Monadic translation of classical sequent calculus

JOSÉ ESPÍRITO SANTO, RALPH MATTHES, KOJI NAKAZAWA and LUÍS PINTO

Mathematical Structures in Computer Science / *FirstView* Article / January 2013, pp 1 - 52
DOI: 10.1017/S0960129512000436, Published online: 18 January 2013

Math. Struct. in Comp. Science: page 1 of 52. © Cambridge University Press 2013
doi:10.1017/S0960129512000436

Monadic translation of classical sequent calculus

JOSÉ ESPÍRITO SANTO[†], RALPH MATTHES[‡],
KOJI NAKAZAWA[§] and LUÍS PINTO[†]

[†]*Centro de Matemática, Universidade do Minho, Portugal*
Email: {jes,luis}@math.uminho.pt
[‡]*I.R.I.T. (C.N.R.S. and University of Toulouse), France*
Email: matthes@irit.fr
[§]*Graduate School of Informatics, Kyoto University, Japan*
Email: knak@kuis.kyoto-u.ac.jp

Received 21 December 2010; revised 8 May 2012

We study monadic translations of the call-by-name (cbn) and call-by-value (cbv) fragments of the classical sequent calculus $\overline{\lambda}\mu\tilde{\mu}$ due to Curien and Herbelin, and give modular and syntactic proofs of strong normalisation. The target of the translations is a new meta-language for classical logic, named monadic $\lambda\mu$. This language 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 a rule expressing the interaction between monadic binding and μ-abstraction.

Our monadic translations produce very tight simulations of the respective fragments of $\overline{\lambda}\mu\tilde{\mu}$ within monadic $\lambda\mu$, with reduction steps of $\overline{\lambda}\mu\tilde{\mu}$ being translated in a 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 $\lambda\mu$ within simply typed λ-calculus with β- and η-reduction. Through strict simulation, the strong normalisation of simply typed λ-calculus is inherited by monadic $\lambda\mu$, and then by cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$, thus reproving strong normalisation in an elementary syntactical way 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 the target, giving new strong normalisation 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 paper, 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.

[†] José Espírito Santo and Luís Pinto 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.
[‡] Ralph Matthes thanks the Centro de Matemática of Universidade do Minho for funding research visits to José Espírito Santo and Luís Pinto.
[§] Koji Nakazawa has been supported by the Kyoto University Foundation for an extended research visit to Ralph Matthes.

1. Introduction

The $\bar{\lambda}\mu\tilde{\mu}$ calculus (Curien and Herbelin 2000) is a way to present classical sequent calculus in an operationalised form as an extension of λ-calculus. Such a 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 thus important to consider confluent fragments (where all reduction sequences lead to the same result, if any). Non-confluence of $\bar{\lambda}\mu\tilde{\mu}$ is due to a single critical pair, which can be resolved in two ways, which determine the call-by-name (cbn) and call-by-value (cbv) fragments of $\bar{\lambda}\mu\tilde{\mu}$ (Curien and Herbelin 2000). In addition, it is desirable that, within each fragment, all reduction sequences starting from typed expressions do indeed produce a result (that is, they end in a term that cannot be reduced any further). This is the strong normalisation property.

In this paper we study embeddings of the cbn and cbv fragments of $\bar{\lambda}\mu\tilde{\mu}$ 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 normalisation for the mentioned fragments of $\bar{\lambda}\mu\tilde{\mu}$. 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 normalisation in the source is reduced to strong normalisation in the target, where it holds and has been proved in many different ways.

The interest in this new proof lies not only in its elementary character, but also in its concepts. The CPS compilations that simulate the fragments of $\bar{\lambda}\mu\tilde{\mu}$ are factored into a *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, which stands for a *monad* and is an ingredient in the categorical semantics originally put forward by Moggi. The monadic translation is thus parameterised by M. Here we will only consider the instantiation of M to the so-called *continuations monad*. This corresponds to interpreting M as *double negation*, which is a type transformation of simple types that 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 paper. This target is a reworking of Parigot's $\lambda\mu$-calculus, which we call *monadic $\lambda\mu$-calculus*, and denote by $\lambda\mu_{\mathsf{M}}$. 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, that is, 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.

Unlike the original monadic meta-language (Moggi 1991), but in line with Hatcliff and Danvy (1994) and Sabry and Wadler (1997), our classical meta-language is equipped with reduction rules. The cbn and cbv monadic translations produce strict simulations of the

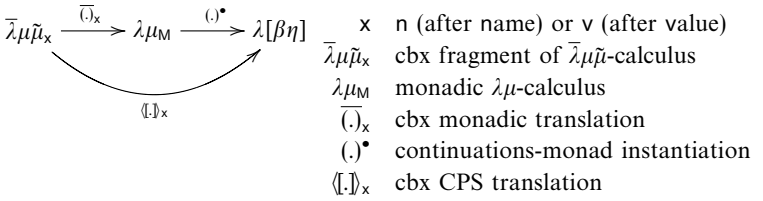

Fig. 1. Overview

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 normalising: this can be seen either by observing that $\lambda\mu_{\mathsf{M}}$ has previously been proved to be strongly normalising 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 Figure 1 for an overview of the systems and translations.

All the systems considered in this paper can be straightforwardly 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 normalisation in cases where no arithmetic proofs are possible. This is because we are content with a simulation result, thereby inheriting the strong normalisation property from second-order λ-calculus, which is widely known and has been established using a multitude of distinct proof strategies.

In an appendix to the paper, we describe some technical refinements to the CPS translations of cbn and cbv $\overline{\lambda}\mu\tilde{\mu}$ we have given. The questions of 'administrative reductions' and the indifference property (Plotkin 1975) are analysed. We propose two variants of our CPS translations: one that performs administrative reductions at compile time and another enjoying strict simulation by β_{v} only. The main point is that the modular approach of having decomposition through $\lambda\mu_{\mathsf{M}}$ is retained since the refinements are confined to the continuations-monad instantiation, and the refined CPS translations are obtained by composition.

1.1. *Structure of the paper*

Section 2 provides a brief summary of the relevant features of $\overline{\lambda}\mu\tilde{\mu}$. 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 normalisation 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.

1.2. *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, that is, the uniqueness of declaration of (co-)variables, which is implicitly enforced in the sense that, for example, when writing the enlarged context $\Gamma, x : A$, we assume that x is not declared in Γ. We write Γ, Γ' for the union of the contexts Γ and Γ', again implicitly assuming that the declarations do not have any variables in common. If F is some type operation, 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. The same holds trivially for co-contexts Δ.

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

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

Plotkin's cbv restriction

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

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

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

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, that is, 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 some 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 $\bar{\lambda}\mu\tilde{\mu}$-calculus (Curien and Herbelin 2000).

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

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

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

$$\frac{\Gamma \vdash t : A | \Delta \quad \Gamma | e : A \vdash \Delta}{\langle t | e \rangle : (\Gamma \vdash \Delta)}$$

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

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

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

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

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

Expressions are ranged over by T, T'. Variables (respectively, co-variables) are ranged over by v, w, x, y, z (respectively, a, b). We assume disjoint countably infinite supplies of variables, and they may also be denoted 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 Figure 2.

There are 6 substitution operations altogether:

$$[t/x]c \qquad [t/x]u \qquad [t/x]e \qquad [e/a]c \qquad [e/a]u \qquad [e/a]e'\,.$$

We consider the 5 reduction rules in Figure 3, where we reuse the name β of λ-calculus (rule names are considered relative to some term system). These are the reductions considered 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), the π rule is replaced in the cbn fragment $\bar{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ of $\bar{\lambda}\mu\tilde{\mu}$ by its restriction

to evaluation contexts:

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

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:

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

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

3.1. *The calculus*

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

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

Note that a bind, and 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 in C with t, and is defined as at (respectively, $\mathsf{bind}(t, x.c)$ or $\mathsf{bind}(\eta t, x.c)$) if C is $a[\,]$ (respectively, $\mathsf{bind}([\,], x.c)$ or $\mathsf{bind}(\eta[\,], x.c)$). Note 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

[†] 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' is related to the monadic translations to be introduced later in the paper. The form $\mathsf{bind}(\eta[\,], x.c)$ is used in the cbn translation only.

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

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

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

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

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

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. Note 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, that is, to define the cbn context $[C/a]C'$ in the obvious way. In particular,

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

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

3.1.2. *Typing rules.* Types are given by

$$A, B \ ::= \ X \,|\, A \supset B \,|\, MA\,.$$

Types of the form MA are called *monadic types*. The typing rules are given in Figure 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 declarations $a : MA$, and thus with monadic types. Apart from the last two rules, these are the rules for Parigot's $\lambda\mu$, but 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, should be contrasted with the usual rule in the framework of λ-calculus:

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

[†] If the restriction on the type of co-variables were not imposed, and accordingly the typing rules *Pass* and *Act* could act with any type instead of just monadic types, we could build the term $\mu a.\mathsf{bind}(t, x.ax)$ of type A from any term t of type MA. This would represent a trivialisation of the system as a monadic language. We are grateful to Dan Licata for pointing this out.

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

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

Instead of a term t of monadic type MB, we now have a command c where no type can be assigned. Still, we 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 obtain 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 $x \notin C$:

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

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

3.1.3. *Reduction rules.* The base reduction rules of $\lambda\mu_{\mathsf{M}}$ are shown in Figure 5. Thus, as in the $\overline{\lambda}\mu\tilde{\mu}$ case, rule π causes substitution for co-variables, while 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:

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

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 the 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. Note 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 it is typed with a monadic type, which is required for the principal (first) argument of bind in order to type the whole expression.

