Monad Translating Inductive and Coinductive Types

Tarmo Uustalu

Inst. of Cybernetics, Tallinn Technical University
Akadeemia tee 21, EE-12618 Tallinn, Estonia
tarmo@cs.ioc.ee

Abstract. We show that the call-by-name monad translation of simply typed lambda calculus extended with sum and product types extends to special and general inductive and coinductive types so that its crucial property of preserving typings and β- and commuting reductions is maintained. Specific similar-purpose translations such as CPS translations follow from the general monad translations by specialization for appropriate concrete monads.

1 Introduction

Thanks to the work of Moggi [14], monads have become a popular structuring device in programming language semantics, as many notions of computation have the structure of a monad together with additional operations. Monad translations are a generic tool used for assigning semantics to impure languages from which specific similar-purpose translations such as continuation passing style (CPS) translations can be derived by specialization.

Inductive and coinductive types are a mechanism for introducing initial algebras and final coalgebras into programming languages and type theory to deal with wellfounded resp. non-wellfounded data structures. They are central in type-theoretic proof assistants, but also appear in experimental programming languages inspired from type theory and categorical logic such as Charity [6].

In this paper, our project is to investigate monad translatability of languages with inductive and coinductive types. The question has not been studied, but makes sense, as one may well conceive a language that has both inductive and coinductive types and impure features and then the question arises what its semantics should be. We show that Moggi's call-by-name monad translation of simply typed lambda calculus with sum and product types extends to natural number and stream types and further to general positive (co)inductive types, but also to (co)inductive types à la Mendler [13, 12]. The property of preserving typings and β- and commuting reductions is maintained in each case except for that of positive (co)inductive types in which case a special reformulation of the system is needed to achieve it. This is, however, not to be pitied as the translation of the alternative system of Mendler-style (co)inductive types turns out to be smoother anyway.

The project grew out of earlier work of Gilles Barthe and the author [4] on CPS translatability of inductive and coinductive types. The call-by-name CPS translations proposed in that paper follow from the translations developed here as specializations.

The organization of the paper is the following. In Sec. 2, we recall the call-by-name monad translation of simply typed lambda calculus with sum and product types, which we then generalize to natural number and stream types in Sec. 3 and further to positive inductive and coinductive types in Sec. 4. Sec. 5 contains the smoother translation of (co)inductive types à la Mendler. In Sec. 6, we show how the monad translations can be specialized into CPS translations. Sec. 7 contains some conluding remarks.

2 Monad translation of simply typed lambda calculus with sums and products

We begin with a recapitulation of the call-by-name monad translation of the system $\lambda_{+,\times}$, the standard extension of simply typed lambda calculus with sums and products that also is the term calculus of full intuitionistic propositional logic. The language of types and terms of $\lambda_{+,\times}$ is given by the grammar

$$\begin{aligned} A, B, C &::= X \mid A \to B \mid A_1 + A_2 \mid 0 \mid A_1 \times A_2 \mid 1 \\ M, N, P &::= x \mid \lambda x.\ M \mid N\ P \\ &\qquad \mid \mathsf{inl}(M) \mid \mathsf{inr}(M) \mid \mathsf{case}(N, u.\ P_1, u.\ P_2) \mid \nabla(N) \\ &\qquad \mid \langle M_1, M_2 \rangle \mid \langle\rangle \mid \mathsf{fst}(N) \mid \mathsf{snd}(N) \end{aligned}$$

and the typing and reduction rules are as given in Fig. 1. $\mathsf{elim}(N)$ is our general notation for any destructor-term; in the case of, e.g., $\lambda_{+,\times}$, this covers the term forms $N\ P$, $\mathsf{case}(N, u.\ P_1, u.\ P_2)$, $\nabla(N)$, $\mathsf{fst}(N)$, $\mathsf{snd}(N)$.

The target system of the translation, $\lambda_{\Diamond,+,\times}$, is Moggi's computational lambda calculus [14] extended with sum and product types. This calculus adds to $\lambda_{+,\times}$ a type constructor $\Diamond$ producing what are called computation types. The types and terms are determined by the grammar

$$\begin{aligned} A, B, C &::= \ldots \mid \Diamond A \\ M, N, P &::= \ldots \mid \mathsf{val}(M) \mid \mathsf{bind}(N, u.\ P) \end{aligned}$$

and the rules of typing and reduction are those presented in Fig. 2.

The semantic intention is that $\Diamond$ should denote an unspecified strong monad; the unit would be $\lambda x.\ \mathsf{val}(x)$, the (internalized) extension operation would be $\lambda f.\lambda x.\ \mathsf{bind}(x, u.f\ u)$. The typing and reduction rules for $\Diamond$ force those properties of a strong monad that fit into an intensional system (the reasonable η-reduction rule, not included here, is $\mathsf{bind}(N, u.\ \mathsf{val}(u)) \rhd N$). The Curry-Howard counterpart of $\lambda_\Diamond$ is a meaningful system of intuitionistic modal logic known variously as computational logic or lax logic [5, 8] and first conceived already by Curry [7]. In this logic, $\Diamond$ is a modality reminding of the S4 possibility modality (the difference

Typing environment formation rules:

$$\frac{}{\diamond \text{ env}} \qquad \frac{\Gamma \text{ env}}{\Gamma, x : C \text{ env}}$$

Typing rules:

$$\frac{}{\Gamma \vdash x : C} \quad \text{if } (x : C) \text{ in } \Gamma \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : A \to B} \qquad \frac{\Gamma \vdash N : A \to B \quad \Gamma \vdash P : A}{\Gamma \vdash N\ P : B}$$

$$\frac{\Gamma \vdash M : A_1}{\Gamma \vdash \mathsf{inl}(M) : A_1 + A_2} \qquad \frac{\Gamma \vdash M : A_2}{\Gamma \vdash \mathsf{inr}(M) : A_1 + A_2}$$

$$\frac{\Gamma \vdash N : A_1 + A_2 \quad \Gamma, u : A_1 \vdash P_1 : C \quad \Gamma, u : A_2 \vdash P_2 : C}{\Gamma \vdash \mathsf{case}(N, u.\ P_1, u.\ P_2) : C} \qquad \frac{\Gamma \vdash N : 0}{\Gamma \vdash \nabla(N) : C}$$

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \times A_2} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash N : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}(N) : A_1} \qquad \frac{\Gamma \vdash N : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}(N) : A_2}$$

β-reduction rules:

$$\begin{aligned}
(\lambda x.\ M)\ P &\rhd M[P/x] \\
\mathsf{case}(\mathsf{inl}(M), u.\ P_1, u.\ P_2) &\rhd P_1[M/u] \\
\mathsf{case}(\mathsf{inr}(M), u.\ P_1, u.\ P_2) &\rhd P_2[M/u] \\
\mathsf{fst}(\langle M_1, M_2 \rangle) &\rhd M_1 \\
\mathsf{snd}(\langle M_1, M_2 \rangle) &\rhd M_2
\end{aligned}$$

c-reduction rules (commuting conversion rules):

$$\mathsf{elim}(\mathsf{case}(N, u.\ P_1, u.\ P_2)) \rhd \mathsf{case}(N, u.\ \mathsf{elim}(P_1), u.\ \mathsf{elim}(P_2))$$

Fig. 1. Typing and reduction rules of $\lambda_{+,\times}$

Typing rules:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{val}(M) : \Diamond A} \qquad \frac{\Gamma \vdash N : \Diamond A \quad \Gamma, u : A \vdash P : \Diamond C}{\Gamma \vdash \mathsf{bind}(N, u.\ P) : \Diamond C}$$

