a natural extension of the analysis for normal forms in the cbn and the cbv fragments of $\overline{\lambda}\mu\tilde{\mu}$: also in the second-order extensions of the fragments of $\overline{\lambda}\mu\tilde{\mu}$, any normal and typable command is either $\langle x|E\rangle$ or $\langle V|a\rangle$, where normal forms are w. r. t. the rules of the respective first-order system, extended by rule $\beta 2$ given below.

The typing rules for the additional expressions respectively correspond to the right- and the left-introduction rules for the second-order quantifier:

$$\frac{\Gamma \vdash t : B|\Delta}{\Gamma \vdash \Lambda X.t : \forall X.B|\Delta}\ RIntro2 \qquad \frac{\Gamma|e : [A/X]B \vdash \Delta}{\Gamma|A :: e : \forall X.B \vdash \Delta}\ LIntro2$$

provided, in $RIntro2$, X does not occur free in any of the types in Γ, Δ. Notice that, in $LIntro2$, type B is not determined from $[A/X]B$ and A, so this introduces a further source of ambiguity of the type of a given term. Still, in the extended system, we do not attach a type to variable x in $\lambda x.t$, as would be done in Church-style formulations of second-order λ-calculus. We thus obtain *domain-free* systems in the sense of (Barthe and Sørensen, 2000), the style that was also adopted for the formulation of second-order $\lambda\mu$-calculus by (Fujita, 1999; Ikeda and Nakazawa, 2006). ‡‡

In order to formulate the additional reduction rule, we have to assume a notion of type substitution in terms, $[A/X]t$, that will be defined simultaneously with $[A/X]c$ and $[A/X]e$. As

‡‡ The following discussion can also be done for the Church-style systems by defining each translation on terms as a mapping from terms with their type derivations. On the other hand, the Curry-style formulation does not seem suitable, since some evidences (Harper and Lillibridge, 1993; Fujita, 1999; Summers, 2011) show that Curry-style cbv polymorphic calculi with control operators are unsound.

admissible typing rules, we get, for example,

$$\frac{\Gamma \vdash t : B|\Delta}{[A/X]\Gamma \vdash [A/X]t : [A/X]B|[A/X]\Delta}$$

with the intuitive reading of the substituted contexts (following the convention in Section 1).

The extra reduction rule for $\overline{\lambda}2\mu\tilde{\mu}$ is

$$(\beta 2) \qquad \langle \Lambda X.t | A :: e \rangle \to \langle [A/X]t | e \rangle \ .$$

The cbn and cbv fragments of $\overline{\lambda}2\mu\tilde{\mu}$ are defined in the same way as for first-order $\overline{\lambda}\mu\tilde{\mu}$ and are called $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{v}}$, respectively. Thanks to the proviso of rule $RIntro2$, subject reduction also holds for $\overline{\lambda}2\mu\tilde{\mu}$. Since $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{v}}$ have only trivial critical pairs (no new critical pair arises with the extension to the second order), these fragments are confluent.

The monadic calculus $\lambda\mu_{\mathsf{M}}$ is similarly extended as follows. The grammar of expressions of $\lambda 2\mu_{\mathsf{M}}$ is:

$$\begin{array}{rcl@{\qquad\qquad}rcl} V & ::= & x \mid \lambda x.t \mid \Lambda X.t & c & ::= & at \mid \mathsf{bind}(t, x.c) \\ r, s, t, u & ::= & V \mid tu \mid tA \mid \mu a.c \mid \eta t & & & \end{array}$$

and the typing rules for the new terms $\Lambda X.t$ and tA are respectively corresponding to the introduction and the elimination rules for the second-order quantifier:

$$\frac{\Gamma \vdash t : B|\Delta}{\Gamma \vdash \Lambda X.t : \forall X.B|\Delta}\ Intro2 \qquad \frac{\Gamma \vdash t : \forall X.B|\Delta}{\Gamma \vdash tA : [A/X]B|\Delta}\ Elim2$$

provided, in $Intro2$, X does not occur free in any of the types in Γ, Δ.[§§]

The additional reduction rule for $\lambda 2\mu_{\mathsf{M}}$ is the ordinary rule of polymorphic λ-calculus:

$$(\beta 2) \qquad (\Lambda X.t)A \to [A/X]t \ .$$

Similarly to $\overline{\lambda}2\mu\tilde{\mu}$, $\lambda 2\mu_{\mathsf{M}}$ enjoys subject reduction. No new critical pair arises from the extension to the second order, hence also $\lambda 2\mu_{\mathsf{M}}$ is locally confluent.

As the target calculus of the continuations-monad instantiation, we consider the second-order extension of λ-calculus in the domain-free style, which is introduced in (Barthe and Sørensen, 2000) in the framework of the domain-free pure type systems, and denoted here $\lambda 2$. The grammar of expressions is extended by $\Lambda X.t$ and tA, for which we add the typing rules $Intro2$ and $Elim2$ without the Δ parts. The additional reduction rule $\beta 2$ is given in the same form as $\beta 2$ for $\lambda\mu_{\mathsf{M}}$.

It is important to stress again that for $\lambda 2$, we do not consider the second-order η-rule. It is not needed for our simulation results and is therefore left out. In the sequel, we will concentrate on $\lambda 2[\beta, \beta 2, \eta]$.

Although we are not aware of a strong normalization result for $\lambda 2[\beta, \beta 2, \eta]$, this result holds, and it can be proved along the lines of (Barthe and Sørensen, 2000), inheriting strong normalization of Church-style second-order λ-calculus with first-order η-reduction rule.

Proposition 21. $\lambda 2[\beta, \beta 2, \eta]$ enjoys strong normalization.

[§§] The rules $RIntro2$ and $Intro2$ are superficially the same rule, but they range over different systems of types and terms.

Proof. In this proof, we consider the Church-style second-order λ-calculus with β, $\beta 2$ and η (where we mean the Church-style versions of β and η with type superscripts at the variable bindings), the strong normalization of which has already been known.¶¶

The erasure function (see (Geuvers, 1993, Section 4.4.2)) $\lceil\cdot\rceil$ from the Church-style calculus to the domain-free style calculus, that is $\lambda 2$, is defined by $\lceil \lambda x^A.t\rceil = \lambda x.\lceil t\rceil$, and the other cases are homomorphic. Then the following are proved by induction straightforwardly: (i) for any domain-free term t which has a type A in context Γ, there exists a Church-style term t' such that t' has the type A in Γ and $\lceil t'\rceil = t$, and (ii) for any Church-style term t' and any s in domain-free style, if $\lceil t'\rceil \to s$ holds in $\lambda 2[\beta, \beta 2, \eta]$, then there exists a Church-style term s' such that $t' \to s'$ and $\lceil s'\rceil = s$. As a consequence from (i) and (ii), we can translate any potential infinite reduction sequence in $\lambda 2[\beta, \beta 2, \eta]$ from a typable domain-free term into an infinite reduction sequence in the Church-style second-order λ-calculus, starting from a typable term. Such a sequence is impossible, hence our result. □

This completes the presentation of the second-order extensions of the systems in this article. Notice that, unlike for second-order $\lambda\mu$-calculus of Parigot (Parigot, 1997), nothing is added on the classical side to accommodate the second order.

6.2. *Extension of translations*

We will now extend the monadic translations from $\overline{\lambda}\mu\tilde{\mu}$ into $\lambda\mu_{\mathsf{M}}$ to monadic translations from $\overline{\lambda}2\mu\tilde{\mu}$ into $\lambda 2\mu_{\mathsf{M}}$. On types, the definitions are as follows:

$$\begin{array}{lcllcllcl}
\overline{X}_{\mathsf{n}} & = & MX & \overline{(A \supset B)}_{\mathsf{n}} & = & M(\overline{A}_{\mathsf{n}} \supset \overline{B}_{\mathsf{n}}) & \overline{\forall X.B}_{\mathsf{n}} & = & M(\forall X.\overline{B}_{\mathsf{n}}) \\
\overline{X}_{\mathsf{v}} & = & MX & \overline{(A \supset B)}_{\mathsf{v}} & = & M(A^{\dagger}_{\mathsf{v}} \supset \overline{B}_{\mathsf{v}}) & \overline{\forall X.B}_{\mathsf{v}} & = & M(\forall X.\overline{B}_{\mathsf{v}})
\end{array}$$

$A^{\dagger}_{\mathsf{x}}$ is again $\overline{A}_{\mathsf{x}}$ without the outermost application of M. As usual, the letter x ranges over the set $\{\mathsf{n}, \mathsf{v}\}$. In particular, $(\forall X.B)^{\dagger}_{\mathsf{x}} = \forall X.\overline{B}_{\mathsf{x}}$, and the cases for type variables and implication are unchanged. Thus, on the surface, the extension for the second-order universal quantifier is the same for cbn and cbv, but it still recursively relies on the different treatment of implication according to the two paradigms. On the surface, there is no difference between the translations of expressions either: add

$$\overline{(\Lambda X.t)}_{\mathsf{x}} = \eta(\Lambda X.\bar{t}_{\mathsf{x}}) \qquad \overline{(A :: e)}_{\mathsf{x}} = \mathsf{bind}([\,], z.\bar{e}_{\mathsf{x}}[zA^{\dagger}_{\mathsf{x}}])$$

to the cbn translation given in Fig. 7 and to the cbv translation in Fig. 8, respectively. In the cbv case, this agrees with the general rule $\overline{V}_{\mathsf{v}} = \eta V^{\dagger}$ by setting $(\Lambda X.t)^{\dagger} = \Lambda X.\bar{t}_{\mathsf{v}}$ for the value $\Lambda X.t$, which seems to be the only reasonable definition.

We have the following properties of type substitutions that become relevant only now although they could have been stated already in Section 4.

¶¶ Unfortunately, we were not able to find a canonical source to cite. Weak normalization of a second-order system was first established by (Girard, 1971), and strong normalization is not essentially harder (see (Barthe et al., 2001) for very general results about that). Many different published proofs of strong normalization exist for second-order systems with type annotations on all variable occurrences, and a proof for Church-style typing would only inessentially deviate from them. There are also different styles for the treatment of the η-reduction rule. One way is to remark that there is an inductive characterization of SN terms that is indifferent to η-reduction (Matthes, 1999), another is by postponement of η-reduction steps.

Lemma 22. The monadic translations satisfy:
1. $\overline{([A/X]B)}_{\mathsf{x}} = [A^\dagger_{\mathsf{x}}/X]\overline{B}_{\mathsf{x}}$ and $([A/X]B)^\dagger_{\mathsf{x}} = [A^\dagger_{\mathsf{x}}/X]B^\dagger_{\mathsf{x}}$,
2. $\overline{([A/X]T)}_{\mathsf{x}} = [A^\dagger_{\mathsf{x}}/X]\overline{T}_{\mathsf{x}}$ and $([A/X]V)^\dagger = [A^\dagger_{\mathsf{v}}/X]V^\dagger$.

Proof. 1. Simultaneous induction on B.
2. Simultaneous induction on T. □

It is easy to establish that the admissible typing rules in Fig. 6 and in Fig. 9, respectively, still hold for this extension. Key to verify $\overline{(A :: e)}_{\mathsf{x}}$ is that $zA^\dagger_{\mathsf{x}}$ gets type $\overline{([A/X]B)}_{\mathsf{x}}$ in a context with $z : (\forall X.B)^\dagger_{\mathsf{x}}$ – recall that the hole of $\overline{(A :: e)}_{\mathsf{x}}$ is filled with a variable of type $\overline{(\forall X.B)}_{\mathsf{x}}$ in both paradigms.

Theorem 23 (Strict simulation for second-order monadic translation). Let $\mathsf{x} \in \{\mathsf{n}, \mathsf{v}\}$.
1. If $T \to T'$ in $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{x}}$, then $\overline{T}_{\mathsf{x}} \to^+ \overline{T'}_{\mathsf{x}}$ in $\lambda 2\mu_{\mathsf{M}}$, where T, T' are either two terms or two commands.
2. If $e \to e'$ in $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{x}}$, then $\overline{e}_{\mathsf{x}}[\overline{t}_{\mathsf{x}}] \to^+ \overline{e'}_{\mathsf{x}}[\overline{t}_{\mathsf{x}}]$ in $\lambda 2\mu_{\mathsf{M}}$ for any $t \in \overline{\lambda}2\mu\tilde{\mu}$.
The same restrictions of the rules in the target system as in the Theorems 2 and 6 are sufficient.