A particular case of π_{bind} is

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

for $a \notin r, t$. This is an 'associativity' rule, and it is 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:

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

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

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

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

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

Note that the cbn versions properly contain the respective cbv versions. It is obvious that none of the seven restricted versions of the $\lambda\mu_{\mathsf{M}}$ reduction rules are closed under term substitution, that is, we do *not* have $T \to_\rho T'$ 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 Figure 5 satisfy closure under term substitution, as well as a final restriction of σ that we will also consider:

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

It goes without saying that $\lambda\mu_{\mathsf{M}}$ enjoys subject reduction: types of terms and commands are preserved under $\to$ (the term closure of the base reduction rules that should more precisely be called the 'closure under all expression constructors'). There are five critical pairs between the reduction rules, and it is not surprising that they all appear in connection with either the η_μ or η_{bind} rule. 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 to both

(i) $[\mathsf{bind}([\,], x.b(\eta x))/a]c$
(ii) $b(\mu a.c)$,

but these terms both reduce to $[b/a]c$: (ii) by one π_{covar} step and (i) with zero or more η_{bind} steps (a result that can be proved together with its analogue for terms). Since all critical pairs are joinable, $\lambda\mu_{\mathsf{M}}$ enjoys local confluence.

3.2. *The 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, the only difference being 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 right-hand side. The typing and reduction rules of the intuitionistic fragment are the expected restrictions to the typing and reduction rules of $\lambda\mu_{\mathsf{M}}$, so we will not spell them out here. Instead, we will develop an isomorphic variant of the intuitionistic fragment, where $*$ and the μ-binder are completely 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 judgment are $\Gamma \vdash t : A$, and $c : (\Gamma \vdash MA)$. Note that these simplified judgment forms reflect both the restriction to only one formula on the right-hand sides and the complete absence of co-variables.

The typing rules *Pass* and *Act* are now

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

We omit the other typing rules.

The β and σ reduction rules are the same as for $\lambda\mu_{\mathsf{M}}$, and the other rules are now

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

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

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

We will now 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

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

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 $\pi_{\ulcorner\cdot\urcorner}$ and $\eta_{\{\cdot\}}$ become identities, just as in the case $c = \ulcorner t \urcorner$ of π_{bind}, and we are left with the following rules:

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

where @ is the same 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 between these rules and the usual reduction rules for Moggi's monadic meta-language, as in Hatcliff and Danvy (1994) and Sabry and Wadler (1997), is rule π_{bind}. There the rule used is

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

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

4. Monadic translation

In this section, we give two translations of $\overline{\lambda}\mu\tilde{\mu}$ into $\lambda\mu_{\mathsf{M}}$. The first, which is denoted by $\overline{(.)}_{\mathsf{n}}$, allows us 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, which is denoted by $\overline{(.)}_{\mathsf{v}}$, gives strict simulation of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ within $\lambda\mu_{\mathsf{M}}$. Hence, they are monadic cbn and cbv translations, respectively, of $\overline{\lambda}\mu\tilde{\mu}$.

4.1. *Call-by-name translation*

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

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

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

We use $A^{\dagger}$ to denote the type $\overline{A}$ without the outermost application of M, that is, we have

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

so $\overline{A} = MA^{\dagger}$.

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

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

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

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

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

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

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

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

Lemma 4.1. The translation satisfies:

(1) $\overline{[t/y]T} = [\bar{t}/y]\overline{T}$.
(2) $\overline{[e/a]T} = [\bar{e}/a]\overline{T}$.

Proof.

(1) The proof is by induction on T.
(2) The proof is by induction on T:

— Case $T = a$:

$$\begin{aligned}\overline{[e/a]a} &= \overline{e}\\ &= [\overline{e}/a](a[\,])\\ &= [\overline{e}/a]\overline{a}\,.\end{aligned}$$

The second equation follows by definition of structural substitution on the context $a[\,]$. □

Theorem 4.2 (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. Also, we do not need full σ, just the restrictions σ_{n} and σ_{C}.

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

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

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

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

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

where the marked equality comes from Lemma 4.1 (1)[†].

[†] Lemma 4.1 (1) refers to item (1) in the statement of Lemma 4.1 – this kind of cross-reference will be used throughout the paper.

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

$$\begin{array}{rcl} \overline{LHS} & = & \overline{E}[\mu a.\overline{c}] \\ & \to_{\pi} & [\overline{E}/a]\overline{c} \\ & \stackrel{*}{=} & \overline{[E/a]c} \\ & = & \overline{RHS}, \end{array}$$

where the marked equality comes from Lemma 4.1 (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 that $y \notin \overline{e}$.

— Case η_{μ}: $\mu a.\langle t | a \rangle \to t$, with $a \notin t$:
Hence, we also have $a \notin \overline{t}$, so

$$\begin{array}{rcl} \overline{LHS} & = & \mu a.a\overline{t} \\ & \to_{\eta_{\mu \mathsf{n}}} & \overline{t} \\ & = & \overline{RHS}. \end{array}$$

□

Remark 4.3. Note the structural 'tightness' of the simulation. Every reduction step of the form σ, π_{n}, η_{μ} or $\eta_{\tilde{\mu}}$ in $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ corresponds to exactly one step in $\lambda\mu_{\mathsf{M}}$ of the form σ_{n}, π, $\eta_{\mu \mathsf{n}}$ or σ_{C}, respectively, and only the β steps of $\overline{\lambda}\mu\tilde{\mu}$ are decomposed into two steps of $\lambda\mu_{\mathsf{M}}$, and these are restricted to the σ_{v} and β_{var} forms[†].

Remark 4.4. 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, when T is a co-term);
(ii) if co-term e is a subexpression of T, then, for some t', we have $\overline{e}[t']$ is a subexpression of $\overline{T}$ (and of $\overline{T}[t]$, for any t, when T is a co-term).

4.2. *Call-by-value translation*

In this section we define and study the translation $\overline{(.)}_{\mathsf{v}}$. To simplify the notation, the subscript $_{\mathsf{v}}$ will be omitted throughout this section.

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

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

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

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

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

The cbv translation of types is the same as the cbn translation, apart from implication:

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

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

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

It can be seen 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 within $\overline{\lambda}\mu\tilde{\mu}$. That part, namely the rules for types, values and typing of terms uses the standard ones in monadic translations. The new elements are also, for the most part, treated in the same way: μ-abstraction is translated homomorphically; commands are translated 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 the cbn translation of $u :: e$ still remains a base context (this is clearly already dictated by typing considerations since the type translation leaves no room once values have been treated in the standard way).

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

Lemma 4.5. The translation satisfies:

(1) $\overline{[V/v]T} = [V^\dagger/v]\overline{T}$.
(2) $\overline{[e/a]T} = [\overline{e}/a]\overline{T}$.

Proof. The proof is by induction on T. □

Theorem 4.6 (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 $\bar{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, then $\overline{e}[\overline{t}] \to^{+} \overline{e'}[\overline{t}]$ in $\lambda\mu_{\mathsf{M}}$ for any $t \in \bar{\lambda}\mu\tilde{\mu}$.
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 obvious since t is a subterm of $\overline{e}[t]$, so we will just show the base cases:

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

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

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

$$\begin{array}{rcl} \overline{LHS} & = & \overline{e}[\mu a.\overline{c}] \\ & \to_{\pi} & [\overline{e}/a]\overline{c} \\ & = & \overline{RHS}\,, \end{array}$$

where the last equality comes from Lemma 4.5 (2).

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

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

where the last equality comes from Lemma 4.5 (1).

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

$$\begin{array}{rcl} \overline{LHS} & = & \mu a.a\overline{t} \\ & \to_{\eta_\mu} & \overline{RHS}\,. \end{array}$$

We now argue that the restriction of η_μ to $\eta_{\mu\mathsf{v}}$ is sufficient in the target system. If t is a value V, then $\overline{t} = \eta V^{\dagger}$, and $V^{\dagger}$ is again a value, 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 $\overline{LHS}[u] = \mathsf{bind}(u, v.\bar{e}[\eta v])$. If e is a co-variable a, we have

$$\begin{array}{rcl} \mathsf{bind}(u, v.\bar{e}[\eta v]) & = & \mathsf{bind}(u, v.a(\eta v)) \\ & \to_{\eta_{\mathsf{bind}}} & au \\ & = & \overline{RHS}[u]\,. \end{array}$$

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

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

□

Remark 4.7. Rule η_{bind} is now required. As in the cbn case, the simulation is quite 'tight'. Every reduction step of the form σ_{v}, π, η_{μ} or $\eta_{\tilde{\mu}}$ in $\bar{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ corresponds to exactly one step in $\lambda\mu_{\mathsf{M}}$ of the form σ_{v}, π, $\eta_{\mu\mathsf{v}}$ or π, or η_{bind} or σ_{v}, respectively, and, again, only the β steps of $\bar{\lambda}\mu\tilde{\mu}$ are decomposed into two steps of $\lambda\mu_{\mathsf{M}}$ with the restricted forms σ_{v} and β_{var}.

Remark 4.8. The cbv monadic translation satisfies the same Subexpression Property as the cbn translation.

5. Continuations-monad instantiation

The monad operation M of $\lambda\mu_{\mathsf{M}}$ can be instantiated to be double negation, which 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 normalisation for cbn and cbv $\bar{\lambda}\mu\tilde{\mu}$ will follow. Composition of instantiation with the monadic translations will yield cbn and cbv CPS translations of $\bar{\lambda}\mu\tilde{\mu}$, whose recursive definition is given at the end of the present section.