β-reduction rules:

$$\mathsf{bind}(\mathsf{val}(M), u.\ P) \rhd P[M/u]$$

c-reduction rules (commuting conversion rules):

$$\mathsf{bind}(\mathsf{bind}(N, u.\ P), u'.\ P') \rhd \mathsf{bind}(N, u.\ \mathsf{bind}(P, u'.\ P')) \quad (u \notin \mathrm{FV}(P'))$$

Fig. 2. Typing and reduction rules of $\lambda_{\Diamond,+,\times}$

being that, while the S4 possibility only is a monad, the computational logic modality is a strong monad). (The most common notation for the modality of computational logic these days is $\bigcirc$. We follow [7] in using $\Diamond$.) The natural deduction formulations of the introduction and elimination rules of $\Diamond$ are

$$\dfrac{A}{\Diamond A}\ \Diamond\mathcal{I} \qquad\qquad \dfrac{\Diamond A \quad \begin{array}{c} A \\ \vdots \\ \Diamond C \end{array}}{\Diamond C}\ \Diamond\mathcal{E}$$

The reduction rules for the type constructor $\Diamond$ reappear as proof normalization rules for removing detours where a $\Diamond$-elimination immediately follows a $\Diamond$-introduction and pushing an elimination up past a $\Diamond$-elimination (observe that, in fact, only another $\Diamond$-elimination can immediately follow a $\Diamond$-elimination):

$$\dfrac{\begin{array}{c} \vdots \\ A \\ \hline \Diamond A \end{array}\ \Diamond\mathcal{I} \quad \begin{array}{c} A \\ \vdots \\ \Diamond C \end{array}}{\Diamond C}\ \Diamond\mathcal{E} \quad \rhd \quad \begin{array}{c} \vdots \\ A \\ \vdots \\ \Diamond C \end{array}$$

$$\dfrac{\dfrac{\begin{array}{c} \vdots \\ \Diamond A \end{array} \quad \begin{array}{c} A \\ \vdots \\ \Diamond C \end{array}}{\Diamond C}\ \Diamond\mathcal{E} \quad \begin{array}{c} C \\ \vdots \\ \Diamond D \end{array}}{\Diamond D}\ \Diamond\mathcal{E} \quad \rhd \quad \dfrac{\begin{array}{c} \vdots \\ \Diamond A \end{array} \quad \dfrac{\begin{array}{c} A \\ \vdots \\ \Diamond C \end{array} \quad \begin{array}{c} C \\ \vdots \\ \Diamond D \end{array}}{\Diamond D}\ \Diamond\mathcal{E}}{\Diamond D}\ \Diamond\mathcal{E}$$

The translation, cf. [14, 18], is given in figure Fig. 3. The main translation function for types prefixes every type in a type by $\Diamond$. In the term translation, a term's constructor subterms are prefixed by val while each destructor subterm is replaced by bind applied to its decomposition into a term and an evaluation context for it. The crucial metatheoretic property the translation enjoys is correctness in the sense of preservation of typings and non-identity (at least one step) reductions; the proof depends in part on a substitution lemma that essentially just says that the translation is compositional.

Lemma 1 (Substitution lemma).

1. $\textsc{mst}[\![A]\!][\textsc{ms}[\![C]\!]/Z] = \textsc{mst}[\![A[C/Z]]\!]$.
2. $\textsc{ms}[\![M]\!][\textsc{ms}[\![P]\!]/u] = \textsc{ms}[\![M[P/u]]\!]$.

Proof. Induction on A resp. M.

Theorem 1 (Correctness of the translation of $\lambda_{+,\times}$).

1. *If* $\Gamma \vdash M : A$ *in* $\lambda_{+,\times}$, *then* $\textsc{mst}[\![\Gamma]\!] \vdash \textsc{ms}[\![M]\!] : \textsc{mst}[\![A]\!]$ *in* $\lambda_{\Diamond,+,\times}$.
2. *If* $M \rhd_{\beta c} M'$ *in* $\lambda_{+,\times}$, *then* $\textsc{ms}[\![M]\!] \rhd^{+}_{\beta c} \textsc{ms}[\![M']\!]$ *in* $\lambda_{\Diamond,+,\times}$.

Proof. 1. Induction on the derivation of $\Gamma \vdash M : A$. 2. Induction on the derivation of $M \rhd_{\beta c} M'$.

Translation of typing environments:

$$
\begin{aligned}
\textsc{mst}[\![\diamond]\!] &= \diamond \\
\textsc{mst}[\![\Gamma, x : C]\!] &= \textsc{mst}[\![\Gamma]\!], x : \textsc{mst}[\![C]\!]
\end{aligned}
$$

Translation of types:

$$
\begin{aligned}
\textsc{mst}[\![A]\!] &= \Diamond \textsc{ms}[\![A]\!] \\
\textsc{ms}[\![X]\!] &= X \\
\textsc{ms}[\![A \to B]\!] &= \textsc{mst}[\![A]\!] \to \textsc{mst}[\![B]\!] \\
\textsc{ms}[\![A_1 + A_2]\!] &= \textsc{mst}[\![A_1]\!] + \textsc{mst}[\![A_2]\!] \\
\textsc{ms}[\![0]\!] &= 0 \\
\textsc{ms}[\![A_1 \times A_2]\!] &= \textsc{mst}[\![A_1]\!] \times \textsc{mst}[\![A_2]\!] \\
\textsc{ms}[\![1]\!] &= 1
\end{aligned}
$$

Translation of terms:

$$
\begin{aligned}
\textsc{ms}[\![x]\!] &= x \\
\textsc{ms}[\![\lambda x.\ M]\!] &= \mathsf{val}(\lambda x.\ \textsc{ms}[\![M]\!]) \\
\textsc{ms}[\![N\ P]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ n\ \textsc{ms}[\![P]\!]) \\
\textsc{ms}[\![\mathsf{inl}(M)]\!] &= \mathsf{val}(\mathsf{inl}(\textsc{ms}[\![M]\!])) \\
\textsc{ms}[\![\mathsf{inr}(M)]\!] &= \mathsf{val}(\mathsf{inr}(\textsc{ms}[\![M]\!])) \\
\textsc{ms}[\![\mathsf{case}(N, u.\ P_1, u.\ P_2)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{case}(n, u.\ \textsc{ms}[\![P_1]\!], u.\ \textsc{ms}[\![P_2]\!])) \\
\textsc{ms}[\![\nabla(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \nabla(n)) \\
\textsc{ms}[\![\langle M_1, M_2 \rangle]\!] &= \mathsf{val}(\langle \textsc{ms}[\![M_1]\!], \textsc{ms}[\![M_2]\!] \rangle) \\
\textsc{ms}[\![\mathsf{fst}(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{fst}(n)) \\
\textsc{ms}[\![\mathsf{snd}(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{snd}(n)) \\
\textsc{ms}[\![\langle\rangle]\!] &= \mathsf{val}(\langle\rangle)
\end{aligned}
$$

Fig. 3. Monad translation of $\lambda_{+,\times}$

3 Natural numbers and streams

Natural numbers and streams are the simplest non-trivial examples of inductive and coinductive types, but at the same time also quite representative examples. Hence it makes sense to first find out how these special inductive and coinductive types should be monad translated and only then proceed to the general case.