Proof. The proof has to proceed as the proofs of Theorems 2 and 6. We will show only the case for the new reduction rule. Note, however, that $\overline{(A :: e)}_{\mathsf{x}}$ is a base context, so that simulation of rule π is not hampered in the case of $E = A :: e$.
Case $\beta 2$: This case is proved as follows, using Lemma 22.2 (we omit the index x everywhere):

$$\begin{array}{rcl} \overline{\langle \Lambda X.t | A :: e\rangle} & = & \mathsf{bind}(\eta(\Lambda X.\overline{t}), z.\overline{e}[zA^\dagger]) \\ & \to_{\sigma_{\mathsf{v}}} & \overline{e}[(\Lambda X.\overline{t})A^\dagger] \\ & \to_{\beta 2} & \overline{e}[[A^\dagger/X]\overline{t}] \\ & = & \overline{\langle [A/X]t | e\rangle} \ . \end{array}$$

□

We will now extend the continuations-monad instantiation, and will obtain the CPS translations by composing the continuations-monad instantiation.
The continuations-monad instantiation for types is extended to

$$X^\bullet = X \qquad (A \supset B)^\bullet = A^\bullet \supset B^\bullet \qquad (MA)^\bullet = \neg\neg A^\bullet \qquad (\forall X.A)^\bullet = \forall X.A^\bullet,$$

and, for terms and commands, add

$$(\Lambda X.t)^\bullet = \Lambda X.t^\bullet \qquad (tA)^\bullet = t^\bullet A^\bullet$$

to the translation given in Fig. 10, i. e., every second-order element is translated homomorphically.

Lemma 24. The continuations-monad instantiation satisfies:
1. $([A/X]B)^\bullet = [A^\bullet/X]B^\bullet$,
2. $([A/X]T)^\bullet = [A^\bullet/X]T^\bullet$.

Proof. Induction on B and T, respectively. □

Using this lemma, it is easily checked that the rules in Fig. 11 are admissible, and the extended continuations-monad instantiation strictly preserves the reduction steps.

Proposition 25. 1. If $T \to T'$ in $\lambda 2\mu_{\mathsf{M}}$, then $T^\bullet \to^+ T'^\bullet$ in $\lambda 2[\beta, \beta 2, \eta]$.
2. The same variants as in statement 2 of Proposition 10 hold again.

Proof. 1. We prove only the case $\beta 2$:

$$((\Lambda X.t)A)^\bullet = (\Lambda X.t^\bullet)A^\bullet \to_{\beta 2} [A^\bullet/X]t^\bullet = ([A/X]t)^\bullet.$$

2. It is proved similarly to Proposition 10.2. □

The first part of the above proposition, together with both Proposition 21 and preservation of typability shown in Fig. 11, immediately gives:

Corollary 26. $\lambda 2\mu_{\mathsf{M}}$ enjoys strong normalization.

Strong normalization of the cbn- and cbv-fragments of $\overline{\lambda} 2\mu\tilde{\mu}$ is now obtained.

Corollary 27. $\overline{\lambda} 2\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda} 2\mu\tilde{\mu}_{\mathsf{v}}$ are strongly normalizing.

Proof. Use the previous corollary, Theorem 23, and preservation of typability, shown in Fig. 6 and in Fig. 9, respectively (and verified to hold even for the second-order extension). □

Despite the little work invested into the second-order extension, we obtained a strong result. This is thanks to CPS translation and its monadic generalization that is known to scale up well, while the negative translation is confined to first-order types/formulas (see, e. g., (Parigot, 1997)).

The CPS translations, which are obtained by composing the monadic translations and the continuations-monad instantiation in form of the equations (3), (4) and (6), satisfy the following additional recursive clauses (that could again be used to give a direct recursive definition of the CPS translations):

$$(\forall X.A)^*_{\mathsf{x}} = \forall X.(\!|A|\!)_{\mathsf{x}}$$

$$(\!|\Lambda X.t|\!)_{\mathsf{x}} = \mathsf{Eta}(\Lambda X.(\!|t|\!)_{\mathsf{x}}) \qquad (\!|A :: e|\!)_{\mathsf{x}} = [\,](\lambda z.(\!|e|\!)_{\mathsf{x}}[zA^*_{\mathsf{x}}]),$$

which are common to both cbn and cbv but refer to otherwise quite different translations of types and expressions.

7. Related and future work

In this paper we proved strong normalization for the cbn and cbv fragments of $\overline{\lambda}\mu\tilde{\mu}$ through a syntactic embedding into the λ-calculus, that extends to the second order with domain-free polymorphic λ-calculus as target. The embeddings are CPS translations with the strict simulation property, obtained as the composition of a monadic translation into an intermediate, monadic language, and an instantiation of the formal monad of this language to the continuations monad. The intermediate language is itself new, combining in a non-trivial way syntax for classical logic in the style of $\lambda\mu$-calculus with syntax for a monad as found in Moggi's monadic meta-language.

We now show how this work relates to the literature and can be developed in the future.

7.1. *Related work*

Strong normalization for $\overline{\lambda}\mu\tilde{\mu}$. Strong normalization for full $\overline{\lambda}\mu\tilde{\mu}$ has been shown directly in (Polonovski, 2004), using the reducibility candidates method, and in (David and Nour, 2007), using subtle proof structures that are complex although formalizable in arithmetic. Before that, (Lengrand, 2003) also achieves strong normalization of $\overline{\lambda}\mu\tilde{\mu}$, using an embedding into the sequent calculus for classical logic of (Urban, 2000), which was proven strongly normalizing by the reducibility method. A more syntactic approach is followed in (Rocheteau, 2005), where $\overline{\lambda}\mu\tilde{\mu}$ is mapped into $\lambda\mu$ extended with some sort of contexts and weak simulation is proved. It is not clear from the proof provided in (Rocheteau, 2005) whether strict simulation is actually achieved, and strong normalization for this extension of $\lambda\mu$ is not addressed.

Our proofs of strong normalization are syntactic in nature, combinatorially simple, and although only applicable to the cbn fragment and the cbv fragment, they are conceptually related to questions of semantics of programming languages. In addition, our results for the second-order extensions are new. The extensibility of our method to the second order is in contrast with the direct arithmetical proof of (David and Nour, 2007) that is confined to the first-order fragment. In fact, we are not even aware of systems extending $\overline{\lambda}\mu\tilde{\mu}$ to the second order, besides the one considered in this paper.

Monadic translation. The technique of *monadic translation* to prove strong normalization was applied first to the intuitionistic fragment of cbn $\overline{\lambda}\mu\tilde{\mu}$ (Espírito Santo et al., 2009b). Two sources for this technique are (Hatcliff and Danvy, 1994), where the idea of factorizing CPS translations into monadic translations and monad instantiations is found; and (Sabry and Wadler, 1997), where reduction steps (instead of equations) in the monadic meta-language are given central importance.

In (Espírito Santo et al., 2009b) the intuitionistic fragment of cbn $\overline{\lambda}\mu\tilde{\mu}$ is the domain of a monadic translation into an intuitionistic monadic meta-language (resulting from the enrichment of Moggi's monadic meta-language with some permutative reduction rules); and that monadic translation is afterwards composed with an instantiation to the *identity* monad. Simulation works only if an extra permutative reduction rule, usually named "assoc", is added to the target (λ-calculus), and extension to the second order looks problematic. Composition with an instantiation to the continuations monad produced a CPS translation, but no simulation.

The present paper highly improves these results by: (i) treating classical logic, both the first-order and second-order systems, and both the cbn and the cbv paradigms; (ii) producing a much "tighter" monadic translation (with non-β reduction steps translated in 1-1 fashion); (iii) producing strict simulation through CPS obtained by factorization via a monadic language; (iv) offering the new monadic language required for this factorization.

CPS translations with strict simulation. A key issue of strict simulation is that, not only the reduction steps at the root, but also deeper inside a term, have to be considered. This was sometimes overlooked in the literature, as pointed out with some examples in (Nakazawa and Tatsuta, 2003), and led to "incorrect proofs" of strong normalization by CPS. A CGPS-translation (*continuation-and-garbage-passing* style translation) of $\lambda\mu$ achieving strict simulation in $\lambda[\beta]$ is developed in (Ikeda and Nakazawa, 2006). This style of translation, that passes around both continuations and

"garbage" terms (so that all parts of the source term are kept), can be applied in various settings, and, in particular, extends to second-order $\lambda\mu$. It was not successfully extended to $\overline{\lambda}\mu\tilde{\mu}$ so far, but a simplification of the technique (where only *units* of garbage are passed) delivered strong normalization for the intuitionistic fragment of cbn $\overline{\lambda}\mu\tilde{\mu}$ (Espírito Santo et al., 2007; Espírito Santo et al., 2009a). This result is extensible to the second order (with simulation in domain-free polymorphic λ-calculus).

CPS translations for $\overline{\lambda}\mu\tilde{\mu}$. CPS translations for the cbn and cbv fragments of $\overline{\lambda}\mu\tilde{\mu}$, denoted by $(\_)^{\triangleright}$ and $(\_)^{\triangleleft}$ resp., were already present in (Curien and Herbelin, 2000). Both translations map into λ-calculus enriched with products. The cbn translation generalises a translation in (Hofmann and Streicher, 2002), and the cbv translation is a dualized version of the cbn translation. Although no precise statement of preservation of reduction by the translations is made, the article states that each translation "validates" the respective evaluation discipline, which suggests that the translations map a reduction in $\overline{\lambda}\mu\tilde{\mu}$ into convertible terms in the target. In fact, one verifies that $(\_)^{\triangleleft}$ does not simulate the β-rule for the subtraction connective, it only obtains convertible terms. By duality, the same happens with $(\_)^{\triangleright}$ w. r. t. the β-rule for implication.

In (Herbelin, 2005), there is both a CPS translation $(\_)^{n}$ of the cbn fragment $\overline{\lambda}\mu\tilde{\mu}_T$ and a CPS translation $(\_)^{v}$ of the cbv fragment $\overline{\lambda}\mu\tilde{\mu}_Q$. The fragments $\overline{\lambda}\mu\tilde{\mu}_T$ and $\overline{\lambda}\mu\tilde{\mu}_Q$, which are introduced in (Curien and Herbelin, 2000), are smaller than $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, respectively. The smaller domains allow a slightly simplified definition of the CPS translations. These are obtained by extending the respective maps for the "logic-free" fragment $\mu\tilde{\mu}$, and for this fragment weak simulation is stated. (Recall weak simulation means: each reduction step of the source is mapped into zero or more reduction steps in the target.) Again, one verifies that β-reduction for implication is mapped to β-equality only, this time for both the cbn and cbv translations.

CPS translations for both fragments of $\overline{\lambda}\mu\tilde{\mu}$ are also considered in (Lengrand, 2003). (For the correct definition of the cbn translation one needs to look at the *erratum*.) When compared to our CPS translations, the differences are (besides the fact that Lengrand does not consider the η_μ and $\eta_{\tilde{\mu}}$ rules): (i) Lengrand's cbv translation takes $(A \supset B)^* = \neg B^* \supset \neg A^*$, whereas we have the intuitionistically equivalent $(A \supset B)^* = A^* \supset \neg\neg B^*$, where the double negation results directly from the instantiation to the continuations monad; (ii) Lengrand's cbn translation of commands reads as $\langle\!\langle \langle t|e\rangle \rangle\!\rangle = \langle\!\langle e \rangle\!\rangle \langle\!\langle t \rangle\!\rangle$, which forces co-term translations to a have a type of the form $\neg\langle\!\langle A \rangle\!\rangle$ (*vs.* our $\langle\!\langle \langle t|e\rangle \rangle\!\rangle = \langle\!\langle e \rangle\!\rangle [\langle\!\langle t \rangle\!\rangle]$). The development of the CPS translations in (Lengrand, 2003) was geared by semantic considerations, and the results of "preservation of semantics" by the CPS translations state that when a term reduces to another, their images are β-convertible. Having said this, we were able to verify that Lengrand's cbv translation shares the simulation property of our Corollary 18 and the need for η-reduction in the target, while β-conversions cannot be avoided in the target of Lengrand's cbn translation.