5.1. *Instantiation and strong normalisation*

We define a translation from $\lambda\mu_{\mathsf{M}}$ into $\lambda[\beta\eta]$ – see Section 1 for the meaning of this notation, and for the definition of simple types. We also write $A \supset B$ 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. Finally, recall 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 Figure 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, thereby 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 k_a notion rather suggests we should have one fixed association that works

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

Fig. 10. Continuations-monad instantiation

for all source expressions. We will adopt this uniform choice, but will go one step further and assume that the co-variables of $\lambda\mu_{\mathsf{M}}$ (which are those of $\overline{\lambda}\mu\tilde{\mu}$) are also included in the variable supply of the λ-calculus, so we can associate the co-variable a with the λ-calculus variable a. This assumption simplifies the notation: in particular, in the extension of type operations to co-contexts Δ (see Section 1, where the extension works the same on Γ and Δ), and because k_a is just $a^\dagger$.

The translation of 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 Figure 10. Note the mapping of co-variables a in the source calculus $\lambda\mu_{\mathsf{M}}$ into ordinary variables of λ-calculus, which is done silently 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 $L[t]$.

We define $L^{\bullet-}$ to be the argument to $[\,]$ of the context $L^\bullet$, that is,

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

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

Consider the following operators:

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

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

† Viewed from the λ-calculus, there is no difference between these two variable supplies, which are guaranteed by our assumptions. For example, 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.

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

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

into

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

We also define

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

for use in cbn translations. This immediately implies

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

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 Figure 11 are admissible (the third rule is a special case of the second one, and is included for use in later proofs), where $\Delta^{\bullet-}$ follows the usual pattern, with the type operation $(.)^{\bullet-}$ with

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

(recall that $a : A \in \Delta$ 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 in $L^\bullet$.

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

Lemma 5.1. 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) The proof is by induction on T.

(2) The proof is by induction on T. The case of $T = at$ is the only non-trivial case and is proved as follows:

$$\begin{aligned}
([L/a](at))^\bullet &= (L[[L/a]t])^\bullet && \text{(by the definition of structural substitution)}\\
&= ([L/a]t)^\bullet L^{\bullet -} && \text{(by the definition of } (.)^\bullet \text{ and } L^\bullet[u^\bullet] = u^\bullet L^{\bullet -})\\
&= [L^{\bullet -}/a]t^\bullet L^{\bullet -} && \text{(by the induction hypothesis)}\\
&= [L^{\bullet -}/a](t^\bullet a) && \text{(by the definition of substitution in the } \lambda\text{-calculus)}\\
&= [L^{\bullet -}/a](at)^\bullet . && \text{(by the definition of } (.)^\bullet)
\end{aligned}$$

□

Proposition 5.2 (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, the η reduction rule can be omitted from the target. If the β and σ reduction rules in the source are restricted to the β_{v} and σ_{v} forms, respectively, then the β reduction rule in the target can be restricted to β_{v}.

Proof.

(1) The proof is by induction on $T \to T'$. We will just show the base cases since term closure is evident by the Subexpression Property:

— Case β: $(\lambda x.t)s \to [s/x]t$:

$$\begin{aligned}
LHS^\bullet &= (\lambda x.t^\bullet)s^\bullet\\
&\to_\beta [s^\bullet/x]t^\bullet\\
&= RHS^\bullet . && \text{(by Lemma 5.1 (1))}
\end{aligned}$$

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

$$\begin{aligned}
LHS^\bullet &= (\lambda k.ks^\bullet)(\lambda x.c^\bullet)\\
&\to_{\beta_{\mathsf{v}}} (\lambda x.c^\bullet)s^\bullet\\
&\to_\beta [s^\bullet/x]c^\bullet\\
&= RHS^\bullet . && \text{(by Lemma 5.1 (1))}
\end{aligned}$$

— Case π: $L[\mu a.c] \to [L/a]c$:

$$\begin{aligned}
LHS^\bullet &= (\lambda a.c^\bullet)L^{\bullet -}\\
&\to_{\beta_{\mathsf{v}}} [L^{\bullet -}/a]c^\bullet\\
&= RHS^\bullet . && \text{(by Lemma 5.1 (2))}
\end{aligned}$$

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

$$\begin{aligned}
LHS^\bullet &= \lambda a.t^\bullet a\\
&\to_\eta t^\bullet\\
&= RHS^\bullet .
\end{aligned}$$

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

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

(2) Since $V^\bullet$ is always a value, the β steps in the cases β and σ of part (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 that only need $\lambda[\beta_{\mathsf{v}}]$ as target. They only work for subsystems of $\lambda\mu_{\mathsf{M}}$, which, however, cover the images of the monadic translations.

Corollary 5.3.

(1) Every typable expression of $\lambda\mu_{\mathsf{M}}$ is strongly normalisable.
(2) The system $\lambda\mu_{\mathsf{M}}$ is confluent for typable expressions.

Proof.

(1) The statement follows from the previous proposition, the strong normalisation of $\lambda[\beta\eta]$ and the fact that typability is preserved by the instantiation, which is shown in Figure 11.
(2) The statement follows from the strong normalisability and local confluence of $\lambda\mu_{\mathsf{M}}$ (using Newman's Lemma). □

Corollary 5.4. The systems $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ are strongly normalising.

Proof. The statement follows from the previous corollary, together with the strict simulation results from Section 4 and the preservation of typability, which are shown in Figures 6 and 9, respectively. □

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

5.2. *CPS translations through the instantiation of monadic translations*

Our proof of strong normalisation for $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$ gives syntactic embeddings of these systems into $\lambda[\beta\eta]$, which are obtained by composing the 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 since they are the composition of mappings with the strict simulation property. In the following we make this precise and obtain a direct inheritance of strong normalisation 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 since they are the composition of type-sound mappings. In the following, we will make the typing rules they obey explicit. Finally, we will discover the recursive structure of the CPS translations.

We 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

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

Note that there is no index v in $V^\dagger$, and consequently none in V^* either. This seems justified since there is simply no such concept in the cbn translations.

An easy calculation shows that $\langle\![A]\!\rangle_{\mathsf{x}} = \neg\neg A^*_{\mathsf{x}}$, so $\langle\![A]\!\rangle^-_{\mathsf{x}} = \neg A^*_{\mathsf{x}}$ (again, the minus sign warns us that $\langle\![A]\!\rangle^-_{\mathsf{x}}$ has one negation less than $\langle\![A]\!\rangle_{\mathsf{x}}$). It is obvious that $X^*_{\mathsf{x}} = X$.

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) in the cbn case satisfy

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

Corollary 5.5 (typing). The typing rules of Figure 12 are admissible.

Proof. We 'compose' the rules in Figure 6 for $\overline{(.)}_{\mathsf{n}}$ with those in Figure 11 for $(.)^\bullet$. We will just show the typing rules for co-terms since the others are analogous but simpler:

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

where the labelled steps are jusitified as follows:

(a) This follows from the third typing rule in Figure 6.

(b) This follows from the third typing rule in Figure 11 and $\overline{A}_{\mathsf{n}} = MA^\dagger_{\mathsf{n}}$.

(c) Using the compositionality of the extension of type operations to (co-)contexts (see Section 1), we get $\langle\![\Gamma]\!\rangle_{\mathsf{n}} = (\overline{\Gamma}_{\mathsf{n}})^\bullet$ and $\langle\![\Delta]\!\rangle^-_{\mathsf{n}} = (\overline{\Delta}_{\mathsf{n}})^{\bullet -}$ from (4) and (5), respectively. Moreover,

$$\neg\neg(A^\dagger_{\mathsf{n}})^\bullet = (\overline{A}_{\mathsf{n}})^\bullet = \langle\![A]\!\rangle_{\mathsf{n}},$$

using (4), and

$$(\overline{e}_{\mathsf{n}}[y])^\bullet = (\overline{e}_{\mathsf{n}})^\bullet[y^\bullet] = \langle\![e]\!\rangle_{\mathsf{n}}[y]$$

using (1) and (6). □

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

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

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

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

Corollary 5.6 (strict simulation).

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

Proof. The method of the proof is to 'compose' strict simulation for $\overline{(.)}_{\mathsf{n}}$ (Theorem 4.2) with strict simulation for $(.)^{\bullet}$ (Proposition 5.2). More precisely:

(1) Let $T \to T'$. From Theorem 4.2, we get $\overline{T}_{\mathsf{n}} \to^{+} \overline{T'}_{\mathsf{n}}$ in $\lambda\mu_{\mathsf{M}}$, and thus $(\overline{T}_{\mathsf{n}})^{\bullet} \to^{+} (\overline{T'}_{\mathsf{n}})^{\bullet}$ in $\lambda[\beta\eta]$, by Proposition 5.2. We now apply the definition of $\langle\![T]\!\rangle_{\mathsf{n}}$ in (6).
(2) Let $e \to e'$ and $t \in \overline{\lambda}\mu\tilde{\mu}$. Then $\langle t|e\rangle \to \langle t|e'\rangle$, and we conclude by applying part (1).

Proposition 5.7 (recursive characterisation). $\langle\![T]\!\rangle_{\mathsf{n}}$ satisfies the equations in Figure 13.

Proof. To prove the statment, we take the recursive characterisation as the definition of $\langle\![T]\!\rangle_{\mathsf{n}}$ and then use simultaneous induction on t, c and e to prove:

(i) $(\overline{t}_{\mathsf{n}})^{\bullet} = \langle\![t]\!\rangle_{\mathsf{n}}$.
(ii) $(\overline{c}_{\mathsf{n}})^{\bullet} = \langle\![c]\!\rangle_{\mathsf{n}}$.
(iii) $(\overline{e}_{\mathsf{n}})^{\bullet} = \langle\![e]\!\rangle_{\mathsf{n}}$.