We consider a system with Burroni natural numbers (natural numbers over multiple zeros) and streams, coming equipped with iteration and coiteration. (We could have made some more advanced disciplined recursion and corecursion schemes available as primitive, but that would only have added complexity). The language of the system $\lambda_{\mathsf{Nat,Str}}$ is given by the grammar

$$\begin{aligned} A, B, C &::= \ldots \mid \mathsf{Nat}(A) \mid \mathsf{Str}(A) \\ M, N, P &::= \ldots \mid \mathsf{o}(M) \mid \mathsf{s}(M) \mid \mathsf{niter}(N, u.\ P_o, u.\ P_s) \\ &\quad \mid \mathsf{scoit}(M, y.\ Q_h, y, Q_t) \mid \mathsf{hd}(N) \mid \mathsf{tl}(N) \end{aligned}$$

while the typing and reduction calculus is given by the rules in Fig. 4. o, s stand, of course, for the constructors zero and successor, hd, tl are for head and tail. niter and scoit are operators for defining functions by iteration resp. coiteration.

Typing rules:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{o}(M) : \mathsf{Nat}(A)} \qquad \frac{\Gamma \vdash M : \mathsf{Nat}(A)}{\Gamma \vdash \mathsf{s}(M) : \mathsf{Nat}(A)}$$

$$\frac{\Gamma \vdash N : \mathsf{Nat}(A) \quad \Gamma, u : A \vdash P_o : C \quad \Gamma, u : C \vdash P_s : C}{\Gamma \vdash \mathsf{niter}(N, u.\ P_o, u.\ P_s) : C}$$

$$\frac{\Gamma \vdash M : D \quad \Gamma, y : D \vdash Q_h : A \quad \Gamma, y : D \vdash Q_t : D}{\Gamma \vdash \mathsf{scoit}(M, y.\ Q_h, y.\ Q_t) : \mathsf{Str}(A)}$$

$$\frac{\Gamma \vdash N : \mathsf{Str}(A)}{\Gamma \vdash \mathsf{hd}(N) : A} \qquad \frac{\Gamma \vdash N : \mathsf{Str}(A)}{\Gamma \vdash \mathsf{tl}(N) : \mathsf{Str}(A)}$$

$\beta-$reduction rules:

$$\begin{aligned} \mathsf{niter}(\mathsf{o}(M), u.\ P_o, u.\ P_s) &\rhd P_o[M/u] \\ \mathsf{niter}(\mathsf{s}(M), u.\ P_o, u.\ P_s) &\rhd P_s[\mathsf{niter}(M, u.\ P_o, u.\ P_s)/u] \\ \mathsf{hd}(\mathsf{scoit}(M, y.\ Q_h, y.\ Q_t)) &\rhd Q_h[M/y] \\ \mathsf{tl}(\mathsf{scoit}(M, y.\ Q_h, y.\ Q_t)) &\rhd \mathsf{scoit}(Q_t[M/y], y.\ Q_h, y.\ Q_t) \end{aligned}$$

Fig. 4. Typing and reduction rules of $\lambda_{\mathsf{Nat,Str}}$

How should $\lambda_{\mathsf{Nat,Str}}$ be translated? The type $\mathsf{Nat}(A) = \mu X.\ A + X$ is, in extensional settings, for any A, the least solution to the type equation $X \cong A + X$

Typing rules:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{o}^\diamond(M) : \mathsf{Nat}^\diamond(A)} \qquad \frac{\Gamma \vdash M : \Diamond\mathsf{Nat}^\diamond(A)}{\Gamma \vdash \mathsf{s}^\diamond(M) : \mathsf{Nat}^\diamond(A)}$$

$$\frac{\Gamma \vdash N : \mathsf{Nat}^\diamond(A) \quad \Gamma, u : A \vdash P_o : C \quad \Gamma, u : \Diamond C \vdash P_s : C}{\Gamma \vdash \mathsf{niter}^\diamond(N, u.\ P_o, u.\ P_s) : C}$$

$$\frac{\Gamma \vdash M : D \quad \Gamma, y : D \vdash Q_h : A \quad \Gamma, y : D \vdash Q_t : \Diamond D}{\Gamma \vdash \mathsf{scoit}^\diamond(M, y.\ Q_h, y.\ Q_t) : \mathsf{Str}^\diamond(A)}$$

$$\frac{\Gamma \vdash N : \mathsf{Str}^\diamond(A)}{\Gamma \vdash \mathsf{hd}^\diamond(N) : A} \qquad \frac{\Gamma \vdash N : \mathsf{Str}^\diamond(A)}{\Gamma \vdash \mathsf{tl}^\diamond(N) : \Diamond\mathsf{Str}^\diamond(A)}$$

$\beta-$reduction rules:

$$\begin{aligned}
\mathsf{niter}^\diamond(\mathsf{o}^\diamond(M), u.\ P_o, u.\ P_s) &\rhd P_o[M/u] \\
\mathsf{niter}^\diamond(\mathsf{s}^\diamond(M), u.\ P_o, u.\ P_s) &\rhd P_s[\mathsf{bind}(M, n.\ \mathsf{val}(\mathsf{niter}^\diamond(n, u.\ P_o, u.\ P_s)))/u] \\
\mathsf{hd}^\diamond(\mathsf{scoit}^\diamond(M, y.\ Q_h, y.\ Q_t)) &\rhd Q_h[M/y] \\
\mathsf{tl}^\diamond(\mathsf{scoit}^\diamond(M, y.\ Q_h, y.\ Q_t)) &\rhd \mathsf{bind}(Q_t[M/y], m.\ \mathsf{val}(\mathsf{scoit}^\diamond(m, y.\ Q_h, y.\ Q_t)))
\end{aligned}$$

Fig. 5. Typing and reduction rules of $\lambda_{\Diamond,\mathsf{Nat}^\diamond,\mathsf{Str}^\diamond}$

and the type $\mathsf{Str}(A) = \nu X.\ A \times X$ the greatest solution to the type equation $X \cong A \times X$. We may conjecture there is a viable translation preserving these properties. Hence we want $\text{MS}[\![\mathsf{Nat}(A)]\!]$ to be the least solution to $X \cong \text{MST}[\![A]\!] + \Diamond X$ and $\text{MS}[\![\mathsf{Str}(A)]\!]$ to be the greatest solution to $X \cong \text{MST}[\![A]\!] \times \Diamond X$. This will be the case if $\text{MS}[\![\mathsf{Nat}(A)]\!] = \mu X.\ \text{MST}[\![A]\!] + \Diamond X$ and $\text{MS}[\![\mathsf{Str}(A)]\!] = \nu X.\ \text{MST}[\![A]\!] \times \Diamond X$. In the target system instead of the types $\mathsf{Nat}(A)$, $\mathsf{Str}(A)$ we therefore need types $\mathsf{Nat}^\diamond(A) = \mu X.\ A + \Diamond X$, $\mathsf{Str}^\diamond(A) = \nu X.\ A \times \Diamond X$. Hence we choose that the target system will be $\lambda_{\Diamond,\mathsf{Nat}^\diamond,\mathsf{Str}^\diamond}$, a system with computation types and modified natural number and stream types whose language we give by the grammar

$$\begin{aligned}
A, B, C &= \ldots \mid \mathsf{Nat}^\diamond(A) \mid \mathsf{Str}^\diamond(A) \\
M, N, P &= \ldots \mid \mathsf{o}^\diamond(M) \mid \mathsf{s}^\diamond(M) \mid \mathsf{niter}^\diamond(N, u.\ P_o, u.\ P_s) \\
&\quad \mid \mathsf{scoit}^\diamond(M, y.\ Q_h, y, Q_t) \mid \mathsf{hd}^\diamond(N) \mid \mathsf{tl}^\diamond(N)
\end{aligned}$$

and whose typing and reduction calculus we determine by the rules appearing in Fig. 5. A sensible translation of $\lambda_{\mathsf{Nat},\mathsf{Str}}$ to $\lambda_{\Diamond,\mathsf{Nat}^\diamond,\mathsf{Str}^\diamond}$ is presented in Fig. 6. The translations of o, s, niter, scoit, hd, tl nearly copy those of inl, inr, case, $\langle\cdot,\cdot\rangle$, fst, snd except that the translation of niter has an extra bind in the second side argument to $\mathsf{niter}^\diamond$ and the translation of scoit has an extra val in the second side argument to $\mathsf{scoit}^\diamond$. This translation has the correctness property.