In all, we may say that, rather than strict simulation, the literature on CPS translations for $\overline{\lambda}\mu\tilde{\mu}$ had other preoccupations like duality, simplicity, and semantic considerations. Our CPS translations for $\overline{\lambda}\mu\tilde{\mu}$ with the strict simulation property turn out to be a contribution to the field, in spite of being a by-product of our approach to strong normalization.

7.2. *Future work.*

The meta-language introduced in this paper has good meta-theoretic properties (subject reduction, confluence, strong normalization), and smoothly extends Moggi's meta-language. We think it deserves further study.

One direction is the investigation of subsystems. We are studying a subsystem of values and computations, originating in the natural idea of restricting arrow types to the form $A \supset MB$ (see e.g. (Hatcliff and Danvy, 1994)). This may lead to new connections with polarized formulations of logic, into which embeddings of cbn and cbv calculi have been studied in (Curien and Munch-Maccagnoni, 2010). Moreover, following (Sabry and Wadler, 1997; Dyckhoff and Lengrand, 2007), we have already found that the monadic cbv translation gives an equational correspondence between the system $\overline{\lambda}\mu\tilde{\mu}_Q$ and a subsystem of $\lambda\mu_{\mathsf{M}}$. We would like to identify subsystems of $\overline{\lambda}\mu\tilde{\mu}$, for which our monadic translations and meta-language, even in the cbn case, can produce neater relationships such as reflections.

The use of monadic meta-languages as generic frameworks for the study of CPS translations was started in (Hatcliff and Danvy, 1994). In that article the goal was to make a comprehensive and uniform analysis of extant translations of an intuitionistic source calculus. In the present article, the monadic meta-language has been a vehicle to discover new translations – with a single crucial property (strict simulation) – of a classical source calculus. So, there is plenty of room for using our classical meta-language in more comprehensive studies, along the lines of *op. cit.*, of CPS translations of λ-calculi with control operators. Although such studies are beyond the scope of the present article, we nevertheless already offer some supplementary analysis of our CPS translations in an appendix to this article.

Based on past experience (Espírito Santo et al., 2009a), we believe there should be no major obstacle in extending the present work to higher-order classical logic. Clearly, also positive connectives such as disjunction and the second-order existential quantifier, together with their usual permutative conversions, would be worth considering.

None of the three mappings from $\lambda\mu$ to $\overline{\lambda}\mu\tilde{\mu}$ given in (Curien and Herbelin, 2000) enjoys strict simulation (see also the errata to *op. cit.*). So, strong normalization for $\overline{\lambda}\mu\tilde{\mu}$ is not immediately inherited to $\lambda\mu$. On the other hand, strong normalization of $\lambda\mu$ has been proved with the technique of CGPS translation (Ikeda and Nakazawa, 2006), but this technique has not yet been extended to $\overline{\lambda}\mu\tilde{\mu}$. Here is still some room for systematization of techniques for proving strong normalization.

References

Asada, K. (2008). Extensional universal types for call-by-value. In Ramalingam, G., editor, *The 6th Asian Symposium on Programming Languages and Systems (APLAS 2008)*, volume 5356 of *LNCS*, pages 122–137. Springer Verlag.

Barthe, G., Hatcliff, J., and Sørensen, M. H. (2001). Weak normalization implies strong normalization in a class of non-dependent pure type systems. *Theoretical Computer Science*, 269(1-2):317–361.

Barthe, G. and Sørensen, M. H. (2000). Domain-free pure type systems. *Journal of Functional Programming*, 10(5):417–452.

Curien, P.-L. and Herbelin, H. (2000). The duality of computation. In *Proc. of ICFP 2000*, pages 233–243. IEEE. (*Errata* available from the second author's homepage).

Curien, P.-L. and Munch-Maccagnoni, G. (2010). The duality of computation under focus. In Calude, C. S. and Sassone, V., editors, *IFIP TCS*, volume 323 of *IFIP*, pages 165–181. Springer.

Danvy, O. and Filinski, A. (1992). Representing control: a study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391.

David, R. and Nour, K. (2007). Arithmetical proofs of strong normalization results for symmetric λ-calculi. *Fundamenta Informaticae*, 77(4):489–510.

Dyckhoff, R. and Lengrand, S. (2007). Call-by-value λ-calculus and LJQ. *Journal of Logic and Computation*, 17(6):1109–1134.

Espírito Santo, J., Matthes, R., and Pinto, L. (2007). Continuation-passing style and strong normalisation for intuitionistic sequent calculi. In Ronchi Della Rocca, S., editor, *Proc. of TLCA 2007*, volume 4583 of *LNCS*, pages 133–147. Springer Verlag.

Espírito Santo, J., Matthes, R., and Pinto, L. (2009a). Continuation-passing style and strong normalisation for intuitionistic sequent calculi. *Logical Methods in Computer Science*, 5(2:11).

Espírito Santo, J., Matthes, R., and Pinto, L. (2009b). Monadic translation of intuitionistic sequent calculus. In Berardi, S., Damiani, F., and de'Liguoro, U., editors, *Post-proc. of TYPES 2008*, volume 5497 of *LNCS*, pages 100–116. Springer-Verlag.

Fujita, K. (1999). Explicitly typed $\lambda\mu$-calculus for polymorphism and call-by-value. In Girard, J.-Y., editor, *Proc. of TLCA 1999*, volume 1581 of *LNCS*, pages 162–177. Springer-Verlag.

Geuvers, H. (1993). *Logics and Type Systems*. Proefschrift (PhD thesis), University of Nijmegen.

Girard, J.-Y. (1971). Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In Fenstad, J. E., editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland.

Harper, R. and Lillibridge, M. (1993). Polymorphic type assignment and CPS conversion. *Lisp and Symbolic Computation*, 6(3-4):361–380.

Hatcliff, J. and Danvy, O. (1994). A generic account of continuation-passing styles. In *Proc. of POPL 1994*, pages 458–471. ACM.

Herbelin, H. (2005). *C'est maintenant qu'on calcule – au cœur de la dualité*. Habilitation thesis, University Paris 11.

Hofmann, M. and Streicher, T. (2002). Completeness of continuation models for lambda-mu-calculus. *Information and Computation*, 179(2):332–355.

Ikeda, S. and Nakazawa, K. (2006). Strong normalization proofs by CPS-translations. *Information Processing Letters*, 99:163–170.

Lengrand, S. (2003). Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In Gramlich, B. and Lucas, S., editors, *Post-proc. of WRS 2003*, volume 86 of *ENTCS*. Elsevier. (*Erratum* available from the author's homepage).

Matthes, R. (1999). Monotone fixed-point types and strong normalization. In Gottlob, G., Grandjean, E., and Seyr, K., editors, *CSL, 12th International Workshop, Brno, Czech Republic, August 24–28, 1998, Proceedings*, volume 1584 of *LNCS*, pages 298–312. Springer Verlag.

Moggi, E. (1991). Notions of computation and monads. *Information and Computation*, 93(1):55–92.

Nakazawa, K. and Tatsuta, M. (2003). Strong normalization proof with CPS-translation for second order classical natural deduction. *Journal of Symbolic Logic*, 68(3):851–859. Corrigendum: vol. 68 (2003), no. 4, pp. 1415–1416.

Parigot, M. (1992). $\lambda\mu$-calculus: an algorithmic interpretation of classic natural deduction. In *Int. Conf. Logic Prog. Automated Reasoning*, volume 624 of *LNCS*, pages 190–201. Springer Verlag.

Parigot, M. (1997). Proofs of strong normalisation for second order classical natural deduction. *Journal of Symbolic Logic*, 62(4):1461–1479.

Plotkin, G. (1975). Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159.

Polonovski, E. (2004). Strong normalization of $\overline{\lambda}\mu\tilde{\mu}$ with explicit substitutions. In Walukiewicz, I., editor, *Proc. of FoSSaCS 2004*, volume 2987 of *LNCS*, pages 423–437. Springer-Verlag.

Rocheteau, J. (2005). $\lambda\mu$-calculus and duality: call-by-name and call-by-value. In (Ed.), J. G., editor, *Proc. of RTA 2005*, volume 3467 of *LNCS*, pages 204–218. Springer-Verlag.

Sabry, A. and Wadler, P. (1997). A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941.

Summers, A. J. (2011). Soundness and principal contexts for a shallow polymorphic type system based on classical logic. *Logic Journal of the IGPL*, 19(6):848–896.

Urban, C. (2000). *Classical Logic and Computation*. Phd thesis, University of Cambridge.

Appendix A. Monadic approach to refinement of CPS translations

In this appendix, we show how to use the decomposition of CPS translations via $\lambda\mu_{\mathsf{M}}$ in order to obtain refined translations of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, accumulating the properties enjoyed so far with other desirable properties. The decomposition allows one to discover opportunities of improvement in the components of the CPS translation. The refinements we introduce are actually confined to the continuations-monad instantiation, and so the monadic translations remain an invariant of the approach. The refined CPS translations are still obtained by composition, with properties still obtained by "composition" of the properties of the components, as happened with the CPS translations studied above.

We analyze the CPS translations of Section 5.2, which we refer to as the *main* CPS translations. They are sound w. r. t. typing, decompose via $\lambda\mu_{\mathsf{M}}$, and – most importantly for our purposes – enjoy strict simulation. We give an analysis of other desirable properties: *readiness* (to reduce source redexes) and *indifference* (to evaluation order).

Let us say that a redex in a λ-term in the range of a CPS translation of $\overline{\lambda}\mu\tilde{\mu}$ is a *source redex* if it corresponds to some redex in the source $\overline{\lambda}\mu\tilde{\mu}$-term. A CPS translation has the readiness property (or is *ready*) if a λ-term in its range is always ready to reduce a source redex (if one such exists). CPS translations are not always ready in this sense, because the translation itself may generate "administrative" redexes (Plotkin, 1975; Danvy and Filinski, 1992), whose reduction is required prior to the reduction of source redexes. The well-known indifference property (Plotkin, 1975), in turn, says in particular that the CPS translation achieves (strict) simulation with β_{v} alone in the target.

We show that slight variations of the main CPS translations achieve one of the extra properties we mentioned – on top of the properties already enjoyed by the main translations. None of the variants, however, achieves both extra properties, although a more extensive modification of the main translations, not pursued in this paper, might have collected all properties.

A first refinement defines the *ready* instantiation, where administrative redexes introduced by the main instantiation are reduced "on the fly". After composing the ready instantiation with the monadic translations, one obtains CPS translations enjoying the readiness property. However, the simulation by ready CPS still employs full β and η in the target.

Next, we discuss the defects of the main and ready CPS translations in connection with the need for full $\beta\eta$-reduction in the target; and we introduce two refinements of the main continuations-monad instantiation, dedicated to cbn and cbv, respectively. Through composition

with the respective monadic translations, new *optimized* CPS translations are obtained, which introduce even more administrative reductions than the main translations, but which enjoy strict simulation by β_{v} only.

A.1. *Ready CPS translations and administrative reductions*

Strict simulation requires each reduction step in the source of the CPS translation to correspond to at least one reduction step in the target, but not conversely. It is easy to see that the main CPS translations of Section 5 do not map reduction steps in 1-1 fashion, even though the monadic translations essentially do (Remarks 3 and 7). As one can observe in the proof of Proposition 10, the main instantiation $(.)^{\bullet}$ generates itself reductions of the form

$$(\mathsf{admin}) \qquad (\lambda k.ku)K \to Ku \quad (k \notin u,\, K \text{ a value}) \ .$$

This is a specific instance of β_{v}, and the redex can also be written as $\mathsf{Eta}(u)K$. Through composition with the monadic translations, these reduction steps become administrative reductions of the main CPS translations.