The case $c = \langle t|e\rangle$ makes use of $(C[t])^{\bullet} = C^{\bullet}[t^{\bullet}]$. □

We could use this recursive characterisation to give direct proofs of the typing rules and strict simulation for $\langle\![.]\!\rangle_{\mathsf{n}}$, but such proofs would not be as modular as the ones given above.

Remark 5.8. Given the recursive characterisation, statement (2) in Corollary 5.6 reads as follows:

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

$$\frac{\Gamma \vdash t : A|\Delta}{\Gamma^*_{\mathsf{v}}, \langle\![\Delta]\!\rangle^-_{\mathsf{v}} \vdash \langle\![t]\!\rangle_{\mathsf{v}} : \langle\![A]\!\rangle_{\mathsf{v}}} \qquad \frac{\Gamma \vdash V : A|\Delta}{\Gamma^*_{\mathsf{v}}, \langle\![\Delta]\!\rangle^-_{\mathsf{v}} \vdash V^* : A^*_{\mathsf{v}}}$$

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

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

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

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

This statement can be easily generalised so that $\langle\!\langle [e] \rangle\!\rangle_{\mathsf{n}}[u] \to^+ \langle\!\langle [e'] \rangle\!\rangle_{\mathsf{n}}[u]$ holds for any λ-term u. The case $u = y$ is a particular case of the statement already proved, since $y = \langle\!\langle [y] \rangle\!\rangle_{\mathsf{n}}$. The case where u is an arbitrary λ-term then follows from this particular case, since $\langle\!\langle [e] \rangle\!\rangle_{\mathsf{n}}[u] = [u/y](\langle\!\langle [e] \rangle\!\rangle_{\mathsf{n}}[y])$ 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) in the cbv case satisfy

$$(A \supset B)^*_{\mathsf{v}} = A^*_{\mathsf{v}} \supset \langle\!\langle [B] \rangle\!\rangle_{\mathsf{v}}\,.$$

Corollary 5.9 (typing). The typing rules in Figure 14 are admissible.

Proof. The proof is similar to the cbn case (Corollary 5.5). We 'compose' the rules in Figure 9 for $\overline{(.)}_{\mathsf{v}}$ with those in Figure 11 for $(.)^\bullet$, but this time we use the equations $\Gamma^*_{\mathsf{v}} = (\Gamma^\dagger_{\mathsf{v}})^\bullet$ and $\langle\!\langle [\Delta] \rangle\!\rangle^-_{\mathsf{v}} = (\overline{\Delta}_{\mathsf{v}})^{\bullet -}$, which follow from (3) and (5), respectively. □

Corollary 5.10 (strict simulation).

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

Proof. The proof is similar to the cbn case (Corollary 5.6). We 'compose' strict simulation for $\overline{(.)}_{\mathsf{v}}$ (Theorem 4.6) with strict simulation for $(.)^\bullet$ (Proposition 5.2). As observed in Theorem 4.6, $\overline{(.)}_{\mathsf{v}}$ only requires $\beta_{\mathsf{var}} \subset \beta_{\mathsf{v}}$ and σ_{v} from the target $\lambda\mu_{\mathsf{M}}$ rather than β and σ. So, Proposition 5.2(2) applies, and β_{v} rather than β is sufficient in the λ-calculus. □

Proposition 5.11 (recursive characterisation). $\langle\!\langle [T] \rangle\!\rangle_{\mathsf{v}}$ satisfies the equations in Figure 15.

Proof. To prove the statment, we take the recursive characterisation as the definition of $\langle\![T]\!\rangle_{\mathsf{v}}$ and V^* and then use simultaneous induction on V, t, c and e to prove:

(i) $(V^{\dagger})^{\bullet} = V^*$.
(ii) $(\bar{t}_{\mathsf{v}})^{\bullet} = \langle\![t]\!\rangle_{\mathsf{v}}$.
(iii) $(\bar{c}_{\mathsf{v}})^{\bullet} = \langle\![c]\!\rangle_{\mathsf{v}}$.
(iv) $(\bar{e}_{\mathsf{v}})^{\bullet} = \langle\![e]\!\rangle_{\mathsf{v}}$. □

Remark 5.12. Given the recursive characterisation, statement (2) in Corollary 5.10 reads as follows:

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

This statement can be generalised 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 5.8 cannot be repeated with $\langle\![.]\!\rangle_{\mathsf{v}}$ since $v \neq \langle\![v]\!\rangle_{\mathsf{v}}$ and rule β_{v} is not closed under substitution. The generalisation requires a new induction; however, since we have already proved Corollary 5.10, it is enough to prove the generalised statement (2), together with the trivial statement for terms and commands saying that 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). The inductive cases are routine (in the single case $t :: e \to t :: e'$ due to $e \to e'$, we use the induction hypothesis, otherwise, we appeal to Corollary 5.10 (1)).

We will only treat the single base case of generalised statement (2):

— Case $\eta_{\tilde{\mu}}$: $\tilde{\mu}v.\langle v|e\rangle \to e$, with $v \notin e$:
Hence $v \notin \langle\![e]\!\rangle_{\mathsf{v}}^{-}$, so

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

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

6. Extension to second-order logic

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

6.1. *Extension of systems*

In this section we present the systems $\bar{\lambda}2\mu\tilde{\mu}$, $\lambda 2\mu_{\mathsf{M}}$ and $\lambda 2$, which are our second-order versions of $\bar{\lambda}\mu\tilde{\mu}$, $\lambda\mu_{\mathsf{M}}$ 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 should be understood that all notions in the rules (for example, the notion of type in

the new grammar for terms and the notions of types and terms in the old and 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 just given by structural induction. However, the new grammatical elements have to be treated as such, but it will only be mentioned when this leads to non-trivial new cases.

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

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

The type variable X is considered bound in $\forall X.A$. We can define type substitution $[A/X]B$ in an obvious way, and we denote the result of capture-avoiding substitution of all free occurrences of type variable X in type B by type A. As an example in $\lambda\mu_{\mathsf{M}}$, we have

$$[A/X](MB) = M([A/X]B).$$

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 $\Lambda X.t$ to be 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, we do not consider such an η-rule in the second-order calculi discussed in this paper, and, as we will see, these calculi (or the cbn and cbv fragments in the case of $\overline{\lambda}2\mu\tilde{\mu}$) have good properties as reduction systems in terms of subject reduction, strong normalisation 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, any normal and typable command in the second-order extensions of the fragments of $\overline{\lambda}\mu\tilde{\mu}$ is either $\langle x|E\rangle$ or $\langle V|a\rangle$, where normal forms are with respect to the rules of the respective first-order system extended by the $\beta 2$ rule given below.

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

$$\frac{\Gamma \vdash t : B|\Delta}{\Gamma \vdash \Lambda X.t : \forall X.B|\Delta}\ RIntro2$$

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

where X does not occur free in any of the types in Γ, Δ in $RIntro2$. Note that, type B in $LIntro2$ is not determined from $[A/X]B$ and A, so this introduces a further source of ambiguity of the type of a given term. However, we do not attach a type to variable

x in $\lambda x.t$ in the extended system, as would be done in Church-style formulations of second-order λ-calculus. In this way, we obtain *domain-free* systems in the sense of Barthe and Sørensen (2000), which is the style that was also adopted for the formulation of second-order $\lambda\mu$-calculus in Fujita (1999) and Ikeda and Nakazawa (2006).†

In order to formulate the additional reduction rule, we have to assume a notion of type substitution in terms, $[A/X]t$, which will be defined simultaneously with $[A/X]c$ and $[A/X]e$. As 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 $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{v}}$ only have trivial critical pairs (no new critical pair arises with the extension to 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:

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

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

$$\frac{\Gamma \vdash t : B|\Delta}{\Gamma \vdash \Lambda X.t : \forall X.B|\Delta}\; Intro2$$

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

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

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

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

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

† The following analysis can also be applied to Church-style systems by defining each translation of terms as a mapping from terms with their type derivations. On the other hand, Curry-style formulations do not appear to be suitable since there is some evidence that Curry-style cbv polymorphic calculi with control operators are unsound (Harper and Lillibridge 1993; Fujita 1999; Summers 2011).

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

We consider the second-order extension of λ-calculus in the domain-free style, which we will denote by $\lambda 2$, as the target calculus of the continuations-monad instantiation. This calculus was introduced in Barthe and Sørensen (2000) in the framework of the domain-free pure type systems. 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 we do not consider the second-order η-rule for $\lambda 2$: it is not reuqired for our simulation results, so we omit it. In the following, we will concentrate on $\lambda 2[\beta, \beta 2, \eta]$.

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

Proposition 6.1. $\lambda 2[\beta, \beta 2, \eta]$ enjoys strong normalisation.

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 normalisation of which is already known[†].

The erasure function (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. We then prove the following by straightforward induction:

(i) For any domain-free term t that has a type A in context Γ, there exists a Church-style term t' such that t' has the type A in Γ and $\lceil t' \rceil = t$.
(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 of (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, which gives us our result. □

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

[†] Unfortunately, we have been unable to find a canonical source to cite. The weak normalisation of a second-order system was first established in Girard (1971), and strong normalisation is not, essentially, any harder (see Barthe *et al.* (2001) for very general results on this issue). There are many different published proofs of strong normalisation for second-order systems with type annotations on all variable occurrences, and a proof for Church-style typing would only differ from them in inessential details. There are also different styles for the treatment of the η-reduction rule: one way is to note that there is an inductive characterisation of SN terms that is indifferent to η-reduction (Matthes 1999); another is to postpone the η-reduction steps.