Theorem 2 (Correctness of the translation of $\lambda_{\mathsf{Nat},\mathsf{Str}}$).

1. *If $\Gamma \vdash M : A$ in $\lambda_{\mathsf{Nat},\mathsf{Str}}$, then $\text{MST}[\![\Gamma]\!] \vdash \text{MS}[\![M]\!] : \text{MST}[\![A]\!]$ in $\lambda_{\Diamond,\mathsf{Nat}^\diamond,\mathsf{Str}^\diamond}$.*

Translation of types:

$$\begin{aligned}
\textsc{ms}[\![\mathsf{Nat}(A)]\!] &= \mathsf{Nat}^{\Diamond}(\textsc{mst}[\![A]\!]) \\
\textsc{ms}[\![\mathsf{Str}(A)]\!] &= \mathsf{Str}^{\Diamond}(\textsc{mst}[\![A]\!])
\end{aligned}$$

Translation of terms:

$$\begin{aligned}
\textsc{ms}[\![\mathsf{o}(M)]\!] &= \mathsf{val}(\mathsf{o}^{\Diamond}(\textsc{ms}[\![M]\!])) \\
\textsc{ms}[\![\mathsf{s}(M)]\!] &= \mathsf{val}(\mathsf{s}^{\Diamond}(\textsc{ms}[\![M]\!])) \\
\textsc{ms}[\![\mathsf{niter}(N, u.\ P_o, u.\ P_s)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], \\
&\qquad n.\ \mathsf{niter}^{\Diamond}(n, u.\ \textsc{ms}[\![P_o]\!], u.\ \textsc{ms}[\![P_s]\!][\mathsf{bind}(u, v.\ v)/u])) \\
\textsc{ms}[\![\mathsf{scoit}(M, y.\ Q_h, y.\ Q_t)]\!] &= \mathsf{val}(\mathsf{scoit}^{\Diamond}(\textsc{ms}[\![M]\!], y.\ \textsc{ms}[\![Q_h]\!], y.\ \mathsf{val}(\textsc{ms}[\![Q_t]\!]))) \\
\textsc{ms}[\![\mathsf{hd}(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{hd}^{\Diamond}(n)) \\
\textsc{ms}[\![\mathsf{tl}(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{tl}^{\Diamond}(n))
\end{aligned}$$

Fig. 6. Monad translation of $\lambda_{\mathsf{Nat},\mathsf{Str}}$

2. *If* $M \rhd_{\beta} M'$ *in* $\lambda_{\mathsf{Nat},\mathsf{Str}}$*, then* $\textsc{ms}[\![M]\!] \rhd_{\beta c}^{+} \textsc{ms}[\![M']\!]$ *in* $\lambda_{\Diamond,\mathsf{Nat}^{\Diamond},\mathsf{Str}^{\Diamond}}$.

It is worth noting that simulating of the second β-reduction rule of Nat (the niter-after-s rule) takes an application of the commuting reduction rule of $\Diamond$.

4 Positive inductive and coinductive types

Once we have a good monad translation for natural numbers and streams, finding a similar translation for general inductive and coinductive types is not hard. The standard approach to general (co)inductive types is based on the syntactic notion of positivity of a type transformer. For the sake of simplicity, we forbid interleaved (co)inductive types. The language of the extension of $\lambda_{+,\times}$ with positive non-interleaving inductive and coinductive types (with iteration and coiteration as the primitively available (co)recursion schemes), $\lambda_{+,\times,\mu,\nu}$ (cf., e.g., [11, 9]) is given by the grammar

$$\begin{aligned}
A, B, C &::= \ldots \mid \mu Z.\ A \mid \nu Z.\ A \\
M, N, P &::= \ldots \mid \mathsf{in}_{Z.\,A}(M) \mid \mathsf{iter}(N, u.\ P) \mid \mathsf{coit}(M, y.\ Q) \mid \mathsf{out}_{Z.\,A}(N) \\
&\qquad (\lambda Z.\ A \text{ positive and with no free occurrence of } Z \text{ under a } \mu \text{ or a } \nu)
\end{aligned}$$

The typing and reduction rules of $\lambda_{+,\times,\mu,\nu}$ are those in Fig. 7. The operator $Map_{Z.\,A}$ ($\lambda Z.\ A$ positive and with no free occurrence of Z under a μ or a ν), that appears in the reduction rules, is a defined operator such that

$$\frac{\Gamma \vdash N : A[D/Z] \quad \Gamma, u : D \vdash P : C}{\Gamma \vdash Map_{Z.\,A}(N, u.\ P) : A[C/Z]}$$

and

$$Map_{Z.\,A}(Map_{Z.\,A}(N, u.\ P), u'.\ P') \rhd^{+}_{\beta c} Map_{Z.\,A}(N, u.\ P'[P/u']) \quad (u \notin \mathrm{FV}(P'))$$

These lemmata say that *Map* delivers composition-preserving monotonicity witnesses for positive type transformers. The definition of *Map* is given in Fig. 8. It is simultaneous with the definition of another operator $\overline{Map}$ delivering composition-preserving antimonotonicity witnesses for negative type transformers. In a system supporting interleaved (co)inductive types, *Map* and $\overline{Map}$ would have to be defined differently and then some commuting reduction rules would have to be postulated for μ, ν to validate the reduction for *Map*—we do not want to discuss the controversial issue of commuting reductions for μ and ν here.