For a variety of reasons, both theoretical and practical, it is desirable to reduce administrative redexes at compile time. This is achievable by several means, for instance by the introduction of the so-called colon-operation (Plotkin, 1975), or by a classification of constructions in the generated code as static or dynamic (Danvy and Filinski, 1992). In this paper we achieve the same goal in a modular way, profiting from the decomposition of CPS translations via $\lambda\mu_{\mathsf{M}}$. Indeed, in our case, it suffices to introduce a slight improvement in the definition of the continuations-monad instantiation.

We define the ready continuations-monad instantiation, denoted $(.)^{\circ}$. In the definition of Fig. 10 we just open an exception in the clause for the instantiation of a command:

$$\begin{array}{rcll} (L[t])^{\circ} & = & L^{\circ}[t^{\circ}] & \text{if } t \neq \eta u \\ (a(\eta u))^{\circ} & = & au^{\circ} & \\ \mathsf{bind}(\eta u, x.c)^{\circ} & = & (\lambda x.c^{\circ})u^{\circ} & \end{array} \tag{10}$$

The remaining clauses of $(.)^{\bullet}$ are not changed. It is immediate that $T^{\bullet} \to^{*}_{\mathsf{admin}} T^{\circ}$.

Define $L^{\circ -}$ as the argument to the hole of L° (hence $L^{\circ} = [\,]L^{\circ -}$). Then, the last two equations of (10) can be uniformly written as

$$(L[\eta u])^{\circ} = L^{\circ -}u^{\circ} \ . \tag{11}$$

We also define

$$\mathsf{bind}(\eta[\,], x.c)^{\circ} = (\lambda x.c^{\circ})[\,] \ , \tag{12}$$

so that the following holds:

$$\text{if } C \text{ is not a base context or } t \neq \eta u, \text{ then } (C[t])^{\circ} = C^{\circ}[t^{\circ}]. \tag{13}$$

Lemma 9 has to be modified as follows.

Lemma 28. The translation satisfies:
1. $[u^{\circ}/x]T^{\circ} \to^{*}_{\mathsf{admin}} ([u/x]T)^{\circ}$, with equality holding if $u \neq \eta r$.
2. $([L/a]T)^{\circ} = [L^{\circ -}/a]T^{\circ}$.

Proof. 1. Induction on T. We just show the cases where administrative steps are generated. These have the form $T = L[t]$, with $[u/x]t = \eta r$ and $t \neq \eta s$, whence $t = x$ and $u = \eta r$.

Case $c = ax$, with $u = \eta r$.

$$\begin{array}{rcl} [u^\circ/x](ax)^\circ & = & [u^\circ/x](xa) \\ & = & u^\circ a \\ & \to_{\mathsf{admin}} & ar^\circ \\ & = & (a(\eta r))^\circ \\ & = & ([u/x](ax))^\circ \end{array}$$

Case $c = \mathsf{bind}(x, y.c')$, with $u = \eta r$.

$$\begin{array}{rcll} [u^\circ/x]\mathsf{bind}(x, y.c')^\circ & = & [u^\circ/x](x(\lambda y.c'^\circ)) & \\ & = & u^\circ(\lambda y.[u^\circ/x]c'^\circ) & \\ & \to_{\mathsf{admin}} & (\lambda y.[u^\circ/x]c'^\circ)r^\circ & \\ & \to^*_{\mathsf{admin}} & (\lambda y.([u/x]c')^\circ)r^\circ & \text{(by IH)} \\ & = & \mathsf{bind}(\eta r, y.[u/x]c')^\circ & \\ & = & ([u/x]\mathsf{bind}(x, y.c'))^\circ & \end{array}$$

2. Induction on T. No administrative steps are generated because we cannot have $[L/a]t = \eta r$, if $t \neq \eta s$. □

In the following, $t \to^= u$ means $t \to u$ or $t = u$, i. e., $\to^=$ is the reflexive closure of $\to$. We will use the symbol ρ to denote reduction rules in the sequel. There is no risk of confusion since we do not study ρ-reduction in our paper.

Proposition 29 (Ready instantiation). Let $T \to_\rho T'$ in $\lambda\mu_{\mathsf{M}}$.

— If $\rho \in \{\beta, \beta_{\mathsf{v}}, \beta_{\mathsf{var}}\}$ then $T^\circ \to_\rho u \to^*_{\mathsf{admin}} T'^\circ$, for some u.
— If $\rho = \eta_\mu$ then $T^\circ \to^=_\eta u \to^*_{\mathsf{admin}} T'^\circ$, for some u.
— If $\rho = \sigma$ then $T^\circ \to_\beta u \to^*_{\mathsf{admin}} T'^\circ$, for some u.
— If $\rho \in \{\sigma_{\mathsf{v}}, \pi\}$ then $T^\circ \to_{\beta_{\mathsf{v}}} T'^\circ$.
— If $\rho = \eta_{\mathsf{bind}}$ then $T^\circ \to_\eta T'^\circ$.

Proof. By induction on $T \to_\rho T'$. For the base cases, we follow the proof of Proposition 10, paying attention to the variants β_{v}, β_{var}, and σ_{v}, and using Lemma 28 instead of Lemma 9. In the base case for β, the call to Lemma 28 may generate administrative reductions. The base case for π is exactly as in Proposition 10. The remaining base cases are as follows:

Case σ: $\mathsf{bind}(\eta s, x.c) \to [s/x]c$.

$$LHS^\circ = (\lambda x.c^\circ)s^\circ \to_\beta [s^\circ/x]c^\circ \to^*_{\mathsf{admin}} RHS^\circ \ ,$$

where the administrative reductions come from Lemma 28.

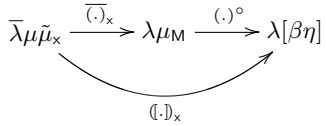
Case σ_{v}: $\mathsf{bind}(\eta V, x.c) \to [V/x]c$.

$$LHS^\circ = (\lambda x.c^\circ)V^\circ \to_{\beta_{\mathsf{v}}} [V^\circ/x]c^\circ = RHS^\circ \ ,$$

where the last equality is by Lemma 28.

Case η_μ: $\mu a.at \to t$, with $a \notin t$. If $t \neq \eta u$, then one η-step is generated, exactly as in

Fig. 16. Ready instantiation and CPS translations



Proposition 10. Otherwise:

$$LHS^{\circ} = \lambda a.au^{\circ} = RHS^{\circ} \ .$$

(Rule η_{μ} may generate administrative steps, but only through one of the inductive cases below.)

Case η_{bind}: $\mathsf{bind}(t, x.a(\eta x)) \to at$. If $t \neq \eta u$, then

$$LHS^{\circ} = t^{\circ}(\lambda x.ax) \to_{\eta} t^{\circ}a = RHS^{\circ} \ .$$

Otherwise

$$LHS^{\circ} = (\lambda x.ax)u^{\circ} \to_{\eta} au^{\circ} = RHS^{\circ} \ .$$

As to inductive cases, all but one is routine. Suppose $L[t_1] \to_{\rho} L[t_2]$, with $t_1 \to_{\rho} t_2$. If $t_2 \neq \eta u_2$, then $t_1 \neq \eta u_1$ and we apply IH. If $t_1 = \eta u_1$ and $t_2 = \eta u_2$, with $u_1 \to_{\rho} u_2$, then we apply again IH. For $\rho \in \{\sigma, \sigma_{\mathsf{v}}, \pi, \eta_{\mathsf{bind}}\}$ there are no more possibilities, and 1-1 simulation holds if $\rho \neq \sigma$. There is a third possibility, only when $\rho \in \{\beta, \beta_{\mathsf{v}}, \beta_{\mathsf{var}}, \eta_{\mu}\}$: $t_1 \neq \eta u_1$, but $t_2 = \eta u_2$. Then:

$$\begin{array}{lcll}
(L[t_1])^{\circ} & = & L^{\circ}[t_1{}^{\circ}] & \text{(since } t_1 \neq \eta u_1\text{)} \\
 & \to^{*} & L^{\circ}[(\eta u_2)^{\circ}] & \text{(by IH, and in the form according to the IH)} \\
 & = & (\lambda k.ku_2{}^{\circ})L^{\circ -} & \text{(by def. of } L^{\circ -} \text{ and } (.)^{\circ}\text{)} \\
 & \to_{\mathsf{admin}} & L^{\circ -}u_2{}^{\circ} & \\
 & = & (L[\eta u_2])^{\circ} & \text{(by def. of } (.)^{\circ}\text{)}
\end{array}$$

□

By composing the monadic translations with the ready continuations-monad instantiation

$$(\![T]\!)_{\mathsf{x}} := (\overline{T}_{\mathsf{x}})^{\circ} \tag{14}$$

we obtain new CPS translations (see Fig. 16). In the cbn case we set

$$(\![E]\!)^{-}_{\mathsf{n}} := (\overline{E}_{\mathsf{n}})^{\circ -} \ . \tag{15}$$

In the cbv case we also set[|||]

$$V^{\star} := (V^{\dagger})^{\circ} \qquad\qquad (\![e]\!)^{-}_{\mathsf{v}} := (\overline{e}_{\mathsf{v}})^{\circ -} \ .$$

Nothing changed at the level of typing w.r.t. the main CPS translations. So $(\![.]\!)_{\mathsf{n}}$ enjoys the typing rules of Fig. 12 and $(\![.]\!)_{\mathsf{v}}$ enjoys the typing rules of Fig. 14.

Suppose $T \to_{\rho} T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ or $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$. By composing the simulation properties of the monadic

[|||] Recall $V^{*} := (V^{\dagger})^{\bullet}$. Typographically, V^{*} (with the multiplication symbol as superscript) may be hard to tell apart from $V^{\star}$ now introduced. Since these symbols appear in different sections, there should be no problem of confusion.

Fig. 17. Ready cbn CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$
\begin{array}{rcl}
(\!|y|\!)_{\mathsf{n}} & = & y \\
(\!|\lambda y.t|\!)_{\mathsf{n}} & = & \mathsf{Eta}(\lambda y.(\!|t|\!)_{\mathsf{n}}) \\
(\!|\mu a.c|\!)_{\mathsf{n}} & = & \lambda a.(\!|c|\!)_{\mathsf{n}} \\
(\!|a|\!)_{\mathsf{n}} & = & [\,]a \\
(\!|u :: e|\!)_{\mathsf{n}} & = & [\,](\lambda f.(\lambda z.(\!|e|\!)_{\mathsf{n}}[fz])(\!|u|\!)_{\mathsf{n}}) \\
(\!|\tilde{\mu} y.c|\!)_{\mathsf{n}} & = & (\lambda y.(\!|c|\!)_{\mathsf{n}})[\,] \\
(\!|\langle t|e\rangle|\!)_{\mathsf{n}} & = & \begin{cases} (\!|E|\!)^{-}_{\mathsf{n}}(\lambda y.(\!|u|\!)_{\mathsf{n}}) & \text{if } e = E \text{ and } t = \lambda y.u \\ (\!|e|\!)_{\mathsf{n}}[(\!|t|\!)_{\mathsf{n}}] & \text{otherwise} \end{cases}
\end{array}
$$

translations and the ready instantiation, it follows that there exists a reduction between the λ-terms $(\!|T|\!)_{\mathsf{x}}$ and $(\!|T'|\!)_{\mathsf{x}}$, consisting of 0, 1 or 2 source reduction steps (the exact number depends on ρ), possibly followed by administrative steps. So, in such reduction no administrative step comes before the reduction steps corresponding to the source reduction $T \to_\rho T'$. We now make these remarks precise.

Definition 30 (Ready reduction). In the λ-calculus:
Cbn case. Given s, s' λ-terms and ρ a reduction rule of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, define $s \overset{\rho}{\Rightarrow}_{\mathsf{n}} s'$ as:

— If $\rho = \beta$ then $s \to_{\beta_{\mathsf{v}}} r \to_{\beta_{\mathsf{var}}} r' \to^*_{\mathsf{admin}} s'$, for some λ-terms r, r'.
— If $\rho = \eta_\mu$ then $s \to^{=}_{\eta} r \to^*_{\mathsf{admin}} s'$, for some λ-term r.
— If $\rho \in \{\sigma, \eta_{\tilde{\mu}}\}$ then $s \to_\beta r \to^*_{\mathsf{admin}} s'$, for some λ-term r.
— If $\rho = \pi_{\mathsf{n}}$ then $s \to_{\beta_{\mathsf{v}}} s'$.