6.2. *Extension of translations*

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

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

$A^{\dagger}_{\mathsf{x}}$ 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 relies recursively on the different treatment of implication according to the two paradigms. On the surface, there is also no difference between the translations of expressions: we add

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

to the cbn translation given in Figure 7 and the cbv translation in Figure 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 – we could have stated them in Section 4, but they only become relevant now.

Lemma 6.2.

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) The proof is by simultaneous induction on B.
(2) The proof is by simultaneous induction on T. □

It is easy to establish that the admissible typing rules in Figures 6 and 9, respectively, still hold for this extension. The key to verifying $\overline{(A :: e)}_{\mathsf{x}}$ 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 in $\overline{(A :: e)}_{\mathsf{x}}$ is filled with a variable of type $\overline{(\forall X.B)}_{\mathsf{x}}$ in both paradigms.

Theorem 6.3 (strict simulation for second-order monadic translation). Let $\mathsf{x} \in \{\mathsf{n}, \mathsf{v}\}$.

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

The same restrictions on the rules in the target system as in Theorems 4.2 and 4.6 are sufficient.

Proof. The proof proceeds in the same way as the proofs of Theorems 4.2 and 4.6. We will only show the case for the new reduction rule. Note, however, that $\overline{(A :: e)}_{\mathsf{x}}$ is a base context, so the simulation of rule π is not hampered in the case of $E = A :: e$.

— Case $\beta 2$:
Using Lemma 6.2 (2) (we omit the index x everywhere), we have:

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

□

We will now extend the continuations-monad instantiation and 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, we add

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

to the translation given in Figure 10, that is, every second-order element is translated homomorphically.

Lemma 6.4. 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. The proof is by induction on B and T, respectively. □

Using this lemma, it is easy to check that the rules in Figure 11 are admissible, and the extended continuations-monad instantiation strictly preserves the reduction steps.

Proposition 6.5.
(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 Proposition 5.2 (2) again hold.

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

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

(2) This part is proved in a similar way to Proposition 5.2 (2). □

The part (1) of the above proposition, together with Proposition 6.1 and the preservation of typability shown in Figure 11, immediately gives the following corollary.

Corollary 6.6. $\lambda 2\mu_{\mathsf{M}}$ enjoys strong normalisation.

The strong normalisation of the cbn- and cbv-fragments of $\bar{\lambda}2\mu\tilde{\mu}$ now follows.

Corollary 6.7. $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{n}}$ and $\overline{\lambda}2\mu\tilde{\mu}_{\mathsf{v}}$ are strongly normalising.

Proof. We use the previous corollary, together with Theorem 6.3 and the preservation of typability, which are shown in Figures 6 and 9, respectively (and can be verified to hold even for the second-order extension). □

Despite the little work invested in the second-order extension, we have obtained a strong result here. This is thanks to the CPS translation and its monadic generalisation, which is known to scale up well, while the negative translation is confined to first-order types/formulas (see, for example, Parigot (1997)).

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

$$\begin{aligned}(\forall X.A)^*_{\mathsf{x}} &= \forall X.(\![A]\!]_{\mathsf{x}}\\ (\![\Lambda X.t]\!]_{\mathsf{x}} &= \mathsf{Eta}(\Lambda X.(\![t]\!]_{\mathsf{x}})\\ (\![A :: e]\!]_{\mathsf{x}} &= [\,](\lambda z.(\![e]\!]_{\mathsf{x}}[zA^*_{\mathsf{x}}])\,.\end{aligned}$$

These 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 have proved the strong normalisation of the cbn and cbv fragments of $\overline{\lambda}\mu\tilde{\mu}$ through a syntactic embedding into the λ-calculus, which extends to the second order with domain-free polymorphic λ-calculus as target. The embeddings are CPS translations with the strict simulation property, which are 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, and combines in a non-trivial way the syntax for classical logic in the style of $\lambda\mu$-calculus with the syntax for a monad as found in Moggi's monadic meta-language.

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

7.1. *Related work*

7.1.1. *Strong normalisation for* $\overline{\lambda}\mu\tilde{\mu}$. Strong normalisation 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 formalisable in arithmetic. Before that, Lengrand (2003) had also proved the strong normalisation of $\overline{\lambda}\mu\tilde{\mu}$ by using an embedding into the sequent calculus for classical logic of Urban (2000), which was proved to be strongly normalising by the reducibility method. A more syntactic approach was 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 then proved. It is not clear from the proof

provided in Rocheteau (2005) whether strict simulation is actually achieved, and strong normalisation for this extension of $\lambda\mu$ is not addressed.

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

7.1.2. *Monadic translation.* The technique of *monadic translation* to prove strong normalisation 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 factorising 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 then composed with an instantiation to the *identity* monad. Simulation only works if an extra permutative reduction rule, usually named 'assoc', is added to the target (λ-calculus), and extension to second order looks problematic. Composition with an instantiation to the continuations monad produced a CPS translation, but no simulation.

The present paper greatly 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 factorisation via a monadic language;
(iv) offering the new monadic language required for this factorisation.

7.1.3. *CPS translations with strict simulation.* A key issue of strict simulation is that we not only have to consider the reduction steps at the root, but also those deeper within a term. This has sometimes been overlooked in the literature, as pointed out with some examples in Nakazawa and Tatsuta (2003), and has led to 'incorrect proofs' of strong normalisation by CPS. A CGPS-translation (*continuation-and-garbage-passing* style translation) of $\lambda\mu$ achieving strict simulation in $\lambda[\beta]$ was developed in Ikeda and Nakazawa (2006). This style of translation, which 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 has not been 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 normalisation 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 second order (with simulation in domain-free polymorphic λ-calculus).

7.1.4. *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 $(\_)^{\triangleleft}$, respectively, were given 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 dualised version of the cbn translation. Although no precise statement of the preservation of reduction by the translations is made, the paper 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 may verify that $(\_)^{\triangleleft}$ does not simulate the β-rule for the subtraction connective, it only obtains convertible terms. By duality, the same happens with $(\_)^{\triangleright}$ with respect to 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 $\overline{\lambda}\mu\tilde{\mu}_{T}$ and $\overline{\lambda}\mu\tilde{\mu}_{Q}$ fragments, which were 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 weak simulation is then stated for this fragment (weak simulation means that each reduction step of the source is mapped into zero or more reduction steps in the target). Again, one can verify that β-reduction for implication is mapped to β-equality only, but this time for both the cbn and cbv translations.

CPS translations for both fragments of $\overline{\lambda}\mu\tilde{\mu}$ were also considered in Lengrand (2003) (see the *erratum* for the correct definition of the cbn translation). When compared with 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 $\langle\!|\langle t|e\rangle|\!\rangle = \langle\!|e|\!\rangle\langle\!|t|\!\rangle$, which forces co-term translations to a have a type of the form $\neg\langle\!|A|\!\rangle$ (compared with our $\langle\!|\langle t|e\rangle|\!\rangle = \langle\!|e|\!\rangle\,[\langle\!|t|\!\rangle]$).

The development of the CPS translations in Lengrand (2003) was guided by semantic considerations, and the results showing the 'preservation of semantics' by the CPS translations state that when a term reduces to another, their images are β-convertible. Having said this, we have been able to verify that Lengrand's cbv translation shares the simulation property of our Corollary 5.10, and the need for η-reduction in the target, while β-conversions cannot be avoided in the target of Lengrand's cbn translation.

Summarising, we may say that rather than strict simulation, the existing literature on CPS translations for $\overline{\lambda}\mu\tilde{\mu}$ has had other preoccupations such as 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, despite only being a by-product of our approach to strong normalisation.

7.2. *Future work*

The meta-language introduced in this paper has good meta-theoretic properties (subject reduction, confluence and strong normalisation), 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, for example, Hatcliff and Danvy (1994)). This may lead to new connections with polarised 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) and 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 paper, the goal was to make a comprehensive and uniform analysis of existing translations of an intuitionistic source calculus. In the current paper, the monadic meta-language has provided a vehicle for discovering 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 Hatcliff and Danvy (1994), of CPS translations of λ-calculi with control operators. Although such studies are beyond the scope of the current paper, we give some supplementary analysis of our CPS translations in the appendix.

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, positive connectives such as disjunction and the second-order existential quantifier, together with their usual permutative conversions, would also 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 Curien and Herbelin (2000)). So strong normalisation for $\overline{\lambda}\mu\tilde{\mu}$ is not immediately inherited by $\lambda\mu$. On the other hand, the strong normalisation of $\lambda\mu$ has been proved using the technique of CGPS translation (Ikeda and Nakazawa 2006), though this technique has not yet been extended to $\overline{\lambda}\mu\tilde{\mu}$. There is clearly still some room for greater systematisation of techniques for proving strong normalisation.

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 us to discover opportunities for improvement in the components of the CPS translation. The refinements we introduce are actually confined to the continuations-monad instantiation, 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 analyse the CPS translations of Section 5.2, which we refer to as the *main* CPS translations. They are sound with respect to 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).

We will 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 since 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 can achieve one of the extra properties we have mentioned on top of the properties already enjoyed by the main translations, but none of the variants achieves both extra properties, though a more extensive modification of the main translations, not pursued in this paper, might do so.

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, we obtain 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, which are dedicated to cbn and cbv, respectively. Through composition with the respective monadic translations, new *optimised* 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 a 1–1 fashion, even though the monadic translations essentially do (see Remarks 4.3 and 4.7). As can be seen in the proof of Proposition 5.2, the main instantiation $(.)^{\bullet}$ itself generates 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 the current paper, we achieve the same goal in a modular way, profiting from the decomposition of CPS translations via $\lambda\mu_{\mathsf{M}}$. Indeed, all we need to do in our case is to introduce a slight improvement in the definition of the continuations-monad instantiation.