Typing rules:

$$\frac{\Gamma \vdash M : A[\mu Z.\ A/Z]}{\Gamma \vdash \mathsf{in}_{Z.\,A}(M) : \mu Z.\ A} \qquad \frac{\Gamma \vdash N : \mu Z.\ A \quad \Gamma, u : A[C/Z] \vdash P : C}{\Gamma \vdash \mathsf{iter}(N, u.\ P) : C}$$

$$\frac{\Gamma \vdash M : D \quad \Gamma, y : D \vdash Q : A[D/Z]}{\Gamma \vdash \mathsf{coit}(M, y.\ Q) : \nu Z.\ A} \qquad \frac{\Gamma \vdash N : \nu Z.\ A}{\Gamma \vdash \mathsf{out}_{Z.\,A}(N) : A[\nu Z.\ A/Z]}$$

β-reduction rules:

$$\mathsf{iter}(\mathsf{in}_{Z.\,A}(M), u.\ P) \rhd P[Map_{Z.\,A}(M, n.\ \mathsf{iter}(n, u.\ P))/u]$$
$$\mathsf{out}_{Z.\,A}(\mathsf{coit}(M, y.\ Q)) \rhd Map_{Z.\,A}(Q[M/y], m.\ \mathsf{coit}(m, y.\ Q))$$

Fig. 7. Typing and reduction rules of $\lambda_{+,\times,\mu,\nu}$

A monad translation of $\lambda_{+,\times,\mu,\nu}$, with $\lambda_{\Diamond,+,\times,\mu,\nu}$ as the target system, is presented in Fig. 9. It mimicks the translation of the previous section, but with one important difference: in the translation of the natural numbers, $\mathsf{Nat}(A) = \mu Z.\ A + Z$ is rendered by $\textsc{ms}[\![\cdot]\!]$ not as $\mu Z.\ \Diamond A + \Diamond Z$, but instead as $\mu Z.\ \Diamond(\Diamond A + \Diamond Z)$. The reason for the change is that while we previously essentially considered only the connection μ-after-$+$, now μ is an independent type-forming operator. Note that in the clauses for iter and coit there are references to $\textsc{mst}[\![A]\!]$ subjected to a "backwards substitution" $[Z/\Diamond Z]$. In $\textsc{mst}[\![A]\!]$, all occurrences of Z are diamonded; $[Z/\Diamond Z]$ is meant as a notation for the operation of undiamonding them. These backwards substitutions are, of course, avoidable: one may, alternatively, introduce two auxiliary translations $\textsc{mst}_Z[\![\cdot]\!]$, $\textsc{ms}_Z[\![\cdot]\!]$, defined exactly as $\textsc{mst}[\![\cdot]\!]$, $\textsc{ms}[\![\cdot]\!]$ except that $\textsc{mst}_Z[\![Z]\!] = Z$ and $\textsc{mst}_Z[\![A]\!] = \Diamond\textsc{ms}_Z[\![A]\!]$ only if $A \neq Z$. Then $\textsc{mst}[\![A]\!][Z/\Diamond Z] = \textsc{mst}_Z[\![A]\!]$.

This translation, unfortunately, is defective compared to the translations previously considered. It has the substitution property and preserves typings, but to prove preservation of reductions, one would need a lemma that

$$Map_{Z.\ \textsc{mst}[\![A]\!][Z/\Diamond Z]}(\textsc{ms}[\![M]\!], z.\ \textsc{ms}[\![P]\!]) \rhd^{+}_{\beta c} \textsc{ms}[\![Map_{Z.\ A}(M, z.\ P)]\!]$$

$$
\begin{aligned}
Map_{Z.\ Z}(N, z.\ P) &= P[N/u] \\
Map_{Z.\ Y}(N, z.\ P) &= N \quad (Z \neq Y) \\
Map_{Z.\ A \to B}(N, z.\ P) &= \lambda x.\ Map_{Z.\ B}(N\ \overline{Map}_{Z.\ A}(x, z.\ P), z.\ P) \\
Map_{Z.\ A_1 + A_2}(N, z.\ P) &= \mathsf{case}(N, u.\ \mathsf{inl}(Map_{Z.\ A_1}(u, z.\ P)), u.\ \mathsf{inr}(Map_{Z.\ A_2}(u, z.\ P))) \\
Map_{Z.\ 0}(N, z.\ P) &= \nabla(N) \\
Map_{Z.\ A_1 \times A_2}(N, z.\ P) &= \langle Map_{Z.\ A_1}(\mathsf{fst}(M), z.\ P), Map_{Z.\ A_2}(\mathsf{snd}(M), z.\ P)\rangle \\
Map_{Z.\ 1}(N, z.\ P) &= \langle\rangle \\
Map_{Z.\ \mu Z'.\ A}(N, z.\ P) &= N \quad (Z \notin \mathrm{FV}(\lambda Z'.\ A)) \\
Map_{Z.\ \nu Z'.\ A}(N, z.\ P) &= N \quad (Z \notin \mathrm{FV}(\lambda Z'.\ A))
\end{aligned}
$$

$$
\begin{aligned}
\overline{Map}_{Z.\ Y}(N, z.\ P) &= N \quad (Z \neq Y) \\
\overline{Map}_{Z.\ A \to B}(N, z.\ P) &= \lambda x.\ \overline{Map}_{Z.\ B}(N\ Map_{Z.\ A}(x, z.\ P), z.\ P) \\
\overline{Map}_{Z.\ A_1 + A_2}(N, z.\ P) &= \mathsf{case}(N, u.\ \mathsf{inl}(\overline{Map}_{Z.\ A_1}(u, z.\ P)), u.\ \mathsf{inr}(\overline{Map}_{Z.\ A_2}(u, z.\ P))) \\
\overline{Map}_{Z.\ 0}(N, z.\ P) &= \nabla(N) \\
\overline{Map}_{Z.\ A_1 \times A_2}(N, z.\ P) &= \langle \overline{Map}_{Z.\ A_1}(\mathsf{fst}(M), z.\ P), \overline{Map}_{Z.\ A_2}(\mathsf{snd}(M), z.\ P)\rangle \\
\overline{Map}_{Z.\ 1}(N, z.\ P) &= \langle\rangle \\
\overline{Map}_{Z.\ \mu Z'.\ A}(N, z.\ P) &= N \quad (Z \notin \mathrm{FV}(\lambda Z'.\ A)) \\
\overline{Map}_{Z.\ \nu Z'.\ A}(N, z.\ P) &= N \quad (Z \notin \mathrm{FV}(\lambda Z'.\ A))
\end{aligned}
$$

Fig. 8. Definition of Map and $\overline{Map}$ for $\lambda_{+,\times,\mu,\nu}$

Translation of types:

$$
\begin{aligned}
\textsc{ms}[\![\mu Z.\ A]\!] &= \mu Z.\ \textsc{mst}[\![A]\!] \\
\textsc{ms}[\![\nu Z.\ A]\!] &= \nu Z.\ \textsc{mst}[\![A]\!]
\end{aligned}
$$

Translation of terms:

$$
\begin{aligned}
\textsc{ms}[\![\mathsf{in}_{Z.\ A}(M)]\!] &= \mathsf{val}(\mathsf{in}_{Z.\textsc{mst}[\![A]\!]}(\textsc{ms}[\![M]\!])) \\
\textsc{ms}[\![\mathsf{iter}(N, u.\ P)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], \\
&\qquad n.\ \mathsf{iter}(n, u.\ \textsc{ms}[\![P]\!][Map_{Z.\textsc{mst}[\![A]\!][Z/\Diamond Z]}(u, z.\ \mathsf{bind}(z, v.\ v))/u])) \\
\textsc{ms}[\![\mathsf{coit}(M, y.\ Q)]\!] &= \mathsf{val}(\mathsf{coit}(\textsc{ms}[\![M]\!], y.\ Map_{Z.\textsc{mst}[\![A]\!][Z/\Diamond Z]}(\textsc{ms}[\![Q]\!], z.\ \mathsf{val}(z)))) \\
\textsc{ms}[\![\mathsf{out}_{Z.\ A}(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{out}_{Z.\textsc{mst}[\![A]\!]}(n))
\end{aligned}
$$