Cbv case. Given s, s' λ-terms and ρ a reduction rule of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, define $s \overset{\rho}{\Rightarrow}_{\mathsf{v}} s'$ as:

— If $\rho = \beta$ then $s \to_{\beta_{\mathsf{v}}} r \to_{\beta_{\mathsf{var}}} r' \to^*_{\mathsf{admin}} s'$, for some λ-terms r, r'.
— If $\rho = \eta_\mu$ then $s \to^{=}_{\beta_{\mathsf{v}}\eta} r \to^*_{\mathsf{admin}} s'$, for some λ-term r.
— If $\rho \in \{\sigma_{\mathsf{v}}, \pi\}$ then $s \to_{\beta_{\mathsf{v}}} s'$.
— If $\rho = \eta_{\tilde{\mu}}$ then $s \to_{\beta_{\mathsf{v}}\eta} s'$.

It may happen that no target step corresponds to a source η_μ-step. Accordingly, in the case $\rho = \eta_\mu$, the following result gives the readiness property in a slightly extended sense.

Corollary 31 (Strict simulation with readiness property). Let $\mathsf{x} \in \{\mathsf{n}, \mathsf{v}\}$.

1 If $T \to_\rho T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{x}}$, where T, T' are two terms or two commands, then $(\!|T|\!)_{\mathsf{x}} \overset{\rho}{\Rightarrow}_{\mathsf{x}} (\!|T'|\!)_{\mathsf{x}}$.
2 If $e \to_\rho e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{x}}$ and $t \in \overline{\lambda}\mu\tilde{\mu}$, then $(\!|\langle t|e\rangle|\!)_{\mathsf{x}} \overset{\rho}{\Rightarrow}_{\mathsf{x}} (\!|\langle t|e'\rangle|\!)_{\mathsf{x}}$.

Proof. As was done in Corollaries 14 and 18, the proof is by "composition" - this time with Proposition 29 - of the simulation theorems of the monadic translations (Theorems 2 and 6), including Remarks 3 and 7. □

We now see that the recursive characterization of $(\!|.|\!)_{\mathsf{n}}$ differs from Fig. 13 in the clauses for $u :: e$, $\tilde{\mu}y.c$ and $\langle t|e\rangle$.

Proposition 32 (Recursive characterization). $(\!|T|\!)_{\mathsf{n}}$ satisfies the equations in Fig. 17 .

Proof. One does the same induction as in the proof of Proposition 15. For the sake of the proof, take the recursive characterization in Fig. 17 as the definition of $(\!|T|\!)_{\mathsf{n}}$ and define $(\!|E|\!)^{-}_{\mathsf{n}}$ as the argument to the hole of $(\!|E|\!)_{\mathsf{n}}$ (hence $(\!|E|\!)_{\mathsf{n}} = [\,](\!|E|\!)^{-}_{\mathsf{n}}$). Then we prove (i) $(\overline{t}_{\mathsf{n}})^{\circ} = (\!|t|\!)_{\mathsf{n}}$; (ii) $(\overline{c}_{\mathsf{n}})^{\circ} = (\!|c|\!)_{\mathsf{n}}$; (iii) $(\overline{E}_{\mathsf{n}})^{\circ -} = (\!|E|\!)^{-}_{\mathsf{n}}$; (iv) $(\overline{e}_{\mathsf{n}})^{\circ} = (\!|e|\!)_{\mathsf{n}}$ by simultaneous induction on t, c, E and e. We show the cases that need update.

Case $e = \tilde{\mu}y.c$.

$$\begin{array}{rcll}
(\overline{(\tilde{\mu}y.c)}_{\mathsf{n}})^{\circ} & = & \mathsf{bind}(\eta[\,], y.\overline{c}_{\mathsf{n}})^{\circ} & \text{(by def. of } \overline{(.)}_{\mathsf{n}}) \\
& = & (\lambda y.(\overline{c}_{\mathsf{n}})^{\circ})[\,] & \text{(by (12))} \\
& = & (\lambda y.(\!|c|\!)_{\mathsf{n}})[\,] & \text{(by IH)} \\
& = & (\!|\tilde{\mu}y.c|\!)_{\mathsf{n}} & \text{(by recursive def.)}
\end{array}$$

Case $E = u :: e$.

$$\begin{array}{rcll}
(\overline{(u :: e)}_{\mathsf{n}})^{\circ -} & = & \mathsf{bind}([\,], f.\mathsf{bind}(\eta\overline{u}_{\mathsf{n}}, z.\overline{e}_{\mathsf{n}}[fz]))^{\circ -} & \text{(by def. of } \overline{(.)}_{\mathsf{n}}) \\
& = & \lambda f.\mathsf{bind}(\eta\overline{u}_{\mathsf{n}}, z.\overline{e}_{\mathsf{n}}[fz])^{\circ} & \text{(by def. of } (.)^{\circ -}) \\
& = & \lambda f.(\lambda z.(\overline{e}_{\mathsf{n}})^{\circ}[fz])\overline{u}_{\mathsf{n}}{}^{\circ} & \text{(by def. of } (.)^{\circ}) \\
& = & \lambda f.(\lambda z.(\!|e|\!)_{\mathsf{n}}[fz])(\!|u|\!)_{\mathsf{n}} & \text{(by IH)} \\
& = & (\!|E|\!)^{-}_{\mathsf{n}} & \text{(by recursive def.)}
\end{array}$$

Case $c = \langle t|e\rangle$. Suppose $e = E$ and $t = \lambda y.u$.

$$\begin{array}{rcll}
(\overline{\langle \lambda y.u | E\rangle}_{\mathsf{n}})^{\circ} & = & (\overline{E}_{\mathsf{n}}[\eta(\lambda y.\overline{u}_{\mathsf{n}})])^{\circ} & \text{(by def. of } \overline{(.)}_{\mathsf{n}}) \\
& = & (\overline{E}_{\mathsf{n}})^{\circ -}(\lambda y.(\overline{u}_{\mathsf{n}})^{\circ}) & \text{(by (11))} \\
& = & (\!|E|\!)^{-}_{\mathsf{n}}(\lambda y.(\!|u|\!)_{\mathsf{n}}) & \text{(by IH)} \\
& = & (\!|\langle \lambda y.u | E\rangle|\!)_{\mathsf{n}} & \text{(by recursive def.)}
\end{array}$$

Otherwise, $\overline{e}_{\mathsf{n}}$ is not a base context or $\overline{t}_{\mathsf{n}} \neq \eta u$. Then:

$$\begin{array}{rcll}
(\overline{\langle t|e\rangle}_{\mathsf{n}})^{\circ} & = & (\overline{e}_{\mathsf{n}}[\overline{t}_{\mathsf{n}}])^{\circ} & \text{(by def. of } \overline{(.)}_{\mathsf{n}}) \\
& = & (\overline{e}_{\mathsf{n}})^{\circ}[(\overline{t}_{\mathsf{n}})^{\circ}] & \text{(by (13))} \\
& = & (\!|e|\!)_{\mathsf{n}}[(\!|t|\!)_{\mathsf{n}}] & \text{(by IH)} \\
& = & (\!|\langle t|e\rangle|\!)_{\mathsf{n}} & \text{(by recursive def.)}
\end{array}$$

□

The recursive characterization of $(\!|.|\!)_{\mathsf{v}}$ differs from Fig. 15 in the clauses for $u :: e$ and $\langle t|e\rangle$.

Proposition 33 (Recursive characterization). $(\!|T|\!)_{\mathsf{v}}$ satisfies the equations in Fig. 18.

Proof. One does the same induction as in proof of Proposition 19. For the sake of the proof, take the recursive characterization in Fig. 18 as the definition of $(\!|T|\!)_{\mathsf{v}}$ and $V^{\star}$ and define $(\!|e|\!)^{-}_{\mathsf{v}}$ as the argument to the hole of $(\!|e|\!)_{\mathsf{v}}$ (hence $(\!|e|\!)_{\mathsf{v}} = [\,](\!|e|\!)^{-}_{\mathsf{v}}$). One proves: (i) $(V^{\dagger})^{\circ} = V^{\star}$; (ii) $(\overline{t}_{\mathsf{v}})^{\circ} = (\!|t|\!)_{\mathsf{v}}$; (iii) $(\overline{c}_{\mathsf{v}})^{\circ} = (\!|c|\!)_{\mathsf{v}}$; (iv) $(\overline{e}_{\mathsf{v}})^{\circ -} = (\!|e|\!)^{-}_{\mathsf{v}}$ and $(\overline{e}_{\mathsf{v}})^{\circ} = (\!|e|\!)_{\mathsf{v}}$ by simultaneous induction on V, t, c, and e.

The cases that need update are $e = u :: e'$ and $c = \langle t|e\rangle$. We just show the latter.

Fig. 18. Ready cbv CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$
\begin{array}{rcl@{\qquad}rcl}
(\!|V|\!)_{\mathsf{v}} & = & \mathsf{Eta}(V^\star) & v^\star & = & v \\
(\!|\mu a.c|\!)_{\mathsf{v}} & = & \lambda a.(\!|c|\!)_{\mathsf{v}} & (\lambda v.t)^\star & = & \lambda v.(\!|t|\!)_{\mathsf{v}} \\
 & & & (\!|\langle t|e\rangle|\!)_{\mathsf{v}} & = & \begin{cases} (\!|e|\!)_{\mathsf{v}}^-(V^\star) & \text{if } t = V \\ (\!|e|\!)_{\mathsf{v}}[(\!|t|\!)_{\mathsf{v}}] & \text{otherwise} \end{cases} \\
(\!|a|\!)_{\mathsf{v}} & = & [\,]a & & & \\
(\!|\tilde{\mu}v.c|\!)_{\mathsf{v}} & = & [\,](\lambda v.(\!|c|\!)_{\mathsf{v}}) & & & \\
(\!|u :: e|\!)_{\mathsf{v}} & = & \begin{cases} [\,](\lambda f.(\lambda w.(\!|e|\!)_{\mathsf{v}}[fw])V^\star) & \text{if } u = V \\ [\,](\lambda f.(\!|u|\!)_{\mathsf{v}}(\lambda w.(\!|e|\!)_{\mathsf{v}}[fw])) & \text{otherwise} \end{cases} & & &
\end{array}
$$

Suppose $t = V$.

$$
\begin{array}{rcll}
(\overline{\langle V|e\rangle}_{\mathsf{v}})^\circ & = & (\overline{e}_{\mathsf{v}}[\eta V^\dagger])^\circ & \text{(by def. of } \overline{(.)}_{\mathsf{v}}) \\
 & = & (\overline{e}_{\mathsf{v}})^{\circ -}((V^\dagger)^\circ) & \text{(by (11))} \\
 & = & (\!|e|\!)_{\mathsf{v}}^-(V^\star) & \text{(by IH)} \\
 & = & (\!|\langle V|e\rangle|\!)_{\mathsf{v}} & \text{(by recursive def.)}
\end{array}
$$

Now let $t \neq V$.

$$
\begin{array}{rcll}
(\overline{\langle t|e\rangle}_{\mathsf{v}})^\circ & = & (\overline{e}_{\mathsf{v}}[\overline{t}_{\mathsf{v}}])^\circ & \text{(by def. of } \overline{(.)}_{\mathsf{v}}) \\
 & = & (\overline{e}_{\mathsf{v}})^\circ[(\overline{t}_{\mathsf{v}})^\circ] & \text{(by (10), as } \overline{t}_{\mathsf{v}} \neq \eta u) \\
 & = & (\!|e|\!)_{\mathsf{v}}[(\!|t|\!)_{\mathsf{v}}] & \text{(by IH)} \\
 & = & (\!|\langle t|e\rangle|\!)_{\mathsf{v}} & \text{(by recursive def.)}
\end{array}
$$

□

Remark 34. Contrary to what was done for the main CPS translations in Remarks 16 and 20, we are not going to generalize statement 2 of Corollary 31, given the non-uniform translation of commands, made explicit in the recursive characterizations.