We will now define the ready continuations-monad instantiation, denoted $(.)^\circ$, by creating an exception in the clause for the instantiation of a command in the definition of Figure 10:

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

The remaining clauses of $(.)^\bullet$ are unchanged. It is then immediate that $T^\bullet \to^*_{\mathsf{admin}} T^\circ$.

We define $L^{\circ -}$ as the argument to the hole in L° (hence $L^\circ = []L^{\circ -}$). Then, the last two equations of (10) can be written uniformly 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 5.1 has to be modified as follows.

Lemma A.1. 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) The proof is by induction on T. We will 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 $u = \eta r$:

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

(2) The proof is by 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, we write $t \to^= u$ to mean $t \to u$ or $t = u$, that is, $\to^=$ is the reflexive closure of $\to$. We will use the symbol ρ to denote reduction rules from now on – there is no risk of confusion since we do not consider ρ-reduction in this paper.

Proposition A.2 (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. The proof is by induction on $T \to_\rho T'$. For the base cases, we follow the proof of Proposition 5.2, paying attention to the variants β_{v}, β_{var} and σ_{v}, and using Lemma A.1 instead of Lemma 5.1. In the base case for β, the use of Lemma A.1 may generate administrative reductions. The base case for π is exactly as in Proposition 5.2. 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 A.1.

— 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 A.1.

— 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 Proposition 5.2. 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

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

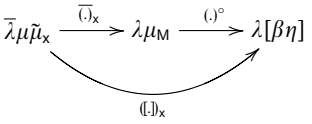

Fig. 16. Ready instantiation and CPS translations

Otherwise,

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

All but one of the inductive cases 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 the induction hypothesis. If $t_1 = \eta u_1$ and $t_2 = \eta u_2$, with $u_1 \to_{\rho} u_2$, then we again apply the induction hypothesis. 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, but only when $\rho \in \{\beta, \beta_{\mathsf{v}}, \beta_{\mathsf{var}}, \eta_{\mu}\}$: $t_1 \neq \eta u_1$ and $t_2 = \eta u_2$. In this case

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

□

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 Figure 16). In the cbn case, we set

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

In the cbv case, we also set[†]

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

Nothing is changed at the level of typing with respect to the main CPS translations, so $([.])_{\mathsf{n}}$ enjoys the typing rules of Figure 12, and $([.])_{\mathsf{v}}$ enjoys the typing rules of Figure 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 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 some administrative steps. So, in such

[†] Recall that $V^{*} := (V^{\dagger})^{\bullet}$. Typographically, V^{*} (with the multiplication symbol as superscript) may be hard to distinguish from the $V^{\star}$ introduced here, but since these symbols appear in different sections, there should be no risk of confusion.

a reduction, no administrative step comes before the reduction steps corresponding to the source reduction $T \to_\rho T'$. We will now make these remarks precise.

Definition A.3 (ready reduction). In the λ-calculus:

— Cbn case:
Given s, s' λ-terms and ρ a reduction rule of $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{n}}$, we define $s \overset{\rho}{\Rightarrow}_{\mathsf{n}} s'$ by:

- 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 $\overline{\lambda}\mu\tilde{\mu}_{\mathsf{v}}$, we define $s \overset{\rho}{\Rightarrow}_{\mathsf{v}} s'$ by:

- 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, if $\rho = \eta_\mu$, the following result gives the readiness property in a slightly extended sense.

Corollary A.4 (strict simulation with the readiness property). Let $\mathsf{x} \in \{\mathsf{n}, \mathsf{v}\}$. Then:

(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 in Corollaries 5.6 and 5.10, the proof is by 'composition' (but this time using Proposition A.2) of the simulation theorems of the monadic translations (Theorems 4.2 and 4.6), including Remarks 4.3 and 4.7. □

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

Proposition A.5 (recursive characterisation). $(\!|T|\!)_{\mathsf{n}}$ satisfies the equations in Figure 17 .

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

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

Proof. We use the same induction as in the proof of Proposition 5.7. To prove the statement, we take the recursive characterisation in Figure 17 as the definition of $([T])_{\mathsf{n}}$ and define $([E])^{-}_{\mathsf{n}}$ as the argument to the hole in $([E])_{\mathsf{n}}$ (hence $([E])_{\mathsf{n}} = [\,]([E])^{-}_{\mathsf{n}}$). Then we use simultaneous induction on t, c, E and e to 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) $(\overline{e}_{\mathsf{n}})^{\circ} = ([e])_{\mathsf{n}}$.

We will only show the cases that need to be updated:

— Case $e = \tilde{\mu} y.c$:

$$
\begin{array}{rll}
(\overline{(\tilde{\mu} y.c)}_{\mathsf{n}})^{\circ} & = \mathsf{bind}(\eta[\,], y.\overline{c}_{\mathsf{n}})^{\circ} & \text{(by the definition of } \overline{(.)}_{\mathsf{n}}) \\
 & = (\lambda y.(\overline{c}_{\mathsf{n}})^{\circ})[\,] & \text{(by (12))} \\
 & = (\lambda y.([c])_{\mathsf{n}})[\,] & \text{(by the induction hypothesis)} \\
 & = ([\tilde{\mu} y.c])_{\mathsf{n}}\,. & \text{(by recursive definition)}
\end{array}
$$

— Case $E = u :: e$:

$$
\begin{array}{rll}
(\overline{(u :: e)}_{\mathsf{n}})^{\circ -} & = \mathsf{bind}([\,], f.\mathsf{bind}(\eta\overline{u}_{\mathsf{n}}, z.\overline{e}_{\mathsf{n}}[fz]))^{\circ -} & \text{(by the definition of } \overline{(.)}_{\mathsf{n}}) \\
 & = \lambda f.\mathsf{bind}(\eta\overline{u}_{\mathsf{n}}, z.\overline{e}_{\mathsf{n}}[fz])^{\circ} & \text{(by the definition of } (.)^{\circ -}) \\
 & = \lambda f.(\lambda z.(\overline{e}_{\mathsf{n}})^{\circ}[fz])\overline{u}_{\mathsf{n}}{}^{\circ} & \text{(by the definition of } (.)^{\circ}) \\
 & = \lambda f.(\lambda z.([e])_{\mathsf{n}}[fz])([u])_{\mathsf{n}} & \text{(by the induction hypothesis)} \\
 & = ([E])^{-}_{\mathsf{n}}\,. & \text{(by recursive definition)}
\end{array}
$$

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

$$
\begin{array}{rll}
(\overline{\langle \lambda y.u|E\rangle}_{\mathsf{n}})^{\circ} & = (\overline{E}_{\mathsf{n}}[\eta(\lambda y.\overline{u}_{\mathsf{n}})])^{\circ} & \text{(by the definition of } \overline{(.)}_{\mathsf{n}}) \\
 & = (\overline{E}_{\mathsf{n}})^{\circ -}(\lambda y.(\overline{u}_{\mathsf{n}})^{\circ})) & \text{(by (11))} \\
 & = ([E])^{-}_{\mathsf{n}}(\lambda y.([u])_{\mathsf{n}})) & \text{(by the induction hypothesis)} \\
 & = ([\langle \lambda y.u|E\rangle])_{\mathsf{n}}\,. & \text{(by recursive definition)}
\end{array}
$$

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

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

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

$$
\begin{aligned}
(\overline{\langle t|e\rangle}_{\mathsf{n}})^\circ &= (\overline{e}_{\mathsf{n}}[\overline{t}_{\mathsf{n}}])^\circ && \text{(by the definition of } \overline{(.)}_{\mathsf{n}}) \\
&= (\overline{e}_{\mathsf{n}})^\circ[(\overline{t}_{\mathsf{n}})^\circ] && \text{(by (13))} \\
&= ([e])_{\mathsf{n}}[([t])_{\mathsf{n}}] && \text{(by the induction hypothesis)} \\
&= ([\langle t|e\rangle])_{\mathsf{n}}\,. && \text{(by recursive definition)}
\end{aligned}
$$

□

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

Proposition A.6 (recursive characterisation). $([T])_{\mathsf{v}}$ satisfies the equations in Figure 18.

Proof. We use the same induction as in proof of Proposition 5.11. To carry out the proof, we take the recursive characterisation in Figure 18 as the definition of $([T])_{\mathsf{v}}$ and $V^\star$ and define $([e])^-_{\mathsf{v}}$ as the argument to the hole in $([e])_{\mathsf{v}}$ (hence $([e])_{\mathsf{v}} = [\,]([e])^-_{\mathsf{v}}$). We then use simultaneous induction on V, t, c and e to prove:

(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 $(\overline{e}_{\mathsf{v}})^\circ = ([e])_{\mathsf{v}}$.