Fig. 9. Monad translation of $\lambda_{+,\times,\mu,\nu}$

which is to say that *Map* and $\text{MS}[\![\cdot]\!]$ commute in a certain technical sense. This lemma, however, fails for $\lambda_{+,\times,\mu,\nu}$ as presented. The technical cause of the failure is the non-destructor format of some *Map*-terms, e.g., $\mathit{Map}_{Z.\ A_1\times A_2}(M, z.\ P) = \langle \mathit{Map}_{Z.\ A_1}(\mathsf{fst}(M), z.\ P), \mathit{Map}_{Z.\ A_2}(\mathsf{snd}(M), z.\ P)\rangle$, which are constructor terms. For such *Map*-terms, the reduction rules for $\Diamond$ are too weak to validate the commutation of *Map* and $\text{MS}[\![\cdot]\!]$. The problem would disappear for product types, if they came with splitting

$$\frac{\Gamma \vdash N : A_1 \times A_2 \quad \Gamma, x_1 : A_1, x_2 : A_2 \vdash P : C}{\Gamma \vdash \mathsf{split}(N, x_1.\ x_2.\ P) : C}$$

$$\mathsf{split}(\langle M_1, M_2\rangle, x_1.\ x_2.\ P) \rhd P[M_1/x_1, M_2/x_2]$$
$$\mathsf{elim}(\mathsf{split}(N, x_1.\ x_2.\ P)) \rhd \mathsf{split}(N, x_1.\ x_2.\ \mathsf{elim}(P))$$

instead of projections, in which case one would define $\mathit{Map}_{Z.\ A_1\times A_2}(M, z.\ P) = \mathsf{split}(M, x_1.\ x_2.\ \langle \mathit{Map}_{Z.\ A_1}(x_1, z.\ P), \mathit{Map}_{Z.\ A_2}(x_2, z.\ P)\rangle)$. But is this not really a direction to go as the clumsiness of the system and the translation remains. We take an alternative direction.

5 Inductive and coinductive types à la Mendler

The problems with the monad translation are symptomatic to $\lambda_{+,\times,\mu,\nu}$. The source of troubles with $\lambda_{+,\times,\mu,\nu}$ lies in the deployment of the notion of positivity and the operator *Map*: these are defined outside the system by induction over the language of types, so that if, e.g., a new connective or modality is added, these definitions would have to be revised adequately. An alternative approach, due to Nax Mendler [12, 13] (see also [11, 9, 17]), which is free of this shortcoming and therefore sometimes gives smoother solutions, is essentially based on the type isomorphisms $A[C/Z] \cong \exists Z.\ (Z \to C) \times A$, $A[C/Z] \cong \forall Z.\ (C \to Z) \to A$, which hold in extensional settings with parametricity whenever $\lambda Z.\ A$ is a functor (in particular, when it is positive). In this approach, $\mu Z.\ A$ and $\nu Z.\ A$ are always legal types, which is achieved by treating them as notations of the least and greatest fixedpoint of $\lambda Z.\ A$, but those of $\lambda Z'.\ \exists Z.\ (Z \to Z') \times A$ resp. $\lambda Z'.\ \forall Z.\ (Z' \to Z) \to A$ ($Z' \notin \mathrm{FV}(A)$). These type transformers are positive independent of whether the original type transformer $\lambda Z.\ A$ is or is not. We now proceed to providing a monad translation for Mendler-style (co)inductive types.

We choose one possible setup of a system with Mendler-style (co)inductive types, which we here denote $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$. Its language is given by the grammar

$$\begin{aligned} A, B, C &::= \ldots \mid \mu Z.\ A \mid \nu Z.\ A \\ M, N, P &::= \ldots \mid f[M] \\ &\quad \mid \ldots \mid \mathsf{in}(M) \mid \mathsf{iter}^{\mathrm{m}}(N, u.\ f.\ P) \mid \mathsf{coit}^{\mathrm{m}}(M, y.\ f.\ Q) \mid \mathsf{out}(N) \end{aligned}$$

and the typing and reduction calculus is specified by the rules in Fig. 10. Here, f is a metavariable for a context variable; $f[M]$ is the result of instantiating the context f with term M. The notation $M[P/f[z]]$ stands for the result of

Typing environment formation rules:

$$\frac{\Gamma \text{ env}}{\Gamma, (z : D \vdash f[z] : C) \text{ env}}$$

Typing rules:

$$\frac{\Gamma \vdash N : D}{\Gamma \vdash f[N] : C} \quad (z : D \vdash f[z] : C) \text{ in } \Gamma$$

$$\frac{\Gamma \vdash M : A[\mu Z.\ A/Z]}{\Gamma \vdash \mathsf{in}(M) : \mu Z.\ A} \qquad \frac{\Gamma \vdash N : \mu Z.\ A \quad \Gamma, u : A, (z : Z \vdash f[z] : C) \vdash P : C}{\Gamma \vdash \mathsf{iter}^{\mathrm{m}}(N, u.\ f.\ P) : C} \quad Z \text{ fresh}$$

$$\frac{\Gamma \vdash M : D \quad \Gamma, y : D, (z : D \vdash f[z] : Z) \vdash Q : A}{\Gamma \vdash \mathsf{coit}^{\mathrm{m}}(M, y.\ f.\ Q) : \nu Z.\ A} \quad Z \text{ fresh} \qquad \frac{\Gamma \vdash N : \nu Z.\ A}{\Gamma \vdash \mathsf{out}(N) : A[\nu Z.\ A/Z]}$$

β-reduction rules:

$$\mathsf{iter}^{\mathrm{m}}(\mathsf{in}(M), u.\ f.\ P) \rhd P[M/u, \mathsf{iter}^{\mathrm{m}}(z, u.\ f.\ P)/f[z]]$$
$$\mathsf{out}(\mathsf{coit}^{\mathrm{m}}(M, y.\ f.\ Q)) \rhd Q[M/y, \mathsf{coit}^{\mathrm{m}}(z, y.\ f.\ Q)/f[z]]$$

Fig. 10. Typing and reduction rules of $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$

replacing in M every subterm $f[N]$ with $P[N/z]$ (cf. the structural substitutions of Parigot's $\lambda\mu$-calculus [15]).

Although this system involves elements of higher-order abstract syntax (as used in logical frameworks), this higher-orderness is quite shallow. It features context variables and instantiation of a context variable, but no context-producing operators, in particular, no schematization of a term. The operators $\mathsf{iter}^{\mathrm{m}}$ and $\mathsf{coit}^{\mathrm{m}}$ are third-order. (Cf. the formulation of Parigot's $\lambda\mu$ in [1] where μ-variables are treated as context variables and the μ-operator is third-order.) Alternatively, context variables could have been replaced by ordinary function-type variables and context instantiation by application. The chosen formulation is, nevertheless, in some ways neater, e.g., several essentially bureaucratic β-reductions in simulations become instantiations, i.e., happen silently.

The monad translation of $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$ is given in Fig. 11. It is clearly smoother than that of $\lambda_{+,\times,\mu,\nu}$ in Fig. 9. Also, it is worth noting that while an application is translated into bind of its decomposition into a term and an evaluation context, a context instantiation translates back into a context instantiation. This is because although function types are translated into function types prefixed by $\Diamond$, subordinate entailments should reasonably be translated into subordinate entailments (similarly to top-level entailments). As a consequence, we avoid the production in the translation of a number of val's and bind's that would only be bureaucratic. The translation preserves typings and reductions.

Theorem 3 (Correctness of the translation of $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$).