A.2. *Defects of the obtained CPS translations*

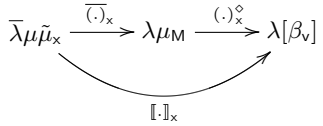
If a CPS translation is also viewed as a computational interpretation and not just as a device for proving strong normalization, it is unfortunate to have η in the target system. Moreover, η-rules are problematic in theories of dependent types, to which we would eventually want to extend our results. These remarks apply to both the main translations of Section 5.2 and the ready translations of the previous section. We concentrate on the former.

Rule η is not just used in Proposition 10.1, but is even needed for soundness of the main cbv CPS translation: as an example, consider

$$c_1 := \langle z|y :: \tilde{\mu}x.\langle x|a\rangle\rangle \to_{\eta_{\tilde{\mu}}} \langle z|y :: a\rangle =: c_2 \ .$$

The β-normal form of $(\!|c_1|\!)_{\mathsf{v}}$ is $zy(\lambda x.ax)$, while the β-normal form of $(\!|c_2|\!)_{\mathsf{v}}$ is zya. Hence, regardless of simulation, not even β-equality is obtained.

Fig. 19. Optimized instantiation and CPS translations



For the cbn translation, the problem is even easier to see: since $\langle\!\langle y \rangle\!\rangle_{\mathsf{n}} = y$ is not a λ-abstraction, the step $\mu a.\langle y|a\rangle \rightarrow_{\eta_\mu} y$ needs η in the CPS translation target for soundness.

For the cbn translation, it is also disappointing that full β is even needed after applying the CPS translation. This is in contrast with cbn CPS for simply-typed lambda-calculus, that was shown to yield terms whose evaluation is even indifferent to the cbn/cbv paradigms (Plotkin, 1975).

Again, rule β beyond β_{v} is not just used in Proposition 10.1, but even needed for soundness of the cbn CPS translation: we can reuse the commands c_1 and c_2 of the example before, but calculate the β_{v}-normal forms of their cbn CPS translations, yielding $z(\lambda f.(\lambda x.xa)(fy))$ and $z(\lambda f.fya)$, respectively. The problem here is that fy is not a value.

Still, as shown next, the composition of the monadic translations with dedicated refined continuations-monad instantiations for cbn and cbv yields CPS translations that provide strict simulation with only $\lambda[\beta_{\mathsf{v}}]$ as target.

A.3. *Optimized CPS translations and indifference property*

We give refinements of the main continuations-monad instantiation, hence of the CPS translations – see the summary in Fig. 19. The goal is to get rid of η-reduction and to restrict to cbv β-reduction in the CPS target (even for the cbn translation).

We start by refining the main continuations-monad instantiation by inserting *η-expansions*

$$\uparrow t := \lambda x.tx \ ,$$

with $x \notin t$. In the cbv case, this is only done for the translation of co-variables, while in the cbn case, the variables are expanded and also the arguments of the unit η of the monad. Relatively to the continuations-monad instantiation $(.)^{\bullet}$, we change the translation at the level of expressions, with different refinements for cbn and cbv. For this reason, we introduce the notations $T^{\diamond}_{\mathsf{n}}$ and $T^{\diamond}_{\mathsf{v}}$. At the level of types, contexts and co-contexts, we keep the translation unchanged. However, for the sake of coherence, we introduce the notations $A^{\diamond}_{\mathsf{n}}$, $\Gamma^{\diamond}_{\mathsf{n}}$, and $\Delta^{\diamond -}_{\mathsf{n}}$, and the cbv variants, even though $A^{\diamond}_{\mathsf{n}} = A^{\bullet} = A^{\diamond}_{\mathsf{v}}$, etc.

A.3.1. *Cbn case* We define a refinement of the continuations-monad instantiation, denoted $T^{\diamond}_{\mathsf{n}}$. The only change, relatively to the definition of $T^{\bullet}$ in Fig. 10, is in the case of a variable which has been $x^{\bullet} = x$, and now is defined as:

$$x^{\diamond}_{\mathsf{n}} = \uparrow x \ ;$$

and in the case of the unit η of the monad, which has been $(\eta t)^{\bullet} = \mathsf{Eta}(t^{\bullet})$, and now becomes

$$(\eta t)^{\diamond}_{\mathsf{n}} = \mathsf{Eta}(\uparrow(t^{\diamond}_{\mathsf{n}})) \ .$$

The first re-definition will be used to get rid of η-reduction in the final target, while the second then allows to restrict β-reduction in the target to β_{v}.

Consequently, we re-define

$$(\mathsf{bind}(\eta[\,], x.c))^{\diamond}_{\mathsf{n}} = \mathsf{Eta}(\uparrow[\,])(\lambda x.c^{\diamond}_{\mathsf{n}}) \ ,$$

in order to maintain

$$(C[t])^{\diamond}_{\mathsf{n}} = C^{\diamond}_{\mathsf{n}}[t^{\diamond}_{\mathsf{n}}] \tag{16}$$

for all cbn contexts C. We also define $L^{\diamond -}_{\mathsf{n}}$ as the unique term such that $L^{\diamond}_{\mathsf{n}} = [\,]L^{\diamond -}_{\mathsf{n}}$. The term $L^{\diamond -}_{\mathsf{n}}$ is a value, like $L^{\bullet -}$.

Obviously, the re-definition invalidates the admissible typing rules of Fig. 11, since $x^{\diamond}_{\mathsf{n}}$ can only be typed by an implication, but $A^{\diamond}_{\mathsf{n}}$ can be a type variable (when $A = X$). Likewise, the argument term t of ηt might be typed by a type variable, and so, the η-expansion of $t^{\diamond}_{\mathsf{n}}$ would not be typable.

A very simple solution is a global exclusion of term typings with atomic types, i. e., type variables. Let us say that a sequent of $\lambda\mu_{\mathsf{M}}$ is *non-atomic* if it is a sequent $c : (\Gamma \vdash \Delta)$ whatsoever; or a sequent $\Gamma \vdash t : A|\Delta$ with A a type that is not a type variable. We call *non-atomic typing system* the subsystem of the typing system for $\lambda\mu_{\mathsf{M}}$ (Fig. 4) that only operates with non-atomic sequents. For emphasis, we write non-atomic sequents (and derivability in the non-atomic typing system) thus:

$$c : (\Gamma \vdash_{\mathsf{nat}} \Delta) \qquad\qquad \Gamma \vdash_{\mathsf{nat}} t : A|\Delta$$

Then, the typing rules of Fig. 11 hold of $(.)^{\diamond}_{\mathsf{n}}$, if the premisses are replaced by non-atomic derivability. On the other hand, the non-atomic system suffices for typing any expression in the range of the cbn monadic translation, as is readily verified, in particular by studying the types that the bound variables in the bind expressions receive, and also by verifying the type of the term fz that appears in the translation of $u :: e$. So, no typing constraint will be observable after forming the CPS translation by composition (see Corollary 37).

Lemma 9 has to be refined as follows.

Lemma 35. The translation satisfies:
1. $[u^{\diamond}_{\mathsf{n}}/x]T^{\diamond}_{\mathsf{n}} \to^{*}_{\beta_{\mathsf{var}}} ([u/x]T)^{\diamond}_{\mathsf{n}}$ for any u which is not an application,
2. $([L/a]T)^{\diamond}_{\mathsf{n}} = [L^{\diamond -}_{\mathsf{n}}/a]T^{\diamond}_{\mathsf{n}}$.

Proof. 1. Induction on T. The case of $T = x$ is the only non-trivial case, and its proof is as follows:

$$[u^{\diamond}_{\mathsf{n}}/x](\uparrow x) = \uparrow u^{\diamond}_{\mathsf{n}} \to_{\beta_{\mathsf{var}}} u^{\diamond}_{\mathsf{n}} = ([u/x]x)^{\diamond}_{\mathsf{n}} \ ,$$

where $u^{\diamond}_{\mathsf{n}}$ is a λ-abstraction, since u is not an application.

2. Induction on T, unchanged from the proof of Lemma 9. □

Proposition 36 (Optimized instantiation for cbn). If $T \to T'$ in $\lambda\mu_{\mathsf{M}}$, where we omit reduction rule η_{bind} and restrict rules β and η_{μ} to the cases β_{n} and $\eta_{\mu\mathsf{n}}$, resp., and σ to the union of σ_{n} and σ_{C}, then $T^{\diamond}_{\mathsf{n}} \to^{+} T'^{\diamond}_{\mathsf{n}}$ in $\lambda[\beta_{\mathsf{v}}]$.

Proof. Induction on $T \to T'$. Remark that, if s is not an application, then $s^{\diamond}_{\mathsf{n}}$ is a λ-abstraction. We study what the proofs of the base cases of Proposition 10.1 yield in the present situation.

Case β: $(\lambda x.t)s \to [s/x]t$. By our restriction to β_{n}, s is not an application.

$$\begin{aligned} LHS^{\diamond}_{\mathsf{n}} &= (\lambda x.t^{\diamond}_{\mathsf{n}})s^{\diamond}_{\mathsf{n}} \\ &\to_{\beta_{\mathsf{v}}} [s^{\diamond}_{\mathsf{n}}/x]t^{\diamond}_{\mathsf{n}} \\ &\to^{*}_{\beta_{\mathsf{var}}} RHS^{\diamond}_{\mathsf{n}} \qquad \text{(by Lemma 35.1)} \end{aligned}$$

Case σ: $\mathsf{bind}(\eta s, x.c) \to [s/x]c$.

$$\begin{aligned} LHS^{\diamond}_{\mathsf{n}} &= (\lambda k.k(\uparrow s^{\diamond}_{\mathsf{n}}))(\lambda x.c^{\diamond}_{\mathsf{n}}) \\ &\to_{\beta_{\mathsf{v}}} (\lambda x.c^{\diamond}_{\mathsf{n}})(\uparrow s^{\diamond}_{\mathsf{n}}) \\ &\to_{\beta_{\mathsf{v}}} [\uparrow s^{\diamond}_{\mathsf{n}}/x]c^{\diamond}_{\mathsf{n}} \\ &\to^{*}_{\beta_{\mathsf{var}}} [s^{\diamond}_{\mathsf{n}}/x]c^{\diamond}_{\mathsf{n}} \qquad \text{(a)} \\ &\to^{*}_{\beta_{\mathsf{v}}} RHS^{\diamond}_{\mathsf{n}} \qquad \text{(b)} \end{aligned}$$

where the last two steps (marked (a) and (b)) are justified by cases, as follows:

Subcase $\sigma = \sigma_{\mathsf{n}}$. Then s is not an application, so $s^{\diamond}_{\mathsf{n}}$ is a λ-abstraction, and $\uparrow s^{\diamond}_{\mathsf{n}} \to_{\beta_{\mathsf{var}}} s^{\diamond}_{\mathsf{n}}$, justifying (a). The step (b) is justified by Lemma 35.1 (which needs the assumption that s is not an application).