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

— Suppose $t = V$:

$$
\begin{aligned}
(\overline{\langle V|e\rangle}_{\mathsf{v}})^\circ &= (\overline{e}_{\mathsf{v}}[\eta V^\dagger])^\circ && \text{(by the definition of } \overline{(.)}_{\mathsf{v}}) \\
&= (\overline{e}_{\mathsf{v}})^{\circ -}((V^\dagger)^\circ) && \text{(by (11))} \\
&= ([e])^-_{\mathsf{v}}(V^\star) && \text{(by the induction hypothesis)} \\
&= ([\langle V|e\rangle])_{\mathsf{v}}\,. && \text{(by recursive definition)}
\end{aligned}
$$

— Now let $t \neq V$:

$$
\begin{aligned}
(\overline{\langle t|e\rangle}_{\mathsf{v}})^{\circ} &= (\bar{e}_{\mathsf{v}}[\bar{t}_{\mathsf{v}}])^{\circ} && \text{(by the definition of } \overline{(.)}_{\mathsf{v}}) \\
&= (\bar{e}_{\mathsf{v}})^{\circ}[(\bar{t}_{\mathsf{v}})^{\circ}] && \text{(by (10), as } \bar{t}_{\mathsf{v}} \neq \eta u) \\
&= ([\![e]\!]_{\mathsf{v}}[([\![t]\!]_{\mathsf{v}}] && \text{(by the induction hypothesis)} \\
&= ([\![\langle t|e\rangle]\!]_{\mathsf{v}}. && \text{(by recursive definition)}
\end{aligned}
$$

□

Remark A.7. Unlike what we did for the main CPS translations in Remarks 5.8 and 5.12, we are not going to generalise Corollary A.4 (2) because of the non-uniform translation of commands, which is made explicit in the recursive characterisations.

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

If a CPS translation is also viewed as a computational interpretation and not just as a device for proving strong normalisation, 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 will just concentrate on the former now.

Rule η is not just used in Proposition 5.2 (1), but is also needed for soundness of the main cbv CPS translation. As an example, consider

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

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

The problem is even easier to see for the cbn translation: since $\langle\!|[\![y]\!]_{\mathsf{n}}|\!\rangle = y$ 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.

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

Again, rule β, and not just β_{v}, is not only used in Proposition 5.2 (1), but also needed to show the soundness of the cbn CPS translation: to see this, we can reuse the commands c_1 and c_2 of the earlier example, but calculate the β_{v}-normal forms of their cbn CPS translations to get $z(\lambda f.(\lambda x.xa)(fy))$ and $z(\lambda f.fya)$, respectively – the problem here is that fy is not a value.

Still, as we are about to show, 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.

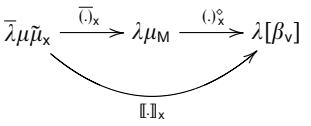

Fig. 19. Optimised instantiation and CPS translations

A.3. *Optimised CPS translations and the indifference property*

In this section we will give refinements of the main continuations-monad instantiation, and thus of the CPS translations – see the summary in Figure 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 will begin 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, both the variables and the arguments of the unit η of the monad are expanded. Relative to the continuations-monad instantiation $(.)^\bullet$, we will change the translation at the level of expressions, with different refinements for cbn and cbv, which leads us to introduce the notations $T^\diamond_n$ and $T^\diamond_v$. The translation is unchanged at the level of types, contexts and co-contexts, but for uniformity we introduce the notations $A^\diamond_n$, $\Gamma^\diamond_n$ and $\Delta^{\diamond-}_n$, together with the cbv variants, even though $A^\diamond_n = A^\bullet = A^\diamond_v$, and so on.

A.3.1. *Cbn case.* We will now define a refinement of the continuations-monad instantiation, denoted $T^\diamond_n$. The only changes relative to the definition of $T^\bullet$ in Figure 10 are in the case of a variable, which was $x^\bullet = x$, and is now defined by

$$x^\diamond_n = \uparrow x\,,$$

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

$$(\eta t)^\diamond_n = \mathsf{Eta}(\uparrow(t^\diamond_n))\,.$$

The first redefinition will be used to get rid of η-reduction in the final target, while the second then allows us to restrict β-reduction in the target to β_v.

Consequently, we redefine

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

in order to maintain

$$(C[t])^\diamond_n = C^\diamond_n[t^\diamond_n] \tag{16}$$

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

Obviously, the redefinition invalidates the admissible typing rules of Figure 11 since $x^\diamond_n$ can only be typed by an implication, but $A^\diamond_n$ can be a type variable (when $A = X$). Similarly, the argument term t of ηt might be typed by a type variable, so the η-expansion of $t^\diamond_n$ would not be typable.

A very simple solution is a global exclusion of term typings with atomic types, that is, type variables. We will 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 the subsystem of the typing system for $\lambda\mu_{\mathsf{M}}$ (Figure 4) that only operates with non-atomic sequents the *non-atomic typing system*. For emphasis, we write non-atomic sequents (and derivability in the non-atomic typing system) thus:

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

The typing rules of Figure 11 then hold for $(.)^{\diamond}_{\mathsf{n}}$, if the premisses are replaced by non-atomic derivability. On the other hand, the non-atomic system is sufficient for typing any expression in the range of the cbn monadic translation: this is easy to verify, 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 A.10).

Lemma 5.1 has to be refined as follows.

Lemma A.8. 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 that is not an application.
(2) $([L/a]T)^{\diamond}_{\mathsf{n}} = [L^{\diamond -}_{\mathsf{n}}/a]T^{\diamond}_{\mathsf{n}}$.

Proof.

(1) The proof is by induction on T. The case for $T = x$ is the only non-trivial case, and its proof is as follows:

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

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

(2) The proof is by induction on T and is unchanged from the proof of Lemma 5.1. □

Proposition A.9 (optimised 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 β_{n} and $\eta_{\mu\mathsf{n}}$, respectively, 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. The proof is by induction on $T \to T'$. Note that if s is not an application, then $s^{\diamond}_{\mathsf{n}}$ is a λ-abstraction. We will now consider what the proofs of the base cases of Proposition 5.2 (1) yield in this situation:

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

$$\begin{array}{rcll} LHS^{\diamond}_{\mathsf{n}} & = & (\lambda x.t^{\diamond}_{\mathsf{n}})s^{\diamond}_{\mathsf{n}} & \\ & \to_{\beta_{\mathsf{v}}} & [s^{\diamond}_{\mathsf{n}}/x]t^{\diamond}_{\mathsf{n}} & \\ & \to^{*}_{\beta_{\mathsf{var}}} & RHS^{\diamond}_{\mathsf{n}}\,. & \text{(by Lemma A.8 (1))} \end{array}$$

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

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

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

- Subcase $\sigma = \sigma_{\mathsf{n}}$:
  Here s is not an application, so $s^{\diamond}_{\mathsf{n}}$ is a λ-abstraction and $\uparrow s^{\diamond}_{\mathsf{n}} \to_{\beta_{\mathsf{var}}} s^{\diamond}_{\mathsf{n}}$, which justifies (a).
  Step (b) is justified by Lemma A.8 (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 reduction step each (recall that $L^{\diamond -}_{\mathsf{n}}$ is a value):

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

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

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

Case $\pi\colon L[\mu a.c] \to [L/a]c$:

$$\begin{array}{rcll} LHS^{\diamond}_{\mathsf{n}} & = & (\lambda a.c^{\diamond}_{\mathsf{n}})L^{\diamond -}_{\mathsf{n}} & \\ & \to_{\beta_{\mathsf{v}}} & [L^{\diamond -}_{\mathsf{n}}/a]c^{\diamond}_{\mathsf{n}} & \\ & = & RHS^{\diamond}_{\mathsf{n}}. & \text{(by Lemma A.8 (2))} \end{array}$$

Case η_μ: $\mu a.at \to t$, with $a \notin t$:
By our restriction to $\eta_{\mu\mathsf{n}}$, we know that $t^\diamond_\mathsf{n}$ is a λ-abstraction. So,

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

Note that we have omitted rule η_{bind}. □

We now define the cbn optimised CPS translation:

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

For an evaluation context E, we also put

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

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

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

Corollary A.10 (typing). The typing rules of Figure 12 are admissible for $[\![.]\!]_\mathsf{n}$.

Proof. The proof method of Corollary 5.5 applies again. We 'compose' the rules in Figure 6 for $\overline{(.)}_\mathsf{n}$ with conclusions in the non-atomic system, with the rules in Figure 11, which hold for $(.)^\diamond_\mathsf{n}$ provided the premisses are in the non-atomic system as well. We clearly need

$$[\![\Gamma]\!]_\mathsf{n} = (\overline{\Gamma}_\mathsf{n})^\diamond_\mathsf{n}$$
$$[\![\Delta]\!]^-_\mathsf{n} = (\overline{\Delta}_\mathsf{n})^{\diamond -}_\mathsf{n},$$

which, as usual for these composition lemmas, are obtained by the observation in Section 1. As in Section 5.2, we will just show the typing rule for co-terms:

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

where the labelled steps are justified as follows:

(a) This follows from the third typing rule in Figure 6 with non-atomic conclusions.
(b) This follows from the third typing rule in Figure 11 with non-atomic premisses.
(c) This follows from

$$\neg\neg(A^\dagger_\mathsf{n})^\diamond_\mathsf{n} = (\overline{A}_\mathsf{n})^\diamond_\mathsf{n} = [\![A]\!]_\mathsf{n}$$

and then

$$(\overline{e}_\mathsf{n}[y])^\diamond_\mathsf{n} = (\overline{e}_\mathsf{n})^\diamond_\mathsf{n}[y^\diamond_\mathsf{n}] = [\![e]\!]_\mathsf{n}[\uparrow y]$$

using (16) and (17).

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

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

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

Corollary A.11 (strict simulation with the 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 we did in the proof of Corollary 5.6, we 'compose' the simulation theorem of the cbn monadic translation $\overline{(.)}_{\mathsf{n}}$ (Theorem 4.2), but this time with Proposition A.9. Note the provisos in this proposition are met due to constraints noted in the extra statement of Theorem 4.2 and since $\beta_{\mathsf{var}} \subset \beta_{\mathsf{n}}$. □

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

Proposition A.12 (recursive characterisation). $[\![T]\!]_{\mathsf{n}}$ satisfies the equations in Figure 20.