Translation of typing environments:

$$\textsc{mst}[\![\Gamma, (z : D \vdash f[z] : C)]\!] = \textsc{mst}[\![\Gamma]\!], (z : \textsc{mst}[\![D]\!] \vdash f[z] : \textsc{mst}[\![C]\!])$$

Translation of types:

$$\begin{aligned} \textsc{ms}[\![\mu Z.\ A]\!] &= \mu Z.\ \textsc{mst}[\![A]\!] \\ \textsc{ms}[\![\nu Z.\ A]\!] &= \nu Z.\ \textsc{mst}[\![A]\!] \end{aligned}$$

Translation of terms:

$$\textsc{ms}[\![f[N]]\!] = f[\textsc{ms}[\![N]\!]]$$

$$\begin{aligned} \textsc{ms}[\![\mathsf{in}(M)]\!] &= \mathsf{val}(\mathsf{in}(\textsc{ms}[\![M]\!])) \\ \textsc{ms}[\![\mathsf{iter}^{\mathrm{m}}(N, u.\ f.\ P)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{iter}^{\mathrm{m}}(n, u.\ f.\ \textsc{ms}[\![P]\!][\mathsf{bind}(z, v.\ f[v])/f[z]] \\ \textsc{ms}[\![\mathsf{coit}^{\mathrm{m}}(M, y.\ f.\ Q)]\!] &= \mathsf{val}(\mathsf{coit}^{\mathrm{m}}(\textsc{ms}[\![M]\!], y.\ f.\ \textsc{ms}[\![Q]\!][\mathsf{val}(f[z])/f[z]])) \\ \textsc{ms}[\![\mathsf{out}(N)]\!] &= \mathsf{bind}(\textsc{ms}[\![N]\!], n.\ \mathsf{out}(n)) \end{aligned}$$

Fig. 11. Monad translation of $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$

1. *If* $\Gamma \vdash M : A$, *then* $\textsc{mst}[\![\Gamma]\!] \vdash \textsc{ms}[\![M]\!] : \textsc{mst}[\![A]\!]$.
2. *If* $M \rhd_{\beta c} M'$, *then* $\textsc{ms}[\![M]\!] \rhd^{+}_{\beta c} \textsc{ms}[\![M']\!]$.

6 Example specialization: CPS translations

Specific translations are obtainable from monad translations by specializing for an appropriate specific monad, cf. [10]. Some standard examples of monads relevant for programming language semantics are the following:

- The continuations monad:

$$\begin{aligned} \Diamond A &= \neg\neg A \\ \mathsf{val}(M) &= \lambda k.\ k\ M \\ \mathsf{bind}(N, u.\ P) &= \lambda k.\ N\ (\lambda u.\ (P\ k)) \end{aligned}$$

  ($\neg A = A \to \bot$ where $\bot$ is some chosen type for answers)
- The state transformer monad:

$$\begin{aligned} \Diamond A &= S \to A \times S \\ \mathsf{val}(M) &= \lambda s.\ \langle M, s\rangle \\ \mathsf{bind}(N, u.\ P) &= \lambda s.\ P[\mathsf{fst}(N\ s)/u]\ \mathsf{snd}(N\ s) \end{aligned}$$

  (S is some chosen type for states)

– The exceptions monad:

$$\begin{aligned} \Diamond A &= A + E \\ \mathsf{val}(M) &= \mathsf{inl}(M) \\ \mathsf{bind}(N, u.\ P) &= \mathsf{case}(N, u.\ P, u.\ \mathsf{inr}(u)) \end{aligned}$$

(E is some chosen type for exceptions)

For the specialization to validate the β- and commuting reductions of $\Diamond$ in the case of the exceptions monad, it is essential to have the commuting reductions of $+$ in the target system. In the cases of the continuation and state monad, even the η-equality rules of $\to$ and $\times$ are needed, but the specialized monad translation is easily fine-tuned so that β- and commuting reductions are preserved also without the presence of these rules in the target system.

We consider the example of continuations. Inlining of the concrete definitions of $\Diamond$, val, bind followed by a minor optimization move take the monad translations of $\lambda_{+,\times}$, $\lambda_{+,\times,\mu,\nu}$ and $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$ from Figs. 3, 9, 11 to translations presented in Figs. 12, 13, 14, which are call-by-name CPS translations à la Plotkin [16] in a "near-colon" version; a discussion of these translations (except for the translation for $\lambda^{\mathrm{m}}_{+,\times,\mu,\nu}$) and of the application of combining (co)inductive types and control can be found in [4].

7 Conclusion

We have shown that the monad translation of simply typed lambda calculus to computational lambda calculus extends to inductive and coinductive types so that β- and commuting reductions are preserved. In the general case this works best for (co)inductive types à la Mendler, as the notions of positivity and monotonicity witness are then avoided. From the general monad translations, specific translations such as translations to continuation and state passing styles can be derived by specialization. This provides a way to assign semantics to languages combining inductive and coinductive types with impure features.

One possible continuation would be to consider extensions to dependently typed and higher-order settings along the lines of [2, 3]. But a more important and interesting exercise will be to give a semantic (category-theoretic) foundation to the translations. This worked out, we intend to consider also tighter monad translations (such as call-by-value) and their behavior wrt. stronger notions of reduction or equality of terms (such as $\beta c\eta$-equality). In the present paper, we deliberately chose to confine ourselves to a purely syntactic ("type-theoretic") study of the problem of finding call-by-name monad translations behaving well wrt. βc-reduction.

Acknowledgements This work was prompted by the outcomes from earlier joint work of Gilles Barthe and the author on CPS translation for (co)inductive types. The author is grateful to Gilles Barthe for several insights. He is also grateful to his three referees for a number of useful comments and suggestions.

Translation of types:

$$
\begin{aligned}
\textsc{cpst}[\![A]\!] &= \neg\neg\textsc{cps}[\![A]\!] \\
\textsc{cps}[\![X]\!] &= X \\
\textsc{cps}[\![A \to B]\!] &= \textsc{cpst}[\![A]\!] \to \textsc{cpst}[\![B]\!] \\
\textsc{cps}[\![A_1 + A_2]\!] &= \textsc{cpst}[\![A_1]\!] + \textsc{cpst}[\![A_2]\!] \\
\textsc{cps}[\![0]\!] &= 0 \\
\textsc{cps}[\![A_1 \times A_2]\!] &= \textsc{cpst}[\![A_1]\!] \times \textsc{cpst}[\![A_2]\!] \\
\textsc{cps}[\![1]\!] &= 1
\end{aligned}
$$

Translation of terms:

$$
\begin{aligned}
\textsc{cps}[\![M]\!] &= \lambda k.\ [\![M]\!]{:}k \\
[\![x]\!]{:}K &= x\ K \\
[\![\lambda x.\ M]\!]{:}K &= K\ (\lambda x.\ \textsc{cps}[\![M]\!]) \\
[\![N\ P]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ n\ \textsc{cps}[\![P]\!]\ K) \\
[\![\mathsf{inl}(M)]\!]{:}K &= K\ \mathsf{inl}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{inr}(M)]\!]{:}K &= K\ \mathsf{inr}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{case}(N, u.\ P_1, u.\ P_2)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{case}(n, u.\ \textsc{cps}[\![P_1]\!], u.\ \textsc{cps}[\![P_2]\!])\ K) \\
[\![\nabla(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \nabla(n)\ K) \\
[\![\langle M_1, M_2\rangle]\!]{:}K &= K\ \langle \textsc{cps}[\![M_1]\!], \textsc{cps}[\![M_2]\!]\rangle \\
[\![\mathsf{fst}(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{fst}(n)\ K) \\
[\![\mathsf{snd}(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{snd}(n)\ K) \\
[\![\langle\rangle]\!]{:}K &= K\ \langle\rangle
\end{aligned}
$$