Subcase $\sigma = \sigma_{\mathsf{C}}$, and $c = L[x]$ ($x \notin L$). Here, (a) and (b) contain exactly one step each (recall $L^{\diamond -}_{\mathsf{n}}$ is a value):

$$\begin{aligned} [\uparrow s^{\diamond}_{\mathsf{n}}/x]c^{\diamond}_{\mathsf{n}} = (\uparrow(\uparrow s^{\diamond}_{\mathsf{n}}))L^{\diamond -}_{\mathsf{n}} &\to_{\beta_{\mathsf{var}}} (\uparrow s^{\diamond}_{\mathsf{n}})L^{\diamond -}_{\mathsf{n}} \ (= [s^{\diamond}_{\mathsf{n}}/x]c^{\diamond}_{\mathsf{n}}) \\ &\to_{\beta_{\mathsf{v}}} s^{\diamond}_{\mathsf{n}}L^{\diamond -}_{\mathsf{n}} = L^{\diamond}_{\mathsf{n}}[s^{\diamond}_{\mathsf{n}}] = ([s/x]c)^{\diamond}_{\mathsf{n}} \end{aligned}$$

Subcase $\sigma = \sigma_{\mathsf{C}}$, and $c = \mathsf{bind}(\eta x, y.c')$ ($x \notin c'$). Here, again (a) and (b) contain exactly one step each:

$$\begin{aligned} [\uparrow s^{\diamond}_{\mathsf{n}}/x]c^{\diamond}_{\mathsf{n}} = \mathsf{Eta}(\uparrow(\uparrow(\uparrow s^{\diamond}_{\mathsf{n}})))(\lambda y.c'^{\diamond}_{\mathsf{n}}) &\to_{\beta_{\mathsf{var}}} \mathsf{Eta}(\uparrow(\uparrow(s^{\diamond}_{\mathsf{n}})))(\lambda y.c'^{\diamond}_{\mathsf{n}}) \ (= [s^{\diamond}_{\mathsf{n}}/x]c^{\diamond}_{\mathsf{n}}) \\ &\to_{\beta_{\mathsf{var}}} \mathsf{Eta}(\uparrow s^{\diamond}_{\mathsf{n}})(\lambda y.c'^{\diamond}_{\mathsf{n}}) = \mathsf{bind}(\eta s, y.c')^{\diamond}_{\mathsf{n}} = ([s/x]c)^{\diamond}_{\mathsf{n}} \end{aligned}$$

Case π: $L[\mu a.c] \to [L/a]c$.

$$\begin{aligned} LHS^{\diamond}_{\mathsf{n}} &= (\lambda a.c^{\diamond}_{\mathsf{n}})L^{\diamond -}_{\mathsf{n}} \\ &\to_{\beta_{\mathsf{v}}} [L^{\diamond -}_{\mathsf{n}}/a]c^{\diamond}_{\mathsf{n}} \\ &= RHS^{\diamond}_{\mathsf{n}} \qquad \text{(by Lemma 35.2)} \end{aligned}$$

Case η_{μ}: $\mu a.at \to t$, with $a \notin t$. By our restriction to $\eta_{\mu\mathsf{n}}$, $t^{\diamond}_{\mathsf{n}}$ is a λ-abstraction.

$$LHS^{\diamond}_{\mathsf{n}} = \lambda a.t^{\diamond}_{\mathsf{n}}a \to_{\beta_{\mathsf{var}}} t^{\diamond}_{\mathsf{n}} = RHS^{\diamond}_{\mathsf{n}}$$

Note that we omitted rule η_{bind}. □

We now define the cbn optimized CPS translation:

$$[\![T]\!]_{\mathsf{n}} := (\overline{T}_{\mathsf{n}})^{\diamond}_{\mathsf{n}} \ . \tag{17}$$

We also put for an evaluation context E:

$$[\![E]\!]^{-}_{\mathsf{n}} := (\overline{E}_{\mathsf{n}})^{\diamond -}_{\mathsf{n}} \ . \tag{18}$$

Fig. 20. Optimized cbn CPS translation of $\overline{\lambda}\mu\tilde{\mu}$

$$
\begin{array}{rcl}
[\![y]\!]_{\mathsf{n}} & = & \uparrow y \\
[\![\lambda y.t]\!]_{\mathsf{n}} & = & \mathsf{Eta}(\uparrow(\lambda y.[\![t]\!]_{\mathsf{n}})) \\
[\![\mu a.c]\!]_{\mathsf{n}} & = & \lambda a.[\![c]\!]_{\mathsf{n}} \\
[\![a]\!]_{\mathsf{n}} & = & [\,]a \\
[\![u :: e]\!]_{\mathsf{n}} & = & [\,](\lambda f.\mathsf{Eta}(\uparrow[\![u]\!]_{\mathsf{n}})(\lambda z.[\![e]\!]_{\mathsf{n}}[(\uparrow f)(\uparrow z)])) \\
[\![\tilde{\mu}y.c]\!]_{\mathsf{n}} & = & \mathsf{Eta}(\uparrow[\,])(\lambda y.[\![c]\!]_{\mathsf{n}}) \\
[\![\langle t|e\rangle]\!]_{\mathsf{n}} & = & [\![e]\!]_{\mathsf{n}}[\,[\![t]\!]_{\mathsf{n}}\,]
\end{array}
$$

In particular, $[\![E]\!]^{-}_{\mathsf{n}}$ is a value (since any $L^{\diamond -}_{\mathsf{n}}$ is and $\overline{E}_{\mathsf{n}}$ is a base context).

At the level of types, contexts, and co-contexts, $[\![.]\!]_{\mathsf{n}}$ changes nothing relatively to $\langle\!|.|\!\rangle_{\mathsf{n}}$. Nevertheless, we introduce, for the sake of coherence, the notations $[\![A]\!]_{\mathsf{n}}$, $[\![\Gamma]\!]_{\mathsf{n}}$, $[\![\Delta]\!]^{-}_{\mathsf{n}}$, etc.

Corollary 37 (Typing). The typing rules of Fig. 12 are admissible for $[\![.]\!]_{\mathsf{n}}$.

Proof. The proof of Corollary 13 applies again. We "compose" the rules in Fig. 6 for $\overline{(.)}_{\mathsf{n}}$, with conclusions in the non-atomic system, with the rules in Fig. 11, that hold of $(.)^{\diamond}_{\mathsf{n}}$ provided the premisses are in the non-atomic system as well. We clearly need that $[\![\Gamma]\!]_{\mathsf{n}} = (\overline{\Gamma}_{\mathsf{n}})^{\diamond}_{\mathsf{n}}$ and $[\![\Delta]\!]^{-}_{\mathsf{n}} = (\overline{\Delta}_{\mathsf{n}})^{\diamond -}_{\mathsf{n}}$, which are obtained by the observation in Section 1, as usual for these composition lemmas. As before in Section 5.2, we just show the typing rule for co-terms.

$$
\dfrac{\dfrac{\dfrac{\Gamma|e : A \vdash \Delta}{\overline{e}_{\mathsf{n}}[y] : (\overline{\Gamma}_{\mathsf{n}}, y : \overline{A}_{\mathsf{n}} \vdash_{\mathsf{nat}} \overline{\Delta}_{\mathsf{n}})}\,(a)}{(\overline{\Gamma}_{\mathsf{n}})^{\diamond}_{\mathsf{n}}, y : \neg\neg(A^{\dagger}_{\mathsf{n}})^{\diamond}_{\mathsf{n}}, (\overline{\Delta}_{\mathsf{n}})^{\diamond -}_{\mathsf{n}} \vdash (\overline{e}_{\mathsf{n}}[y])^{\diamond}_{\mathsf{n}} : \bot}\,(b)}{[\![\Gamma]\!]_{\mathsf{n}}, y : [\![A]\!]_{\mathsf{n}}, [\![\Delta]\!]^{-}_{\mathsf{n}} \vdash [\![e]\!]_{\mathsf{n}}[\uparrow y] : \bot}\,(c)
$$

Justifications:

(a) By the third typing rule in Fig. 6 with non-atomic conclusions.
(b) By the third typing rule in Fig. 11 with non-atomic premisses.
(c) Since $\neg\neg(A^{\dagger}_{\mathsf{n}})^{\diamond}_{\mathsf{n}} = (\overline{A}_{\mathsf{n}})^{\diamond}_{\mathsf{n}} = [\![A]\!]_{\mathsf{n}}$; and $(\overline{e}_{\mathsf{n}}[y])^{\diamond}_{\mathsf{n}} = (\overline{e}_{\mathsf{n}})^{\diamond}_{\mathsf{n}}[y^{\diamond}_{\mathsf{n}}] = [\![e]\!]_{\mathsf{n}}[\uparrow y]$ using (16) and (17).

Finally, to get rid of the expansion $\uparrow y$, we invoke subject reduction for η-reduction in λ-calculus. □

Corollary 38 (Strict simulation with indifference property).

1. If $T \to T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, then $[\![T]\!]_{\mathsf{n}} \to^{+} [\![T']\!]_{\mathsf{n}}$ in $\lambda[\beta_{\mathsf{v}}]$, where T, T' are either two terms or two commands.
2. If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, then $[\![\langle t|e\rangle]\!]_{\mathsf{n}} \to^{+} [\![\langle t|e'\rangle]\!]_{\mathsf{n}}$ in $\lambda[\beta_{\mathsf{v}}]$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

Proof. As was done in Corollary 14, we "compose" the simulation theorem of the cbn monadic translation $\overline{(.)}_{\mathsf{n}}$ (Theorem 2), this time with Proposition 36. Notice the provisos in this proposition are met due to constraints remarked in the extra statement of Theorem 2 and since $\beta_{\mathsf{var}} \subset \beta_{\mathsf{n}}$. □

Since the optimized cbn CPS translation also preserves typability, we can infer strong normalization of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ from that of $\lambda[\beta_{\mathsf{v}}]$.

Proposition 39 (Recursive characterization). $[\![T]\!]_{\mathsf{n}}$ satisfies the equations in Fig. 20.

Proof. Similar to the proof of Proposition 15. □

In particular, in the proof of the previous proposition, it is established that $[\![E]\!]^-_{\mathsf{n}}$ is the term after the hole $[\,]$ in $[\![E]\!]_{\mathsf{n}}$. Hence $[\![E]\!]_{\mathsf{n}}[t] = t[\![E]\!]^-_{\mathsf{n}}$. This fact is used next.

Remark 40. Given the recursive characterization, statement 2 in Corollary 38 reads

$$\text{If } e \to e' \text{ in } \overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}, \text{ then } [\![e]\!]_{\mathsf{n}}[[\![t]\!]_{\mathsf{n}}] \to^+ [\![e']\!]_{\mathsf{n}}[[\![t]\!]_{\mathsf{n}}] \text{ in } \lambda[\beta_{\mathsf{v}}] \text{ for any } t \in \overline{\lambda}\mu\tilde{\mu}.$$

This statement can be generalized so that $[\![e]\!]_{\mathsf{n}}[u] \to^+ [\![e']\!]_{\mathsf{n}}[u]$ in $\lambda[\beta_{\mathsf{v}}]$, for any λ-term u. As in Remark 20, this is proved by a new simultaneous induction, together with trivial statements for terms and commands. The inductive cases are routine. We only inspect the single base case of statement 2, which is again $\tilde{\mu}y.\langle y|e\rangle \to e$, with $y \notin e$. Using the recursive characterization,

$$[\![LHS]\!]_{\mathsf{n}}[t] = \mathsf{Eta}(\uparrow t)(\lambda y.[\![e]\!]_{\mathsf{n}}[\uparrow y]) \ .$$

If e is an evaluation context E, then $[\![e]\!]^-_{\mathsf{n}}$ is a value and $[\![e]\!]_{\mathsf{n}}[t] = t[\![e]\!]^-_{\mathsf{n}}$; moreover:

$$\begin{array}{rll} \mathsf{Eta}(\uparrow t)(\lambda y.[\![e]\!]_{\mathsf{n}}[\uparrow y]) & \to_{\beta_{\mathsf{v}}} & (\lambda y.[\![e]\!]_{\mathsf{n}}[\uparrow y])(\uparrow t) \\ & \to_{\beta_{\mathsf{v}}} & [\![e]\!]_{\mathsf{n}}[\uparrow(\uparrow t)] = (\uparrow(\uparrow t))[\![e]\!]^-_{\mathsf{n}} \\ & \to_{\beta_{\mathsf{var}}} & (\uparrow t)[\![e]\!]^-_{\mathsf{n}} \\ & \to_{\beta_{\mathsf{v}}} & t[\![e]\!]^-_{\mathsf{n}} = [\![e]\!]_{\mathsf{n}}[t] \\ & = & [\![RHS]\!]_{\mathsf{n}}[t] \end{array}$$

Otherwise $e = \tilde{\mu}z.c$, then $\tilde{\mu}y.\langle y|e\rangle = \tilde{\mu}y.\langle y|\tilde{\mu}z.c\rangle \to_{\sigma_{\mathsf{v}}} \tilde{\mu}y.[y/z]c = \tilde{\mu}z.c$, hence this case can be seen as an inductive case where $\tilde{\mu}y.c_0 \to_{\sigma_{\mathsf{v}}} \tilde{\mu}y.c_0'$, with $c_0 \to_{\sigma_{\mathsf{v}}} c_0'$.