Proof. The proof is similar to the proof of Proposition 5.7. □

In particular, the proof of the previous proposition establishes the fact 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 A.13. Given the recursive characterisation, statement (2) in Corollary A.11 reads as follows:

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

This statement can be generalised so that $[\![e]\!]_{\mathsf{n}}[u] \to^{+} [\![e']\!]_{\mathsf{n}}[u]$ in $\lambda[\beta_{\mathsf{v}}]$ for any λ-term u. As in Remark 5.12, this is proved by a new simultaneous induction, together with trivial statements for terms and commands. The inductive cases are routine, so we will only consider 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 characterisation,

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

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

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

Otherwise, $e = \tilde{\mu}z.c$, so

$$\begin{array}{rcl}
\tilde{\mu}y.\langle y|e\rangle & = & \tilde{\mu}y.\langle y|\tilde{\mu}z.c\rangle \\
& \to_{\sigma_{\mathsf{v}}} & \tilde{\mu}y.[y/z]c \\
& = & \tilde{\mu}z.c\,,
\end{array}$$

so 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 will now refine the continuations-monad instantiation by keeping the definition of $(.)^{\bullet}$ in Section 5 except for setting

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

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

Lemma 5.1 is modified as follows (the only change appears in part (2)).

Lemma A.14.

(1) $([u/x]T)_{\mathsf{v}}^{\diamond} = [u_{\mathsf{v}}^{\diamond}/x]T_{\mathsf{v}}^{\diamond}$.
(2) $[L_{\mathsf{v}}^{\diamond -}/a]T_{\mathsf{v}}^{\diamond} \to_{\beta_{\mathsf{var}}}^{*} ([L/a]T)_{\mathsf{v}}^{\diamond}$.

Proof.

(1) The proof is by induction on T.
(2) The proof is by induction on T:

— Case at:

$$\begin{array}{rcll}
[L_{\mathsf{v}}^{\diamond -}/a](at)_{\mathsf{v}}^{\diamond} & = & [L_{\mathsf{v}}^{\diamond -}/a](t_{\mathsf{v}}^{\diamond}(\uparrow a)) & \\
& = & [L_{\mathsf{v}}^{\diamond -}/a]t_{\mathsf{v}}^{\diamond}(\uparrow L_{\mathsf{v}}^{\diamond -}) & \\
& \to_{\beta_{\mathsf{var}}}^{*} & ([L/a]t)_{\mathsf{v}}^{\diamond}(\uparrow L_{\mathsf{v}}^{\diamond -}) & \text{(by the induction hypothesis)} \\
& \to_{\beta_{\mathsf{var}}} & ([L/a]t)_{\mathsf{v}}^{\diamond} L_{\mathsf{v}}^{\diamond -} & \\
& = & (L[[L/a]t])_{\mathsf{v}}^{\diamond} & \\
& = & ([L/a](at))_{\mathsf{v}}^{\diamond}\,. &
\end{array}$$

The final reduction step is a β_{var} step since, as we noted earlier, $L^{\diamond -}_{\mathsf{v}}$ is a λ-abstraction. □

Proposition A.15 (optimised 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 $T^{\diamond}_{\mathsf{v}} \to^{+} T'^{\diamond}_{\mathsf{v}}$ in $\lambda[\beta_{\mathsf{v}}]$.

Proof. The proof is by induction on $T \to T'$. We will just consider what the proofs of the base cases of Proposition 5.2 (1) yield in the present situation:

— Case β_{v}: $(\lambda x.t)V \to [V/x]t$:
Note that $V^{\diamond}_{\mathsf{v}}$ is a value. So,

$$\begin{aligned} LHS^{\diamond}_{\mathsf{v}} &= (\lambda x.t^{\diamond}_{\mathsf{v}})V^{\diamond}_{\mathsf{v}} \\ &\to_{\beta_{\mathsf{v}}} [V^{\diamond}_{\mathsf{v}}/x]t^{\diamond}_{\mathsf{v}} \\ &= RHS^{\diamond}_{\mathsf{v}}. \qquad \text{(by Lemma A.14 (1))} \end{aligned}$$

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

$$\begin{aligned} LHS^{\diamond}_{\mathsf{v}} &= (\lambda k.kV^{\diamond}_{\mathsf{v}})(\lambda x.c^{\diamond}_{\mathsf{v}}) \\ &\to_{\beta_{\mathsf{v}}} (\lambda x.c^{\diamond}_{\mathsf{v}})V^{\diamond}_{\mathsf{v}} \\ &\to_{\beta_{\mathsf{v}}} [V^{\diamond}_{\mathsf{v}}/x]c^{\diamond}_{\mathsf{v}} \\ &= RHS^{\diamond}_{\mathsf{v}}. \qquad \text{(by Lemma A.14 (1))} \end{aligned}$$

— Case π: $L[\mu a.c] \to [L/a]c$:

$$\begin{aligned} LHS^{\diamond}_{\mathsf{v}} &= (\lambda a.c^{\diamond}_{\mathsf{v}})L^{\diamond -}_{\mathsf{v}} \\ &\to_{\beta_{\mathsf{v}}} [L^{\diamond -}_{\mathsf{v}}/a]c^{\diamond}_{\mathsf{v}} \\ &\to^{*}_{\beta_{\mathsf{var}}} RHS^{\diamond}_{\mathsf{v}}. \qquad \text{(by Lemma A.14 (2))} \end{aligned}$$

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

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

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

$$\begin{aligned} \lambda a.t^{\diamond}_{\mathsf{v}}(\uparrow a) &= \lambda a.(\lambda k.kV^{\diamond}_{\mathsf{v}})(\uparrow a) \\ &\to_{\beta_{\mathsf{v}}} \lambda a.(\uparrow a)V^{\diamond}_{\mathsf{v}} \\ &\to_{\beta_{\mathsf{v}}} \lambda a.aV^{\diamond}_{\mathsf{v}} \\ &= RHS^{\diamond}_{\mathsf{v}}. \end{aligned}$$

In the final reduction we have used the fact that $V^{\diamond}_{\mathsf{v}}$ is a value.

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

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

□

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

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

to obtain the optimised cbv CPS translation. As usual, we also define

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

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

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

The rules in Figure 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 Figure 14 (with $\langle\!|.|\!\rangle_{\mathsf{v}}$ replaced by $[\![.]\!]_{\mathsf{v}}$) also still hold. Thus, the situation is more pleasant than for the cbn case.

Corollary A.16 (strict simulation with the 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 we did in the proof of Corollary 5.10, we 'compose' the simulation theorem of the cbv monadic translation $\overline{(.)}_{\mathsf{v}}$ (Theorem 4.6), but this time with Proposition A.15. 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 4.6 and recall that $\beta_{\mathsf{var}} \subset \beta_{\mathsf{v}}$). □

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

Proposition A.17 (recursive characterisation). The recursive characterisation of $[\![.]\!]_{\mathsf{v}}$ is obtained by changing the clause for co-variables in Figure 15 as follows:

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

Proof. We just adapt the case $e = a$ in the induction we used to prove Proposition 5.11. □

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

Remark A.18. Given the recursive characterisation, statement (2) in Corollary A.16 reads as follows:

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

This statement can be generalised 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 $x \notin e$), which we do as follows:

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

References

Asada, K. (2008) Extensional universal types for call-by-value. In: Ramalingam, G. (ed.) The 6th Asian Symposium on Programming Languages and Systems (APLAS 2008). *Springer-Verlag Lecture Notes in Computer Science* **5356** 122–137.

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: *Proceedings of ICFP 2000*, IEEE 233–243. (*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. (eds.) Theoretical Computer Science: Proceedings TCS 2010. *IFIP Advances in Information and Communication Technology* **323**, Springer-Verlag 165–181.

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. (ed.) Proceedings of TLCA 2007. *Springer-Verlag Lecture Notes in Computer Science* **4583** 133–147.

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. (eds.) Post-Proceedings of TYPES 2008. *Springer-Verlag Lecture Notes in Computer Science* **5497** 100–116.

Fujita, K. (1999) Explicitly typed $\lambda\mu$-calculus for polymorphism and call-by-value. In: Girard, J.-Y. (ed.) Proceedings of TLCA 1999. *Springer-Verlag Lecture Notes in Computer Science* **1581** 162–177.

Geuvers, H. (1993) *Logics and Type Systems*, Proefschrift (Ph.D. 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. (ed.) *Proceedings of the Second Scandinavian Logic Symposium*, Studies in Logic and the Foundations of Mathematics **63**, North-Holland 63–92.

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: *Proceedings of POPL 1994*, ACM 458–471.

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. (eds.) Post-Proceedings of WRS 2003. *Electronic Notes in Theoretical Computer Science* **86**. (*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. (eds.) Proceedings CSL, 12th International Workshop. *Springer-Verlag Lecture Notes in Computer Science* **1584** 298–312.

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* in *Journal of Symbolic Logic* **68** (4) 1415–1416.)

Parigot, M. (1992) $\lambda\mu$-calculus: an algorithmic interpretation of classic natural deduction. In: Voronkov, A. (ed.) Logic Programming and Automated Reasoning: Proceedings International Conference LPAR'92. *Springer-Verlag Lecture Notes in Computer Science* **624** 190–201.

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. (ed.) Proceedings of FoSSaCS 2004. *Springer-Verlag Lecture Notes in Computer Science* **2987** 423–437.

Rocheteau, J. (2005) $\lambda\mu$-calculus and duality: call-by-name and call-by-value. In: Giesl, J. (ed.) Term Rewriting and Applications: Proceedings of RTA 2005. *Springer-Verlag Lecture Notes in Computer Science* **3467** 204–218.

Sabry, A. and Wadler, P. (1997) A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems* **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*, Ph.D. thesis, University of Cambridge.