Fig. 12. CPS translation of $\lambda_{+,\times}$

Translation of types:

$$
\begin{aligned}
\textsc{cps}[\![\mu Z.\ A]\!] &= \mu Z.\ \textsc{cpst}[\![A]\!] \\
\textsc{cps}[\![\nu Z.\ A]\!] &= \nu Z.\ \textsc{cpst}[\![A]\!]
\end{aligned}
$$

Translation of terms:

$$
\begin{aligned}
[\![\mathsf{in}_{Z.\,A}(M)]\!]{:}K &= K\ \mathsf{in}_{Z.\textsc{cpst}[\![A]\!]}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{iter}(N, u.\ P)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{iter}(n, \\
&\qquad u.\ \textsc{cps}[\![P]\!][Map_{Z.\textsc{cpst}[\![A]\!][Z/\neg\neg Z]}(u, z.\ \lambda k'.\ z\ (\lambda z'.\ z'\ k'))/u])\ K) \\
[\![\mathsf{coit}(M, y.\ Q)]\!]{:}K &= K\ \mathsf{coit}(\textsc{cps}[\![M]\!], y.\ Map_{Z.\textsc{cpst}[\![A]\!][Z/\neg\neg Z]}(\textsc{cps}[\![Q]\!], z.\ \lambda k'.\ k'\ z)) \\
[\![\mathsf{out}_{Z.\,A}(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{out}_{Z.\textsc{cpst}[\![A]\!]}(n)\ K)
\end{aligned}
$$

Fig. 13. CPS translation of $\lambda_{+,\times,\mu,\nu}$

Translation of types:

$$\textsc{cps}[\![\mu Z.\ A]\!] = \mu Z.\ \textsc{cpst}[\![A]\!]$$
$$\textsc{cps}[\![\nu Z.\ A]\!] = \nu Z.\ \textsc{cpst}[\![A]\!]$$

Translation of terms:

$$\begin{aligned}
[\![\mathsf{in}(M)]\!]{:}K &= K\ \mathsf{in}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{iter}^{\mathrm{m}}(N, u.\ f.\ P)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{iter}^{\mathrm{m}}(n, u.\ f.\ \textsc{cps}[\![P]\!][\lambda k'.\ f[z]\ (\lambda z'.\ z'\ k')/f[z]])\ K) \\
[\![\mathsf{coit}^{\mathrm{m}}(M, y.\ f.\ Q)]\!]{:}K &= K\ \mathsf{coit}^{\mathrm{m}}(\textsc{cps}[\![M]\!], y.\ f.\ \textsc{cps}[\![Q]\!][\lambda k'.\ k'\ f[z]/f[z]]) \\
[\![\mathsf{out}(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{out}(n)\ K)
\end{aligned}$$

Fig. 14. CPS translation of $\lambda^{m}_{+,\times,\mu,\nu}$

The research was done while the author was on leave to Dep. de Informática, Univ. do Minho, Campus de Gualtar, P-4710-057, Braga, Portugal. It was partially supported by the Portuguese Foundation for Science and Technology under grant No. PRAXIS XXI/C/EEI/14172/98 and by the Estonian Science Foundation under grant No. 4155. The author's participation at TYPES'02 was supported by the FP5 IST thematic network TYPES.

References

1. Abel, A.: A third-order representation of the $\lambda\mu$-calculus. In: Ambler, S. J., Crole, R. L., Momigliano, A. (eds.): Proc. of Wksh. on Mechanised Reasoning about Languages with Variable Binding, MERLIN 2001 (Siena, June 2001). Electr. Notes in Theor. Comput. Sci., Vol. 58(1). Elsevier, Amsterdam (2001)
2. Barthe, G., Hatcliff, J., Sørensen, M. H. B.: CPS translations and applications: The cube and beyond. Higher-Order and Symbolic Comput. **12**(2) (1999) 125–170
3. Barthe, G., Hatcliff, J., Thiemann, P.: Monadic type systems: Pure type systems for impure settings (preliminary report). In: Gordon, A., Pitts, A., Talcott, C. (eds.): Proc. of 2nd Wksh. on Higher-Order Operational Techniques in Semantics, HOOTS'97 (Stanford Univ., CA, Dec. 1997). Electr. Notes in Theor. Comput. Sci., Vol. 10. Elsevier, Amsterdam (1998)
4. Barthe, G., Uustalu, T.: CPS translating inductive and coinductive types (extended abstract). In: Proc. of 2002 ACM SIGPLAN Wksh. on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'02 (Portland, OR, Jan. 2002). SIGPLAN Notices **37**(3). ACM Press, New York (2002) 131–142
5. Benton, P. N., Bierman, G. M., de Paiva, V. C. V.: Computational types from a logical perspective. J. of Funct. Prog. **8**(2) (1998) 177–193
6. Cockett, R., Fukushima, T.: About Charity. Yellow Series Report 92/480/18, Dept. of Computer Science, Univ. of Calgary (1992)
7. Curry, H. B.: The elimination theorem when modality is present. J. of Symb. Logic **17**(4) (1952) 249–265
8. Fairtlough, M., Mendler, M.: Propositional lax logic. Inform. and Comput. **137**(1) (1997) 1–33

9. Geuvers, H.: Inductive and coinductive types with iteration and recursion. In: Nordström, B., Pettersson, K., Plotkin, G. (eds.): Proc. of Wksh. on Types for Proofs and Programs (Båstad, June 1992). Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ. (1992) 193–217
10. Hatcliff, J., Danvy, O.: A generic account of continuation-passing styles. In: Conf. Record of 21st Ann. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'94 (Portland, OR, Jan. 1994). ACM Press, New York (1994) 458–471
11. Leivant, D.: Contracting proofs to programs. In: Odifreddi, P. (ed.): Logic and Computer Science. APIC Studies in Data Processing, Vol. 31. Academic Press, London (1990) 279–327
12. Mendler, N. P.: Recursive types and type constraints in second-order lambda-calculus. In: Proc. of 2nd Ann. IEEE Symp. on Logic in Computer Science, LICS'87 (Ithaca, NY, June 1987). IEEE CS Press, Washington, DC (1987) 30–36
13. Mendler, N. P.: Inductive types and type constraints in the second-order lambda-calculus. Ann. of Pure and Appl. Logic, **51**(1–2) (1991) 159–172
14. Moggi, E.: Notions of computation and monads. Inform. and Comput. **93**(1) (1991) 55–92
15. Parigot, M.: $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.): Proc. of Int. Conf. on Logic Programming and Automated Reasoning, LPAR'92 (St Petersburg, July 1992). Lect. Notes in Artif. Intell., Vol. 624. Springer-Verlag, Berlin (1992) 190–201
16. Plotkin, G. D:: Call-by-name, call-by-value and the λ-calculus. Theor. Comput. Sci. **1**(2) (1975) 125–159
17. Spławski, Z., Urzyczyn, P.: Type fixpoints: Iteration vs. recursion. In: Proc. of 4th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'99 (Paris, Sept. 1999). SIGPLAN Notices **34**(9). ACM Press, New York (1999) 102–113
18. Wadler, P.: Comprehending monads. Math. Struct. in Comp. Sci. **2**(4) (1992) 461–493