A.3.2. *Cbv case* We refine the continuations-monad instantiation by keeping the definition of $(.)^\bullet$ in Section 5, except for setting

$$(a[\,])^\diamond_{\mathsf{v}} = [\,](\uparrow a) \ .$$

Accordingly, $(a[\,])^{\diamond -}_{\mathsf{v}} = \uparrow a$ and, since $\uparrow a$ is a λ-abstraction, every $L^{\diamond -}_{\mathsf{v}}$ is now a λ-abstraction.

Lemma 9 is modified as follows (where the only change appears in 2.):

Lemma 41. 1. $([u/x]T)^\diamond_{\mathsf{v}} = [u^\diamond_{\mathsf{v}}/x]T^\diamond_{\mathsf{v}}$
2. $[L^{\diamond -}_{\mathsf{v}}/a]T^\diamond_{\mathsf{v}} \to^*_{\beta_{\mathsf{var}}} ([L/a]T)^\diamond_{\mathsf{v}}$

Proof. 1. By induction on T.
2. By induction on T.

Case at.

$$\begin{aligned}
[L_{\mathsf{v}}^{\diamond -}/a](at)_{\mathsf{v}}^{\diamond} &= [L_{\mathsf{v}}^{\diamond -}/a](t_{\mathsf{v}}^{\diamond}(\uparrow a)) \\
&= [L_{\mathsf{v}}^{\diamond -}/a]t_{\mathsf{v}}^{\diamond}(\uparrow L_{\mathsf{v}}^{\diamond -}) \\
&\to_{\beta_{\mathsf{var}}}^{*} ([L/a]t)_{\mathsf{v}}^{\diamond}(\uparrow L_{\mathsf{v}}^{\diamond -}) \qquad \text{(by IH)} \\
&\to_{\beta_{\mathsf{var}}} ([L/a]t)_{\mathsf{v}}^{\diamond} L_{\mathsf{v}}^{\diamond -} \\
&= (L[[L/a]t])_{\mathsf{v}}^{\diamond} \\
&= ([L/a](at))_{\mathsf{v}}^{\diamond}
\end{aligned}$$

The last step is a β_{var}-step since, as has been remarked before, $L_{\mathsf{v}}^{\diamond -}$ is a λ-abstraction. □

Proposition 42 (Optimized instantiation for cbv). If $T \to T'$ in $\lambda\mu_{\mathsf{M}}$, where β, σ, and η_μ are restricted to β_{v}, σ_{v}, and $\eta_{\mu\mathsf{v}}$, respectively, then $T_{\mathsf{v}}^{\diamond} \to^{+} T'^{\diamond}_{\mathsf{v}}$ in $\lambda[\beta_{\mathsf{v}}]$.

Proof. Induction on $T \to T'$. We study what the proofs of the base cases of Proposition 10.1 yield in the present situation.

Case β_{v}: $(\lambda x.t)V \to [V/x]t$. Notice that $V_{\mathsf{v}}^{\diamond}$ is a value.

$$\begin{aligned}
LHS_{\mathsf{v}}^{\diamond} &= (\lambda x.t_{\mathsf{v}}^{\diamond})V_{\mathsf{v}}^{\diamond} \\
&\to_{\beta_{\mathsf{v}}} [V_{\mathsf{v}}^{\diamond}/x]t_{\mathsf{v}}^{\diamond} \\
&= RHS_{\mathsf{v}}^{\diamond} \qquad \text{(by Lemma 41.1)}
\end{aligned}$$

Case σ_{v}: $\mathsf{bind}(\eta V, x.c) \to [V/x]c$. Notice again that $V_{\mathsf{v}}^{\diamond}$ is a value.

$$\begin{aligned}
LHS_{\mathsf{v}}^{\diamond} &= (\lambda k.kV_{\mathsf{v}}^{\diamond})(\lambda x.c_{\mathsf{v}}^{\diamond}) \\
&\to_{\beta_{\mathsf{v}}} (\lambda x.c_{\mathsf{v}}^{\diamond})V_{\mathsf{v}}^{\diamond} \\
&\to_{\beta_{\mathsf{v}}} [V_{\mathsf{v}}^{\diamond}/x]c_{\mathsf{v}}^{\diamond} \\
&= RHS_{\mathsf{v}}^{\diamond} \qquad \text{(by Lemma 41.1)}
\end{aligned}$$

Case π: $L[\mu a.c] \to [L/a]c$.

$$\begin{aligned}
LHS_{\mathsf{v}}^{\diamond} &= (\lambda a.c_{\mathsf{v}}^{\diamond})L_{\mathsf{v}}^{\diamond -} \\
&\to_{\beta_{\mathsf{v}}} [L_{\mathsf{v}}^{\diamond -}/a]c_{\mathsf{v}}^{\diamond} \\
&\to_{\beta_{\mathsf{var}}}^{*} RHS_{\mathsf{v}}^{\diamond} \qquad \text{(by Lemma 41.2)}
\end{aligned}$$

Case $\eta_{\mu\mathsf{v}}$: $\mu a.at \to t$, with $a \notin t$.

$$LHS_{\mathsf{v}}^{\diamond} = \lambda a.t_{\mathsf{v}}^{\diamond}(\uparrow a) \to_{\eta} \uparrow t_{\mathsf{v}}^{\diamond} \to_{\eta} t_{\mathsf{v}}^{\diamond} = RHS_{\mathsf{v}}^{\diamond}$$

However, due to our extra restriction $t = \eta V$, we can do without η-reduction:

$$\begin{aligned}
\lambda a.t_{\mathsf{v}}^{\diamond}(\uparrow a) &= \lambda a.(\lambda k.kV_{\mathsf{v}}^{\diamond})(\uparrow a) \\
&\to_{\beta_{\mathsf{v}}} \lambda a.(\uparrow a)V_{\mathsf{v}}^{\diamond} \\
&\to_{\beta_{\mathsf{v}}} \lambda a.aV_{\mathsf{v}}^{\diamond} = RHS_{\mathsf{v}}^{\diamond}
\end{aligned}$$

In the last reduction, we used that $V_{\mathsf{v}}^{\diamond}$ is a value.

Case η_{bind}: $\mathsf{bind}(t, x.a(\eta x)) \to at$.

$$\begin{aligned} LHS^{\diamond}_{\mathsf{v}} &= t^{\diamond}_{\mathsf{v}}(\lambda x.(\lambda k.kx)(\uparrow a)) \\ &\to_{\beta_{\mathsf{v}}} t^{\diamond}_{\mathsf{v}}(\lambda x.(\uparrow a)x) \\ &\to_{\beta_{\mathsf{var}}} t^{\diamond}_{\mathsf{v}}(\uparrow a) = RHS^{\diamond}_{\mathsf{v}} \end{aligned}$$

□

We compose the cbv monadic translation with this new continuations-monad instantiation

$$[\![T]\!]_{\mathsf{v}} := (\overline{T}_{\mathsf{v}})^{\diamond}_{\mathsf{v}}$$

to obtain the optimized cbv CPS translation. We also define, as usual:

$$[\![e]\!]^{-}_{\mathsf{v}} = (\overline{e}_{\mathsf{v}})^{\diamond -}_{\mathsf{v}}$$

In particular $[\![e]\!]^{-}_{\mathsf{v}}$ is always a λ-abstraction.

At the level of types, contexts, and co-contexts, $[\![.]\!]_{\mathsf{v}}$ changes nothing relatively to the $\langle\!|.|\!\rangle_{\mathsf{v}}$. Nevertheless, we introduce, for the sake of coherence, the notations $[\![A]\!]_{\mathsf{v}}$, $[\![\Gamma]\!]_{\mathsf{v}}$, $[\![\Delta]\!]^{-}_{\mathsf{v}}$, etc.

The rules in Fig. 11 (with $(.)^{\bullet}$ replaced by $(.)^{\diamond}_{\mathsf{v}}$) remain admissible since variables a are assigned types of the form $\neg A^{\diamond}_{\mathsf{v}}$ in all the contexts. Therefore, the rules in Fig. 14 (with $\langle\!|.|\!\rangle_{\mathsf{v}}$ replaced by $[\![.]\!]_{\mathsf{v}}$) remain to hold as well. Thus, the situation is more pleasant than for the cbn case.

Corollary 43 (Strict simulation with indifference property).
1. If $T \to T'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $[\![T]\!]_{\mathsf{v}} \to^{+} [\![T']\!]_{\mathsf{v}}$ in $\lambda[\beta_{\mathsf{v}}]$, where T, T' are either two terms or two commands.
2. If $e \to e'$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $[\![\langle t|e\rangle]\!]_{\mathsf{v}} \to^{+} [\![\langle t|e'\rangle]\!]_{\mathsf{v}}$ in $\lambda[\beta_{\mathsf{v}}]$ for any $t \in \overline{\lambda}\mu\tilde{\mu}$.

Proof. As was done in Corollary 18, we "compose" the simulation theorem of the cbv monadic translation $\overline{(.)}_{\mathsf{v}}$ – Theorem 6, this time with Proposition 42. The restrictions of the rules of $\lambda\mu_{\mathsf{M}}$ in this proposition are met in the target of $\overline{(.)}_{\mathsf{v}}$ (see the extra statement in Theorem 6 and recall $\beta_{\mathsf{var}} \subset \beta_{\mathsf{v}}$). □

As in the cbn case, the optimized cbv CPS translation preserves typability, so we can infer strong normalization of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ from that of $\lambda[\beta_{\mathsf{v}}]$.

Proposition 44 (Recursive characterization). The recursive characterization of $[\![.]\!]_{\mathsf{v}}$, is obtained by changing the clause for co-variables in Fig. 15 as follows:

$$[\![a]\!]_{\mathsf{v}} = [\,](\uparrow a) \ .$$

Proof. Just adapt the case $e = a$ in the induction that proved Proposition 19. □

In particular, the proof of the previous proposition established that $[\![e]\!]^{-}_{\mathsf{v}}$ is the term after the hole of $[\![e]\!]_{\mathsf{v}}$. Hence $[\![e]\!]_{\mathsf{v}}[t] = t[\![e]\!]^{-}_{\mathsf{v}}$, a fact that is used next. Another fact used next is that $[\![e]\!]^{-}_{\mathsf{v}}$ is always a λ-abstraction, which fails for $\langle\!|e|\!\rangle^{-}_{\mathsf{v}}$ if e is a co-variable.

Remark 45. Given the recursive characterization, statement 2 of Corollary 43 reads

$$\text{If } e \to e' \text{ in } \overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}, \text{ then } [\![e]\!]_{\mathsf{v}}[[\![t]\!]_{\mathsf{v}}] \to^{+} [\![e']\!]_{\mathsf{v}}[[\![t]\!]_{\mathsf{v}}] \text{ in } \lambda[\beta_{\mathsf{v}}] \text{ for any } t \in \overline{\lambda}\mu\tilde{\mu}.$$

This statement can be generalized so that $[\![e]\!]_{\mathsf{v}}[u] \to^+ [\![e']\!]_{\mathsf{v}}[u]$ in $\lambda[\beta_{\mathsf{v}}]$, for any λ-term u. Again, only the case of base $\eta_{\tilde{\mu}}$-reduction requires fresh verification. We have to show $[\![\tilde{\mu}x.\langle x|e\rangle]\!]_{\mathsf{v}}[t] \to^+ [\![e]\!]_{\mathsf{v}}[t]$ in $\lambda[\beta_{\mathsf{v}}]$ (for $x \notin e$).

$$\begin{array}{rcll} LHS & = & t(\lambda x.[\![x]\!]_{\mathsf{v}}\,[\![e]\!]_{\mathsf{v}}^{-}) & \\ & \to_{\beta_{\mathsf{v}}} & t(\lambda x.[\![e]\!]_{\mathsf{v}}^{-}\,x) & ([\![e]\!]_{\mathsf{v}}^{-} \text{ is a value}) \\ & \to_{\beta_{\mathsf{var}}} & t[\![e]\!]_{\mathsf{v}}^{-} & ([\![e]\!]_{\mathsf{v}}^{-} \text{ is a } \lambda\text{-abstraction}) \\ & = & RHS & \end{array}$$